2010

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

Rez-O-Lution:

A Ticket Management System for The Office of Housing and Residence Life at The University of North Carolina Wilmington

Christopher Cotton

A Capstone Project Submitted to the University of North Carolina Wilmington in Partial Fulfillment of the Requirements for the Degree of Master of Science

Department of Computer Science

Department of Information Systems and Operations Management

University of North Carolina Wilmington

2010

Contents

1 Introduction

This paper documents the planning, analysis, design, and development of Rez-O-lution: An issue submission and tracking "Ticket Management System". The paper will review the need for the system, the key users, the minimum system requirements, and the challenges in developing the solution.

2 Motivation

2.1 Terminology

The terminology section of this document explores those terms that are used within the document that may be unconventional or unclear.

2.1.1 Swimlanes

Section 5.5 discusses swimlane diagrams. These diagrams depict a process, who is involved in it, and what course of events must take place. The reason why these diagrams are called swimlanes is because they resemble the lanes of a swimming pool. All of the diagrams for this project use the lanes to represent a person, actor, or user and how they relate to a particular process.

2.1.2 Rez-O-lution

The name of the system that will be designed and developed for this project is named "Rez-O-lution." Throughout this document, the Rez-O-lution system will be called by its name but assume that any reference to "a system" or "the system" is a reference to the Rez-O-lution system unless otherwise noted.

2.1.3 Happy Path

The term "happy path" is used significantly in section 5.3 in association with use cases and also in section 5.4 in association with the state diagram. A "happy path" describes a path within the system that is most frequently traveled or that is intended to be traveled. Any deviation from this is considered an alternate path. The happy path is important because one can gain certain important metrics on frequency and load within the system.

2.2 Housing and Residence Life

The University of North Carolina Wilmington (UNCW) was established as Wilmington College in 1947 and began with an enrollment of 238 students. Since its inceptions almost 62 years ago, UNCW has grown to enroll nearly 13,000 students a year and has graduated over 53,000 students graduate. The University uses a unique blend of teaching, research experiences, and service learning opportunities that attract high-quality students and gives its graduates a competitive advantage in the marketplace. U.S. News & World Report has ranked UNCW as one of the top 10 public master's universities in the south for 12 straight years and UNCW has also been listed as one of the top 16 "up-and-coming" master's universities in the south for 2 straight years. (http://uncw.edu/facts/)

The University is made up of the College of Arts and Sciences, Cameron School of Business, the School of Nursing, the Watson School of Education, and the Graduate School. UNCW offers 52 majors, 35 master's degrees, a PhD in Marine Biology, and an EdD in Educational Leadership and Administration.

The University Administration is divided into 7 divisions which include the Chancellors Office, Academic Affairs, Business Affairs, Public Service & Continuing Studies, Information Technology Systems, Student Affairs, and University Advancement.

UNCW's department of Housing and Residence Life is a unit within the Student Affairs division. The goal of the Housing and Residence Life (HRL) department is best described in its mission statement (Appendix A.1).

The mission statement declares that the Office of Housing and Residence Life is designed to help the students of UNCW develop their full potential through support of their educational growth and personal development. HRL also strives to provide students with a satisfactory physical environment that emphasizes the utilization of human and material resources in an efficient manner. Housing also tries to provide students with educational opportunities that will help them attain a greater appreciation and understanding of cultural and lifestyle differences as well as leadership and self-governance opportunities. (http://www.uncw.edu/stuaff/housing/mission.htm)

The Office of HRL also states that in order to establish a student community, students need to understand that every action they do has a consequence and they need to take responsibility for those actions. Housing provides trained professional staff to help assist students in understanding the established guidelines for behavioral expectations of all UNCW students.

The organizational chart describing the structure of Housing can be found in appendix D.1. The Director of HRL directly reports to the Vice Chancellor for Student Affairs. Two Associate Directors report to the Director as do two Assignments coordinators, one Receptionist, and one Assistant Director. The rest of the staff report directly to their respective Associate Directors, Assistant Director, Assignments Coordinator, or Receptionist.

2.3 Statement of the Problem

2.3.1 Current Issue Resolving Process

The current process used to identify an issue that needs the attention of the Office of Housing HRL is depicted in the flow chart provided by Appendix F.1. Housing representatives are required to deal with a wide range of issues which can include but are not limited to Housing contract cancellations, roommate conflicts, and maintenance requests. The amounts of issues that are reported to HRL vary depending on the time of year. Students report about half of the issues while parents report the other half. This is because students commonly report issues about living conditions while parents are usually more interested in financial situations. Very few issues are reported by other sources such as other university staff.

When a parent or student has an issue that requires the attention of the Office of HRL, they usually call or, more commonly, email Housing by at housing@uncw.edu. The requests are reviewed by the current Assistant Director of Communications and Technology and delegated to the appropriate Housing staff member. Once an issue is assigned to a staff member, they perform actions to resolve the issue.

One of the identified problems is that in some cases, the person who submitted the problem doesn't receive a confirmation that their issue has been resolved and there is almost no means of tracking the particular request.

There are some cases in which a student may see a HRL staff member and report an issue to them in verbally. If this is the case, the staff member either resolves the issue or they can report it to the Assistant Director of Communications and Technology who then routes the issue to the appropriate HRL staff member. These cases are not as frequent as the cases in which a student or parent calls HRL or emails them, but these cases can occur and therefore reflected in the diagram in appendix F.1.

2.3.2 Problems with the Current Resolution Process

The current issue resolving process is very "trust-based." That is, those people who make HRL aware of an issue expect that their issue comes to some resolution but they have no means of tracking their problem and, as mentioned before, they may not even be aware that their problem has been "resolved."

In the current process, the "Housing Office" is ultimately held responsible for the resolution of an issue. This means there is very little individual accountability within Housing.

The current process also consumes a lot of time and lacks efficiency. As mentioned before, the Assistant Director of Communication and Technology has to read every email sent to Housing and then forward it to a staff member for resolution. This is a time consuming process and creates a "middle-man" with unnecessary delay.

Another problem associated with the current issue resolving process is the fact that there is no way to report on the process. There is no way of telling how many issues were resolved, precisely how long it takes to resolve an issue, or if there are still issues open. In addition no summary report is currently available to detail how many were resolved for a period (i.e. semester). Finally it is hard to determine if the same issue occurs on a regular or semester basis.

2.3.3 Opportunities for Improvement

The anticipated benefit from the proposed system includes:

- Provides users with a convenient way to report issues to Housing
- Reduces the amount of time for Housing to be aware of issues
- Reduces the amount of time to resolve an issue
- Provides a way to audit actions taken by Housing in the process of resolving a ticket
- Increases accountability within Housing
- Generates aggregate reports about issue counts and resolution times
- Notifies the ticket creator and staff members of actions taken during the ticket resolution process
- Reduces the amount of time that the Assistant Director of Communication and Technology has to spend distributing issues to Housing staff members.

2.4 Review of General Ticket Systems

As part of the analysis phase of this project, several third party trouble ticketing/bug tracking systems were evaluated. Two of those systems were found to have a feature set that came close to meeting the project requirements. In the following sections I will review and discuss the two third party systems and evaluate their strengths and weaknesses related to this project. The two ticket systems are:

- *BMC's Remedy Service Desk*
  The developer decided to explore this system is because it is already in use by UNCW as a means for the staff of UNCW to submit technology related tickets to UNCW's information technology systems division. This is a software package that the university the university has purchased. This is an enterprise software package, meaning that UNCW also pays for support and other features. Unfortunately, I was unable to collect any information on how much the university pays for the product and the company that provides it prices it out differently according to the client.
- *TicketDesk*
  TicketDesk is designed to facilitate communications between help desk staff and end users. It is designed to be as simple and frictionless for both the help desk staff and end users as possible. TicketDesk is an asp.NET web application written in the C# language targeting the .NET 3.5 framework. It includes a simple database with support for SQL Server 2005 and 2008 (including the express editions). It can leverage SQL server for membership and role based security or integrate with windows authentication and Active Directory groups.

Both of the systems share a common set of features. These include:

- Notifications to the help desk staff and end users when new tickets arrive and are assigned
- Notifications to the end users when certain events and state changes occur to a ticket (however the BMC Remedy Ticket tells a user that a ticket has been resolved, but does not tell them "WHICH" ticket has been resolved. Thus if a user has more than one ticket in the system, the user really doesn't know which has been resolved. In addition if two different users enter the same request, the 2nd requester is send notice that the ticket has been resolved even if it has not been resolved, since the system only wants one active ticket per incident.
- The ability to view tickets and change information about a ticket
- The ability to enter a trouble ticket via a web form.

Along with the basic features provided by the TicketDesk and Remedy systems, each system provides their own interesting set of features. For example, the TicketDesk system provides an extremely simple interface for the both the help desk staff members and end users. In addition, the TicketDesk system is also open source. This means that an organization that uses this system can completely modify it to meet their needs. The Remedy issue tracking provides an organization that is paying to use the system third-party support and upgrades, but one of the most interesting features that Remedy provides is the ability to provide notifications to help desk staff members through means other than email (i.e. pagers).

2.5 Justification

2.5.1 Build vs. Buy vs. Customize

Systems Analysis methodologies detail three alternative approaches to system development and implementation, they are:

- Build a new system to solve the problem
- Buy a system that can solve the problem at hand
- Customize an existing system

As with every decision, there are pros and cons to each method and the decision to use a certain method completely depends on the situation, time constraints, and money available to the ones in need of the solution.

2.5.1.1 Build

When people choose to develop a new system, they usually do so because:

- The system does not exist
- Significant customization is required for a purchased system
- Built 'exactly' to the client's needs and wants with no unnecessary features
- Little to no upfront costs associated with it

Some of the attributes to building a new system are actually downsides to the methodology as well.

- It may take a significant amount of time to create the new system.
- Even when the system is deployed, it may have some bugs that were not caught during the design and development of the system
    - making it necessary to invest more time into the system
- No outside support should something go wrong
    - The people who have made the system have to rely on themselves to troubleshoot whatever problems that arise
- Long term management as staffing changes over time

The option to develop a new system is beneficial for the office of HRL because it will be completely customized for their needs. The Rez-O-lution system will provide only those features that are important to HRL including a ticket entry page that doesn't require a user to log in. They will also own the code at the completion of this project so that they can make future modifications to the system as they need. HRL made it clear that they wanted a solution that had minimal costs associated with it and the option to develop a new system provides that. The downsides associated with developing a new system for housing is that they may change their requirements during the development process. In this case, it will be very difficult to rework what has already been developed to accommodate these changes in requirements. To confront this problem, the developer has consulted with HRL and developed a clear set of requirements that will be adhered to. The possibility of maintenance and new requests within the system after its completion is also a problem that housing will be incurred in the future in order to

minimize this impact; the developer will provide documentation on the system so that resolving any bugs will be easier. Finally, housing will not have any outside support after the project is completion. UNCW's information technology division will be responsible for hosting the Rez-O-lution system as agreed upon during discussions involving Housing and ITSD in the spring of 2010.

2.5.1.2 Buy

Another alternative to the 'make' is the buy option. This can include a new system or software. This is beneficial because:

- It usually doesn't require a lot of time to implement the solution
    - The installation and training of staff are usually the biggest consumers of time when people decide to buy a system
- The buyer usually gets outside support
    - This can include things like technical support, bug fixing, patching, etc
- Upgrading the system is generally easier because the vendor usually bears the responsibility.

On the other hand, buying a system presents its own set of problems. These problems include:

- The purchased system may not have all of the features needed or wanted or, it may be feature heavy
- Purchasing only the software and not the source code
    - Cannot make any changes deep within the system
- Cost
    - There is usually an upfront cost and most vendors charge monthly/annual fees for using their solution
    - This cost can add up and become very large over time

Housing could benefit from buying a system that has already been developed because they would be a quick, relatively easy, solution that would most likely be supported by the vendor of the product. The problem with this option is that housing is looking for a solution that cost very little, if anything. Another problem with purchasing a solution is that it may not contain all of the features and/or it may contain too many features than they are looking for in a solution.

2.5.1.3 Customize

The customize option combines characteristics of both the make option and buy option. Usually, in a custom system a client will buy a system or a part of a system and then add in or remove parts of the system that they need or don't need. A custom system can be:

- A less expensive solution to a problem than buying a solution
- The client usually has ownership of the code
    - Customized to their needs
- The client may have third-party support
- Less time is needed to implement the system than if a client were to build an entirely new system

- The end system usually contains most if not all of the features a client needs/wants and little to none of the features they don't need.

Some of what makes a custom system an attractive solution to a problem can also pose as problems.

- Sometimes custom solutions are not supported by a third-party
- More expensive time-wise than a solution that is bought and installed
- Custom solution may not cost as much as a bought solution, it usually costs some money
    - As opposed to a solution that is made which usually costs little to no money
- Upgrades
    - If an upgrade is made to the base software on which the custom solution is built, the custom solution may run into problems or break completely as the custom code may not interact well with the upgraded source code

Housing would benefit from a custom solution because it would allow them to take a pre-developed solution and modify it for their needs and wants. The problem with this option is that upgrades can be complicated. Also, even though custom solutions can be relatively inexpensive, there is still a cost associated with them. Housing has made it clear that they are not willing to spend any money for whatever solution they decide to go with.

2.5.2 Why Housing Chose to Have a New System Developed

Representatives from UNCW's Housing and Residence Life sat down with the developer to discuss the current process in Housing and discuss some possible solutions for them. It was decided that the best solution for them would come from designing and building a system to combat the complexity created by their current process. Housing, in addition to the developer, has decided this was the best option because:

- It would be a simple solution and it would be tailored for the desired processes of Housing and Residence Life
    - The system that will be created will be simple in the sense that it will be designed to contain those features specific to housing. The system should be relatively simple to build and once built, it should be easy to maintain. Also, at the completion of the project, Housing will own the code so that they may continue future system development.
- It would provide its creator (in this instance, myself) with much needed experience in system design and development, and implementation.
    - The benefit derived from designing and building the system will be the experience of working with new software tools, patterns and practices. Also, the implementation of analysis and project management best practices will also be learned.
- It would be inexpensive (in the short term). Another attractive benefit of building a new system for Housing is the fact that it will be inexpensive. It will not cost Housing any money to make this system except for the man-hours it takes to design and implement it.

3 Project Environment & Processes

3.1 Basic Development Methodology

For this project, it was decided that a combination of the Spiral Iterative Model with the help of Rapid Application Development (RAD) tools would be used. That is to say that the Spiral model was used for the initial design and analysis of the project while RAD application tools and during the creation and implementation of features not accounted for in the design of the project to gain a better understanding of them. The actual programming paradigm that was used for this project was an object-oriented approach.

3.1.1 Spiral Iterative Model

The spiral model follows the System Development Life-Cycle and is iterative. Every iteration contained 3 activities: Add feature, test feature, re-factor. The System Development Life-Cycle (SDLC) is composed of 5 different stages. Each stage is done in a linear sequence. The five stages are: system/information engineering and modeling, software requirement analysis, system analysis and design, code generation, testing, and maintenance. The first three stages (system/information engineering and modeling, software requirement analysis, and system analysis and design) for this project were done in the fall of 2009 while the code generation and testing phases were done in the spring of 2010. The maintenance phase will be left for ITSD.

During the development of the system, some features that were not identified during the initial planning were developed using Rapid Application Development TOOLS(see section 3.1.2).

3.1.2 Rapid Application Development (RAD)

The RAD software development methodology uses minimal planning in favor of rapid prototyping. Most of the software development using RAD is concerned with the writing of the software itself. The minimal pre-planning usually makes the development of software faster and easier to change requirements on-the-fly. The actual design of this project did not follow the RAD model as seen from the various design documents included in the appendices, but there were times when RAD tools were used to gain a better understanding of RAD application tools as well as create features within the system that were not recognized during the initial planning of the system.

The initial intention, before planning and design had started, was that this application would be built using the RAD methodology. This did not occur as it became clear, early in the design stage, that extensive documentation would make the development of the project smoother.

3.1.3 Project Cycles

Figure 1 depicts the project's development cycle. From this figure, one can see that during the development of this project, a new feature was added then the new feature was tested, and then the whole project was re-factored before another new feature was added. As part of the re-factor process, feedback was collected from all of the primary users (those users who would be utilizing the system on a regular basis) and actions were taken based on this feedback. This cycle continued until the project's completion.

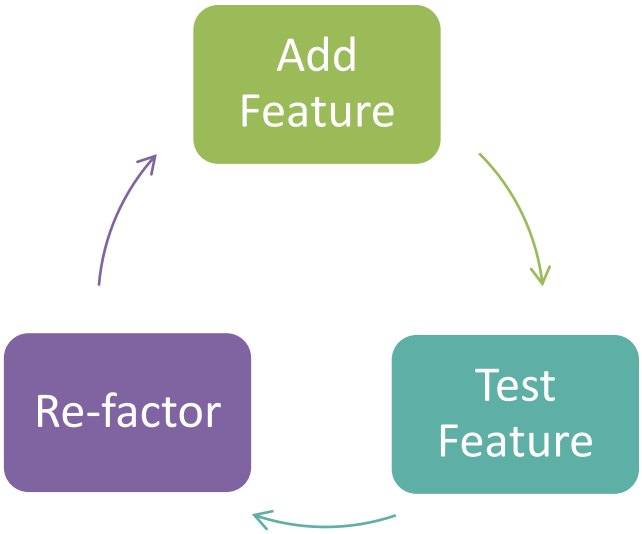


Figure 1: The Project Development Cycle

3.1.4 Object-Oriented Paradigm

The object-oriented approach to application development focuses on the creation of design and implementation of objects. It is the interaction of all of these objects that defines what a system is and what it can do. Some of the key features of object-oriented programming are data abstraction, encapsulation, modularity, polymorphism and inheritance.

Evidence that the object-oriented approach was used for the planning of this project is found in the abstraction of this system. The Rez-O-lution system has a high level of abstraction with distinct layers that need to be traversed when action within the system occurs.

3.1.5 Scaffolding

Scaffolding is a RAD tool that is a dynamic, data-driven, way to create software with minimal to no code. It is able to create fully functional software and web-sties very quickly based on a pre-designed data model.

Scaffolding was not used to develop the Rez-O-lution system. The scaffolding tools provided by Microsoft's Visual Studio were used briefly to see what would be produced and it was concluded that it would take require too much time and effort to modify the pre-constructed web-application that the scaffolding tool created. The result of the scaffolding tool was very detailed but not very cohesive. It was also not very user friendly as what was initially produced required more than a basic understanding of databases. Essentially what was created with the scaffolding tool is a visual representation of the tables in the data base and a web application that allows a user to enter data into the tables of the database.

3.1.6 Justification

The combination of the Spiral model and RAD tools was chosen for this project. This is seen from the amount of up-front planning that was done but there were times during the development of the Rez-O-lution system where something may have been designed and then it may have had to change. This may be because the client may have wanted it to look or act different or a feature was needed and not planned for. The Rez-O-lution system had some changes and additions made to it prior to its completion but because RAD tools were chosen for the development of the system in addition to the planning done with the spiral model, it was created quickly it was presented to the client for feedback, and then changes were made as they needed and wanted.

3.2 Project Locker

The Rez-O-lution system code and diagrams was hosted by ProjectLocker, a free, web-based software management solution, while it is being developed. ProjectLocker is a full-serviced managed development platform for software development teams. It allows developers ways to share code and, track issues, ensure code quality, and analyze project processes; all of which are essential to the software creation process. The decision to use ProjectLocker came about after initial attempts to install Microsoft SharePoint 3.0 and Team Foundation met with issues related to domain account credentials. ProjectLocker worked out well for this project as it was reliable and easy to use.

3.2.1 Source control

Source control allows developers to track and manage changes within code and documentation. ProjectLocker manages the hosting of Subversion, the source control software we are using for this project. The subversion client that was chosen was TortoiseSVN.

3.2.2 Collaboration

For the collaboration and issue tracking portions, ProjectLocker offers Trac. Trac allowed us to create a wiki and a source code browser that connected to our source control repository (Subversion). Trac also allowed us to quickly get a comprehensive view of defects, features, and the overall state of a project. Not all of the features of Trac were used for this project. It really was useful in the early stages of planning for this project when details of the system were still being recognized. Weekly accomplishments were recorded in Trac. Trac was not used much during the development stage of the project as accomplishments were more tangible (i.e. the creation of a stored procedure) and were recorded in SVN check-ins.

3.2.3 Unit testing

The software verification and validation method in which a programmer tests if individual units of source code are fit for use is known as unit testing. In an application, a unit is the smallest testable part. The unit tests written are designed to make sure that code does what it is intended to do and doesn't do anything else.

- As with any design process, testing is an important step. Unit testing is something that the developer of this system did not have much previous experience. Unit testing would have been very helpful and was going to be utilized but because of the narrow time constraint and the

fact that the developer had his plate full learning many other new techniques including, but not limited to, MS SQL design and implementation, application development with C#, application development with ASP .NET, and error handling with ELMAH, unit testing was cut from the project.

Overall testing is an important part of software development and although unit testing was not used, detailed testing was done. Testing was done during all of the iterations of the project. Because the project went through multiple iterations, testing was done often at the completion of each iteration. The results of the testing allow the developer to recognize those things that needed to be added or changed for the next iteration.

3.3 Back Ups

During the project's development, all of the documentation and code associated with this project was hosted on ProjectLocker which guarantees a retrieval back of the project data from their servers. During the project, a development machine had a full, most recent copy of the project on it but to ensure that there was a working copy of the project at all times, a full copy of the project was also checked-out of ProjectLocker to another machine other than the development machine.

4 Platform

4.1 SQL Server

For this project, MS SQL Server 2008 was chosen to develop and deploy the backend data for the Rez-O-lution system. The developer was familiar with SQL Server 2008 as he worked with it on an almost-daily basis. Another reason why MS SQL Server 2008 was chosen is that Housing uses SQL Server 2005 to develop online reports conveying various aspects of their financial and resident occupancy that they deploy through SQL Server Reporting Services. This means that Housing will be comfortable with it as well. Additionally, MS SQL was chosen for this project because it is the University of North Carolina's Information Services Division standard backend database for smaller projects.

From a programming perspective, MS SQL Server was very versatile and easy to use. It was a very user-friendly piece of software and it complimented the fact that I developed this system in ASP.NET.

4.2 ASP.Net

This project is a web-based application and ASP.Net was selected to develop it. ASP.Net is very flexible and user friendly. My experience with ASP.Net prior to this project was minimal and this gave me the opportunity to learn. In addition, as was mentioned earlier, this project with be given to ITSD for long term support and ASP.Net is their chosen development platform for small customized solutions. Finally to assist with developmental endeavors, the chosen committee members for this project all had experience with ASP.Net and were a great resource.

This project gave the developer a chance to practice their knowledge and expand upon it. In preparation for this endeavor, the developer took a class with Dr. Thomas Janicki in which they were able to learn some new things and put them to practice. This experience helped the developer to gain

additional knowledge of the ASP.NET platform. Dr. Janicki was also asked to be a part of this project's committee as he was a valuable resource.

4.3 SQL Server Reporting Services

For the reporting aspect of this system, SQL Server Reporting Services (SSRS) was used. This allowed for the delivery of a variety of interactive reports. These reports were developed with relative ease within Visual Studio. SSRS can be accessed directly or through the Report Manager (the web-based application that interfaces with the Report Server web service). For the purposes of this project, I interacted with SSRS directly through a set of ASP.NET ReportViewer web controls for some of the reports I created. This allowed me to directly embed reports into the system's web pages.

5 System Analysis

An object-oriented approach has been taken during the design of this project. This can be seen in the multitude of design documents provided including, but not limited to, use cases, swimlane diagrams, actor diagrams, etc. A Rapid Application Development (RAD) approach was taken during the development of the project. A RAD approach was decided upon because the ultimate goal of this project was to create a working application for housing with less focus on the business analysis of it. This does not mean business analysis was ignored, but the overall focus was on the completion of this system versus the analysis of it. All of the diagrams and tools used helped aid in that goal.

5.1 Project Charter

The project charter for this system is found in appendix I.1. A project charter details the foundation of the project and allows all parties (clients, developers, and other stakeholders) to view, discuss, and agree upon a plan for the project. This includes everything from the processes within the system to the platform on which it will run.

This document went through two revisions before it was agreed upon by all stakeholders during the months of August, September, and October 2009. Detailed reports of these revisions were not kept because they were done before a version-tracking software had been installed for the use of this project; most of the changes made to the document were to the syntax of the document or the grammar. The most interesting points that were not part of the original document were the ideas of application contingency and integration capabilities. Both issues were discussed in the final project charter document as a result of the discussions I had with the stakeholders.

The project charter shown in appendix I.1 is the final document and it was reviewed and agreed upon by:

- My entire project committee which includes:
    - Dr. Douglas Kline (chair)
    - Dr. Ron Vetter
    - Charles Laymon
    - Dr. Tom Janicki
- The project's clients in UNCW's Housing and Residence life

- Mainly Sean Ahlum (Assistant Director)
- Brad Reid (Director)
- Nathan Miner (Associate Director)
- Allison Tucker (Associate Director)
- Nick Troutman (Associate Director)
- Larry Wray (Associate Director)

A brief discussion of the topics covered in the project charter is found below.

- Business Purpose and Objectives
  - The goal of this section is to make clear to the designer and clients the purpose of the system. This section also defines the problem at hand so that it can be solved as best as possible.
  - ***The purpose and objective of this project is to provide an online system for submitting these issues, routing the issues to the appropriate person, and tracking their resolution***.
- Desired Features
  - The desired features section of the project charted describes those features that HRL is looking for regarding the solution to their problem stated in the business purpose and objectives section.
  - These features are listed in the project charter in appendix I.1.
- Critical Success Factors
  - This section describes those items that will determine whether or not this project will be a success.
  - The things that will make this project a success are:
    - Simplicity for users
    - Ability to track events within the system through some logging mechanism
    - Audit control through a logging mechanism
    - Reliability of system operations
    - Continued ability for ITSD support and maintenance
- Constraints
  - The constraints section describes a set of limitations that the developer must consider during the course of the project
- Risks
  - This section describes those things that may be a risk to the project's success
  - ***One of the risks that were encountered during the development of the project that was not mentioned in the project charter was the idea of time constraints. Often times during the development of the project, some of the tasks that were done pushed the limits of the timeline provided.***
- Roles and Responsibilities
  - The roles section lists and describes those people who will be participating in the project or who have some stake in the project

- Actors
    - The actor section of the project charter describes those users who will be interacting with the system
    - These were identified early to assist in developing the project charter
    - This helps the developer understand who the system key users (internal and external)
        - This, in turn, leads to development decisions based on user experience levels
- Locations of Interest
    - The locations of interest section describes those places that will be important in the development of the proposed solution
        - More specifically, where the project will be developed and hosted
- Event list/Event Table
    - This section depicts the types of action that each actor can take within the system
    - This table gives the developer an idea of the load that will be exerted on the system
    - ***One of the most interesting events that is described in the table is that of the frequency of ticket creation by guests***
        - ***30-200 tickets are projected to be created per day***
- Use Cases
    - The use case section gives a preliminary list of those use cases that will be created for the solution
    - These will eventually be the interactions that a user will have with the system
    - The use cases were identified but not fleshed out until later
- Preliminary Execution Architecture
    - This section lists those tools that will be used during the project
- Project Infrastructure
    - The project infrastructure section describes the platform that will be used for developing and hosting the project
- Project Release Strategy
    - This section describes the method by which the project will be delivered to the client
    - The idea is that the project will be released incrementally (piece-by-piece)
- Application Contingency
    - The application contingency states who will be responsible for the continuation and maintenance of the system after it is completed and delivered to the client.

5.2 Actor Diagram

The final actor diagram is in appendix E.1. It depicts the actors that use the system, the components within the system and how each actor interacts with the system. Following is a list and description of each actor and component as well as an explanation of how they relate to each other.

5.2.1 Actors

The HRL organization chart is re-visited in appendix D.2. The chart has been laid out like the organization of housing but it has been modified and colored to match the actors within the Rez-O-lution system. It is important to note that a person's position within housing does not dictate their role

within the system except for the director as his role in the system does reflect his position within housing.

After the project was proposed to the project's committee, it was decided that a person can have more than one role in the system. That is, a person can be more than one actor.

5.2.1.1 Guest

The Guest actor represents those users that are primarily responsible for the creation of tickets. These are usually students of UNCW and/or their parents. A future enhancement would be to permit the Guests to be able to view tickets that they have previously submitted to the system. This would allow them to add notes to those tickets. This out of scope for this project as it is not a critical feature to the system.

5.2.1.2 Staff Member

The Staff Member actors are those users that are primarily responsible for the handling of issues within Housing. The types of people that can be staff members are represented by blue boxes in the HRL organization chart in appendix D.2. A staff member is able to view tickets that are assigned to them, add notes to tickets, resolve tickets, request that a ticket be assigned to another Staff Member. In addition they are able to update information about themselves within the system Another future enhancement would permit a staff members to manage what types of messages they receive from the system as well as permitting them to create tickets on behalf of Guests, These features were not considered critical to the success of the system so they were cut out of the project's scope.

5.2.1.3 System Manager

The System Manager is the actor that is primarily in charge of the day-to-day operations within the system. The system manager is represented as a green box in the HRL organization chart represented in appendix D.2. Their responsibilities include creating users within the system, managing information on users, assigning roles to users, viewing tickets in the system, adding notes to tickets, creating ticket categories, allocating Staff Members to ticket types, viewing reports, update information about themselves, and approve or deny requests for ticket reassignment by Staff Members.

5.2.1.4 Director

The Director actor plays a relatively minor role within the system. This actor does not directly affect any data within the system; rather they are mostly responsible for viewing and making sense of historical reports about the information within the system. The director actor will be the director of housing and this position is represented as a yellow box in the housing organization chart in appendix D.2.

5.2.1.5 System Administrator

The System Administrator actor plays a smaller role within the system. They cannot make any changes to data within the system. The System Administrator is responsible for the overall functionality of the system. They will be able to create logs of events happening within the system and they will be able to view historical reports to make sure that the data within the system is relevant and accurate. The system administrator is represented as a red box in the housing organization chart provided in appendix D.2.

5.2.2 Components of System

5.2.2.1 Manage Tickets

The manage tickets component of the system is the component in which users are able to create, update, and close tickets. The users who can interact with this component are guests, staff members, and system managers. The use cases (see section 5.3 for more on use cases) that are a part of this component are manage ticket (system manager), *manage ticket* (guest), and manage ticket (staff member). Staff members interact with the manage ticket (staff member) interface through a web form. Most of the time, a staff member will just close a ticket, but all of the ticket management features they are able to use are located in the manage ticket (staff member) use case. Guests are mainly responsible for the creation of tickets as seen in the manage ticket (guest) use case. System managers are primarily responsible for approving and denying requests that are submitted for ticket reassignments made by staff members.

5.2.2.2 Manage Profile

The manage profile component is that component which controls how a user can manage information about themselves. Every actor, except guests are able to utilize this component. The use case that relates to this component is *manage profile*. This component is not essential to the project's success but it is something was added to it as it was relatively easy to create and added to the overall functionality of the system. The manage profile component is driven by the manage profile use case. Users interact with this component through a web form.

5.2.2.3 Manage Messaging Options

The *manage messaging* options component controls how a user can manage which actions generate a notification for them. Every actor is able to utilize this component except for guests. The use case that relates to this component is manage messaging options. This is not an essential component and it was not added to this project but in order to be able to add it later, the data model was designed to supports this option. All of the users that have login information will be able to interact with the manage messaging options component. This component is driven by the manage messaging options use case and will be handled via a web form.

5.2.2.4 Manage Users and Roles

This component controls how users are created and updated within the system. System managers are responsible for creating users in the system and assigning them roles and ticket types but users are able to update personal information about themselves. The only use case that relates to this component is manage users. This component is essential component to the system's functionality. The manage users component is driven by the manage users use case and is represented as a combination of data grid views and web forms. It was decided that users will not be deleted from the Rez-O-lution system. If a system manager decides that a user is no longer needed within the system, they flag that user as inactive but the user remains in the system.

5.2.2.5 Login

The login component gives users with login information the ability to log into the Rez-O-lution system to perform various tasks that their assigned role allows them to perform. Every actor in the actor diagram

is able to log into the system except for guests. The login use case is the interface that drives this component. The *login* use case is controlled via a web form. This is an essential piece to the system as it helps to manage security within the system.

5.2.2.6 View Reports

The view reports component gives users the ability to view premade reports about the data within the system. Only system managers, directors, and system administrators are able to interface with this component. The use case that relates to this component is *view historical reports*. This component is not essential to the system's functionality but is part of the project because it provides important metrics about the system by which to measure its performance. The view reports component is driven by the view reports use case. This interaction is represented as a web form with links to reports provided by SQL Server Reporting Services.

5.2.3 Constraints

One of the biggest limitations that the actor diagram reflects about the actors and the system as a whole is the fact that very little guest information is being stored within the system itself. This is in part due to the fact that the client has requested that the ticket creation process be as easy as possible for the guest to utilize. This means that guests do not have to register to use the system and there is no multiple step verification process when a ticket is initially created. The only information that the system gathers on guests is that information they enter on a ticket when it is submitted. What this means is that guests who enter multiple tickets must re-enter their information multiple times. As this was one of the biggest requirements to the client, it was something that this system had to support.

5.3 Use Cases

The use cases for the Rez-O-lution system can be found in appendices B.1 through B. 8. There are eight use cases which are designed to show how a user interacts with the system. The use cases listed are manage messaging options, manage profile, manage users, view historical reports, manage ticket (system manager), manage ticket (guest), manage ticket (staff member), and login.

In the interest of the timeline for this project, each use case was classified into one of three categories. The categories are Crucial, Non-crucial, and Out-of-Scope. Each category describes the importance of that user interaction to the system's functionality. Crucial interactions are those use cases that were essential the system's functionality and/or this project's scope. These use cases are Manage Tickets (Guest), Manage Tickets (Staff Member), View Historical Reports, and Login. Non-Crucial use cases were those interactions that I would like to have added to this project but in the interest of time, may have been cut from the system. Effort was be made to develop these interactions though. Non-Crucial use cases are Manage Tickets (System Manager), Manage Profile, and Manage Users. Out-of-Scope use cases are those interactions that would have been really nice to have in the system but were not a part of this project's scope. The system was designed in a way that will allow for these interactions to be developed in the future, but they were not included in this project. There is only one use case of the nine that was considered to be Out-of-Scope and that is the Manage Messaging Options interaction. A table showing the use cases and their classifications as shown in Table 1.

Crucial Use Cases	Non-Crucial Use Cases	Out-of-Scope Use Cases
• Manage Tickets (Guest) • Manage Tickets (Staff Member) • View Historical Reports • Login	• Manage Tickets (System Manager) • Manage Profile • Manage Users	• Manage Messaging Options

Table 1: Priorities of Use Cases for Rez-O-lution

5.3.1 A Use Case Description

The use cases in the appendices are all designed in the same manner so it is not necessary to go over each one individually but in order to understand all of them, it is necessary to describe one in detail the following use case details r how a staff member interacts with a ticket (manage ticket (staff member)) which is provided In Table 2 but can also be found in appendix B.6.

Use Case Name	Manage Ticket (Staff Member)	
Use Case Description	Staff member manages a ticket	
Frequency	Episodic	
Actors	Staff Member	
Related Use Cases	Login	
Stakeholders	Guest: Ticket needs to be correct and statuses need to be updated accordingly.	
Happy Pathway	Resolve a ticket	
Preconditions	User exists in the system. Ticket exists in the system. Ticket status is “Submitted” or ”Open”	
Post-Conditions	Ticket status is set to “Closed”	
Flow of Events	**Actor**	**System**
	1. User clicks on “View Tickets.”	
		2. List of current tickets for a user is displayed to the user where the status is “Submitted” or “Open.”
	3. User selects a ticket to work on.	
		4. Selected ticket’s information is displayed to the user.
	5. User changes the state of the ticket to “Closed”. 6. User adds a note to the ticket.	
		7. Ticket information is updated and saved. 8. Guest who entered the ticket is emailed.
Alternate Pathways	View ticket Request re-assignment Add note	
Exception Conditions	Time Out – No information is saved	

Table 2 2: Use Case Example

Components of the use case include:

- As with almost every interaction within this system, a staff member will perform this action in an episodic manner. That is, the frequency of occurrences is not consistent. A staff member may have more tickets to close during certain times in the school year than other times (i.e. move-in week may produce a lot of tickets but winter break may not produce that many).
- The only actors that can perform this action are the staff members. The only use case that relates to this one is the login use case because a user must interact with the login use case before they can manage a ticket.
- There are a few pre-conditions that must exist before a user can interact with this use case and those are that a staff member must already exist in the system, a ticket exists in the system, and the status of the ticket is set to submitted or open.
- There is one post-condition for this use case. When a staff member is done with this interaction, the ticket's status should be set to "closed."
- The next section of the use case describes the flow of events within the use case. This does not describe every possible action that a user can do within the use case, but rather, it describes how a user moves through the "happy path" (the actions most commonly and/or the path by which the use case was designed around). In the case of the manage ticket (staff member) use case, the flow of events describes the set of actions that are taken by a staff member to resolve a ticket (set the tickets state to closed). The flow of events is split between the actions that the user takes and the actions that the system performs. For this use case, the staff member initiates the interaction by viewing a ticket. The system then presents the staff member with a list of tickets that have been assigned to them where the status of the ticket is set to submitted or open. The staff member then selects a ticket to view more information on. The system displays information on that ticket to the staff member. The staff member performs a set of actions to resolve the issue presented in the ticket and set the status of the ticket to close. When the staff member does this, they will add a note to the ticket detailing what actions they took in order to resolve the ticket. Next, the system updates the information about the ticket, including its changed status and note. Finally the guest who submitted the ticket is emailed to inform them that their ticket was closed.
- The alternate path within the use case lists other actions that are part of the interaction that may not be part of the happy path. For the manage ticket (staff member) use case, the alternate paths are view ticket, request reassignment, and add note. The view ticket path allows a staff member to open a ticket and view information about the ticket. A ticket that is viewed is not updated. The system simply presents the staff member with more detailed information on the ticket but does not update any of the information on it. A staff member may also request a reassignment of the ticket to another staff member. When this happens, a staff member chooses another staff member who they think can better handle the ticket they are viewing and submits a request for reassignment. The final alternate path that can be taken within this use case is the add note feature. This happens when a staff member views a ticket and decides to add a note to the ticket. This note contains information that the staff member feels needs to be attached to the ticket for whatever reason.

The final section of the use case describes exception conditions that could cause the flow of events to deviate from its proper course of action. This particular use case only has one exception condition and that is the event of a time out. A time out occurs when a staff member is working on ticket and does not commit their changes before the system logs them out for inactivity.

In the next sections, a more detailed description is given for each use case. The uses cases are ordered by their classification starting with the crucial ones.

5.3.1.1 Manage Ticket (Guest)

This use case is Crucial to the system's operation and to this project's success. This is the means by which tickets are created. Unlike the other use cases, a user does not need to log into the system to utilize this feature. I would have liked to add the ability for a Guest to view tickets that they had previously submitted but that was not an essential feature to the project or for the operation of the system so it was an out-of-scope feature. This interaction is controlled by a web-form containing a combination of field and dropdown controls. It was designed to be as simple as possible for a user to use.

5.3.1.2 Manage Ticket (Staff Member)

This component is that component which gives Staff Members the ability to manage Tickets. This particular component can only be accessed by Staff Members and shares some features of the Manage Ticket (System Manager) that are designed specifically for Staff Members. Staff Members are able to use this component to add notes to a ticket, resolve a ticket, and request that a ticket be reassigned to another Staff Member. Staff members are not be able to submit a ticket on behalf of a guest. If the guest cannot submit a ticket, for whatever reason, and a staff member may go through the steps that the guest would but there is no path within this use case for a staff member to create a ticket on a guest's behalf. This is a crucial interface because it is the primary means by which a ticket is closed and the request for reassignment feature is important in case an error has been made in the auto assignment of a ticket. A lot of the features of this component relate closely to each other so they were designed to be controlled by one page with a data-grid control attached to detail controls to manage more in depth information.

5.3.1.3 View Historical Reports

This interaction is not essential to the system's operations but it is considered a crucial component because it is an important way to view information and add metrics to the data within the system which was one of the biggest requirements of this project. This interaction allows users to view historical reports about the data within the System. In some cases, it may take two or more staff members to resolve a ticket. The reports reflect this fact so that the staff member who closes the ticket is not associated with all of the time it required to resolve the ticket. Access is given to System Managers, System Administrators, and Directors. All of the reports have been designed in advance by the appropriate stakeholders. This interaction is controlled by a web-form. The user is able to see a list of reports that they can view. When they click on one, the report is shown either in that page or will pop-up as another page.

5.3.1.4 Login

This is a very important use case for the system's operation and to the success of the project. This is the interaction through which all users (except for Guests) use to log in to the system to use all of the other features (except Manage Tickets (Guest)). This use case is used by any user that has a username and password. A pre-condition is that a user must exist in the system before they can log in; meaning that a System Manager must create them before they can use this feature. This interaction is controlled by a simplistic web-form with text boxes and validation controls.

5.3.1.5 Manage Ticket (System Manager)

This use case describes how a System Manager can manage some options about a ticket. This interaction is only used by a System Manager. The features offered within this use case will allow a System Manager to view tickets within the system, close a ticket, approve or deny requests for reassignments, and reassign tickets to other Staff Members. This interaction will most likely be controlled by a series of web forms with data-grid controls. For ease of use it would be impossible to put all the features (i.e. close a ticket, approve or deny requests for reassignment, etc.) of this use case to be controlled on one page so each functionality may have its own page but they will all be part of the same use case.

5.3.1.6 Manage Profile

This use case describes how a user manages information about themselves. All users that can log into the system are able to do this. Again a precondition is that the user must exist in the system and therefore a System Manager must create them initially via the Manage Users interaction before they can manage any information about themselves. This was a Non-Crucial use case to this project. This interaction is controlled by a form page. Users are able to see what information exists on them, and then choose which information they want to change and then are able to change it.

5.3.1.7 Manage Users

This interaction is only done by System Managers. Three major events happen within this use case: A System Manager is able to create users within the system and update their information, assign roles to users within the system, and assign Ticket Types to a Staff Member. This is a crucial interaction to this system as it is the primary means of adding and editing information on users and assigning Staff Members Ticket Types. This is controlled by a page that has a data-grid view of all users within the system. The data-grid has the option to edit a current user's assigned Ticket Types via a dropdown control. This is a fast and easy way to edit this information. User specific information is edited inside of a details control. If the System Manager chooses to add a new user, a new web form becomes visible where the System Manager can input information about the new user.

5.3.1.8 Manage Messaging Options

This is the use case that describes how a user is able to choose which types of events will trigger an email to that user from the system. I have decided to exclude this use case from this project as it is not necessary for the operation of the system and will take time to develop (time that can be spent on other interactions), but the system's design will allow for it to be added later as messaging within the system will be table-driven. This piece of the system will be used by all users that receive a message about any

system event that occurs (ticket assigned to them, ticket being closed, etc.). I envision this interaction to be controlled by a single page with radio-button options that a user can select and then save their options.

5.3.1.9 Use Case Review

The use cases for this project have been reviewed primarily by Douglas Kline. Charles Laymon was also able to provide helpful feedback on them, some of which was included in the use cases.

5.4 State Diagram

The ticket state diagram for the Rez-O-lution system is found in Appendix G.1. This diagram is designed to represent the various states of the system and how a ticket can move to each state. State changes happen within the system when an event occurs. A detailed description and definition of each state is given in the following sections. In the state diagram, notice that the happy path is indicated by a thick green line. This is the path in which 80% of all tickets will take when being worked on by Housing.

5.4.1.1 Submitted State

The submitted state is applied to a ticket when it is created within the system. This state is designed to acknowledge that a ticket exists but the system realizes that it has not been viewed yet. A ticket can never "*re-enter*" the submitted state after it has been assigned to another state. A ticket can only move from the Submitted state to the Open state only after a Staff Member has viewed it for the first time. This is when, not only does the system acknowledge the ticket's existence, but so does the staff member it is assigned to and, ultimately, the Housing department.

5.4.1.2 Open State

The open state is meant to describe a ticket that has been acknowledged by the system and by the staff member it is assigned to. A ticket moves to the open state only after the staff member who it is assigned to views the ticket for the first time. Once this initial viewing happens, a time stamp will be given to the ticket that will help keep track of how long it takes for the ticket to get resolved (move to the resolved or voided state). From the open state, a ticket can only move to the closed, voided, or request for reassignment state from the open state. A ticket moves to the closed state after a staff member has worked on the ticket and deems it to be completed. Similarly, a ticket can only move to the voided state when a staff member views a ticket and considers it to be a bad ticket. A bad ticket is a ticket that is false, malicious, or not meant to be handled by housing. A ticket moves to the reassignment requested state when the staff member who is assigned the ticket decides that the ticket might be better handled by another staff member and requests that the ticket be reassigned to another staff member.

A ticket can move back to the open state from the reassigned state and reassignment denied state. A ticket moves back to the open state when the ticket is reassigned to a new staff member after a request for reassignment is approved by a system manager. This ticket will only move to the open state after the newly assigned staff member views the ticket for the first time. A ticket also moves back to the open state after a request for reassignment is denied by a system manager. Again, the ticket only moves back to the open state after the original staff member views the ticket for the first time after the reassignment request has been denied.

Once a ticket moves to the open state initially, it will appear to be open to a guest until is moves to the closed or voided state (until the ticket is resolved). The guest will not see their ticket go through the reassignment requesting process as the process is designed for system and staff use.

5.4.1.3 Closed State

This, along with the voided state, is one of the final states a ticket can enter. A ticket can only move to this state from the open state when a staff member feels that they have dealt with the ticket's issue. A staff member cannot close the ticket without viewing it. Once a ticket has been assigned the closed state (when the ticket is resolved) the ticket cannot be reopened. If the guest that submitted the ticket feels that further work needs to be done on their issue, they will have to submit a new ticket.

5.4.1.4 Voided State

The voided state is similar to the closed state in that it is one of the final states that a ticket can move to, once a ticket is assigned the voided state (when a ticket is deemed invalid, false, etc.) it cannot be reopened, if a guest feels that their issue needs further work then they must submit a new ticket, and a ticket can only move to the voided state from the open state only after a staff member has viewed the ticket. The biggest difference between the voided state and the closed state is that the ticket never gets any work done on it. The staff member who is assigned the ticket makes a choice that the ticket should be voided because it is invalid for whatever reason.

The voided state is primarily designed to help with reporting within the system, especially with the life cycle of a ticket. That is, there needs to be a distinction between "fake" tickets that get closed immediately, and "real" tickets that require some time to resolve. A voided ticket's statistics will not show up in and reports except for those reports that report on the amount of tickets that were voided.

5.4.1.5 Reassignment Requested State

This is the state that a ticket moves to when the staff member who is assigned a ticket requests that another staff member gets assigned the ticket. When this happens, the original staff member not only needs to request a reassignment but they must also chose a staff member who they think can better handle the ticket. A ticket can only move to this state from the open state and only after a request for reassignment action has occurred. A ticket can move to either the reassigned state or the reassignment denied state from this state. A ticket moves to the reassigned state after a system manager approves the reassignment request. Similarly, a ticket can only move to the reassignment denied state after a system manager "denies" a reassignment request.

When a guest checks the status of their ticket (a feature that will probably not make it into the first version of the system), the guest will not see all of the internal states of the system. This means, that when a ticket moves through the reassignment process, the staff members will be able to see it, but to the guest, the ticket will appear to be open. This is because the guest does not need to know about these state changes as it may introduce confusion. Also, the time counter that is keeping track of how long a ticket is open is still running.

5.4.1.6 Reassigned State

This is the state that a ticket enters after a system manager approves a request for reassignment. A ticket can only get to this state from the reassignment request state and only after a system manager "approves" the request for a ticket's reassignment to another staff member. From this state, a ticket can only go to the open state. The ticket actually goes straight through to the open state after a reassignment request has been approved but the ticket passes through this state for reporting purposes and so the system can keep track of how a ticket was able to be resolved (who solved it, what actions took place, when those actions took place, and who performed the action).

5.4.1.7 Reassignment Denied State

This is the state that a ticket enters after a system manager denies a request for reassignment. A ticket can only get to this state from the reassignment request state and only after a system manager "denies" the request for a ticket's reassignment to another staff member. From this state, a ticket can only go to the open state. The ticket actually goes straight through to the open state after the reassignment request has been denied but the ticket passes through the reassignment denied state for reporting purposes and so the system can keep track of how a ticket was able to be resolved (who solved it, what actions took place, when the actions took place, and who performed the action).

5.4.2 Actions that change state

The list of actions that can change the state of a ticket and which state the ticket gets changed to are:

- When a ticket is initially created, the state is set to submitted
- When the ticket is viewed by a staff member for the first time, the state is changed to open
- When a staff member resolves a ticket, they change the state to closed
- When a staff member voids a ticket, they change the state to voided
- When a staff member requests that a ticket that was assigned to them be assigned to another staff member, the ticket's state is changed to reassignment requested
- When a system manager approves a request for a ticket to be reassigned to another staff member, the ticket is changed to reassigned
    - The system immediately moves the ticket to the open state
    - The reassigned state is only for ticket tracking within the system
    - A ticket will never stay in the reassigned state. It passes through it
- When a system manager denies a request for a ticket to be reassigned to another staff member, the ticket is changed to reassignment denied
    - The system immediately moves the ticket to the open state
    - The reassignment denied state is only for ticket tracking within the system
    - A ticket will never stay in the reassignment denied state. It passes through it

5.5 Swimlanes

Appendices H.1 through H.4 contain swimlanes of the ticket resolution processes within the Rez-O-lution system. They depict the paths that a ticket can take towards the closed and/or voided state. The four swimlanes created show how a ticket gets resolved, how a ticket gets voided, how a ticket gets reassigned to another staff member, and what happens when a request for reassignment is denied.

Each lane in the diagram represents an actor within the system or the system itself, the boxes depict an action that is performed within a particular lane, and the flow of events is suggested by the arrows going from one action (box) to another.

5.5.1 Ticket is Resolved Swimlane

The swimlane diagram depicting the process by which a ticket gets resolved can be found in appendix H.1. From this model we can see that a guest will submit a ticket. After the ticket is submitted, the system routes (assign) the ticket to the appropriate staff member in charge of that ticket type. Once the ticket has been assigned to a staff member, the system emails the staff member to notify them that they have been assigned a new ticket. The staff member then opens the ticket and performs some sort of action with the ticket. Once the ticket has been worked on to its completion, the staff member resolves the ticket and creates a note. Finally, the system emails the guest to notify them that their ticket has been completed.

5.5.2 Ticket is Resolved via Void

The swimlane diagram depicting the process by which a ticket gets resolved can be found in appendix H.2. The process by which a ticket is voided is similar to the way that a ticket gets closed. The guest will submit a ticket and then the system will route the ticket to the appropriate staff member and email them. Like how a ticket is resolved normally, the staff member will view the ticket, but instead of performing some action to resolve the issue, the staff member will just void the ticket. Once a ticket has been voided, the system will attempt to email the guest to let them know the status of their ticket.

5.5.3 Reassignment Request Approved

The swimlane diagram depicting this process can be found in appendix H.3. This diagram is designed to show the steps that happen when a staff member decides that another staff member would be better suited to resolve a particular ticket. The process starts when a guest submits a ticket to the Rez-O-lution system. As with the other processes depicted in the swimlane diagrams, the system automatically routes the ticket to a staff member who is designated to handle a particular ticket type. The staff member views the ticket. At this point, the staff member may make the decision that another staff member may be better able to solve the issue so they will request that the ticket be reassigned to another staff member. When they request that the ticket be reassigned, they will also choose another staff member to reassign it to. Next, the system manager will see the request for reassignment and approve it. At this time, the ticket will then get assigned to the staff member that was suggested by the originally assigned staff member.

5.5.4 Reassignment Request Denied

The swimlane diagram depicting the process by which a staff member requests another staff member is reassigned a ticket and that request being denied can be found in appendix H.4. The process starts off as the other ticket processes do with the submission of a ticket to the system by a guest. The ticket is then routed to the appropriate staff member and that staff member is messaged to notify them that they have a new ticket that needs to be resolved. The staff member will then view the ticket and make the decision that they think another staff member may handle the ticket better. They will request that the ticket be reassigned to another staff member and select a staff member that they think it should be

reassigned to. The system manager will see that a request for reassignment was made, and for whatever reason, may chose to deny the request. When this happens, the ticket stay assigned to the original staff member (never gets reassigned) and an email is sent to the staff member to notify them.

5.6 Data Model

The data model that was used for the Rez-O-lution system is shown in Appendix C.2. There are two versions in the appendix (appendix C.1 and C.2). The data model in appendix C.1 was the first generation of the data model. The developer spent a lot of time working on and re-working the data model over the period of September to November 2009 because it is the backbone of the system. Input was collected from Dr. Douglas Kline and Charles Laymon. The developer felt that if he started out with a high quality, descriptive data model, the rest of the system would be fairly easy to assemble. The time spent on the data model also let the developer experiment with the scaffolding technique as described in section 3.1.5. The first version reflects many of the developer's initial views of the systems, some of which changed or were added to.

There are a few reasons why the developer chose to use the second data model. These reasons included the fact that it was much more descriptive than the first model especially in the event handling scenarios. The first model didn't make a clear distinction between those actions that were events, those that were state changes, or those that did both events and state changes. The second model separated events and state changes so that they were much clearer. Also, the second model made state transitions "smarter". Not only is the developer able to see clearly which events cause a state change, but able to label the "arc" between the state changes as events. For example, if a ticket moves from the open to close state, the data model is able to recognize that this is the closing of a ticket just by the state change but the developer is also able to label this as an event, "ticket life cycle end". Another difference between the first and second version of the data model is the fact that ticket categories are recursive. This allowed the developer to remove the category tables and replace it with a TicketType table. This is nice because there is less overhead when assigning categories to tickets, something which was only meant to provide a sense of hierarchy to the categories. The way categories are handled in the first version of the data model was through means of two foreign keys as a part of the ticket. In the second data model, categories are actually part of the TicketType table. Those ticket types with the same ID in both the TicketTypeID and ParentTicketTypeID fields are general categories where those ticket types with different TicketTypeID and ParentTicketTypeIDs are specific ticket types.

When an action occurs within the system, more entries are made in the second version of the data model than in the first version. This may seem like more overhead, but in actuality, the developer thought it would make reporting of, and the tracking of tickets and what events are associated with a ticket easier in the long run. For instance, in the first version of the data model, when a ticket is created, there would be one entry in the TicketEvent table as all of the actions that happen when a ticket is created happen at once without being recorded, but in the second data model, there will be four entries in the TicketEvent table (ticket created, state transition, assign ticket type, and assign responsible user). This gives the developer a detailed, step-by-step description of the actions that occurred during a ticket's life time. Another interesting feature about this is the fact that if one would want to know more about any particular event that happened during a ticket's lifetime, they need only query more

information about that particular event. This is something the first data model does not allow for. That being said, the developer believes the features provided by the second data model will make the system more robust and flexible in the long-term.

In next few sections will provide additional details of each table of the data model. A data dictionary for this data model is found in appendix J.1.

5.6.1 Table-Based Role Handling and Authentication

To make the Rez-O-lution system more flexible in the future, it was decided to allow the data model to drive many of the functions that can happen within it. This includes role management, notifications, and authentication. Roles will be defined in a table and assigned to users. Users can have more than one role. When a user performs any action within the Rez-O-lution system, the system will verify that the user has the correct roles assigned to them by checking the database.

Authentication is handled in much the same way. Users will have a username and password that is stored within the database. When a user attempts to log into the system, the Rez-O-lution system will check a table in the database to verify the user's information.

5.6.2 Database Structures

5.6.2.1 Tables

5.6.2.1.1 TicketType

This table is designed to help describe the type of a ticket. The hierarchy of a ticket type is handled in this one table as opposed to having two tables (see appendix C.1). There is a recursive feature to it, meaning that specific ticket types differ from general ticket types in that they have different TicketTypeID and ParentTicketTypeIDs.

5.6.2.1.2 UserTicketType

This is an associative table that links the TicketType and User tables. This table exists because a user within the system may be assigned to more than one ticket type and a ticket type, but a ticket type can never be assigned to more than one user based on the fact that there is a unique index on the userID and ticketTypeId in this table.

5.6.2.1.3 User

This is the table that stores personal information about a user including their role ID. The way that the data model is designed, a user may have more than role. This functionality will not be added in to this project but the data model will allow for it in future releases.

5.6.2.1.4 Role

The role table stores information about the roles within the system. A role may have more than one user assigned to it and a user may have more than one role assigned to them.

5.6.2.1.5 UserRole

This is an associative table between the User and Role tables. The purpose of this table is to keep track of the various roles that a user might have.

5.6.2.1.6 TicketEventTicketType

This is an associative table between TicketType and TicketEvent. This table exists because a TicketEvent can have more than one TicketType associated with it as a TicketEvent can have more than one TicketType associated with it. This table is important because it gives the system manager the ability to change a ticket's TicktType.

5.6.2.1.7 TicketEventUser

TicketEventUser is an associative table between TicketEvent and User. This table says that a TicketEvent can have more than one User and a User can have more than one TicketEvent. This means that a ticket can be reassigned to another user and when this happens, and entry is made in this table.

5.6.2.1.8 TicketEventNotificationType

This table describes what notifications get created for what ticketEventType. When a ticketEvent occurs that requires a notification, this table controls what notification is sent, who it gets sent to, and what information is filled into the ticketNotification's dynamic fields.

5.6.2.1.9 NotificationType

This table holds the notifications that can be sent out by the system. Notifications in this table have dynamic fields that get filled out when a notification is created. Notifications are grouped by a type. For instance, there is a notificationType for when a ticket is created, closed, voided, etc.

5.6.2.1.10 NotificationRole

This table describes what roles get sent notifications and what notifications they get.

5.6.2.1.11 Ticket

This is table that stores information about a ticket. It is populated when a ticket is created and holds all information about the guest who submitted the ticket. No information is stored about the guest themselves aside from the information they submit when they create a ticket. If a particular guest creates more than one ticket, they must re-enter their information every time they create a new ticket.

5.6.2.1.12 TicketEvent

This is the table that is responsible for keeping track of every event that happens to a ticket throughout its lifecycle. An entry will be made in this table every time an event happens to a ticket. This is the table that makes it possible to follow the path a ticket takes towards its completion.

5.6.2.1.13 TicketEventStateTransition

The TicketEventStateTransition table is an associative table between the TicketEvent and StateTransition tables. This table means that a ticket event can be associated with many state transitions and a state transition can be associated with many ticket events. Basically, this table allows the system to log when a state transition happens to a ticket.

5.6.2.1.14 StateTransition

This table holds information about state transitions including the original state before the transition and the new state after the transition is complete. This table also allows us to name the arc between the states in a state transition. For example, an entry may be made in this table for "Resolve Ticket" where the old state would be open and the new state would be closed.

5.6.2.1.15 State

This is descriptive table about the states of the system. It holds the name of the state and a description of it.

5.6.2.1.16 TicketEventType

TicketEventType is the table that will hold information about the events that can happen to a ticket. A TicketID along with a TicketEventTypeID will be the basis for an entry into the TicketEvent table. There will be an entry in this table for all of the different types of events that can happen to a ticket as it moves towards being resolved.

5.7 System Diagram

The Rez-O-lution system diagram is located in appendix I.1. This diagram is an abstract view of the system and how the platforms and the components within the platforms interact with each other. There are two major platforms that the system will be using (ASP.NET and SQL Server). Within the ASP.NET platform, there will be five components which are the presentation layer, business logic layer, data access layer, messaging layer, and reporting layer. Within the data model platform are the stored procedures and base tables. More detail about the components can be found below. By compartmentalizing the system, layers can be added and removed without affecting other components making the system more versatile.

5.7.1 ASP.NET

ASP.NET is the web platform used in designing the website portion of the Rez-O-lution system. This section gives information on what layers can be found in the ASP.NET environment. For more information on the environment itself, refer to section 4.2.

5.7.1.1 Presentation Layer

The presentation layer is the layer within the ASP.NET platform which a user will directly interact with the system. This is the layer that contains all user interfaces. Aside from providing and receiving information from the users, the only other component that can interact with the presentation layer is the business logic layer. The presentation layer is able to send and receive information from the business logic layer. The presentation layer is not able to directly interact with any other component. All elements of the presentation layer used XHTML and CSS standards.

5.7.1.2 Business Logic Layer

The business logic layer of the system is the component that separates the business logic (code) of the system from the rest of the system. The business logic layer can only directly interact with the presentation layer, messaging layer, and data access layer. The business logic layer can send and receive information back and forth between the presentation layer and the same thing is true for how the

business logic layer and the data access layer interact. The relationship between the business logic layer and the messaging layer is different. Information is passed to the messaging layer but no information is passed back to the business logic layer. That is, the messaging layer is designed to be table driven so that when an action occurs within the system, tables in the database decide what notifications are sent out and to who they go to.

5.7.1.3 Messaging Layer

The messaging layer is the layer within the system that is responsible for the sending of notifications within the system. It is table driven. When an event occurs in the system, the database decides what notifications are sent out and who will get them.

5.7.1.4 Reporting Layer

The reporting layer is the layer within the system that will be responsible for the presentation of reports to the users. This layer will only directly interact with the data access layer. The reporting layer will utilize a set of stored procedures to query the database tables and present the information found in the queries to the user.

5.7.1.5 Data Access Layer

The data access layer is the layer that contains things like classes and objects that interact with the business logic and stored procedures. The data access layer can only directly interact with the business logic layer, reporting layer, messaging layer, and stored. It can pass information back and forth between the business logic layer, stored procedures, and messaging layer but the data access layer can only pass information to the reporting layer. The data access layer was originally going to be handled by the tools provided by .NET but it was decided to design and implement a custom data access layer. This decision was made because it would make the system more flexible and it did not take much time to implement.

5.7.2 MS SQL Server

MS SQL server is the tool that was used in designing the database or backend portion of the Rez-O-lution system. This section gives information on what layers can be found in the SQL environment. For more information on the environment itself, refer to section 3.1.

5.7.2.1 Stored Procedures

The stored procedures are the subroutines that the application will utilize to access information in the system's database. The stored procedures are the only way that the application can communicate with the base tables. This abstraction provides a better level of security, flexibility, reusability and performance. The stored procedures send information back and forth between the data access layer and the base tables.

5.7.2.2 Base Tables

The base tables in the system are the tables that hold the "raw" data of the system. This is the most basic layer of the system and is the most important as it is the layer that provides the data for the rest of the layer. The stored procedures are the only objects that can directly interact with the base tables and

can send information back and forth to them. No other layer can directly access the base tables. This provides security and abstraction and makes the system more versatile.

5.8 Success Criteria

The success of the project is outlined by the success criteria. The success criteria are a set of metrics that determines the overall success of this project. In order for this project to be a success:

- It must reduce the amount of time it takes for Housing to respond to and solve an issue submitted by an individual
- Must be done must be simple for all users, convenient, and reliable
- Should increase accountability within Housing
- Eliminate the staff resource needed to serve as an electronic distributor of email as notifications will be automatically handled
- Provide the ability to track events within the system
- Provide a way to audit actions taken by Housing in the process of resolving a ticket
- Provide a way to generate aggregate reports about ticket issue counts and resolution times
- Provide the ability to notify users when certain actions occur with a ticket including the resolution of it.

The current process by which an issue is presented to and resolved by Housing is time consuming and requires a lot of effort. The current process is described in section 2.3.1. Most issues are brought to the attention of Housing by means of email or telephone call. The time it takes for Housing to acknowledge the issue is, on average, 12 hours. This is when a Housing staff member, usually Sean Ahlum (Assistant Director of Technology and Communications), will go through all of the issues presented to Housing and assign them manually to other Housing staff members who are responsible for that issue type. This takes about an hour a day and is done once-a-day, usually in the mornings. Once the issue has been assigned to a staff member, it takes about another 2.5 hours. This is assuming that a staff member checks their email three times a day and works an 8 hour day. The time it takes to solve an issue depends on the issue itself, but for simplicity's sake, I am assuming that it takes 24 hours to solve a problem. This means that it takes, on average, 39.5 hours to solve a problem, 15.5 of that is spent just getting the issue to the correct person.

The Rez-O-lution system was designed to reduce the majority of the 15.5 hours it takes for an issue to be seen by the appropriate staff member. The system automatically assigns tickets to the appropriate staff members at the ticket's creation. The ticket may still incur the 2.5 hour gap between when the ticket is assigned and when the staff member works on the ticket, but it is anticipated the majority of the 13 additional hours will be eliminated. This also implies that there doesn't need to be anyone taking an hour out of their day to sort through issues. This frees up an individual's 20 hours a month that can be spent doing something else. With this reduction in time and effort, the goal would be to handle 75% of all tickets within 24 hours.

6 Project Continuation and Maintenance

The Rez-O-lution system will be hosted by either UNCW's Information Technology Services Division (ITSD). This agreement was made in the spring of 2010 during the project's development. A working virtual machine will be provided to HRL with all of the source code, running ASP, and a running MS SQL Server database. HRL will then in turn, present this virtual machine to ITSD for deployment on the university owned and operated servers. ITSD will be in charge of maintaining the servers containing the virtual machine that will host Rez-O-lution. The virtual machine that will be hosting Rez-O-lution will be self-sufficient and will have the capability of being moved around. This will make it easy to transfer Rez-O-lution as needed.

7 Project Plan

7.1 Scope

For this project, the plan was to design and build a functional ticket management system for UNCW's Office of Housing and Residence Life. This system should be simplistic in design, reliable, have the ability to automatically assign tickets to Housing staff members, and notify users of events that occur with a ticket. A detailed chart that describes exactly what activities were planned to be performed can be found in section 7.2.

The plan also included the creation of hands-on training with the staff of Housing. This training consists of walkthroughs for the users of the Rez-O-lution system as well as material that can be viewed on the web for the use of the guests entering tickets. This includes a walk-through and some frequently asked questions. Documentation was also delivered in the form of a manual for the reference of the Housing staff. This manual can also be provided online if Housing requests.

At the completion of this project:

- A virtual machine was provided to HRL with:
    - Running MS SQL database
    - Running ASP.NET website
    - All source code written in C#
- User documentation was provided to the users of the Rez-O-lution system

7.2 TimeLine

The timeline below describes in detail the actions the developer planned following the proposal defense in order to complete this project by May 1, 2010. The developer had four months (16 weeks) to complete this project therefore, the timeline below is broken up by activity and how many days he anticipated it would take to complete an activity. A majority of the project's design was completed during the fall 2009 term, leaving the code development for the spring 2010 term.

Weeks	Activity	Activity Details
1 Week (January)	Design System Layouts	• Create Master Sheet • Create Style Sheets • Setup folders and

		permissions within the project • Create error pages
1Week (January)	Create Login User Interface	• Create stared procedures for login page • Create login page • Create links to appropriate pages for users depending on user permissions
1 Week(January)	Create Ticket Entry Page	• Create stored procedures for ticket entry page • Create ticket entry page • Create ticket entry confirmation page
2 Weeks(January – February)	Create System Manager Interface	• Create stored procedures for manage users page • Create manage user page • Create stored procedures for report viewing page • Create view reports page • Create stored procedures for view ticket page • Create view ticket page • Create stored procedures for update ticket page • Create update ticket page • Create stored procedures for reassignment request page • Create reassignment requests page
2 Weeks(February)	Create Staff Member Interface	• Create stored procedures for view tickets page • Create view tickets page • Create stored procedures for update ticket page • Creates stored procedures for ticket reassignment requests • Create update ticket page with ability to request ticket reassignment
1 Week(February)	Create Director Interface	• Create stored procedures for report viewing page • Create view reports page
2 Weeks(March)	Create System Administrator	• Create stored procedures

	Interface	for report viewing page • Create view reports page • Create stored procedures for system log page • Create system log page • Create backend data pages
1 Week(March)	Create Manage User Information Interface	• Create stored procedures for manage user information page • Create manage user information page
2 Weeks(March – April)	Create Documentation and User Training Materials	• Create user handbooks for: o System manager o Director o Staff Member o System Administrator • Train users
3 Weeks(April)	Write Final Defense Paper	• Write final defense sections of project document • Create power points

Table 3: Proposed Timeline

7.3 Resources

Resources are the means required to complete a project. They can include people, equipment, facilities, money, or anything else. Shown in Table 4 are the resources utilized in the development of this project

Resource	**Responsibility**
Dr. Douglas Kline	Capstone Chair– client, business, functional requirements
Dr. Ron Vetter	Capstone Committee – software design
Dr. Thomas Janicki	Capstone Committee and acting Capstone chair from January 2010 until May 2010 – software design, web application programmer
Charles Laymon	Capstone Committee – software design
Sean Ahlum	Assistant Director for Technology and Communications – Provider of information and business requirements
Chris Cotton	Design and implementation of the project
Brad Reid	Director of Housing – provider of information
Associate and Assistant Directors of Housing - Larry Wray, Nic Troutman, Nate Miner, and Kristen Tucker	Provider of information and processes within Housing and Residence Life
Visual Studio 2008	Development environment for the project in C#
MS SQL Server 2008	Database creation
SQL Server Reporting Services	Report generation tool
ProjectLocker	Subversion and Trac hosting site, project hosting

TortoiseSVN	Subversion client used on development machines
Internet Information Services 7.0	Role and authentication management
Deployment Server	Used for the deployment of the project
ELMAH (Error Logging Modules and Handlers)	Used to log errors within the Rez-O-lution system

Table 4: Resources

7.4 Rez-O-lution: Limitations

7.4.1 Rez-O-lution Downtime

If due to intranet, system or power failures the Rez-O-lution system is not operational, students would have to revert back to the old process in order to submit an issue to HRL (refer to section 2.3.1). During this downtime, the problems with the current issue resolving process as discussed in section 2.3.1 could exist.

It would be up to ITSD to get the system active again to minimize the effects of these problems.

7.4.2 Ticket Priorities

Some issue resolving systems that were reviewed for this project allowed users the ability to assign priorities to their tickets (i.e. important, urgent, emergency, etc.). The Rez-O-lution system currently does not allow for this. Rez-O-lution categorizes tickets by specific issue types. It is the responsibility of the staff member who is assigned a ticket to judge which tickets need to be solved in which order. If a user is presented with an emergency that requires HRL's immediate attention or the immediate attention of university resources, the Rez-O-lution system will have a dialogue on the ticket entry page telling them who to contact in such a case.

7.5 Justification

Section 7.2 describes the timeline the developer envisioned this project would take and Section 7.3 describes the resources needed. The developer was confident that with the help of my committee and the other resources available to him that he would be able to complete this project in the 16 weeks that was designed it. The chart below describes how the human resources contributed to the project.

Resource Type	Resource	Task
Internal	Christopher Cotton	See section 7.2
	Dr. Ron Vetter	• Review and proof read final documentation • Approve user training documentation • Review power point presentation
	Dr. Douglas Kline	• Review and approve system stored procedures • Review and approve system pages • Review and approve reports

		• Review and proof read final documentation • Review power point presentation
	Dr. Thomas Janicki	• Review and approve system stored procedures • Review and approve system pages • Review and approve reports • Review and proof read final documentation • Approve user training documentation
	Charles Laymon	• Review and approve system stored procedures • Review and approve system pages • Review and approve reports • Review and proof read final documentation
External	Sean Ahlum	• Approve ticket entry page • Approve system manager interface • Approve reports • User training
	Brad Reid	• Approve director interface • Approve reports
	Assistant and Associate Directors of Housing	• Approve staff member interface

Table 5: Resource Type Chart

8 Project Risks and Mitigation Measures

As with every endeavor, there were risks associated with this project. The chart in appendix K.1 lists some of the possible risks that could have been encountered during this project as well as some possible methods for mitigating them. The chart ranks the risks in order of their danger to the project with the most risky being at the top of the chart and the least risky being at the bottom. Most of these risks were realized early on and noted in the project's charter found in appendix I.1.

9 Predictions

9.1 Lack of Security Features with Ticket Creation

One of the client's biggest requirements was the fact that they wanted the ticket creation process to be as simple and easy as possible. HRL did not want there to be a required login for those individuals submitting tickets or a multiple step process to validate a guest's information after a ticket was created. This could have been a problem as the developer envisioned false tickets being entered and a vulnerability to malicious attacks like scripting. A possible solution to this was implementation of a Captcha-like feature as well as bot checking on the ticket creation page. This requires people to enter in a random string in order to create a ticket. The string changes every time a new page is generated.

9.1.1 Captcha

Captcha is a challenge-response test that is implemented on many websites that tests and grades a response from a user. If the user passes the test, then it is presumed that they are human and not a bot. Most Captcha tests consist of an image box containing a distorted string that the user is required to repeat.

9.1.2 Bot Checking

AJAX provides a tool called No Bot. No Bot attempts to provide Captcha-like bot checking without any human interaction. That is, No Bot runs on the web-site and doesn't require a user to directly interact with it like Captcha does. No Bot is easier to bypass than Captcha but it is completely invisible to the users.

9.2 Data Model Choice

As discussed in section 5.6, the developer had a decision to make on which data model to use. He decided to use the second data model as shown in appendix C.2. He based his decision on the fact that he thought this data model will be more descriptive, robust, and flexible. Also discussed in section 5.6, the second data model separates events and state changes from each other. This makes the data model more descriptive and less confusing as the first data model tended to confuse those actions that were state changes and those that were events. Furthermore, the second data model allows state changes to be smarter through features like "arc naming" (the ability to name an even that happens between two states in a state change). The fact that category types are recursive in the second data model was a significant change as well. This allowed the developer to consolidate two tables describing categories into one. This was a nice change because it was unnecessary to have two tables describing the hierarchy of categories because it added complexity to the data model. Finally, even though more entries need to be made in the database, whenever something happens in the system and the queries may have more joins than the first data model in the stored procedures that will be written, the developer believes that the second data model will allow for a user of the system to more accurately track events within the system. This will help when the time comes to audit actions in the system and writing log reports.

9.3 Learning Curve

The developer did not have extensive experience in the platforms he used and this project is one of the biggest endeavors he undertook. He developed most of this system on his own and he unified many different software widgets and techniques This was an obstacle (as was seen with the developer's

attempts to install and implement Microsoft's Team Foundation Server), but thankfully, he had some very skilled and knowledgeable resources to help him. The developer was confident that he would be able to accomplish this project with the help of these resources.

10 What was Proposed vs. What was Delivered

10.1 Proposed Timeline vs. Actual Timeline

As mention in section 7.2, the developer estimated that the project would take 15 weeks to complete this project. During this time, four to five hours of each day was spent to develop the system. That means, on average, 20 – 25 hours a week were spent working on the project.

Looking back at the time predictions I made before the start of the project, I believe that setting up a weekly timeline was unrealistic. Although it would have been nice to say, things like "This stored procedure will be done in week 12," I found it didn't work out this way. The tasks proposed were completed but I found that often times, they were completed out of order. Some tasks required that other things be done first others were done earlier than proposed because it fit into the "natural flow." Project Management learning occurred as a result of tracking anticipated time lines and order of activities versus the actual implementation of the project.

The timeline I provided attempted to estimate the amount of time it would take to complete each activity. This was a good attempt to estimate the time it would take to complete the project, but it still didn't accurately reflect the timeline of the project. For starters, in the timeline originally proposed, I was really specific in that I provided the time it would take to complete a very specific activity like "Creating the staff member interface". A lot of the times, similar activities were done at the same time. An example of this would be the creation of stored procedures pertaining to the management of tickets. It wouldn't make any sense to create the stored procedures only pertaining to the staff members and then move on when, in reality, staff members, system managers, and system administrators might require similar stored procedures. If I had to project in the future, I would group similar items at the same time.

The original timeline I provided is primarily flawed in that I try to associate a time to an activity associated with a user within the system. (i.e. I estimated that it would take one week to create the director interface.) I should have provided an estimate of the time it would take to complete specific parts of the system. For instance, I should have estimated the time it would take to complete the system's data access layer, the business logic layer, stored procedures, etc.

As mentioned before, all of the activities on the original timeline were completed, but they were done out of order for the most part. That being said, I have recorded in which month certain activities were completed and also the major accomplishments that were completed in the months during which I developed this system.

10.2 Timeline Revisited by Month

Presented in Table 6 is the original proposed timeline but with the addition of another column that states the month in which a particular activity was completed. Additional activities that were not included in the original timeline but were completed as a part of this project are not represented in this chart but are made note of in the section 10.4. The purpose of this revised chart is to show the actual progression of the project, that it went out of its proposed order, and that all of the proposed activities were completed.

Anticipated	Activity	Activity Details	Completed
1 Week (January)	Design System Layouts	• Create Master Sheet • Create Style Sheets • Setup folders and permissions within the project • Create error pages	• April – Created SiteDefault.master and NestedSiteDefault.master • April – Created RezMembershipProvider.cs and RezRoleProvider.cs and added permissions to all pages in the web.config file • March – Error handling done through ELMAH and an error page(Error.aspx) was created to handle system errors
1Week (January)	Create Login User Interface	• Create stared procedures for login page • Create login page • Create links to appropriate pages for users depending on user permissions	• February – Created upValidLogin • April – Login.aspx created • April – Link to UserHome.aspx created from Login.aspx. This is where users will find links to pages appropriate to their role
1 Week(January)	Create Ticket Entry Page	• Create stored procedures for ticket entry page • Create ticket entry page • Create ticket entry confirmation page	• January – Created upTicketCreate • April – Default.aspx handles ticket creation for guests • April - Ticket-Submitted.aspx created
2 Weeks(January – February)	Create System Manager Interface	• Create stored procedures for manage users page • Create manage user page • Create stored procedures for report viewing page • Create view reports page	• January – Created upUserCreate, upUserUpdate • April 2010 – ManagerManageUsers.aspx • April 2010 – No stored procs were needed, used asp report control • April 2010 – viewreports.aspx • February – Created upTicketList, upTicketDetailGet

		• Create stored procedures for view ticket page • Create view ticket page • Create stored procedures for update ticket page • Create update ticket page • Create stored procedures for reassignment request page • Create reassignment requests page	• April – ListTicketDetails.aspx created • February – up_ListTickets • April 2010 – ListAllTickets.aspx • February - upReassignmentRequest, upReassignmentApproveDeny • April 2010 – viewreassignmentrequests.aspx
2 Weeks(February)	Create Staff Member Interface	• Create stored procedures for view tickets page • Create view tickets page • Create stored procedures for update ticket page • Creates stored procedures for ticket reassignment requests • Create update ticket page with ability to request ticket reassignment	• February – Created upTicketList, upTicketDetailGet • • February – Created upTIcketVoid, upTicketClose • February - upReassignmentRequest, upReassignmentApproveDeny • April 2010
1 Week(February)	Create Director Interface	• Create stored procedures for report viewing page • Create view reports page	• April 2010 • April 2010
2 Weeks(March)	Create System Administrator Interface	• Create stored procedures for report viewing page • Create view reports page • Create stored procedures for system log page • Create system log page	• April 2010 • April 2010 • February – Created up_TicketEventsGet • Deemed out of scope for the project defense. Will be added after project is defended. • Deemed out of scope for the project defense. Will be added after project is defended.

		• Create backend data pages	
1 Week(March)	Create Manage User Information Interface	• Create stored procedures for manage user information page • Create manage user information page	• February – Created upUserCreate, upUserUpdate • Deemed out of scope for the project defense. Will be added after project is defended.

2 Weeks(March – April)	Create Documentation and User Training Materials	• Create user handbooks for:
		  ○ System manager
		  ○ Director
		  ○ Staff Member
		  ○ System Administrator
		• Train users
			• April 2010
			• This was not done as the application has not been deployed yet. Once the application is deployed, work will begin to train users on the system.

3 Weeks(April)	Write Final Defense Paper	• Write final defense sections of project document • Create power points	• April – Final document was created and submitted to project committee for approval • April – Power point slide for project defense created

Table 6: Timeline Revisited

10.3 Implementation Using Rapid Application Development Tools

In section 3.1, the model used to develop this system was discussed. Figure 1 also depicts the project development cycle that the Rez-O-lution project resembled. This development cycle is made of three actions that are repeated over and over until the project is complete. The actions are to add a feature, test the new feature, and re-factor. Each month of the project was another iteration of this process. Each phase was designed to create a new piece of the system. Once a piece was "complete", it was tested and presented for feedback, and then eventually re-factored. The testing and feedback made it clear what needed to be re-factored and this re-factorization could be the addition of components that were needed and didn't exist, the deletion of components that were created and not needed, or the modification of already existing components to better fit what was needed. Figure 2 shows a more detailed view of the project's development and how the project's development cycle was applied to each phase. A description of each phase is described in sections 10.3.1 through 10.3.6.

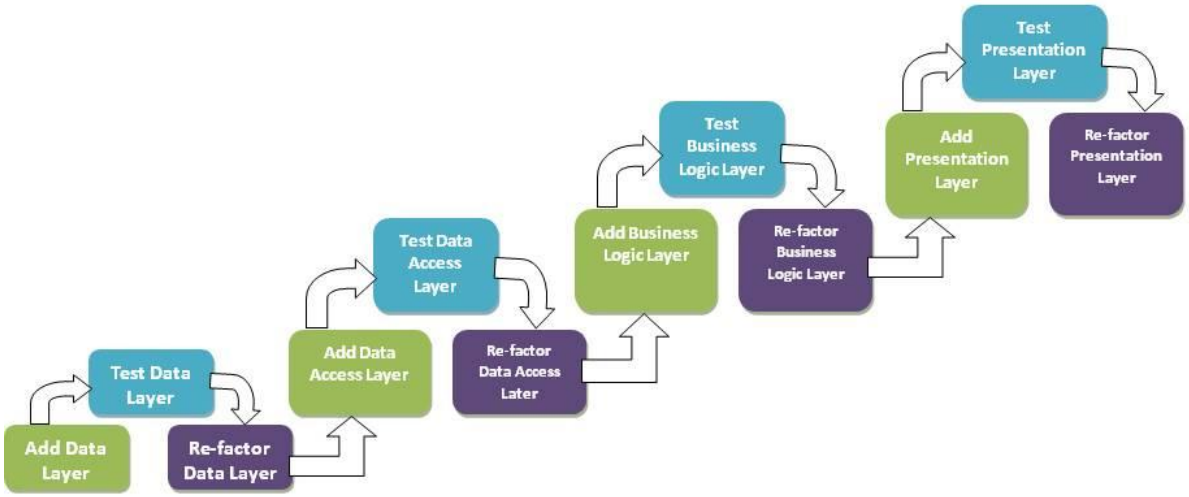


Figure 2: The Project Life Cycle for Rez-O-lution

10.3.1 Fall 2009

The planning phase of this project was done in the fall of 2009 during months of August through December. The initial planning, diagramming and interviewing was done at this time. Below is a list of accomplishments completed at during this phase.

- Interviewing
  - Interviewed Housing staff to gather requirements
  - Project charter was created
- Diagramming
  - Current processes were mapped out
  - Proposed solution was mapped out
  - Rez-O-lution system diagram created
- System Analysis
  - Use cases created
  - Data model was designed

- Actor diagram was created
- State diagram was created

10.3.2 January 2010 to February 2010: Iteration 1

The month of January 2010 marked the start of the project. During this month, the developer was able to set up the project's workspace and begin working on the data layer of the project. This included the stored procedures, functions, changes to the database, as well as any other tasks that required direct interaction with the database. By the end of the month, the data layer was tested and anything that needed to be re-factored was found out.

10.3.3 February 2010 to March 2010: Iteration 2

During February 2010, any final modifications were made to the data layer. After the refactoring of the data layer, the data access layer and business access layer were added. These components were tested by the end of month and changes that needed to be made were found.

10.3.4 March 2010 to April 2010: Iteration 3

The data access layer and business logic layer were re-factored in the beginning of March 2010. When the re-factorization was complete, work began on the presentation layer. The presentation layer was worked on throughout March and into April. Testing of the presentation layer began during the middle of April.

10.3.5 April 2010 to May 2010: Iteration 4

Work continued on the presentation layer. The completion of the presentation layer marked the end of development of the project. The rest of the time spent was on the deployment of the project and the creation of documentation for the users of the Rez-O-lution system.

10.3.6 May 2010: Iteration 5

May 2010 was dedicated to the deployment of the Rez-O-lution system, the creation of documentation for the users of the system, and the creation and presentation of the project to the project's committee.

10.4 Accomplishments by Month

The project was completed during the spring 2010 term. This section lists out what activities were done in which month and if any design changes were made along the way that may disagree with what was originally proposed.

- **January 2010**
  - Setup the Rez-O-lution Project
  - Created System Security Accounts and Roles
  - Data Model Changes
    - The changes made to the data model had to do with notifications
    - Designed to make the process of logging and sending notifications simpler.
    - The previous deign created a notification for individual users when specific events happened.
      - Pros: Notifications were detailed and tailored to the specific user

        - Cons: required more overhead and more confusing
    - New design has a notification types associated with a notification event
        - Pros: Less overhead and less confusing
        - Cons: Generic messages are sent out to the designated parties
        - Caused a lot of overhead, it was confusing
  - Scripted the Database to a Backup File
  - Created Initialization Scripts
    - InitTables fills the tables with information known ahead of time (i.e. User information, ticket types, etc.)
    - InitTestData fills the database with some test data. This script was the basis for many of the stored procedures associated with ticket manipulation as it described all of the things that need to be done when a particular action is performed.
  - Started Work on Needed Stored Procedures and Functions
    - None of these were actually included in the original timeline but were needed in order to complete other activities
    - They include: fn_GenerateTicketCode, fn_ReturnTicketEventID, upTicketAssignTicketType, upTicketTypeCreateSpecific, upTicketNotificationCreate, upTicketEventInsert, upTicketAssignStateTransition, upTicketAssignUser, upTicketCreate
- **February 2010**
  - Created an updated backup of the database
  - Renamed all of the stored procedures to better reflect their purpose
  - Continued to work on necessary stored procedures and functions needed for the system
    - These include: fn_GetTicketUser, fn_ValidateTicketChangeableStatus, fn_GetTicketStatus, upNotesAppend, upTicketClose, upTicketTypeCreateGeneral, upReassignmentRequestTicketsGet, upTicketDetailGet, upTicketEventsGet, upTicketTypeGet, upTicketList, upUserListValid, upReassignmentApproveDeny, upReassignmentRequest, upUserSelect, upTicketOpen, upUserUpdate
- **March 2010**
  - Created the data access layer and the business logic layer
  - Started work on the user interface
  - Implemented ELMAH for error handling
- **April 2010**
  - Created user interface for the project
  - Implmented Captcha for the project
  - Setup role-based permissions on all of the web pages that were created
  - Documentation completed
    - This included: The project's final paper, slides for the defense, and user documentation

11 Predictions: Revisited

11.1 Lack of Security Features

After completion of the project, it became clear that the lack of security was still an issue. Although the implementation of Captcha would help with the reduction of malicious ticket entries, little information was being stored on the people entering tickets. An alternative solution to this would be to implement a login system for all users (even guests) to the system where a guest would have to register or to implement a multiple step authentication feature that would require a user to confirm that they are, in fact, who they say they are.

11.2 Learning Curve

The learning curve of the project was the biggest challenge to the project. There were times when the developer either underestimated the time it would take to complete a task mainly due to time to come up to speed with the implementation tools and/or techniques. Having said that, this was a valuable learning experience

11.3 Data Model Choice

The data model choice that was selected is very robust and provides a high degree of flexibility.

12 General Review of the Project

This section covers a general review of the project. This includes things that were not anticipated, key accomplishments and successes, and future work.

Overall, the Rez-O-lution project was a success. There were some obstacles, but the majority of the initial user requirements were implemented. A working system has been created that incorporates the key features that Housing was looking for in a solution to their current issue resolution process.

The future of the project is also looking bright. Work will still need to be done by whoever decides to continue this project after its completion by the current developer but the foundation for many features have been planned out for and, in some cases, have been started but need to be completed.

When Housing came to the developer of this system, they requested that a system be developed for them that was simple and easy for users to submit requests to housing, expedite the time it would take to resolve issues, would not require a login for students and parents, and notify users of events that pertain to issues they are involved in. Rez-O-lution does these things. Other features were proposed and they can be added in the future, but the essence of what Housing wanted is found within this solution.

12.1 Rez-O-lution Stats

Appendix J contains a complete list of all stored procedures included in Rez-O-lution while appendix k is the table of contents from the Doxygen code documentation tool. The Doxygen table of contents lists out all of the pieces of code contained in the project. Generally though, the Rez-O-lution system is made up of these pieces:

- 17 data tables
- 8 user defined functions
- 48 stored procedures
- 4 data access layer datasets
- 4 business logic layer classes
- 15 asp user controls
- 2 asp master pages
- 19 asp pages

12.2 Code Analysis

Code analysis was not part of this project's original scope. That being said, it was suggested that it be done before the project was deployed. Before the project is handed over to ITSD, code analysis will be done. Visual Studio 2008 contains a code analysis tool in the team edition but not in the professional edition. The development of this project was done in the professional edition. For this reason, the Visual Studio took could not be utilized.

To do the code analysis, fxCop will be used. FxCop is an open-source code analysis tool. Based on its findings, changes to the code of the project will be made to make it more reliable and efficient.

12.3 Threat Analysis

There are three major threats on the structure of the project that were a concern during the development. These threats include SQL injection, authentication, and cross-site scripting. Precautions were taken to mitigate these threats.

To mitigate the threat of SQL injections, parameterized stored procedures were used for all actions that required database interactions.

To handle authentication, it was decided that forms authentication would be used. ITSD has expressed interest in making the authentication become windows authentication or active directory. If this is the case, they will be charged with handling the authorization aspect of security in this system. The developer did use forms authentication and as a precaution, set a very conservative time-out period for the forms authentication cookie.

Finally, ASP .NET has some out-of-the-box default settings to prevent cross-site script attacks. ASP .NET, by default, does not allow html based mark-ups to be posted to the server. This is allowed but it has to be manually turned on. It was decided that, in order to mitigate the risk of cross-site scripting, html mark-ups would not be allowed to be posted to the server.

12.4 Key Accomplishments and Successes

During the development of the Rez-O-lution system, many accomplishments and successes were done. Key accomplishments are detailed in the next sections.

12.4.1 Custom Data Access Layer and Business Logic Layer

After the initial plan, it was decided that a custom data access layer and business logic layer be created. This allowed for a greater abstraction within the system. The data access layer is used most of the features developed in the system but towards the end of the project's development and in the interest of time, some of the features developed using .NET's data access controls. For these features, methods have been added to the data access layer and business logic layer to allow for their addition in the future.

12.4.2 Simple Interface for Ticket Creation

This was one of Housing's primary requirements. The Rez-O-lution system allows for users to create tickets without a user having to log into the system. The interface by which users enter tickets is very simple and Captcha was added to help reduce the amount of false tickets.

12.4.3 Ticket Management

Ticket management was added to the system and per the design documents, can only be handled by the staff member actor. Those users who are assigned to the staff member role must log into the system before they can manage tickets. The management of tickets includes the closing, voiding, and reassignment requests of tickets. These actions are all handled in one user interface.

12.4.4 Maintenance Requests

Housing already has a maintenance request system in place. Rez-O-lution was not created to replace it. Maintenance requests still need to be directed to the maintenance request system. The is done is by providing some text telling guests where to go to enter maintenance requests on the ticket entry page along with a link to Housing's maintenance request page.

12.4.5 ELMAH

During the development of the Rez-O-lution system, it became clear that the design documentation did not allow for error handling. With the help of the developer's committee, ELMAH was implemented to help with the handling of errors. ELMAH provides some interesting features including the ability to automatically generate error reports via XML documents, auto notifications, and an interface by which a system administrator can view logged errors.

12.4.6 Captcha

Captcha was added to the project to verify that users of the system are not using scripts to auto generate tickets. The CaptchaUltimateUserControl by Peter Keller was used and was relatively easy to implement. The Captcha control is used on the ticket creation page as well as the user login page. This is the primary means of security for the system to prevent unwanted visitors from entering false data into the system.

12.4.7 Abstraction

One of the most interesting features of the Rez-O-lution system is the fact that there is a very high level abstraction within the system. A user cannot directly access the data from the user interface but rather, when a user wants to perform an action within the system, the requests have to travel through multiple layers and validations before any data is created or changed. Figure 3 shows those abstractions. This

diagram shows that when a user makes a request to the system, the do so via the user interface. That request is then passed to the business logic layer which is in turn passed to the data access layer. The Data access layer then passes the request to the data layer (or base tables) via stored procedures. The result of the user's request is then passed back to the user through the same channels by which it was received. This is very helpful because if a piece of the system ever needs to be changed, the change can be made in just a few places rather than in multiple places.

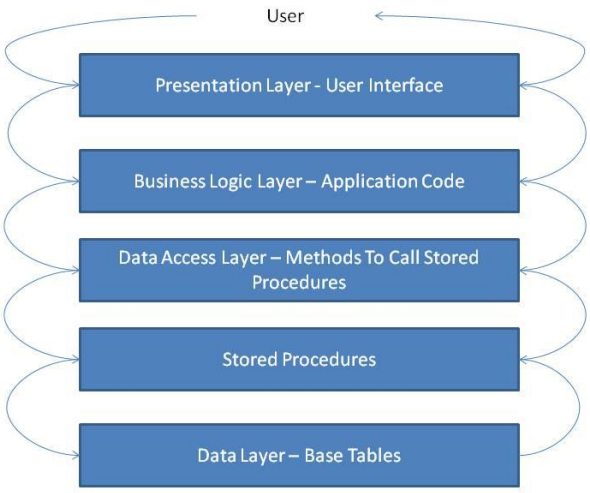


Figure 3: Levels of Abstraction

12.4.8 A Look at Rez-O-lution

It doesn't make sense to list every file and include every line of code into this document, but it is important to show some results of the work that was put into this project. Screen shots of some interesting features can be found in Appendix J.

Appendix K.1 shows a list of stored procedures used in the Rez-O-lution system and appendix K.2 shows a summary of code. Doxygen was used to generate the code summary.

12.5 Unanticipated Factors

Time was a huge obstacle for this project. There were many instances when something needed to be done that was not anticipated. For example, it was decided that creating a custom data access layer and business logic layer would allow for a greater abstraction and more flexibility but the creation of these features took time to do. The time to create these things were not factored into the project's timeline, were necessary (in the developer's mind).

The project's scope was very ambitious. Because of the very large scope of the project, some features were scaled back or cut to make time for those features that are essential to the project.

12.6 Future Work

The future works section covers those things that were not included in the system's initial development but have been recognized as features that could be added in the future.

12.6.1 Deployment of Rez-O-lution

The Rez-O-lution system has not been deployed on any servers yet. The process for this has been initiated though. This was recognized as something that Housing would be in charge of but steps have been taken to get it deployed onto the servers owned by UNCW's Information Technology Services Division. Individuals in ITSD responsible for this deployment have been contacted in the spring of 2010 and some basic requirements for deployment on ITSD servers have been realized.

A SSL certification will be required for deployment. ITSD has assured that the application will run on SSL when it is deployed on their servers.

From a security standpoint, ITSD will also be responsible for the securing of sensitive connection string information.

12.6.2 Guest Ticket Tracking

Again this feature was not in scope but it is something that has been recognized and the ability to add it in the future exists within the Rez-O-lution system. Guest ticket tracking will allow guests to revisit the site and check on the status of their ticket after they have submitted it.

12.6.3 Administration Pages

Administration pages were never in scope. These would be the means by which a system administrator can track events within the system. The ability to add this feature is available for future implementations but the developer does not see it as a necessary feature at this time.

12.6.4 Ticket Event Tracking

Ticket event tracking would allow users (mainly system administrators and system managers) the ability to track events associated with a ticket. This feature was not in the project's scope but as with many of the other features that did not make it into the initial release of the project, the ability to add this feature is possible as stored procedures as well as methods in the data access and business logic layer have been created already. The only thing that would need to be done to add this feature is to add a user interface to it.

12.6.5 Rez-O-lution and Beyond

One of the most interesting features of Rez-O-lution is the fact that, even though it is a custom solution for Housing, it could potentially be used by other organizations. Some modifications would have to be made, but the fact that so much planning, simplification, and abstraction have been incorporated into this solution makes for a light-weight application that almost any organization can use.

12.6.6 Future Enhancements

There were some things that the developer would like to have added to this project to enhance it, but because of time constraints, they were not included in the scope.

12.6.6.1 Clarification of Emails

Feedback from users suggest that the notifications currently included in the project are not detailed enough. Users would like more information included in the emails sent out to them. Specifically, they would like to know things about the ticket that the notification is about including, a link to the ticket that directs the user back to the system, a description of the ticket (the ticketType), and a key contact person who the user can contact if they have any questions about their ticket.

12.6.6.2 Guest Ability to View and/or Edit Tickets

User feedback also suggests that users would like the ability to return to the system and either edit their original ticket's information or view it to see its current status. This was recognized early in the designing of the system, but it was cut from the scope due to time constraints.

12.6.6.3 User Documentation and Training

One thing that would really enhance this project would be a set of documentation for users as well as user training. It is something that was included in the original scope but the developer ran out of time.

13 Skills Acquired during the capstone project

During this project the developer was able to use and gain some valuable skills. These skills include:

- Time management
- MS SQL design and implementation
- Application development with C# .NET
- Application development with ASP
- Application design document creation
- System analysis and design
- Error handling with ELMAH
- Project timelines and execution
- Software development

14 Classes Particularly Valuable

To prepare someone to undertake a project like this one, it is suggested that they take certain classes which may include but are not limited to:

- Information Analysis and Management
- Application development with ASP. NET
- Software Engineering
- Database Management
- Analysis, Modeling, and Design

15 Acknowledgements

There are a lot of people I, the developer, would like to thank for their support throughout this project. First of all, I would like to thank my project committee especially Charles Laymon and Dr. Douglas Kline. I also want to thank Sean Ahlum, and all of Housing for support, cooperation, and understanding during the project's design and development. Thanks to Amy Risher who provided endless amounts support and encouragement. Finally, I need to thank my parents, Donald and Deborah, whose constant moral, emotional, and edible support not only kept me going from day-to-day but also pushed me to new heights, challenging me to do my absolute best, always.

16 Works Cited

1. Brown, David William. An Introduction to Object-Oriented Analysis: Objects and UML in Plain English. New York, New York: John Wiley & Sons, Inc, 2002.
2. Reed, Paul R. Jr. Developing Applications with Visual Basic and UML. Reading, Massachusetts: Addison-Wesley, 2000.
3. Wysocki, Robert K. Effective Project Management: Traditional, Adaptive, Extreme. Indianapolis, Indiana: Wiley Publishing, 2007.
4. "About UNCW." 10/15/2009. <http://uncw.edu/facts/>
5. "UNCW Housing & Residence Life." 10/15/2009. <http://www.uncw.edu/stuaff/housing/>
6. "About UNCW Housing & Residence Life." 10/15/2009. <http://www.uncw.edu/stuaff/housing/mission.htm>
7. "Object-Oriented Programming." 10/15/2009. <http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci212681,00.html>
8. "IIS Authentication." 11/15/2009. <http://msdn.microsoft.com/en-us/library/aa292114(VS.71).aspx>
9. "Configuring .NET Roles in IIS 7." 11/15/2009. <http://technet.microsoft.com/en-us/library/cc731597(WS.10).aspx>
10. "SQL Server Reporting Services." 11/15/2009. <http://en.wikipedia.org/wiki/SQL_Server_Reporting_Services>
11. "Rapid Application Development." 11/15/2009. <http://en.wikipedia.org/wiki/Rapid_application_development>
12. "Unit Testing." 11/15/2009. <http://en.wikipedia.org/wiki/Unit_testing>
13. "What does Remedy do for the University of North Texas?." 11/16/2009. <http://www.unt.edu/benchmarks/archives/2000/march00/what_does_remedy_do.htm>
14. "Remedy Service Desk." 11/16/2009. < http://www.bmc.com/products/product-listing/22743834-121272-1370.html>
15. "Ticket Desk - Help Desk Issue Tracking and Support System." 11/16/2009. <http://www.codeplex.com/TicketDesk>

Appendices

Appendix A.1 : Housing and Residence Life Mission Statement

"It is the mission of the Office of Housing and Residence Life to appropriately challenge residents to develop to their full potential by supporting their educational growth and personal development.

The Office of Housing and Residence Life strives to establish a satisfactory physical environment that emphasizes the utilization of human and material resources in an efficient manner. In order to facilitate each student's personal and academic growth, it is essential that a clean, safe environment be maintained.

The Housing and Residence Life staff will maintain an environment that is conducive to educational excellence by providing academic support, educational opportunities, and challenges for individual residents. Opportunities for students to attain a greater appreciation and understanding of cultural and lifestyle differences and opportunities for leadership and self-governance are integral to the on-campus experience.

In order to establish community, residents must accept responsibility for their actions and understand the natural and logical consequences of their behavior, as it impacts themselves and those who reside around them. The Office of Housing and Residence Life will provide a trained professional and paraprofessional staff that will assist students in understanding the established guidelines for behavioral expectations of all UNCW students."

<http://www.uncw.edu/stuaff/housing/mission.htm>

Appendix B.1: Manage Messaging Options Use Case

Use Case Name	Manage Messaging Options	
Use Case Description	A user chooses the events for which they would like to receive notifications about.	
Frequency	Episodic	
Actors	Staff Member, System Manager, System Administrator	
Related Use Cases	Login	
Stakeholders	Staff Member: Correctly update information regarding messages and verify that it is correct System Manager: Correctly update information regarding messages and verify that it is correct System Administrator: Correctly update information regarding messages and verify that it is correct	
Happy Pathway	Change Messaging Options	
Preconditions	User must be logged in.	
Post-Conditions	The user's messaging information is updated and saved.	
Flow of Events	**Actor**	**System**
	1. the user clicks on the "manage messaging options" link	
		2. The system provides the user with their current messaging options
	3. The user selects the desired messaging options 4. The user clicks submit	
		5. The system saves the updated messaging information
Alternate Pathways	User cancels out of the process – No changes saved.	
Exception Conditions	Session time-outs – equivalent to a user cancel – no changes saved.	

Appendix B.2: Manage Profile

Use Case Name	Manage Profile	
Use Case Description	A User updates their account information.	
Frequency	Episodic	
Actors	Staff Member, Director, System Administrator, System Manager	
Related Use Cases	Login	
Stakeholders	Staff Member: Correctly update or add information about their account.	
Happy Pathway	Change Profile Information	
Preconditions	A User must exist in the system and must be logged in.	
Post-Conditions	The User's information is updated.	
Flow of Events	**Actor**	**System**
	1. The User clicks on the "Edit Profile" link. 3. The User updates/adds their information. 4. The User clicks "Submit".	2. The system displays all current information about the User to the User. 5. The system updates the User's information in the database.
Alternate Pathways	User cancels out of the process – No changes saved.	
Exception Conditions	Session time-outs – equivalent to a user cancel – no changes saved.	

Appendix B.3: Manage Users

Use Case Name	Manage Users	
Use Case Description	Add or edit Staff Members and their information	
Frequency	Episodic	
Actors	System Manager	
Related Use Cases	Login	
Stakeholders	Staff Member: Account must be created in order to log into the system and add or update account information.	
Happy Pathway	Add New User	
Preconditions	System Manager must be logged into the system	
Post-Conditions	A Staff Member exists in the system.	
Flow of Events	**Actor**	**System**
		1. System displays a list of all current Staff Members.
	2. System Manager clicks on the "Add User" link.	
		3. System displays the "New User" form.
	4. System Manager fills out the "New User" form. 5. System Manager clicks "Submit"	
		6. System stores the new user's information
Alternate Pathways	1. System Manager cancels out of the process – No information is stored 2. Edit Information for Current User 3. Delete User	
Exception Conditions	Session Time outs – No information is saved	

Appendix B.4: View Historical Reports

Use Case Name	View Historical Reports	
Use Case Description	Historical reports are displayed to the user.	
Frequency	Episodic	
Actors	System Manager, System Administrator, Director	
Related Use Cases	Login	
Stakeholders	Director: Must be able to see the correct data displayed in a report. System Manager: Must be able to see the correct data displayed in a report. System Administrator: Must be able to see the correct data displayed in a report.	
Happy Pathway	View Reports	
Preconditions	User must be logged into the system. Reports exist within the system.	
Post-Conditions	No data is changed.	
Flow of Events	**Actor**	**System**
	1. User clicks on the "View Reports" link. 3. The user selects a report to view. 5. The user closes the report.	2. The system displays a list of reports that can be viewed. 4. System displays the selected report to the user.
Alternate Pathways	NA	
Exception Conditions	1. Time out – Nothing is changed	

Appendix B.5: Manage Ticket (System Manager)

Use Case Name	Manage Ticket (System Manager)	
Use Case Description	Manager manages a ticket.	
Frequency	Episodic	
Actors	System Manager	
Related Use Cases	Login	
Stakeholders	Staff Member: Need the ticket's information to be correct, mainly the ticket status.	
Happy Pathway	Change Ticket State	
Preconditions	User must be logged in. Ticket exists.	
Post-Conditions	Ticket status is updated.	
Flow of Events	**Actor**	**System**
	1. User clicks on "View Tickets."	
		2. List of current tickets for a user is displayed to the user where the status is "Submitted" or "Open."
	3. User selects a ticket to work on.	
		4. Selected ticket's information is displayed to the user.
	5. User changes the state of the ticket to "Closed", "Reassignment Approved", or "Reassignment Denied"	
		6. Ticket information is updated and saved. 7. Guest and/or Staff Member emailed.
Alternate Pathways	View Ticket	
Exception Conditions	1. User cancels out of the process – no information is saved 2. Time out – No information is saved	

Appendix B.6: Manage Ticket (Staff Member)

Use Case Name	Manage Ticket (Staff Member)	
Use Case Description	Staff member manages a ticket	
Frequency	Episodic	
Actors	Staff Member	
Related Use Cases	Login	
Stakeholders	Guest: Ticket needs to be correct and statuses need to be updated accordingly.	
Happy Pathway	Resolve a ticket	
Preconditions	User exists in the system. Ticket exists in the system. Ticket status is “Submitted” or ”Open”	
Post-Conditions	Ticket status is set to “Closed”	
Flow of Events	**Actor**	**System**
	1. User clicks on “View Tickets.”	
		2. List of current tickets for a user is displayed to the user where the status is “Submitted” or “Open.”
	3. User selects a ticket to work on.	
		4. Selected ticket’s information is displayed to the user.
	5. User changes the state of the ticket to “Closed”. 6. User adds a note to the ticket.	
		7. Ticket information is updated and saved. 8. Guest who entered the ticket is emailed.
Alternate Pathways	View ticket Request re-assignment Add note	
Exception Conditions	Time Out – No information is saved	

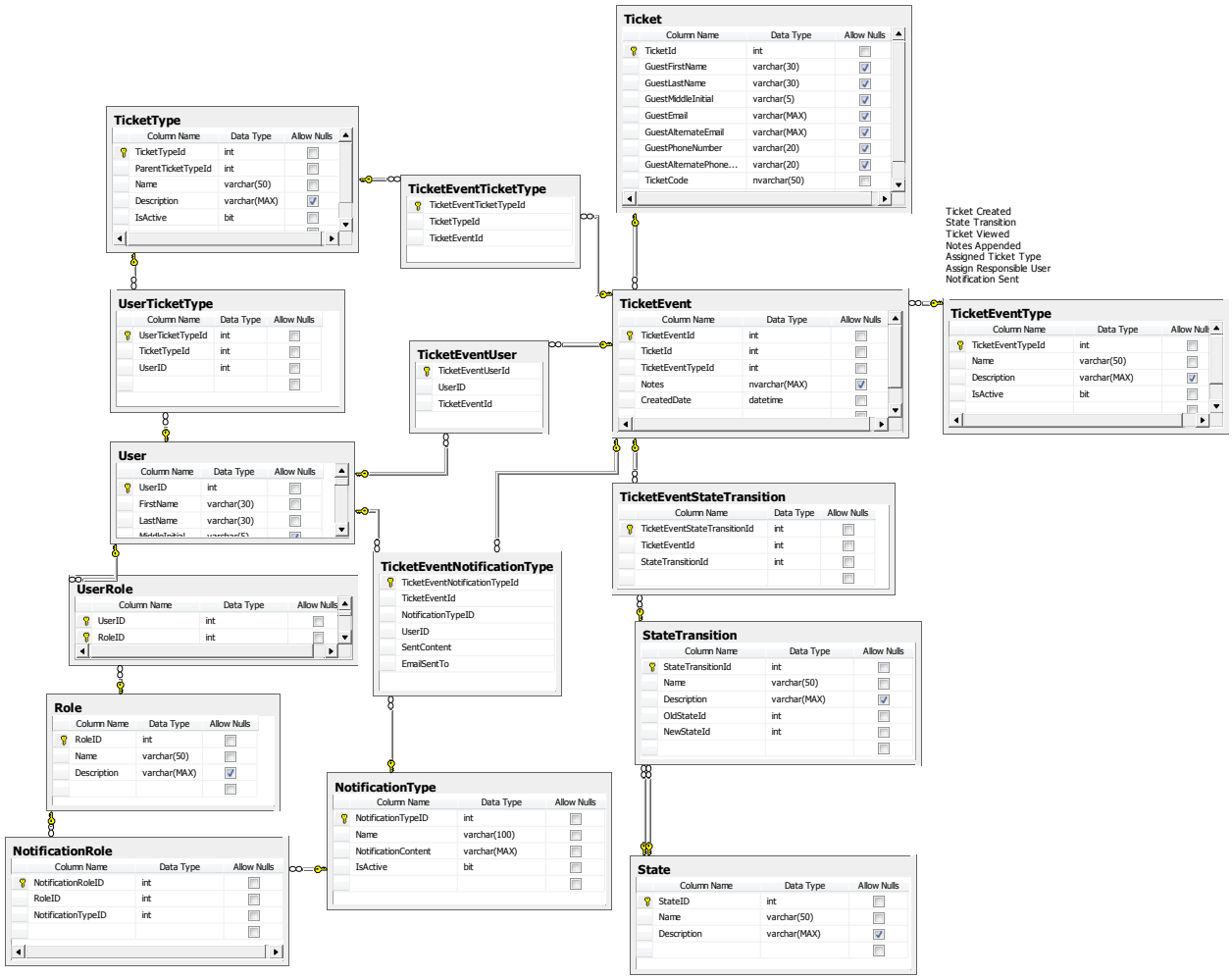
Appendix B.7: Manage Ticket (Guest)

Use Case Name	Manage Ticket (Guest)	
Use Case Description	A guest manages a ticket in the system.	
Frequency	Episodic	
Actors	Guest	
Related Use Cases	None	
Stakeholders	Guest: Ticket's information must be stored with the User's correct information.	
Happy Pathway	Guest enters ticket	
Preconditions	User has an issue that needs the attention of Housing.	
Post-Conditions	Ticket is entered into the system with the correct information. Ticket assigned a Staff Member	
Flow of Events	**Actor**	**System**
	1. User navigates to the "Enter Ticket" page. 4. User fills out the required information. 5. User enters the string in the Captcha image. 6. User clicks "Submit."	 2. System displays the "New Ticket" page. 3. System generates a Captcha image. 7. Ticket information is stored. 8. Ticket is assigned to a Staff Member. 9. Ticket state set to "Submitted" 10. Staff Member is emailed.
Alternate Pathways	Guest views ticket Guest adds note to ticket Guest cancels ticket creation	
Exception Conditions	Time out – No information is stored	

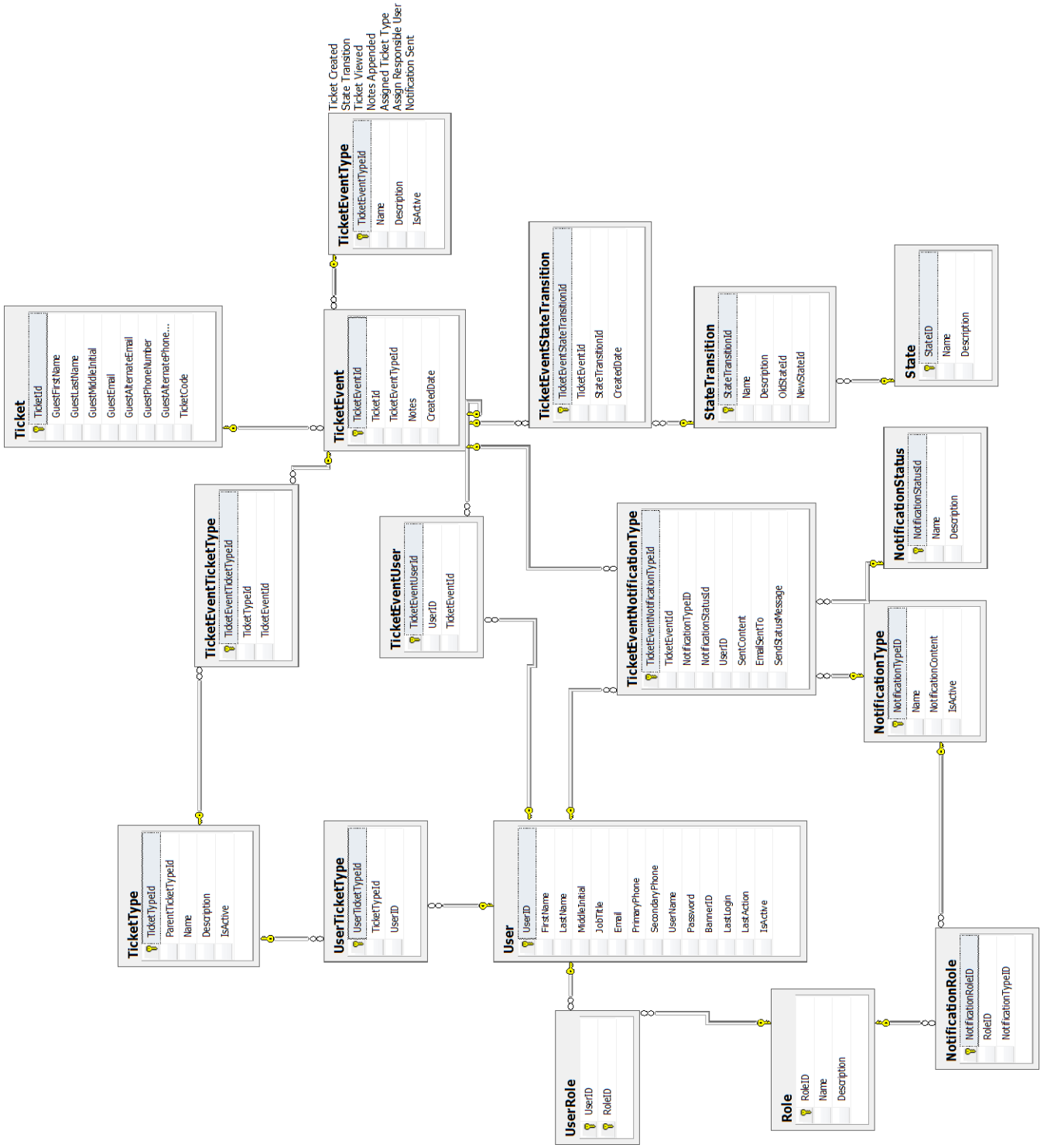
Appendix B.8: Login

Use Case Name	Login	
Use Case Description	A user logs into the system.	
Frequency	Episodic	
Actors	System Manager, System Administrator, Staff Member, Director	
Related Use Cases	Manage Ticket (Staff Member), Manage Ticket (System Manager), View Historical Reports, Manage Users, Manage Profile	
Stakeholders	System Manager: Must be redirected to the correct page depending on role. System Administrator: Must be redirected to the correct page depending on role. Staff Member: Must be redirected to the correct page depending on role. Director: Must be redirected to the correct page depending on role.	
Happy Pathway	Successful login	
Preconditions	A user exists in a system and has a username and password.	
Post-Conditions	A user is logged into the system with the correct roles and is able to see the correct pages.	
Flow of Events	**Actor**	**System**
	1. User navigates to the Login Page. 3. User enters in their username and password 4. User clicks “Login.”	2. System displays the login page for the user. 5. System checks User’s login information against the database. 6. System redirects User to the appropriate page depending on User’s roles
Alternate Pathways	User needs password reminder User submits invalid username/password combination	
Exception Conditions	A user cancels out of the log in process – user unable to pass the login screen Time out – User unable to pass the login screen.	

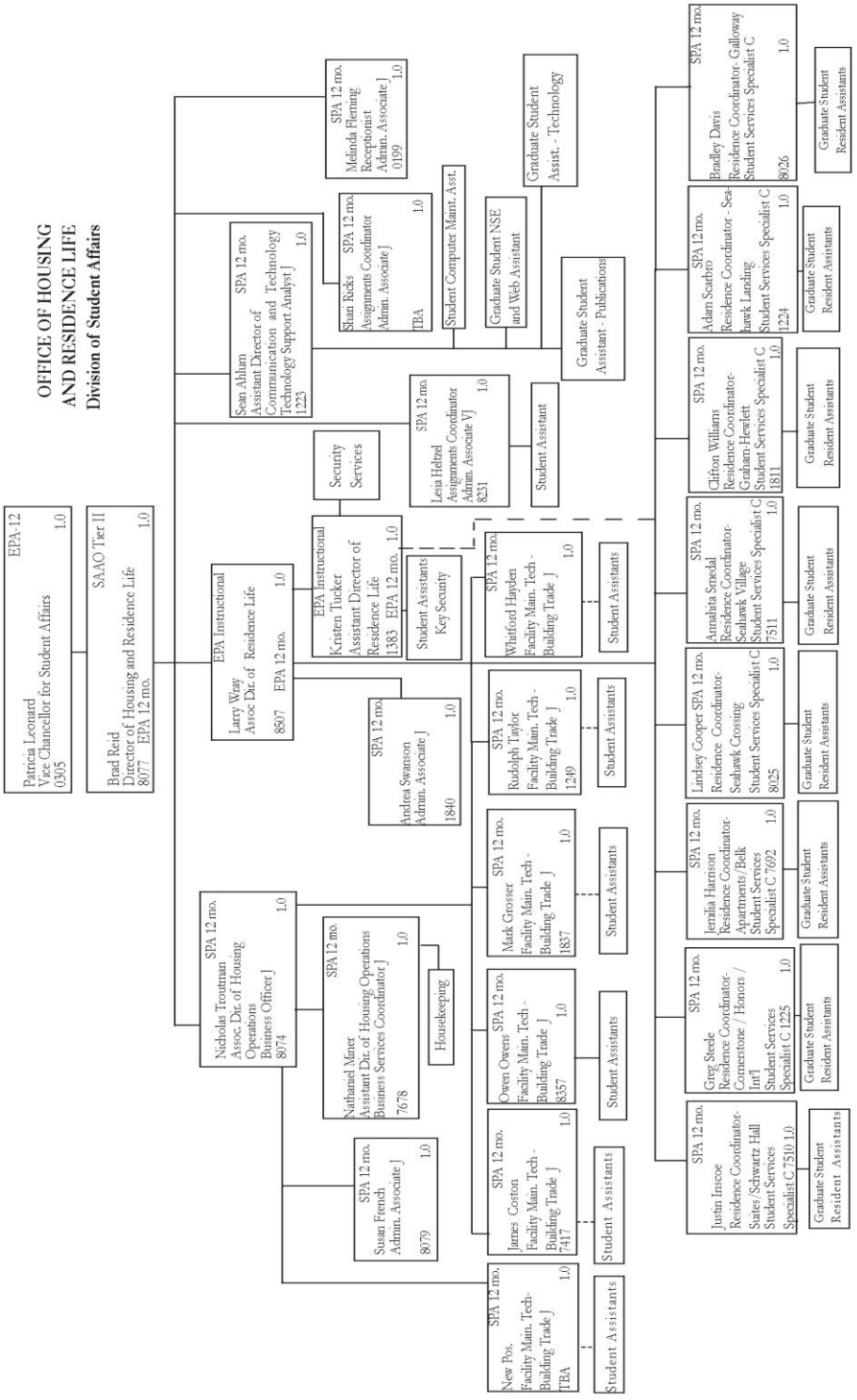
Appendix C.1: Rez-O-lution Data Model Version 1



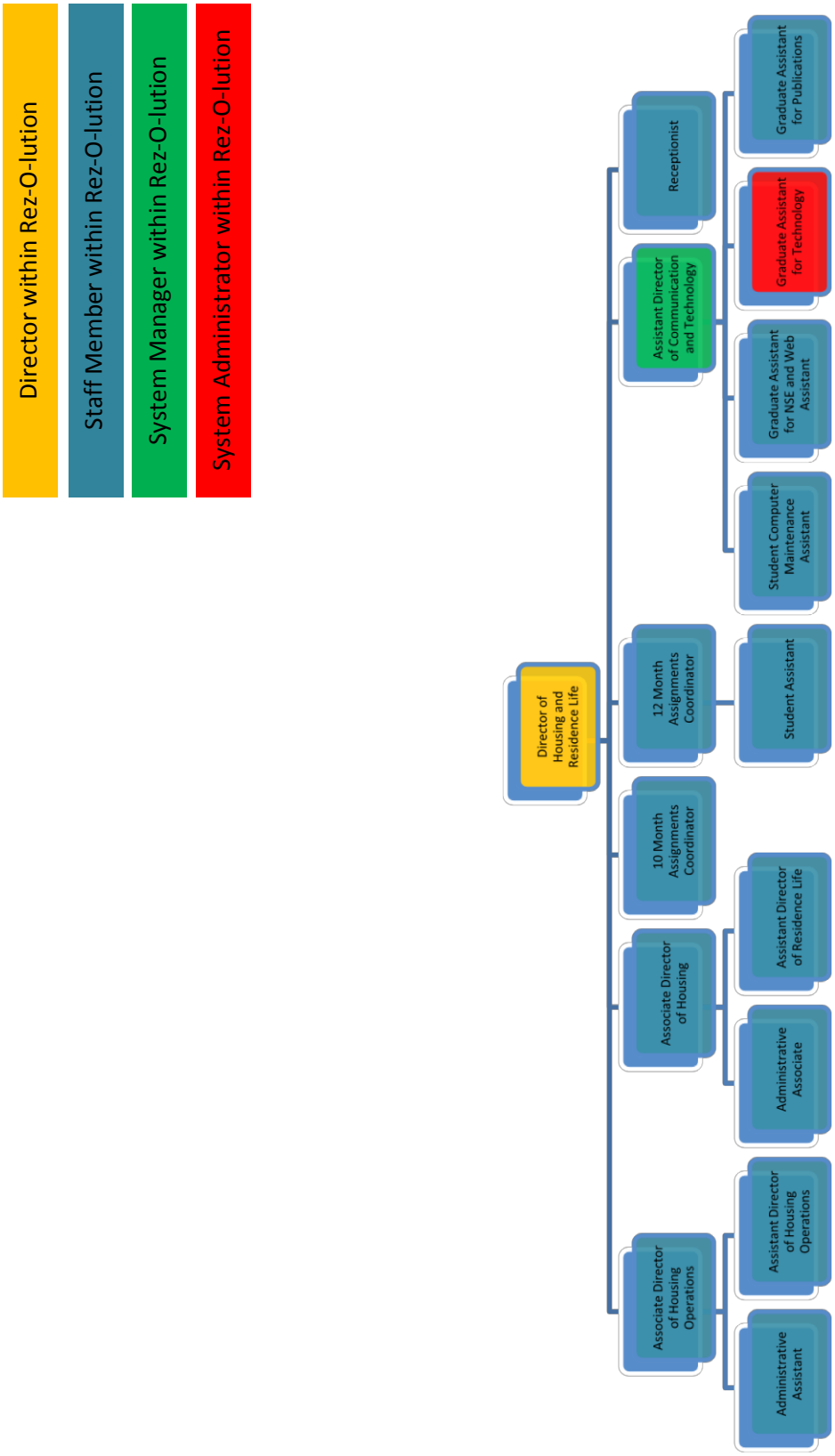
Appendix C.2: Rez-O-lution Data Model Version 2



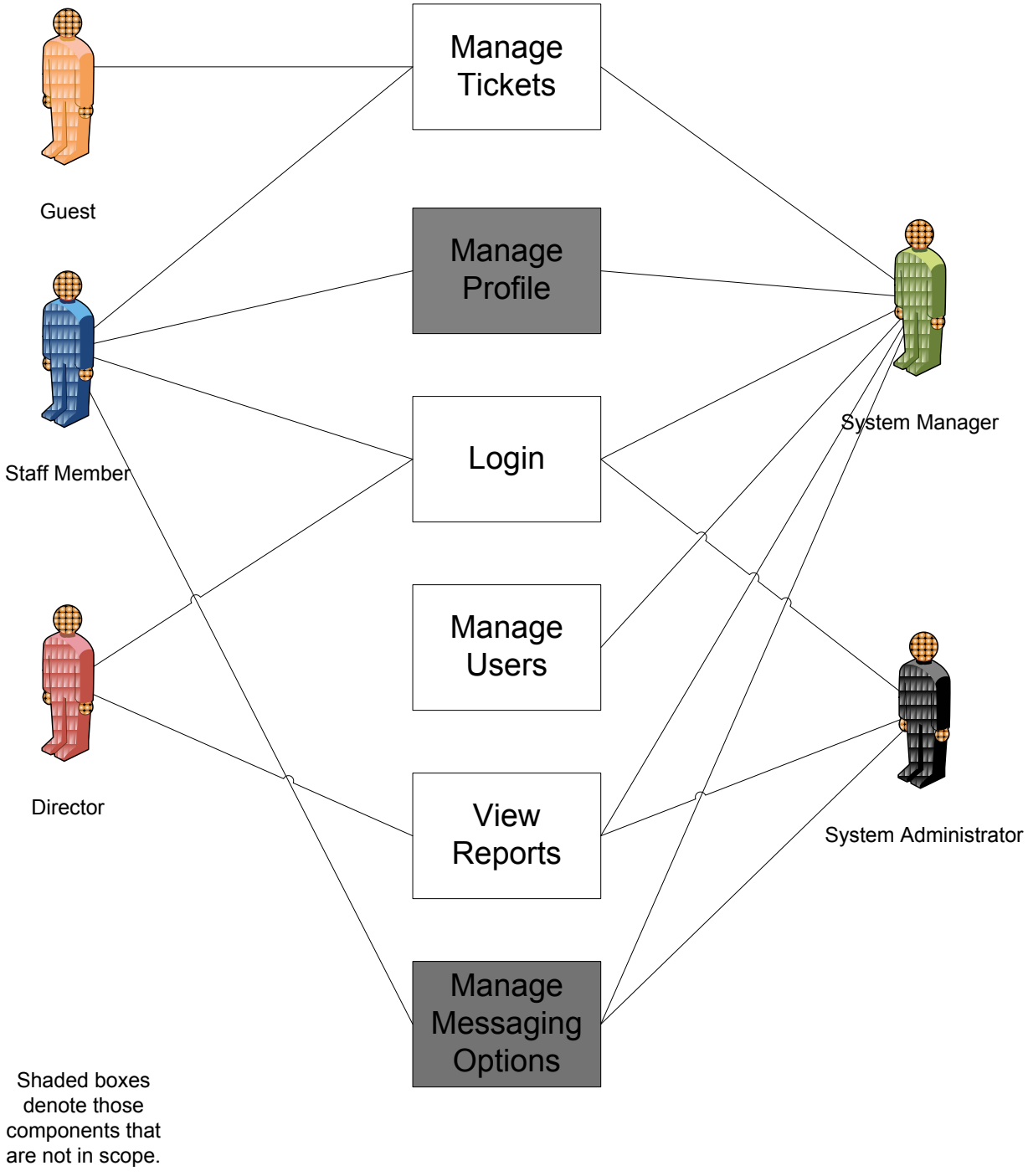
Appendix D.1: Office of Housing and Residence Life Organization Chart



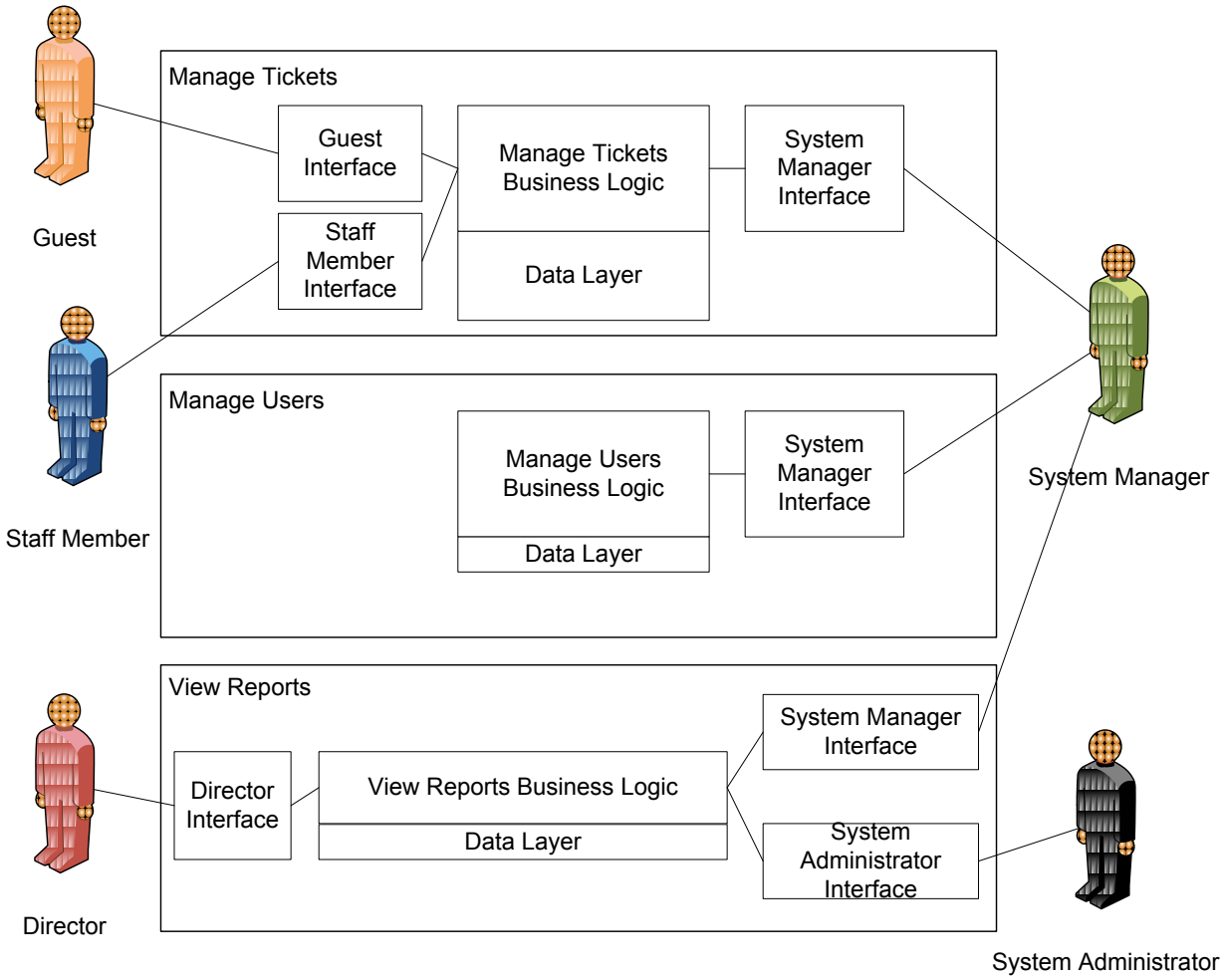
Appendix D.2: HRL Org Chart with Actor Designation

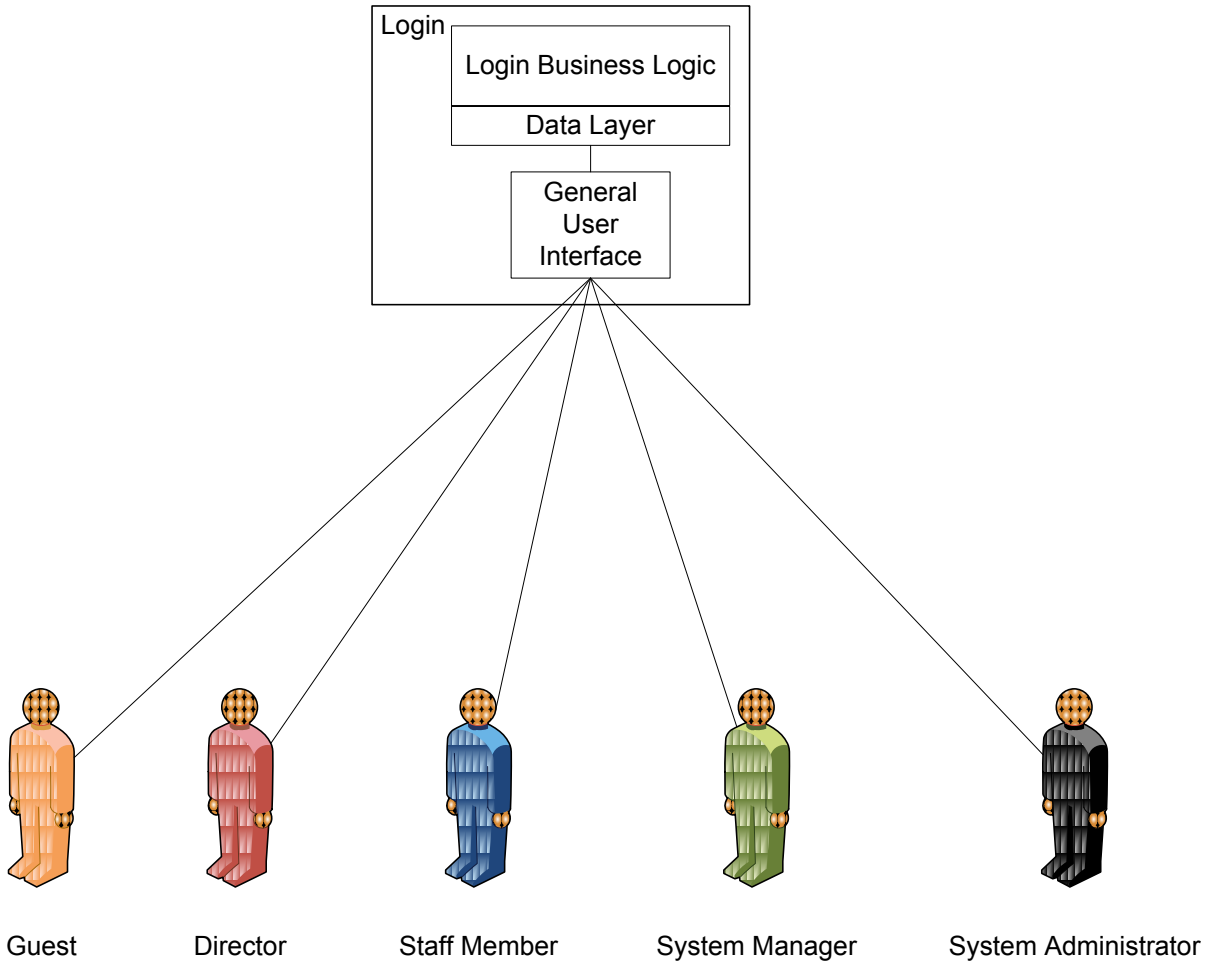


Appendix E.1: Rez-O-lution Actor Diagram

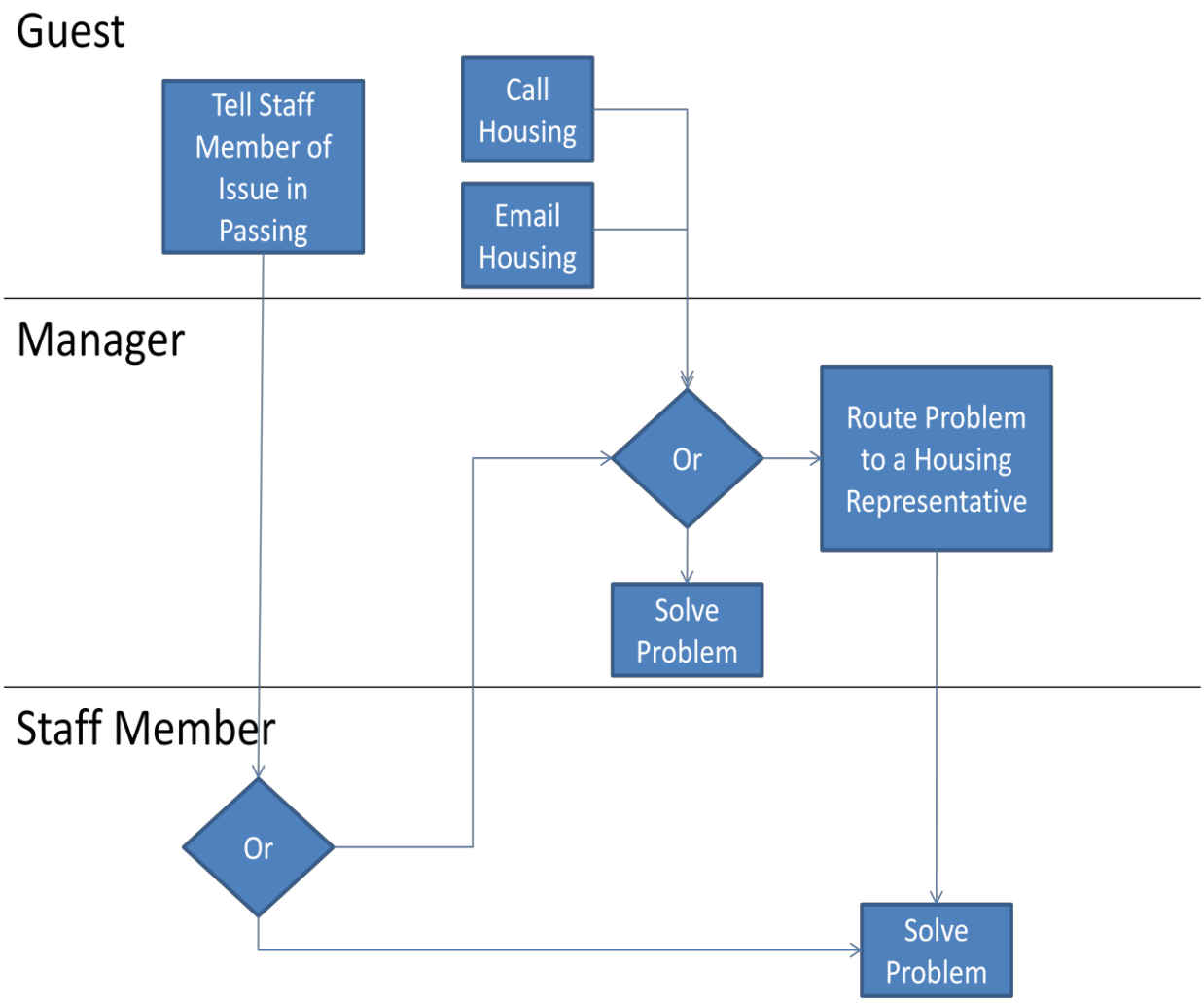


E.2: Detailed Actor Diagram

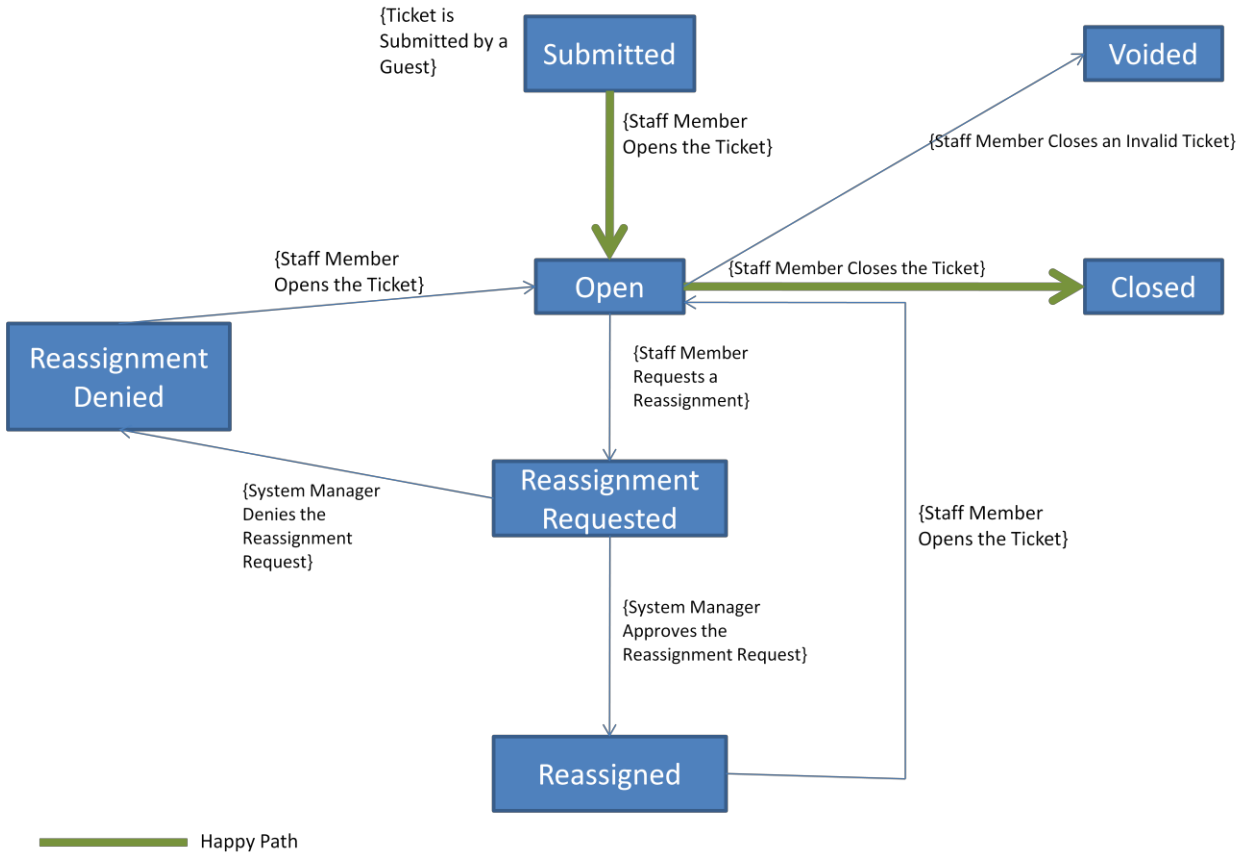



Login
Login Business Logic
Data Layer
General User Interface
Guest
Director
Staff Member
System Manager
System Administrator

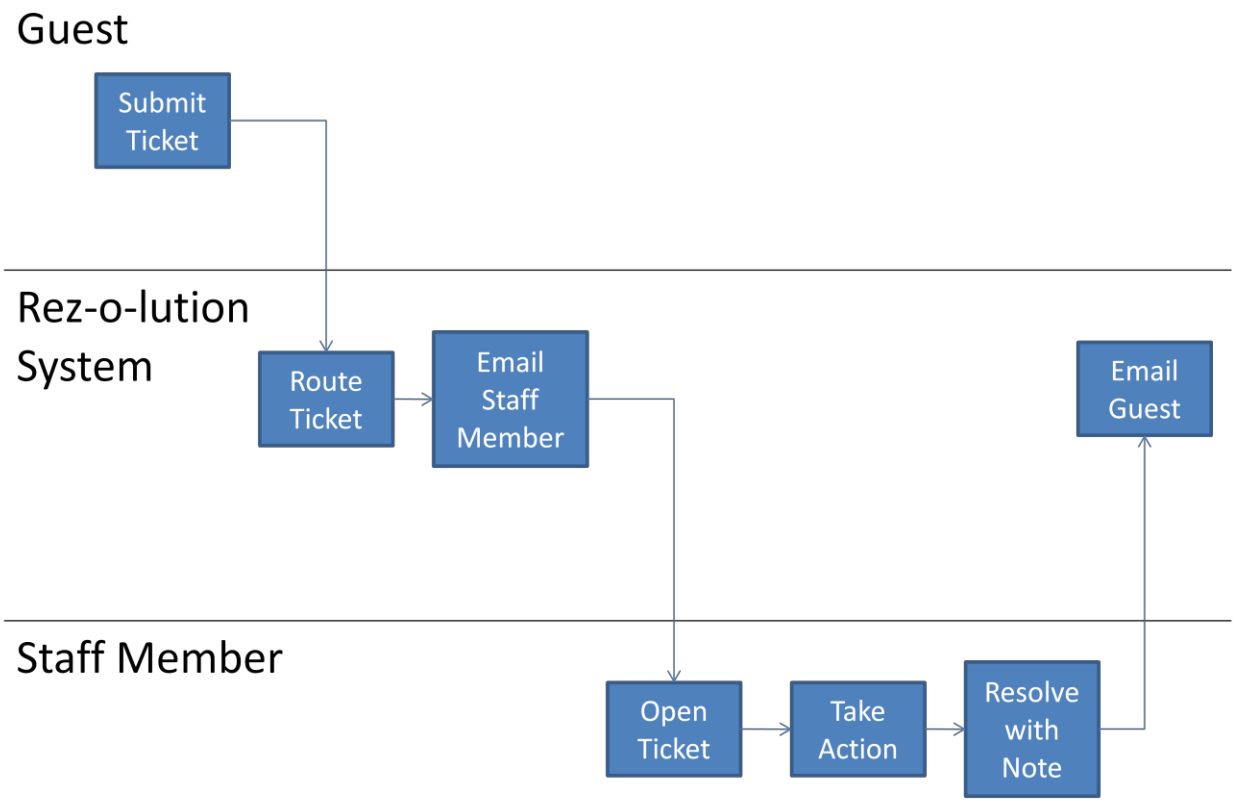
Appendix F.1: Current Housing and Residence Life Problem Solving Process



Appendix G.1: Rez-O-lution State Diagram

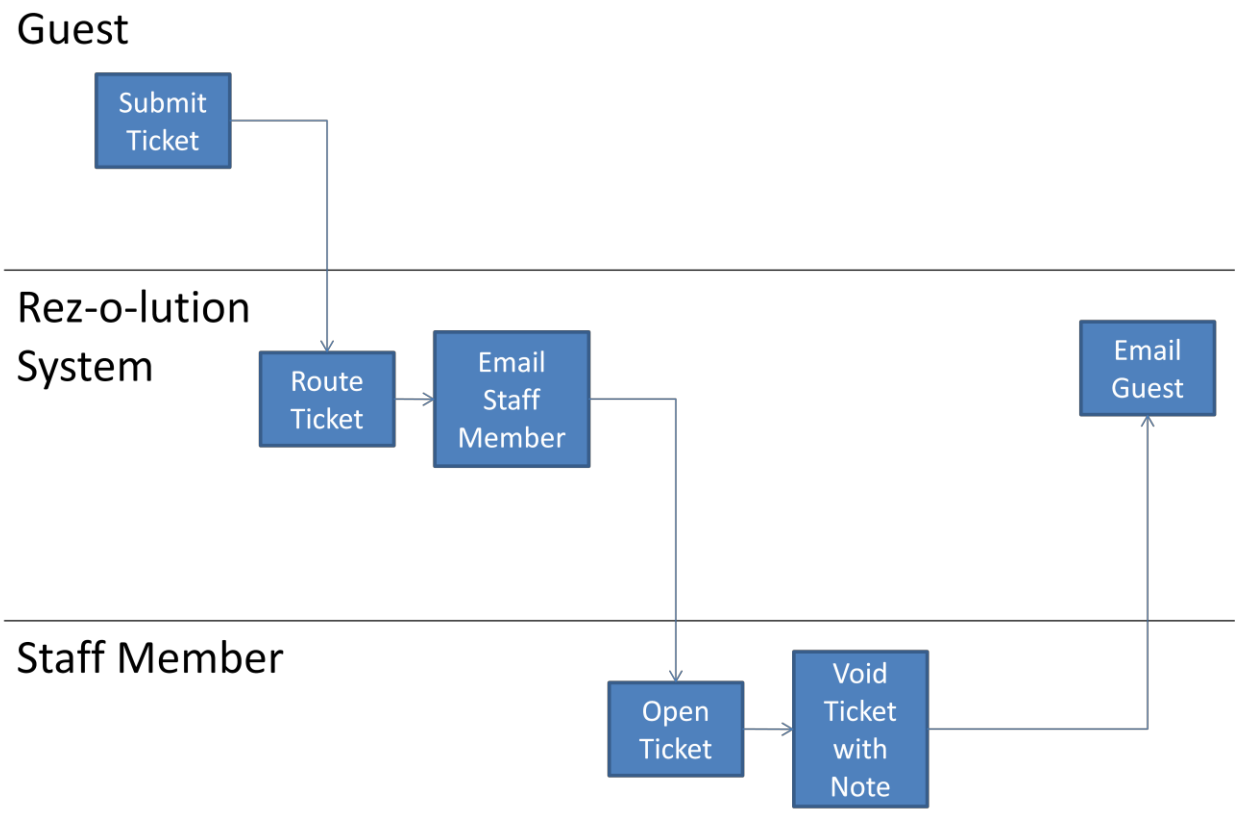


Appendix H.1: Swimlane – Ticket is Resolved Process



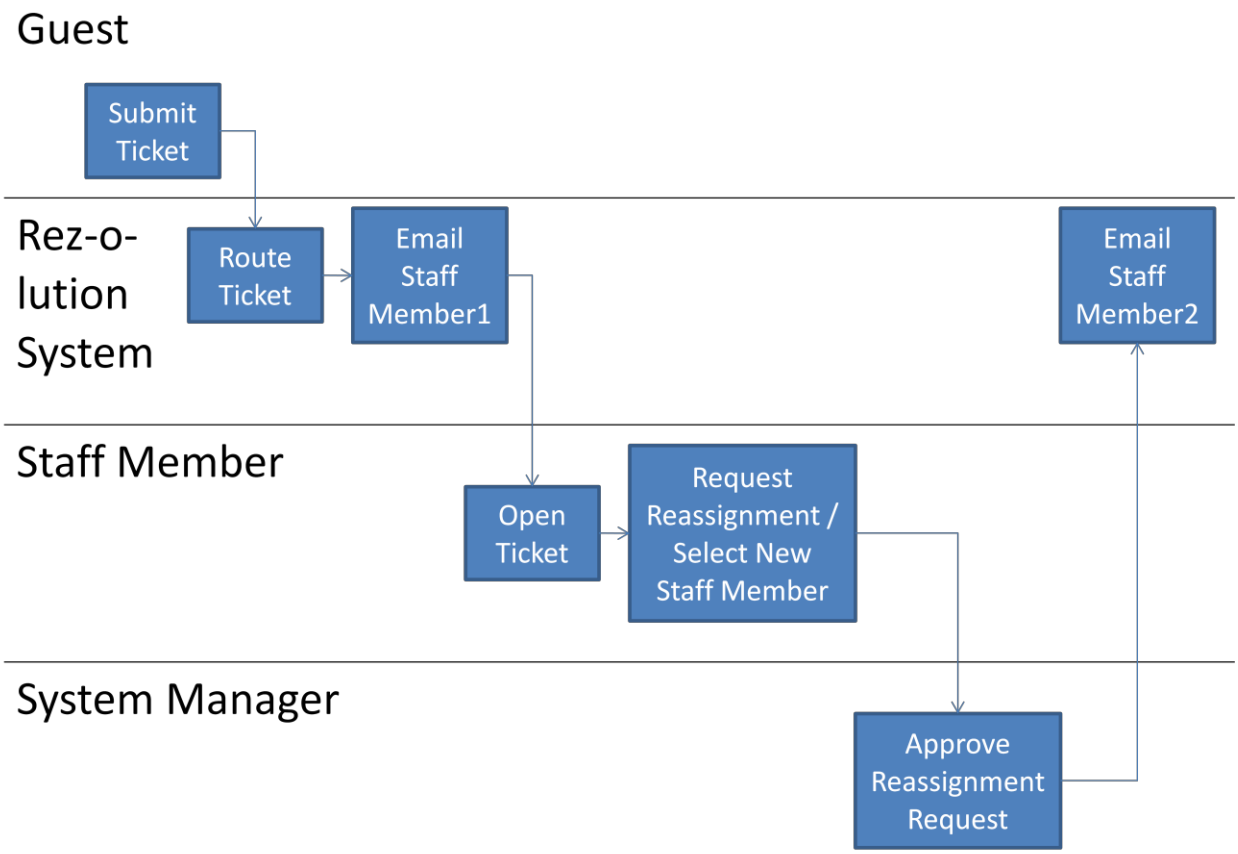
Ticket is Resolved

Appendix H.2: Swimlane – Ticket is Resolved via Void



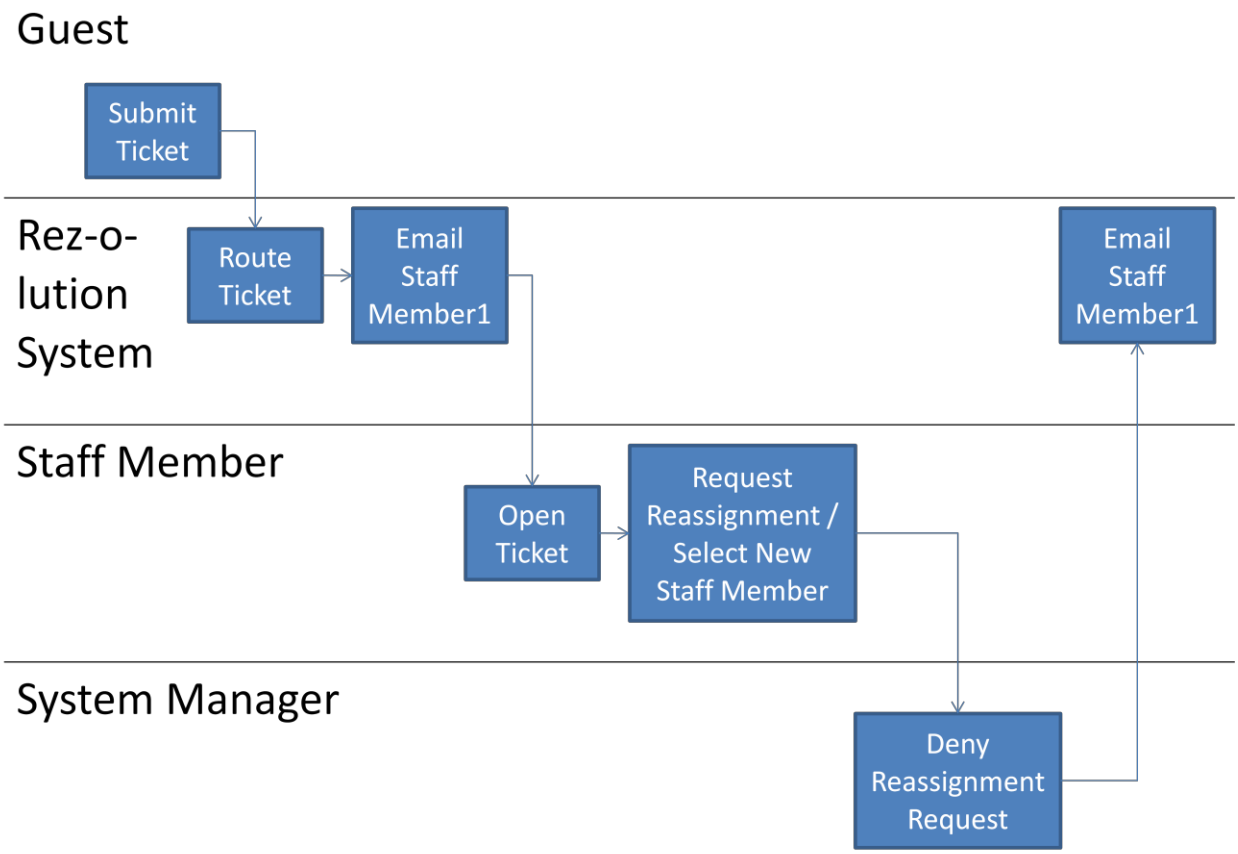
Ticket is Resolved via Void

Appendix H.3: Swimlane – Reassignment Request Approved



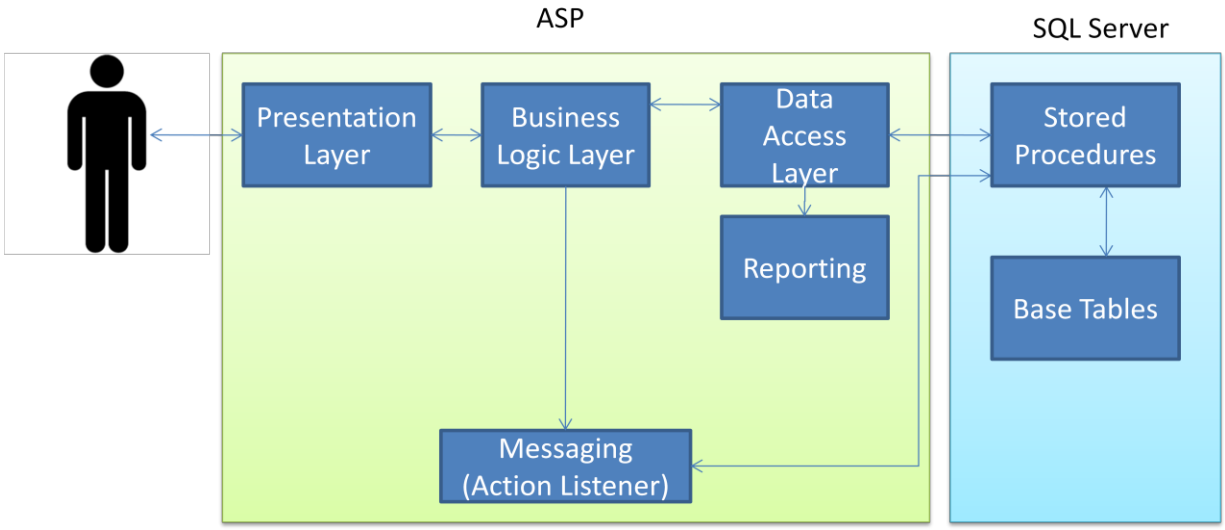
Reassignment Request Approved

Appendix H.4: Swimlane – Reassignment Request Denied



Reassignment Request Denied

Appendix I.1: Rez-O-lution System Diagram



Appendix I.1: Rez-O-lution Project Charter

Project Charter

Rez-O-lution: UNCW Housing and Residence Life Issue Ticket Management System

Chris Cotton

Business Purpose

The Department of Housing and Residence Life at UNCW receives calls and emails from students and parents about issues such as:

- Maintenance
- Housekeeping Early move-in/late move-out
- Contract cancellations
- Roommate conflicts
- The purpose of this project is to provide an online system for submitting these issues, routing the issues to the appropriate person, and tracking their resolution.
    - Current process is not clearly defined and managed by one person

Business Objectives

- Provide various users with a convenient way to report issues to Housing
- Reduce amount of time for Housing to be aware of issues
- Reduce amount of time for issue resolution
- Provide a way to audit actions taken by Housing in the process of resolving an issue
- Increase productivity by reducing the amount time for Housing to respond to issues
- Increase accountability
- Generate aggregate reports about issue counts and resolution times.
- Notify the requester when a ticket is resolved
- Eliminate the staff resource needed to serve as electronic distributor of email
- Specific Goals:
    - 70% of all Housing issues come through this system
    - 75% of tickets are closed within 24 hours
    - Year-end reporting to identify specific areas for targeted resource deployment

Desired Features

- Ability to automatically forward issues (tickets) to the person best suited to solve the issue
- Track events within the system for reporting purposes

- Maintenance issues are redirected to the maintenance request system. These issues are logged in the system for reporting purposes
    - All maintenance requests are handled by the housing maintenance department and through a different system
- Simple interface for guests(the people submitting a ticket) that reduces the ability for malicious activity such as script attacks
- Back-end integration to RMS for seamless CRM capabilities
    - Client desired feature.
    - May be out of scope for this particular project's time line, but may be something we can explore in the future.

Critical Success Factors

- Simplicity for users
- Ability to track events within the system through some logging mechanism
- Reliability of system operations
- Continued ability for ITSD support and maintenance

Constraints

- Authentication – no 2-step activation of tickets
- Integration to other systems will be link-only
    - Maintenance Request System
- Project completion date is before May 2010

Risks

- Learning curve – ASP.Net, Scaffolding, Team Foundation Server
- User feedback – Ability to obtain information required for the development of this system
- Project deadlines – Ability to meet timelines within the project
- Separation between Housing and Residence Life and Project – Ability to continue project should something happen within Housing and Residence Life
- Academic goals and client goals may not align
- Staff member buy-in
- Maintenance of the project after graduation

Roles and Responsibilities

- Christopher Cotton – Developer
- Douglas Kline – Project committee chairman
- Charles Laymon – Consultant and project Committee member
- Ron Vetter – Project committee member
- Thomas Janicki – Project committee member

- Sean Ahlum – Asst Director of UNCW's Housing and Residence Life, client for the system
- Bradley Reid – Director of UNCW's Housing and Residence Life, subject matter expert
- Larry Wray – Assoc Director of UNCW's Housing and Residence Life, subject matter expert
- Nic Troutman – Assoc Director of UNCW's Housing and Residence Life, subject matter expert
- Nate Miner – Asst Director of UNCW's Housing and Residence Life, subject matter expert
- Kristen Tucker – Asst Director of UNCW's Housing and Residence Life, subject matter expert

Locations of Interest

- University of North Carolina Wilmington
- Computer Information Systems Building
- Server Room (CIS 2035a)
- Wilmington, North Carolina

Actors

Guest – parent or student

- Create ticket
- Add ticket note
- View Ticket

Staff Member

- Create ticket
- Update ticket information
- View tickets assigned to staff member
- Request for ticket re-assignment

System Manager

- Create Staff Member
- Manage Staff Member information
- Create Users
- Manage User information
- Create ticket
- Append Ticket Notes
- Create ticket categories
- Manage ticket categories
- View Reports
- Approve a Reassignment Request
- Deny a Reassignment Request
- View Tickets

Director

- View Reports

System Administrator

- Create data logs
- View reports

Event List/Event Table

Subject	Verb	Object	Frequency	Arrival Pattern	Response	Use Case
Guest	Creates	Ticket	30-200/day	Episodic	Routed to appropriate Staff Member. Ticket initialized to Submitted state. Staff Member is messaged.	Manage Tickets (Guest)
Guest	Views	Ticket	30-200/day	Episodic	Ticket information relevant to corresponding ticket is displayed including ticket status and notes	Manage Tickets (Guest)
Guest	Appends	Ticket Notes	10/day	Episodic	Ticket Notes are added. Staff Member is messaged.	Manage Tickets (Guest)
Staff Member	Views	Ticket	200/day	Episodic	Ticket moves from Submitted state to Open state	Manage Tickets (Staff Member)
Staff Member	Resolves	Ticket	30-200/day	Episodic	Ticket moves from Open state to Closed state. Guest is messaged.	Manage Tickets (Staff Member)
Staff Member	Requests Reassignment	Ticket	1/week	Episodic	Ticket moves from open state to reassignment requested state	Manage Tickets (Staff Member)
Staff Member	Appends Ticket Notes	Ticket	10/week	Episodic	Ticket Notes are added	Manage Tickets (Staff Member)
Manager	Approves Reassignment	Ticket	1/week	Episodic	Ticket moves from Open state to Reassigned state	Manage Tickets (System Manager)
Manager	Denies Reassign	Ticket	1/week	Episodic	Ticket moves from Open state to	Manage Tickets (System

	ment				Reassignment Request Denied and then to Open again.	Manager)
Manager	Manages	Assignments	1/month	Episodic	Ticket Category assigned to a Staff member. Staff Member is messaged.	Manage Users (System Manager)
Manager	Resolves	Ticket	3/week	Episodic	Manager sets a ticket status to "Closed". Ticket cannot be re opened.	Manage Tickets (System Manager)
Manager	Creates	Category	1/term	Episodic	Ticket Category is created.	Manage Users (System Manager)
Manager	Views	Reports	2/day	Episodic	Report is displayed.	Historical Reports
Manager	Assigns	Role	2-3/term	Episodic	Manager assigns a user a role. User is messaged.	Manage Users (System Manager)
Manager	Views	Tickets	3/day	Episodic	Manager views a list of tickets.	Manage Tickets (System Manager
Manager	Appends Ticket Notes	Tickets	1/week	Episodic	Ticket notes are added.	Manage Tickets (System Manager)
Manager	Manages	Tickets	3/day	Episodic	Manager updates ticket information.	Manage Tickets (System Manager)
Director	Views	Reports	1/term	Episodic	Appropriate report is displayed	Historical Reports
Manager	Creates	User	5/term	Episodic	User exists in the system.	Manage Users
Manager	Manager s	User	2/term	Episodic	User's information is updated.	Manage User
System Administrator	Views	Reports	2/day	Episodic	Report is displayed.	Historical Report

Use Cases

- Manage Messaging Options
- Edit Profile
- Manage Staff Members
- Historical Reports
- Manage Tickets (Manager)

- Manage Tickets (Staff Member)
- Manage Ticket Guest

Use Case Course of Events

Note – Paths events colored in green are considered to be in the "happy path"

Manage Messaging Options: Responsible for managing the messaging options. Used by every user. Paths: Update Messaging Options

Edit Profile: Helps users update their profile which contains information specific to them. Used by every user. Paths: Update Profile

Manage Users: Responsible for managing users and their roles as well as managing ticket categories. Used by the System Manager. Paths: Manage Assignments, Create Ticket Category, Manage Ticket Category, Assign Role, Create User, Manage User

Historical Reports: Responsible for the viewing of historical reports. Used by the System Manage, System Administrator, and Director. Paths: View Reports

Manage Tickets (Manager): Responsible for anything having to do with the creation of or editing of a ticket. Used by the System Manager. Paths: Approve Reassignment, Deny Reassignment, Manage Tickets, Append Notes, Resolve Ticket, View Ticket

Manage Tickets (Staff Member): Responsible for anything having to do with the creation of or editing of a ticket. Used by the Staff Member. Paths: View Ticket, Resolve Ticket, Request Reassignment, Append Ticket Notes

Messaging: Responsible for messaging users. System use only. Paths: Message User(Event)

Manage Ticket (Guest): Responsible for anything having to do with the creation or editing of a ticket. Used by Guests. Paths: View Ticket, Append Ticket Notes, Create Ticket

Preliminary Execution Architecture

- Visual Studio 2008 (VS08) – development environment, VB.net and C#.net, .aspx
- SQL Server Reporting Services (SSRS) – Manage Reports
- SQL Server Business Intelligence Studio (BIDS) – Create reports
- SQL Server 2008 – Create and manage database
- Microsoft IIS - .Net 3.5, system deployment

Project Infrastructure

Team Foundation Server 2008

- automated build

- version control
- share-point
- task management
- unit tests
- remote deployment
  Development Server
  - VS08
  - SQL Server 08 – SSRS, BIDS
  - Backups of project will be done every Friday. Backups will be located on an external HD off site as well as a mobile memory stick

Project Release Strategy

- The project will be released in stages
- Each stage will include additions to the previous release
- After a release has been tested to satisfaction, a new release will be made
- New releases will contain new parts and/or updates to previous releases
- Decision for what will be in each release will be determined from the previous releases
- Goal is to include every component that is necessary to the system's functionality

Application Contingency

- After completion, the application will be maintained by Housing and Residence Life by Housing staff.

Appendix J.1: Rez-O-lution Data Dictionary

Table Name	Field Name	Data Type	Allow Nulls	Primary Key(PK) / Foreign Key (FK)
TicketType	TicketTypeId	int	No	PK
	ParentTicketTypeId	int	No	FK
	Name	varchar(50)	No	
	Description	varchar(MAX)	Yes	
	IsActive	bit	No	
UserTicketType	UserTicketTypeId	int	No	PK
	TicketTypeId	int	No	FK
	UserID	int	No	FK
User	UserID	int	No	PK
	RoleID	int	No	FK
	FirstName	varchar(30)	No	
	LastName	varchar(30)	No	
	MiddleInitial	varchar(5)	Yes	
	JobTitle	varchar(MAX)	Yes	
	Email	varchar(MAX)	No	
	PrimaryPhone	varchar(15)	Yes	
	SecondaryPhone	varchar(15)	Yes	
	UserName	nvarchar(30)	No	
	Password	varchar(50)	No	
	BannerID	varchar(12)	Yes	
	LastLogin	datetime	Yes	
	LastAction	int	Yes	
Role	RoleID	int	No	PK
	Name	varchar(50)	No	
	Description	varchar(MAX)	Yes	
UserRole	UserID	Int	No	PK
	RoleID	Int	No	PK
TicketEventNotificationType	TicketEventNotificationTypeId	int	No	PK
	TicketEventId	int	No	FK
	NotificationStatusId	Int	No	FK
	NotificationTypeID	int	No	FK
	UserID	Int	No	FK
	SentContent	Varchar(max)	No	
	EmailSentTo	Varchar(250)	No	
TicketEventUser	TicketEventUserId	int	No	PK
	UserID	int	No	FK
	TicketEventId	int	No	FK
TicketEventTicketType	TicketEventTicketTypeId	int	No	PK

	TicketTypeId	int	No	FK
	TicketEventId	int	No	FK
Ticket	TicketId	int	No	PK
	GuestFirstName	varchar(30)	Yes	
	GuestLastName	varchar(30)	Yes	
	GuestMiddleInitial	varchar(5)	Yes	
	GuestEmail	varchar(MAX)	Yes	
	GuestAlternateEmail	varchar(MAX)	Yes	
	GuestPhoneNumber	varchar(20)	Yes	
	GuestAlternatePhoneNumber	varchar(20)	Yes	
	TicketCode	nvarchar(10)	Yes	
TicketEvent	TicketEventId	int	No	PK
	TicketId	int	No	FK
	TicketEventTypeId	int	No	FK
	Notes	nvarchar(MAX)	Yes	
	CreatedDate	datetime	No	
TicketEventStateTransition	TicketEventStateTransitionId	int	No	PK
	TicketEventId	int	No	FK
	StateTransitionId	int	No	FK
StateTransition	StateTransitionId	int	No	PK
	Name	varchar(50)	No	
	Description	varchar(MAX)	Yes	
	OldStateId	int	No	FK
	NewStateId	int	No	FK
State	StateID	int	No	PK
	Name	varchar(50)	No	
	Description	varchar(MAX)	Yes	
TicketEventType	TicketEventTypeId	int	No	PK
	Name	varchar(50)	No	
	Description	varchar(MAX)	Yes	
	IsActive	bit	No	
NotificationType	NotificationTypeID	int	No	PK
	Name	varchar(50)	No	
	NotificationContent	varchar(MAX)	No	
	IsActive	bit	No	
NotificationRole	NotificationRoleID	int	No	PK
	RoleId	Int	No	PK
	NotificationTypeID	Int	No	FK
NotificationStatus	NotificationStatusId	Int	No	PK
	Name	varchar(50)	No	
	Description	Varchar(max)	No	

Appendix K.1: Rez-O-lution Risks and Mitigation Techniques Chart

Risk	Description	Mitigation Technique
No Guest Login	Guests will not have the ability to login to the system. This means Rez-O-lution will not be able to collect much information on those people entering tickets and will make it more vulnerable to malicious behavior.	• Implement Captcha on ticket creation page • Implement AJAX script deterring tools (including No Bot) on the ticket creation page • Storing of all host headers for all guest ticket submissions

Separation Between Housing and Residence Life and Project	This is the risk associated with what would happen if HRL decided that they didn't want to pursue this project at sometime during its development	• Detailed project charted signed off by all participating parties including
		  ○ Clients
		  ○ Developer
		  ○ Project Committee
		• Constant communication between developer and clients

User Feedback	The inability to collect feedback required for the development of Rez-O-lution	• Constant communication between the developer and client • Clear set of requirements laid out in the project charter
Learning Curve	The amount of time and effort it may take in learning and using new software tools and packages and the amount of time and effort required to integrate all of these software tools and packages	• Selection of software experts for project committee • Documentation throughout the project • Attended class in ASP.NET to help gain a better understanding • Most of the project done in Microsoft products for easy integration

Academic Goals and Client Goals May Not Align	The risks associated when the client goals may not meet or may exceed those goals required for the successful completion of a master's project	• Detailed project charter signed off by all participating parties including
		  ○ Clients
		  ○ Developer
		  ○ Project Committee
		• Project proposal document and presentation approved by project committee

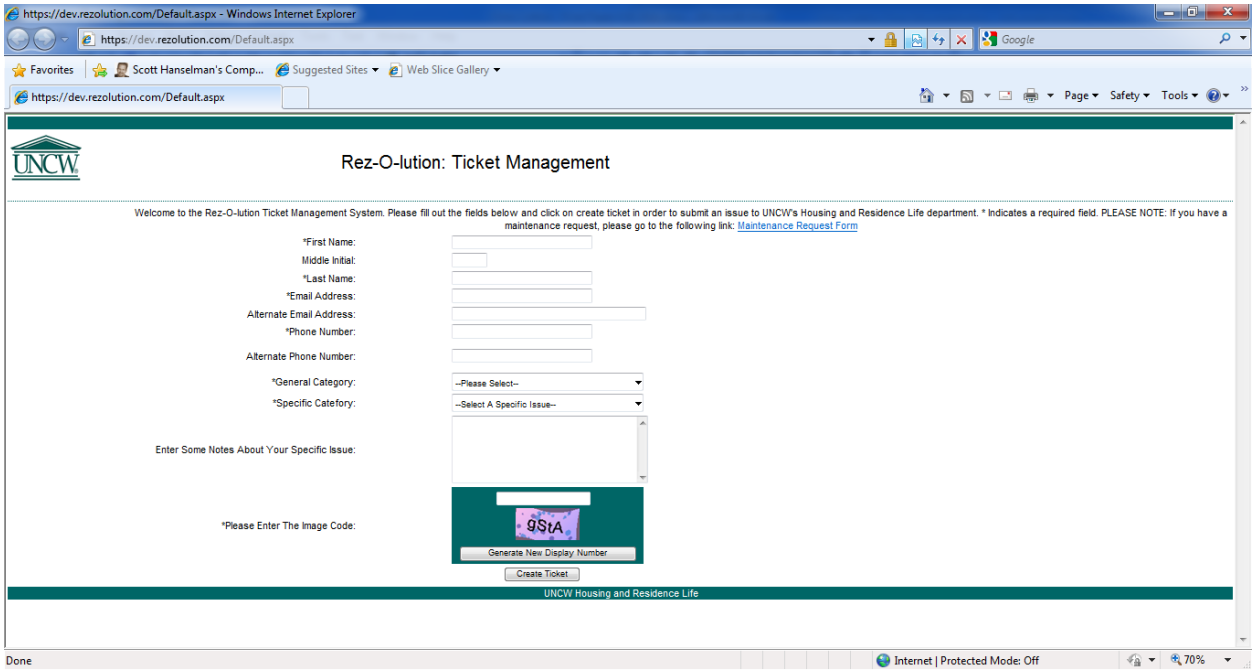
Project deadlines	Ability to meet timelines within the project	• Detailed project charter • Detailed scope laid out in a proposal document • Clear scope as laid out in the

		proposal document

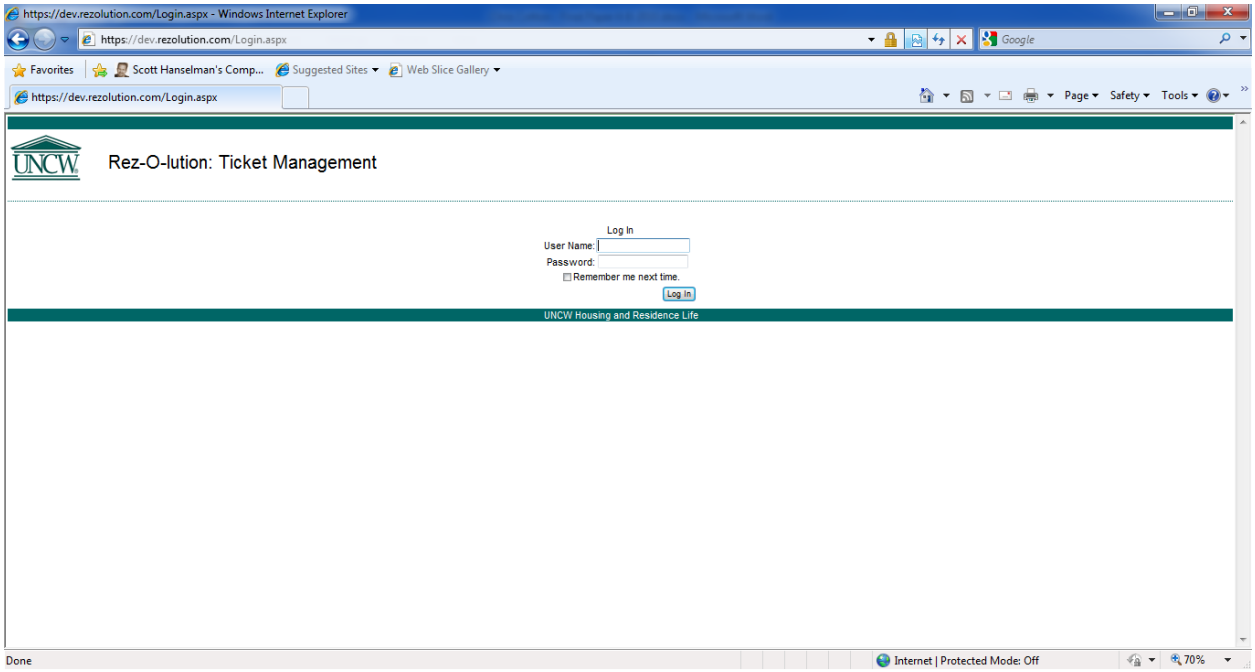
Maintenance of the Project After Developer's Graduation	Who will be responsible for the maintenance of the project after the developer has completed the project and graduated	• Contingency plan laid out in project proposal document and project charter and agreed on by all participating clients
		  ○ Rez-O-lution hosted by ITSD
		  ○ Maintained by HRL

Staff Member Buy-in	The risk associated with whether or not staff members use Rez-O-lution for whatever reason (i.e. complexity, lack of technical knowledge, etc.)	• Simplistic design • User feedback • Training materials and documentation will be provided

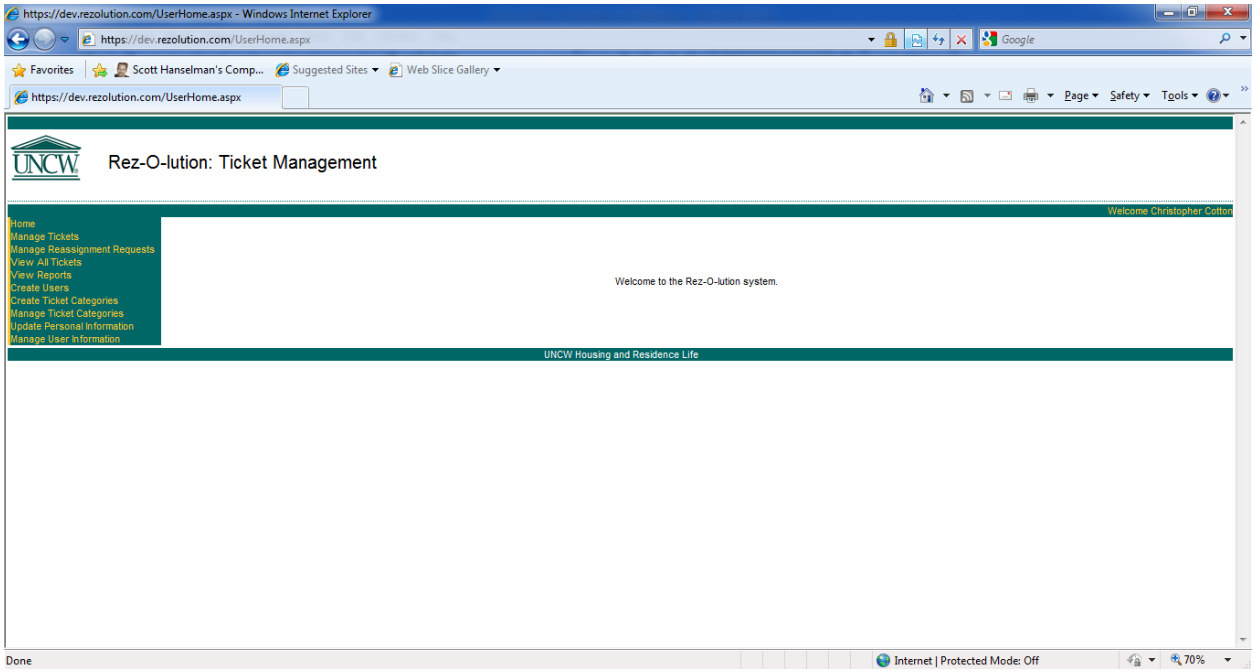
Appendix J.1: Rez-O-lution Ticket Entry Screen



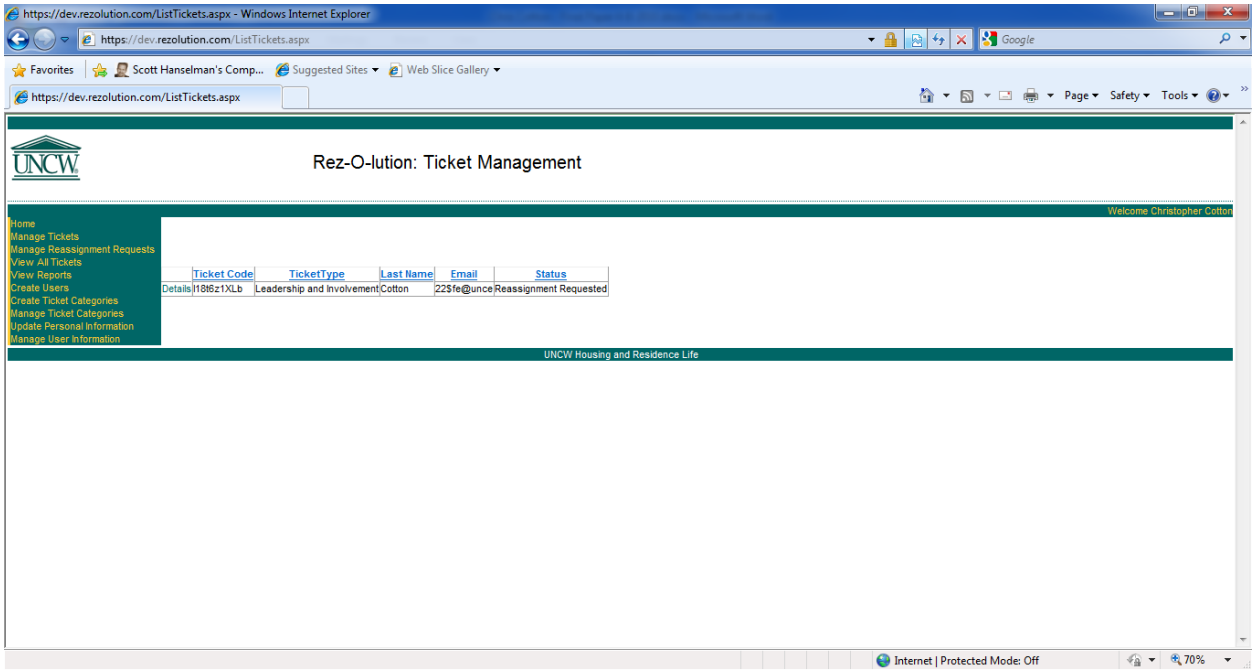
Appendix J.2: Rez-O-lution Login Screen

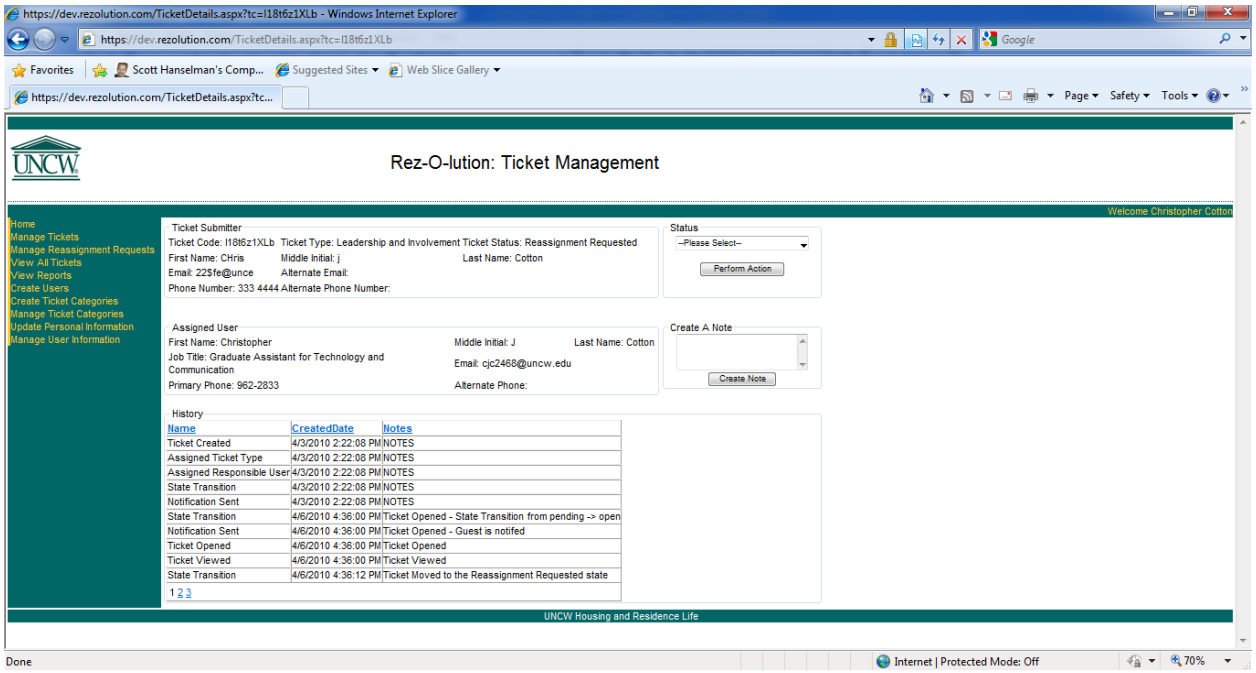


Appendix J.3: Rez-O-lution User Home Page

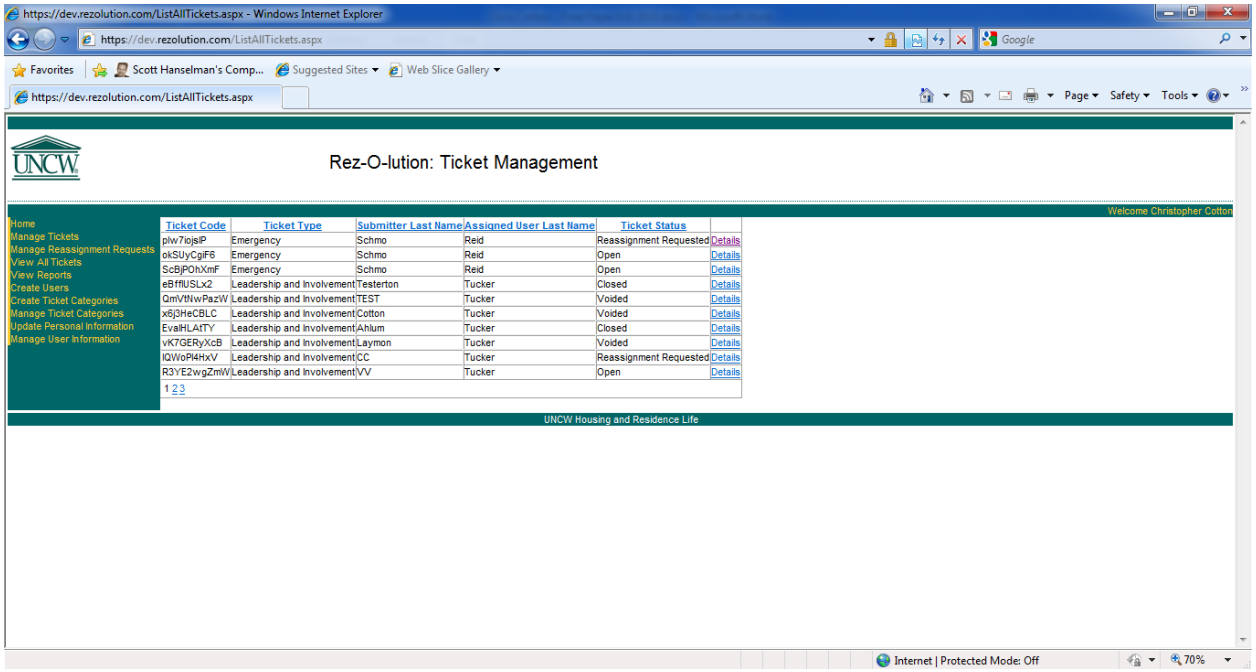


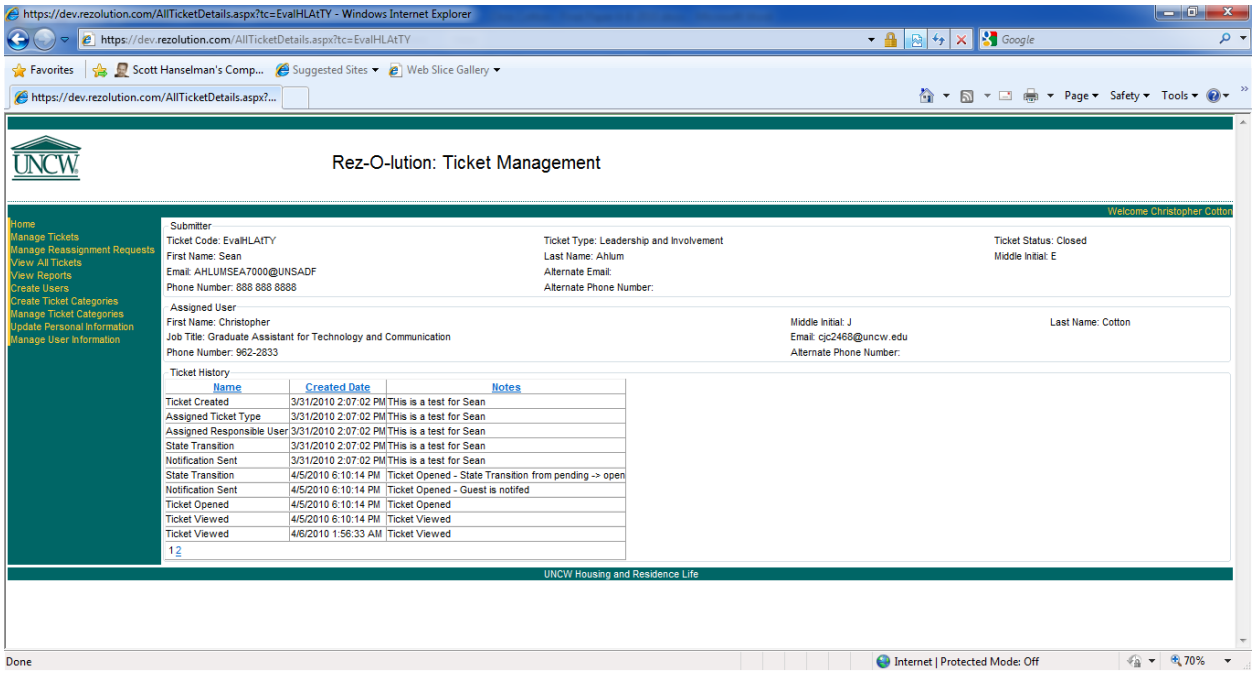
Appendix J.4: Rez-O-lution Ticket Management Screen and Details



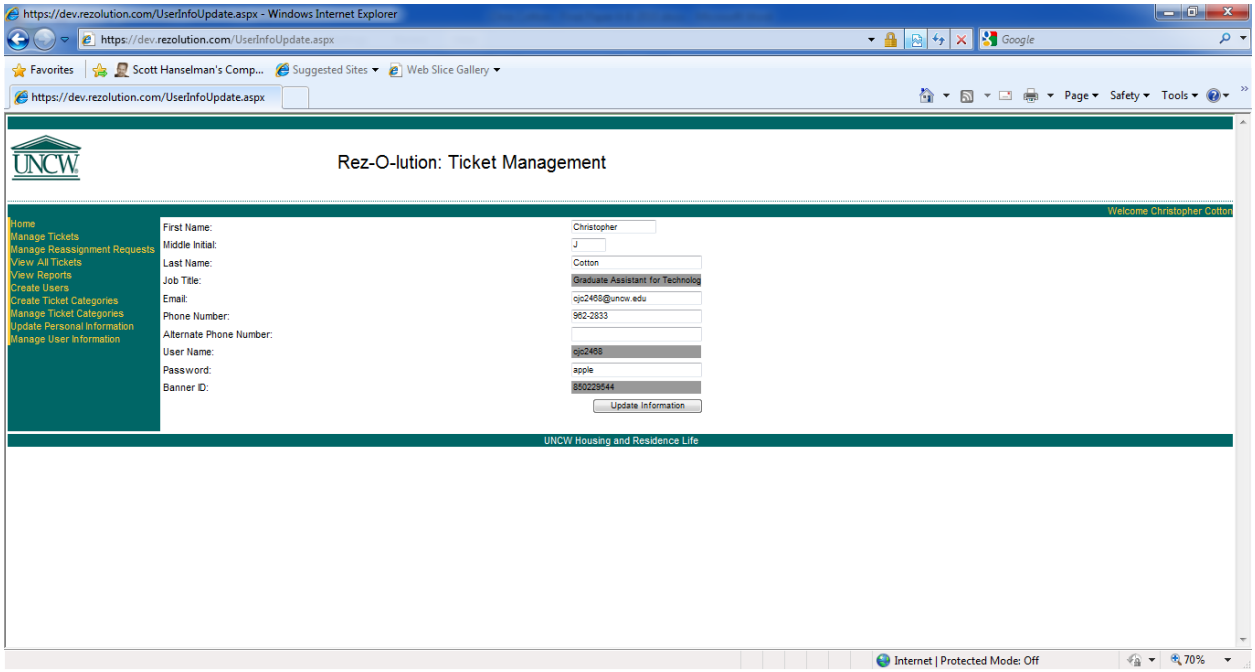


Appendix J.5: List All Tickets Screen and Details

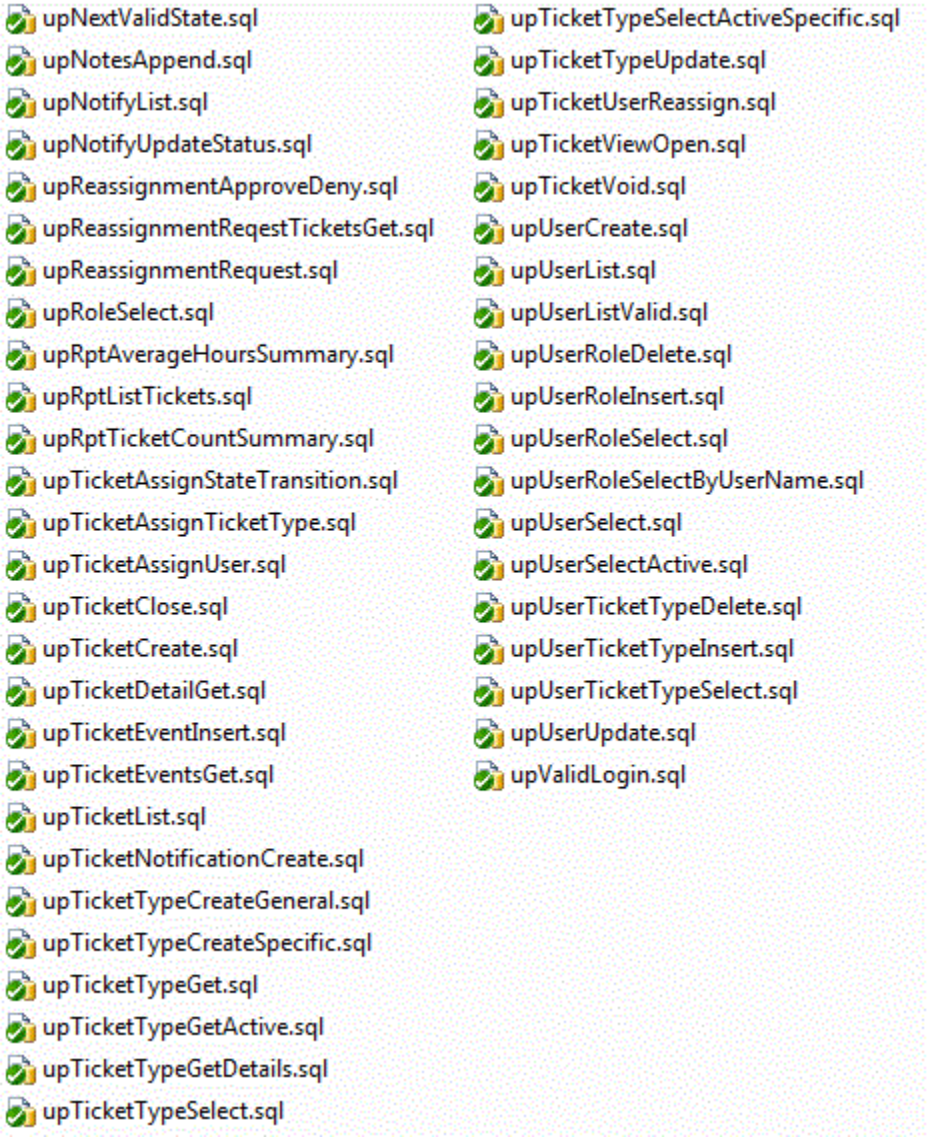




Appendix J.6: Rez-O-lution User Information Management Screen



Appendix K.1: Rez-O-lution Stored Procedures



Appendix K.2.1: Rez-O-lution Doxygen Table of Contents

Appendix K.2.2: Rez-O-lution Doxygen Table of Contents

Appendix K.2.3: Rez-O-lution Doxygen Table of Contents

Appendix K.2.4: Rez-O-lution Doxygen Table of Contents