CTV Design Specials

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Krista Miller

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

April, 2014

CTV Design Specials

By Krista Miller

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____	_____
Dr. Kenny Hunt Examination Committee Chairperson	Date
_____	_____
Dr. Thomas Gendreau Examination Committee Member	Date
_____	_____
Dr. Andrew Berns Examination Committee Member	Date

Abstract

Miller, Krista, R., "CTV Design Specials", Master of Software Engineering, May 2014, Hunt, Kenny.

The CTV Design Specials team at Trane in La Crosse, WI currently relies on a series of whiteboards and manually created spreadsheets to track the design process of special-order chillers. An electronic version of this system was created in order to make data entry and reporting more efficient and robust. During the design process, the system changed drastically and the decision was made to reengineer the entire system to improve speed, usability, flexibility, and maintenance efforts.

This manuscript describes the development of a more efficient, usable, and well-written system to replace the current one. It focuses on the design process, decisions that were made, challenges that arose, and comparisons between the new and old systems.

Acknowledgements

I would like to thank my project advisor, Dr. Kenny Hunt, for his feedback and guidance when it was needed. I would also like to thank all of the Computer Science professors I've had the pleasure of learning from during my time at UW-L. I have genuinely enjoyed learning from all of you over the last several years.

I would also like to thank my project sponsors, Craig Ausrud and Matt Haas for giving me the opportunity to work on this project. I would specifically like to thank Craig for taking the time to help me understand the requirements well enough to use the project for my capstone, even after we got the unfortunate news that the project would no longer be used. I would also like to thank Matt for allowing me to work on the project during work hours, again, even after we found out that the project wouldn't be used. It was a pleasure working with you during the time of my internship and I learned more than I could have ever imagined. It is all being put to good use in my new role with the company.

I would like to thank my parents for being the most amazing supporters I could have ever asked for. Your constant encouragement and unconditional love has gotten me to where I am today. I will never be able to express how thankful and blessed I am to have the two of you to call "Mom" and "Dad".

Finally, I would like to thank my husband, Jeremy, for encouraging me when I needed it. Thank your for your patience during my last semester, when I wasn't able to spend nearly as much time with you as I should have. Your support means more than you can imagine and I'm not convinced that I could have done it without you by my side.

Table of Contents

List of Figures

List of Tables

Glossary

Agile

A software development methodology that emphasizes close collaboration between the programming team and business experts and frequent delivery of new deployable software.

Approval

A portion of design that must be approved before an order can move to the next stage/gate.

Business Layer

Part of a program that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed.

C#

A simple, modern, object-oriented programming developed by Microsoft. It was originally released in 2001.

Class Diagram

A diagram that describes the structure of a system by showing the system's classes and relationships among objects.

CTV

Centrifugal Trane Vacuum

Design Special

A special option that is not offered through standard configuration. A sales associate can request these from a manufacturing location, which determines design costs and provides special pricing authorization.

Entity Relationship (ER) Diagram

A data modeling technique that gives a graphical representation of entities and their relationships to each other.

Graphical User Interface (GUI)

A type of user interface that allows users to interact with electronic devices with images rather than typing commands as text.

Gray-Box Testing

A combination of white and black box testing, which searches for defects due to improper structure or usage.

Issue

A problem that occurs in the design of a chiller.

Iterative

A software development process model in which a set of activities are performed again and again, converging toward some goal.

K001

An individual chiller for an order.

MDI

Manage Daily for Improvement

Microsoft Access

A database management system from Microsoft.

Microsoft Excel

A spreadsheet application that was developed by Microsoft. It was originally released in 1985.

Object-Oriented Programming

A programming model that represents concepts as objects that have data fields (attributes) and associated procedures (methods).

Order Number

An individual chiller order.

Special

A chiller feature that is not part of the standard chiller offering.

SQL Server

A relational database management system developed and marketed by Microsoft. Its primary query language is Transact-SQL, an implementation of the ANSI/ISO standard Structured Query Language (SQL).

Third Normal Form (3NF)

A type of database normalization that minimizes the duplication of data.

Unit Test

Testing a small piece of the application by isolating it from the remainder of the code to ensure correctness before integrating it with the rest of the system.

Visual Studio

An Integrated Development Environment from Microsoft.

Waterfall

A software development process model in which the life cycle is broken up into phases of distinct activities. These activities are performed to completion and are not expected to be performed again once the phase is over. A traditional process may be broken up into the following phases: Requirements, Analysis, Design, Code, Integration, and Testing.

Windows Forms

The graphical application programming interface included in the Microsoft .NET Framework, providing access to native Microsoft Windows interface elements.

1. Introduction

1.1 Background of CTV Design Specials – The Process

Trane, a subsidiary of Ingersoll Rand, is a global provider of heating, ventilating, and air conditioning systems. They provide high-performance and energy efficient systems to buildings of any size, from small homes to large industrial buildings. Chillers are a product offered by Trane and come in many different sizes and configurations. They are used in air conditioning systems to cool and dehumidify air.

A Centravac (CTV) is a high-efficiency Trane chiller manufactured in La Crosse, WI. There are many different options available, but the pre-configured units do not meet all customer requests. Special order chillers are chillers that are not included in Trane's standard offerings due to unique customer needs. There is a specific team, CTV Design Specials, which handles these types of orders for Centravac chillers. The CTV Design Specials team is responsible for a special order from the initial customer request to the completion of manufacturing and installation. They track the special orders through four stages: Pre-Order, Pre-Schedule Release, Pre-Order Shear, and Execute/Follow-Up. Not all orders make it through all four stages of development as the order may be cancelled or it may be found that the configuration is not possible.

The first stage, Pre-Order, is the quoting stage. A chiller enters into this phase when a customer makes a request that is not a standard offering. During this phase, the potential order is reviewed and a quote is sent to the customer. If the customer approves the quote, the chiller is moved to the Pre-Schedule Release stage.

Before production can be scheduled and started, the design work must be done, which is the bulk of the Pre-Schedule Release stage. Many issues can arise in this stage, for example waiting on a third party or discovering design problems. If the design can be completed, the chiller is moved to the Pre-Order Shear stage.

The Pre-Order Shear stage is for manufacturing preparation. The process documentation is developed and the shear date is set. The shear date is the day in which

the first part for the chiller is cut. In order to set a shear date the manufacturing team must be sure that all preparation is done so the factory has all of the necessary information to produce the special order chiller.

When an order is moved to the Execute/Follow-Up stage, the chiller has already gone through production. This stage is simply used to document lessons learned and what should be changed or kept the same if this type of special order is received again in the future.

1.2 Background of CTV Design Specials – The Software

The *CTV Design Specials* software has been in development since June 2012. It was intended to assist the CTV Design Specials team in tracking special orders. Its initial launch was planned for August 2012, but due to growing requirements, loss of a programming intern, and other projects the deadline was extended. Unfortunately, during that extension period, the project was cancelled by company executives who decided that Design Special teams throughout the company needed to standardize their process, instead of each team having their own system. However, the product owner requested that the project be finished in case they were allowed to use it in the future.

The original program has the functionality to move an order through all four stages, referred to as "Gates". An order can be moved back to a previous gate, rejected, or archived for later use. Gates 1 through 3 have a series of review fields consisting of a route, comment, status, and approval.

While the review fields are common between each of the first three gates, they each have unique fields and functionalities as well. For example, Gate 2 allows orders to be prioritized. Changes in priority are tracked by the system in case they need to be reviewed. Gate 2 also has a "review status", which can hold one of several different values. The functionality of the review fields in Gate 2 is based off of this review status. For example, if the review status is "N/A" the review fields cannot be changed since they are not necessary.

Gate 3 is unique in that the status of the review fields is based off of a shear date and cannot be controlled by users. Each field has a certain number of days before the shear date that it will turn yellow and a certain number of days until it will turn red, if it is not marked approved.

Gate 4 is different than the other three because it is for feedback only. It contains issue fields, which allow comments to be associated with certain steps of the design process.

The transition between gates involves various state transitions. An order in Gate 1 can either be archived, rejected, completed. If an order is archived or rejected it is moved into the archive and marked as being incomplete. If an order is rejected it is specifically marked. A user can restore an archived or rejected order at any time. When an order is completed in Gate 1, unless the user specifies differently, the order is moved directly to archive. This is because Gate 1 is the quoting stage and many orders do not make it into Gate 2. When a user wishes to move an order from Gate 1 to Gate 2, they can either do it from the archive or from Gate 2. From Gates 2 and 3 an order can be archived, rejected, or completed. When an order is completed in Gate 2 it is moved directly to Gate 3 and when an order number is completed in Gate 3 it is moved to Gate 4. An order in Gate 4 may not be removed from Gate 4. An order in any gate can be moved to a previous gate by an administrator. The base information of the order is deleted, but approvals, routes, and notes are saved for when the order returns to the gate it was moved from.

The current program also has several other maintenance features that are restricted to administrator access. These features include adding and editing users, adding and editing special or issue categories, and viewing the history of prioritizing in Gate 2.

1.3 Need for Reengineering

The original program was designed by the author and was not part of this capstone project. Throughout the process of the original program's design, additional requirements were added almost weekly. While this may not have been an issue for experienced developers, the author still had much to learn about software design and engineering,

resulting in a disappointing final product. Following the original program's completion, additional functionalities were needed, as well as a restructuring of code and database design. The author proposed a system overhaul to the product owner with the goals of increasing overall speed and efficiency, creating a positive user experience, and increasing future maintainability. The product owner approved the changes. Several additional functionalities were identified:

- An MDI Board must be added for Gate 3 shop documentation.
- "Issue" fields need to be added for review fields in the MDI Board.
- Each time any part of an order is late, users will be forced to choose a reason for the order being late.
- Users must be able to choose an existing issue or add a new one.
- Based off of the issue field, a Pareto chart must be generated to show where the most common issues lie.
- Users must have the ability to use the data from an existing order as a template for a new order.
- Users must be able to search for a K001, order number, or job name.
- The program must generate history reports for a K001 or order number.
- Administrators must have the ability to delete an order in any gate.
- The program must run on a large touchscreen computer as well as a users' individual workstation.
- Additional charts and reports.

The overall goal of the project was to create a system that could go above and beyond what the whiteboards could do by making users' jobs easier and reducing extra steps needed to create reports and transfer data from computer to whiteboard. The system needed to have good performance and be maintainable for other interns who would be making changes in the future.

2. Software Lifecycle Models

2.1 Models Considered

Three life cycle models were considered during the planning for this project: agile, iterative, and waterfall. Agile was not highly examined for the reengineering portion, but the author recognized that it would have been a good choice for the initial design and implementation, since requirements changed frequently. The author initially decided on the waterfall approach, but the project's advisor pointed to iterative, due to the concern of changing requirements. However, shortly into the design of the project, it was discovered that the program would no longer be used by the company. The requirements became static since the project manager was no longer involved. Due to the stable nature of the requirements the waterfall model was ultimately chosen.

2.2 Model Used - Waterfall

The waterfall model is a linear sequence of phases, in which one phase does not begin until the previous phase ends. If a change is required in a later stage, that change should be backtracked to previous phases, all the way to the initial requirements phase [2]. Due to the nature of completing the project in distinct phases, the customer is involved only in the beginning of the project, when requirements are being gathered, and at the end during user-acceptance testing [2].

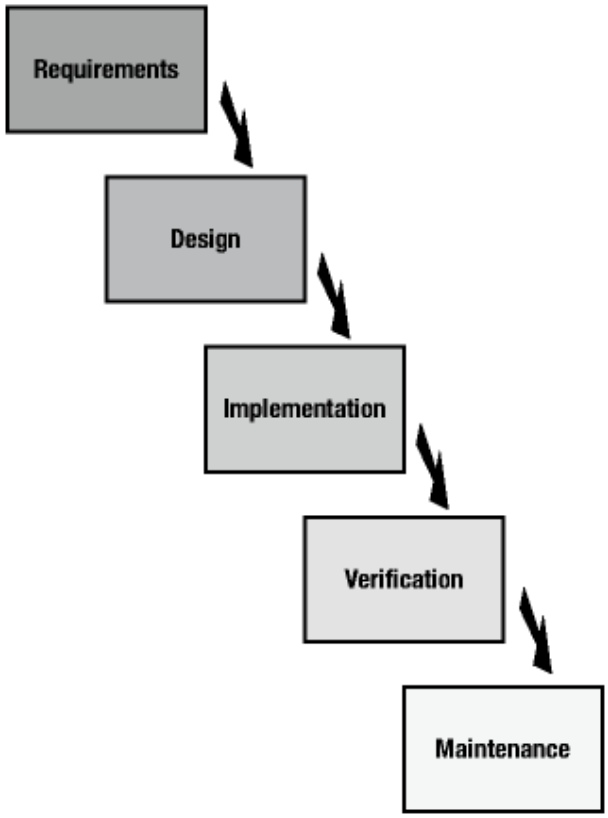


Figure 1: Waterfall Model [4]

There are several known advantages and disadvantages of using the waterfall model. Two advantages that were observed throughout the course of this project are careful and precise project planning and complete documentation [2]. These two aspects made the implementation stage easier and allowed for limited reworking. A disadvantage of the model that was encountered is that it can take an extended time frame to finish a project, which is generally not acceptable in the software industry today [2]. The reengineering of this project took approximately one year to complete. While this worked well for the author's needs, if the final product were actually being used the project manager would not have approved of waiting until the end to view the product. Another disadvantage, seen in the first attempt at the project, is that project planning is conducted in the early stages of the lifecycle, when only limited insight into the project is available [2]. It is easy to see that using the waterfall model in the first attempt at the project would not have been successful.

The waterfall model was decided upon because the author had a complete understanding of the requirements to be implemented. Since the project was no longer going to be used no additional requirements would be added, so completing the requirements and design stages without going back forth would not be a problem. In the requirements and design phases a true waterfall model was followed. Design was not started until the requirements were complete and implementation was not started until the design was complete. However, in the implementation phase a more iterative approach was introduced. The database and stored procedures were designed and then a brief testing phase was completed before code implementation was initiated. The object classes were then created, followed by another testing phase before any database or GUI interactions were integrated. Then, at the end, all three pieces were integrated and final verification took place.

3. Functional Requirements

Requirements were gathered for the original system over the course of a year. Initial requirements were given by the product owner, Craig, in an introductory meeting. There were weekly meetings after that, in which Craig would add additional requirements each time. He would also tune up any previous requirements that were given.

More user-oriented requirements were gathered by attending meetings of the Design Specials team to watch how they interacted with the current whiteboards. The author was able to see what additional requirements were needed and how the interface could be designed in a way that would be the most useful for the users.

Additional requirements that were to be implemented in the reengineered program were given in a large meeting with Craig and the owners of each of the four gates. Each gate owner was given several weeks to go through the original program and decide what else was needed.

The system's requirements were detailed in the requirements document, which complies with IEEE standards [1, 5]. The document explains the purpose and scope of the project, describes user characteristics, system constraints, and assumptions, and includes 82 functional requirements. The following list gives an overview of the functional requirements:

- A new K001 can be added to Gate 1 and a new order number can be added to Gate 2
- An active K001 or order number in any gate can be modified
- A K001 or order number in any gate can be deleted by an administrator
- A K001 or order number in Gates 1-3 can be rejected/restored or archived/reactivated
- An order number in any Gate can be sent back to a previous gate without losing any approvals or notes
- A completed K001 will be moved to the archive, unless otherwise specified
- A K001 can be associated with an active order number

- A completed order number in Gate 2 will be automatically moved to Gate 3
- A completed order number in Gate 3 will be automatically moved to Gate 4
- Approvals can be added to any active K001 or order number in Gates 1-3
- An approval of an active K001 or order number can be modified
- An approval that is not required can be deleted from an active K001 or order number
- An approval of an active K001 or order number can be routed to an active user
- A note can be added to an approval of an active K001 or order number
- A note can be edited/deleted by the original creator
- Specials can be added to any active order number in the MDI Board
- A special of an active order number can be modified
- Issues can be added to an active order number in Gate 4 or an issue of an active order number in the MDI board
- An issue can be modified or deleted
- Any number of notes can be added to an issue of an active order number
- A user must be able to search for a K001 or order number by K001 ID, order number ID, description, or job name
- Order numbers in Gate 2 can be prioritized by any user
- Prioritizations are saved to the database and have a reporting function available
- Users can view which active items have been routed to them
- Reports of various data must be available
- Users can log in and out of the system and must be logged in to make changes
- There must be two user roles: regular and administrator. Administrators must provide a password to access functions requiring advanced permissions.
- Administrators can add/edit/deactivate users
- Administrators can add/edit/deactivate special categories and issue categories
- Administrators can define which approval types are required for all gates

- A startup screen must give a high-level overview of what is contained in each gate

Figure 2 below shows two functional requirements taken from the requirements document.

Index:	3.1.1
Name:	GetGate1
Purpose:	To retrieve a list of all active K001s from Gate 1.
Input parameters:	None
Action:	Open a database connection. Create a list of active K001s and their associated approvals. Close the database connection. Return the list of K001s.
Output parameters:	k001s
Exceptions:	The database connection failed.
Remarks:	This will be used during initialization or during a refresh to create the list of active K001s, which users can select.
Cross-references:	None

Index:	3.2.1
Name:	AddK001
Purpose:	To add a new K001 to Gate 1.
Input parameters:	newK001
Action:	Validate the data in newK001. Open a database connection. Store the data of newK001 in the database. Close the database connection.
Output parameters:	None
Exceptions:	A K001 with the same id as newK001 already exists in the database. A required attribute in newK001 is missing. An attribute in newK001 is in an invalid format. The database connection failed.
Remarks:	newK001 should contain the attributes of a K001 as listed in Appendix A of this document.
Cross-references:	None

Figure 2: Functional Requirements

Each requirement has a unique index, which can be traced back to the corresponding section of the requirements document. The name and purpose describe how the requirement will be used and input parameters are listed. A high-level overview of the steps that need to be taken to complete the requirement are specified and output parameters are shown. Exceptions that may occur during the function's execution are

described and any additional information that is helpful to understanding the requirement is included. Any relationships to other functional requirements are also specified.

The system also has several non-functional requirements, the most important being usability. Since a goal of the system is to be a more advanced version of the whiteboards, the team must be able to complete their tasks quickly and efficiently through quick database accesses and frequently updated data. The system also needs to be usable from a touchscreen, as well as users' individual workstations, meaning users must be able to access the system concurrently, without duplicating data or encountering errors. The system must be secure in order to preserve data integrity and confidential company information through passwords and restricted folder access. Lastly, the system needs to be maintainable, since its upkeep will be the responsibility of interns who only remain in the position for one or two years at a time.

4. Design

Since the bulk of the requirements were gathered in the initial phase of the project, the design phase came relatively quickly. The design is based on an object-oriented approach and included the use of a class diagram, component diagram, and entity relationship diagram. The following section explains the tools and techniques used throughout the design phase.

4.1 Database

Since *CTV Design Specials* is data-oriented, the database design naturally came first. The database was initially going to be designed in Microsoft Access 2010. This was due to the author's position in the company at the time as a Business Tools Software Intern, which is a position responsible for developing and maintaining small applications for internal use. Using anything other than Microsoft Access would have caused the company to classify the program as a software project, rather than a tool, and give it to a software development team overseas. However, once it was discovered that the program was no longer needed, it was decided that SQL Server 2012 would be used, rather than Microsoft Access. This decision was made for several reasons. First, SQL Server is more widely used in the software industry than Microsoft Access, so it would be good experience. Second, SQL Server is a more robust choice for overall database management. It is said to handle simultaneous access better than Microsoft Access and has better database administration tools.

One goal, when designing the entity relationship diagram, was ending with a database in third normal form. The database for the first version of the project was not in third normal form and, therefore, contained redundant data, larger tables, and slower queries. The new database, which is in third normal form except for three tables used to populate dropdown lists, has increased performance and is easier to use overall. Figure 3 shows the resulting entity relationship diagram.

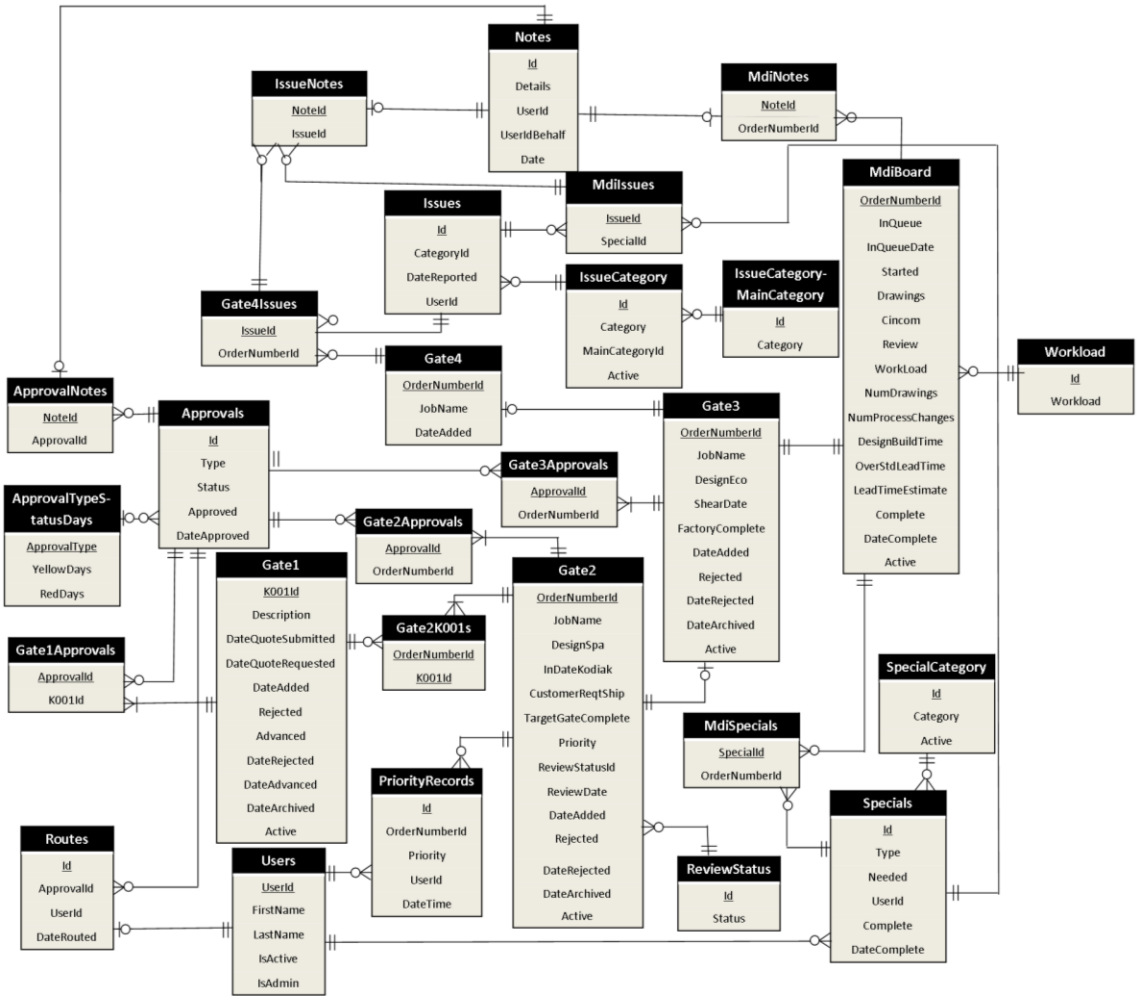


Figure 3: Entity Relationship Diagram

As shown in the entity relationship diagram, there are 28 tables, some responsible for holding data and others responsible for reducing redundancy.

In case the project is needed in the future, all database queries are written as stored procedures within SQL Server. That way, if the program did need to use Microsoft Access, there would not be a large code change required. Using stored procedures also made for cleaner, more readable code. There are 115 stored procedures, each written to retrieve only the necessary data in the quickest way possible, by delaying joins between tables.

4.2 Class Structure

Since *CTV Design Specials* is a data-oriented application, the class structure was inferred from the entity relationship diagram (Figure 3). The class diagram can be seen in Figure 4 below (detailed classes are shown in Appendix B) and detailed definitions of the 18 classes can be seen in the design document [3].

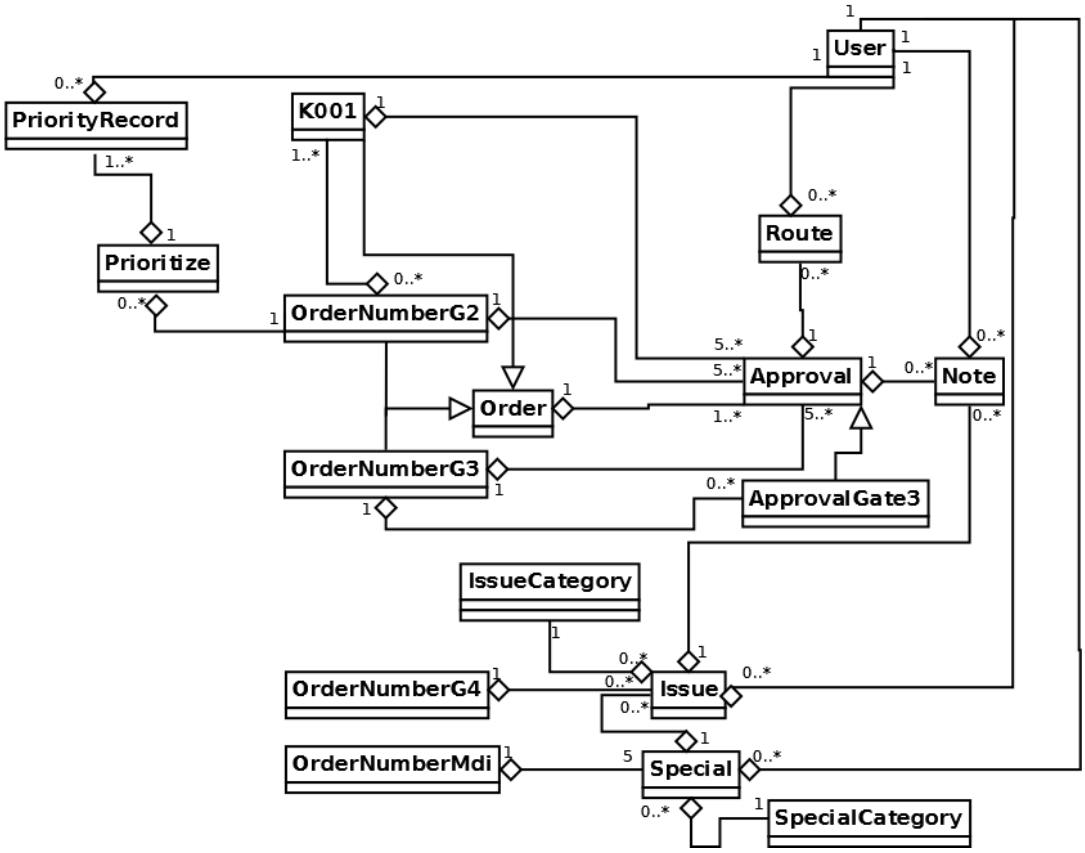


Figure 4: Class Diagram

As the diagram shows, the order number and K001 classes are the main parts of the system, which is fitting because they are also the central portions of the business logic. K001, OrderNumberG2, and OrderNumberG3 all inherit from Order. The various K001 and order number classes also contain approvals, issues, specials, and notes. The Order class can be seen in Figure 5, with a portion of the class narrative in Figure 6.

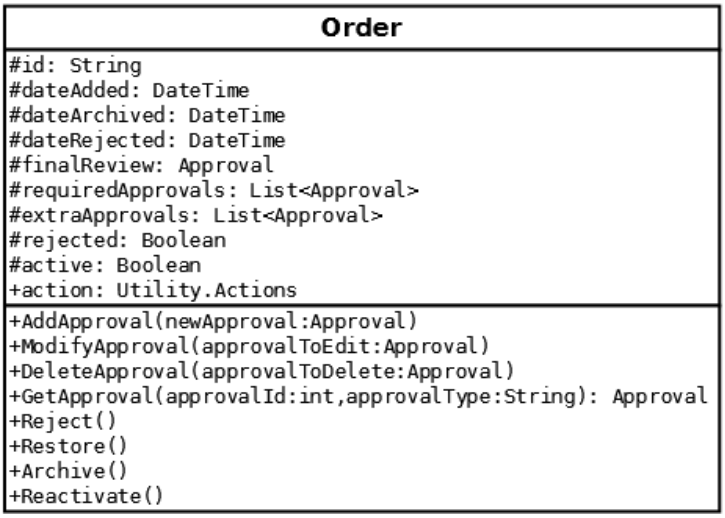


Figure 5: Order Class

Since K001s and order numbers in Gates 2 and 3 inherit from the Order class, most of their details can be seen in Figure 5. The design document goes into further details with class definitions for each class, an example of which can be seen in Figure 6.

Class name: Order

Attributes:

protected	String	id
protected	DateTime	dateAdded
protected	DateTime	dateArchived
protected	DateTime	dateRejected
protected	Approval	finalReview
protected	List<Approval>	requiredApprovals
protected	List<Approval>	extraApprovals
protected	Boolean	rejected
protected	Boolean	active
protected	Utility.Actions	action

*/** There are no public setX() methods for any of the above attributes except for action, the rest are protected. **/*

Methods:

Name:	AddApproval
Synopsis:	AddApproval(newApproval)
Purpose:	To add a new approval to a K001 or order number.
Visibility:	public
Input parameters:	Approval newApproval
Output parameter:	None
Local variables:	None
Exceptions:	ApprovalExistsException – display an error message and terminate the method. ArgumentException - display an error message and terminate the method.
Remarks:	Two approvals for an order number are considered to be equal if they have the same type.

Figure 6: Portion of Order Class Narrative

Between the class diagram and class narratives, the developer is given the information they need to implement the class. For the Order class, each order has a unique identifier. Since order numbers are moved from Gate 2 to Gate 3, the identifiers are repeated across gates. Users must be able to see the history of a K001 or order number, so the Order class also contains attributes to track if and when they have been added, archived, or rejected. Gates 1 through 3 all have a final review field and a list of required approvals. The required approvals are different across gates. A K001 or order number can also have any number of additional, or extra, approvals. The "action" attribute is not part of the business logic, but is used to indicate whether the order is being added, modified, or deleted so the proper database updates can be made.

A few other helper-type classes exist in the system, but are not shown in the diagram, such as a database interfaces, a utility class, and report utility classes. The point of these classes is to keep other parts of the system from having to directly interact with the database or reports.

4.3 Reports

Reporting is done in Microsoft Excel. This decision was made based on the tools offered by the company and the users' preference to have data in Excel where they can manipulate it as needed. Most of the reporting is available to all users and gives information on a K001 or order number. For example, users can choose a K001 and see which order numbers are tied to that K001, which gates they have gone through and when. There is one report that is available to administrators only. It allows them to see how users have prioritized order numbers in Gate 2. This is to protect against a single user repeatedly prioritizing their orders above all others.

4.4 System Security

Security is not generally viewed as a concern with these types of small programs at Trane. They are usually stored on a server, where anyone with access to the server can access them. This is because gaining access to a folder can take several days and most users wish to avoid this. However, a few extra security measures were decided upon for this piece of software. First, the database and all needed files were placed in a folder where only members of the Design Specials and related teams have access. Since that did not restrict access enough, a password was added to the database as well as usernames to the program. The program uses the environment username to see if that user has access to the system. If they do, they are automatically logged in. Otherwise, they can view data, but are not allowed to make any changes. There is also a log in option, which was to be used on the touchscreen, since it was not going to have any specific user logged in. Passwords were discussed, since anyone could use the login option to type someone else's username, but the project manager did not think it was

necessary and did not want users to have to remember a password. As the design addressed, the program has an administrative role, which requires a password. An administrator can do anything that a regular user can do, but they also have maintenance options available, which include adding, editing, or deleting users or different pieces of data as well as viewing advanced reports. This password is the same for every user, but is only given to the administrators.

5. Implementation

Due to the level of detail given in the design phase, the implementation phase proceeded quickly.

Since the user interface was a portion that would be widely reused from the previous version, that design came first. While nothing was directly used, the overall look and feel was kept the same. Changes were made based on user feedback from the original software. Figure 7 below shows a screen used to allow users create a report as seen in the old software versus the new software. Additional comparisons between the new and old GUI can be seen in Appendix A.

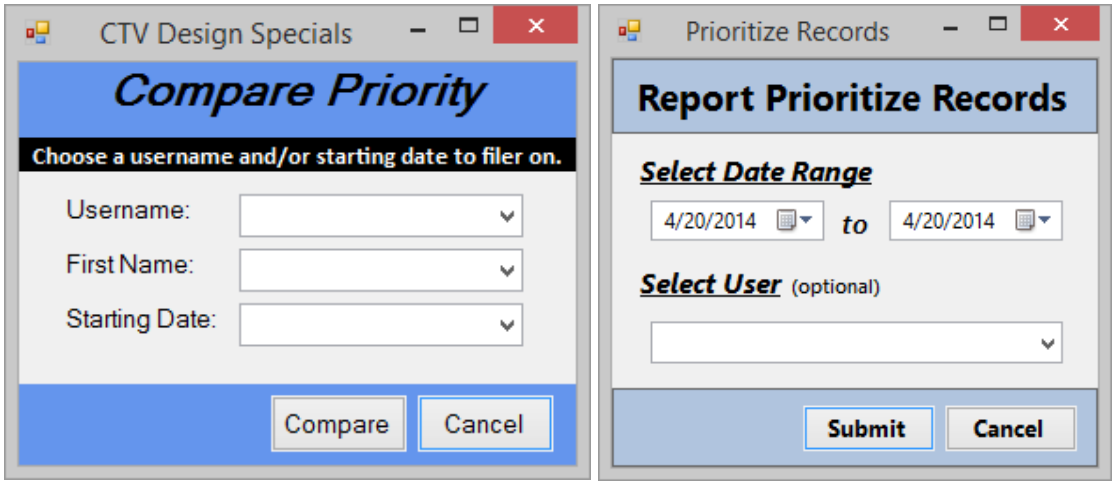


Figure 7: Report Prioritize Records Screen – Before and After

The GUI was designed using Windows Forms in Visual Studio 2013, with C# as the code running and connecting the GUI, database, and business layers. Visual Studio and C# were chosen for an opportunity to learn more about their features and to make maintenance easier for future interns.

The database was created based on the entity relationship diagram and the classes were written based on the class diagram and definitions contained in the design document [3]. The largest challenge in the implementation phase was creating the charts and reports as the programmer was unfamiliar with the technology.

6. Validation and Testing

Since the program is no longer being used and the programmer had since accepted a new position in the company, the testing was left up to the programmer, which is not ideal.

The testing did not follow the typical waterfall approach, as it was done as each component was added to the system. Since the database and stored procedures were completed first, each stored procedure was tested separately to ensure the expected information was being retrieved and modified. Methods were written in separate database interfacing classes to access each stored procedure.

Next came the GUI, without any data connections. After the GUI was developed and wired up, the programmer tested to ensure that each button caused the correct action and that no data could be saved with missing or invalid entries.

After GUI development came the object classes. The classes were tested as they were designed in the design phase. After they were all fully written, unit tests were created to check the error handling and functionality of each method in each class.

Next it was time to integrate the separate pieces. As different portions were added, they were lightly tested to uncover any obvious mistakes. After everything was put together, gray-box testing was performed for each functional requirement. Gray-box testing is a combination of black-box and white-box testing, in which the requirements are used to create test results, but the internal structure of the code is also known. Any issues that were uncovered were fixed and tested again. After all tests passed, they were ran once more to insure nothing was broken as issues were being fixed. Table 1 shows an example of a test case, in which changes to approval statuses in Gate 3 were checked against changes in the shear date.

Test Id	Purpose	Requirements	Steps	Expected Result
G3001	To ensure the status dates for approvals change as they should when the shear date is changed	1. The steps and expected results assume the following about the required approval types: Process Design: 13 days to yellow, 11 to red Material: 12 days to yellow, 10 to red Shop Documentation: 7 Days to yellow, 5 to red Programs: 3 days to yellow, 1 to red	Open an active Gate 3 order number	Order number data is loaded to form
			Unapprove all required approvals	Required approvals should not be marked as approved
			Change the shear date to the current date	All required approvals should have a red status
			Change the shear date to one day past the current date	No change in approval status
			Change the shear date to two days past the current date	Programs status should change to yellow, no other changes
			Change the shear date to four days past the current date	Programs status should change to white, no other changes
			Change the shear date to six days past the current date	Shop Documentation status should change to yellow, no other changes
			Change the shear date to 8 days past the current date	Shop Documentation status should change to white, no other changes
			Change the shear date to 11 days past the current date	Material status should change to yellow, no other changes
			Change the shear date to 12 days past the current date	Process Design status should change to yellow, no other changes
			Change the shear date to 13 days past the current date	Material status should change to white, no other changes
			Change the shear date to 14 days past the current date	Process Design status should change to white no other changes

Table 1: Test Case for Gate 3 Status Changes for Change in Shear Date

The table above does not show the last two columns, which are “Actual Results” and “Pass/Fail”.

If the program were to be used, usability and user acceptance testing would have also been performed by the product owner and owners of the four gates. For about a month-long period, after each of their meetings, they would have used the program to enter the same data that they entered on the current whiteboards to make sure everything was up to their standards and would be easy for everybody to use. However, since that was not an option, the programmer did their best to navigate the system with the mindset of a user and make any changes that would benefit usability.

7. Result of Reengineering

The reengineering of the system was a success. The system has a better overall structure, with increased usability, performance, and ease of maintenance. While none of the code was reused, the overall system structure, database structure, and GUI design were reused and improved.

The old version of the software contained seven classes, only four of which were used to store data objects, which meant that a lot of the code that should have been in a business layer was instead integrated with GUI, making code hard to read and make changes difficult to make. The reengineered software contains 12 database interface classes, 1 utility class, 4 various reporting utility classes, and 17 object model classes (Appendix B), none of which access any of the GUI code. These classes are more readable and flexible, making the system easier to understand.

The reengineering of the database and its access methods also created drastic improvement. The old database contained 12 poorly structured tables. Instead of having separate tables for approvals, routes, and order numbers, these were all attributes in larger tables, meaning that an order number could have a finite number of approvals, all notes were stored in one field, and only one route was allowed per approval. The new database contains 29 tables with no duplicated data. These tables allow an order number or K001 to have any number of approvals, an approval to have any number of notes and routes, and so on. The database queries were moved from the code to the database, as stored procedures, and were optimized to increase speed and only retrieve the necessary data. These changes gave the system a large performance boost.

The GUI was heavily based of off the old one. The largest changes were made to the content on each screen. Instead of fitting everything needed for a K001 or order number on one screen, they are split into separate screens to help the user see what exactly is needed for the task they are trying to accomplish. Comparisons between the new and old GUI can be seen in Appendix A.

The reengineering was an overall success. The good parts of the old system were reused and improved upon and the poorly designed portions were redone for improved performance, usability, and code readability.

8. Conclusion

The creation of this system was a great learning experience. The author was able to experience the challenges and benefits of closely following a software lifecycle model, while also feeling the disappointment of a project being discontinued. While having the patience to complete all documentation before starting implementation, as dictated by the waterfall model, was a challenge, it was a great benefit in the end, making the implementation portion go smoothly.

The original goals of the project would have been met with this system, had it been used. The system would allow users to do everything they were previously able to do, and more. They would have all of their data in once place and could easily view current data as well as looking at past data. All of their reports would have been in once place and they could have accessed everything from the comfort of their own desk.

8.1 Challenges

Multiple challenges were overcome in the process of this system design. The largest of those challenges came when the project owner was told that he was no longer to use the system. Since the author had not gathered enough understanding of some of the new requirements, the project manager had to be kept involved long enough to gather sufficient information.

A smaller challenge was changing requirements, which still occurred, even after the project owner was out of the picture. This was the fault of the author, for not looking back in old notes prior to starting the project. This made it necessary to backtrack through all stages of the waterfall model to accommodate for the rediscovered requirements.

A challenge also arose when it was time to start allowing order numbers and K001s to be added and edited. Several different screens can be opened, while editing either an order number or a K001. For example, if the user wants to add a new approval, they open a form to enter the information of a new approval, from which they can also open additional forms to add notes or routes. At first the programmer made temporary records

for approvals, notes, and routes when this was done so the data could be saved and passed back or deleted, if necessary. However, that caused a very slight pause when closing each form while the information was being saved to the database. When looking for a solution, the programmer found that data could be passed between forms through public methods, so if a change was made that data could be retrieved and marked as being added, modified, or deleted so that all of the data could be saved at once, at the end of the edit.

A database-related challenge that arose was related to having multiple users accessing the system at the same time. One major goal of this system was to allow users to be able to make changes from their desks instead of having to walk, or in some cases travel from different buildings, to get to the main board. This brought the possibility of users not seeing up-to-date data or creating duplicate information in the system. This issue was solved by making sure data was refreshed after each major action (saving or closing a form), which guaranteed that the user was seeing the most up-to-date information. To protect against duplicate information being added, database constraints were used to avoid having duplicate order numbers, the same approval for an order number, etc. A message is shown saying that the information had already been added and they will not be allowed to add it again.

A positive challenge was learning to interface with Excel through C# to create charts and reports. The same can be said for the charts shown in the interface of the program. The programmer did not have any experience with charts or reports, so there was a learning curve involved. The skills and techniques discovered will be useful in future projects.

8.2 Future Work

It is not likely that any future work will take place, since this program will not likely be needed in the future. However, the idea of it may be used in the author's current position at Trane because a Manage Daily for Improvement (MDI) Board is being

considered. The *CTV Design Specials* system could be used as a framework and the MDI Board portion could be abstracted out and used as a generic MDI Board template.

9. Bibliography

[1] IEEE Guide to Software Requirement Specifications, New York, IEEE 1998. ANSI/IEEE Std. 830-1998.

[2] L. A. Maciaszek and B. L. Liong, "Lifecycle Models" in Practical Software Engineering: A Case Study Approach. Harlow, England: Pearson Education Limited, 2005, ch 1, sec. 1.3.1, pp. 22-24.

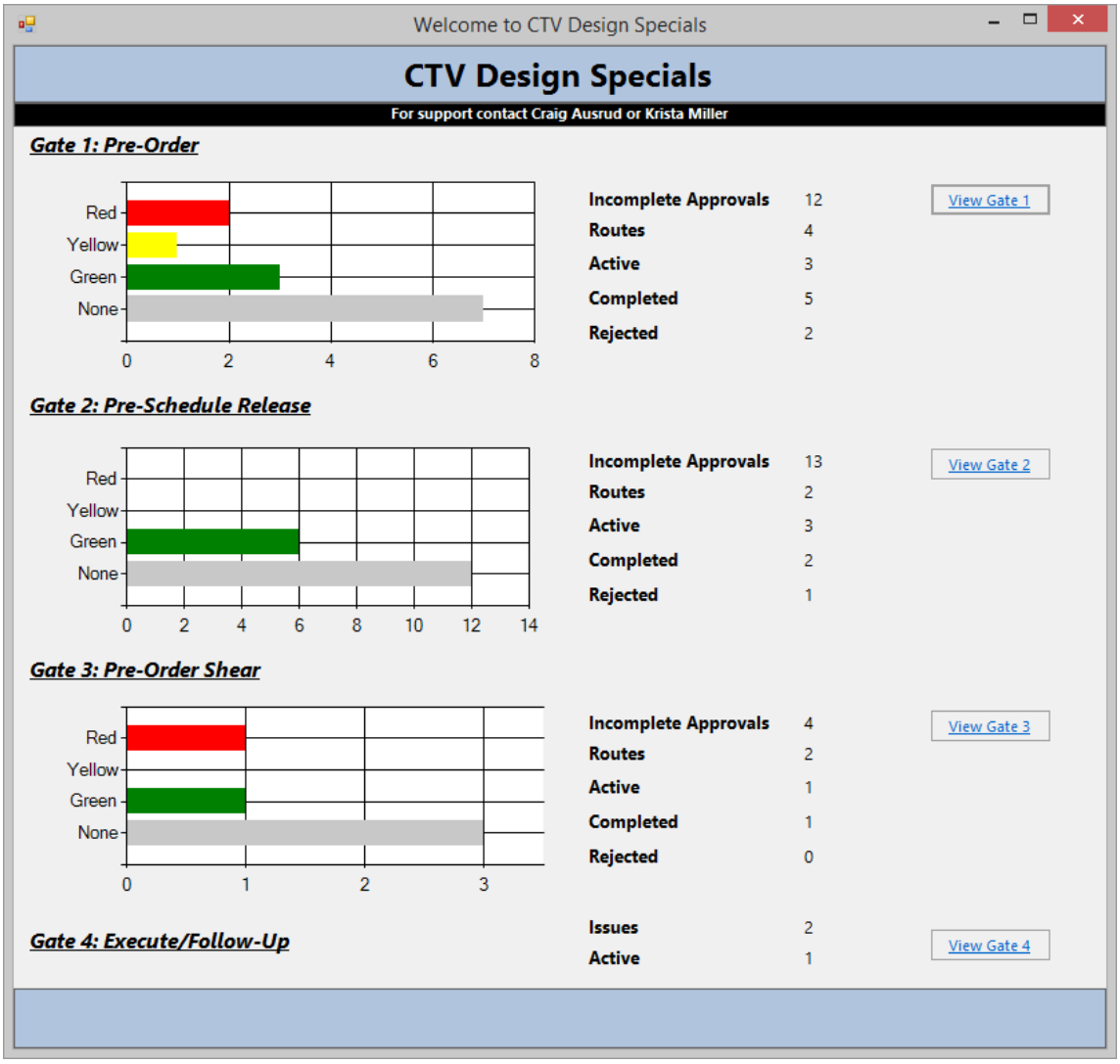
[3] Miller, Krista. "Design Document for CTV Design Specials", Version 1.1, April 2014.

[4] J. Rossberg and M. Olausson, "Development Processes and Frameworks," in Pro Application Lifecycle Management with Visual Studio 2012, 2nd ed. Dordrecht: Springer, 2012, ch. 3, pp. 38.

[5] Schultz, Krista. "Software Requirements Document for CTV Design Specials", July 2013.

Appendix A: GUI Before and After

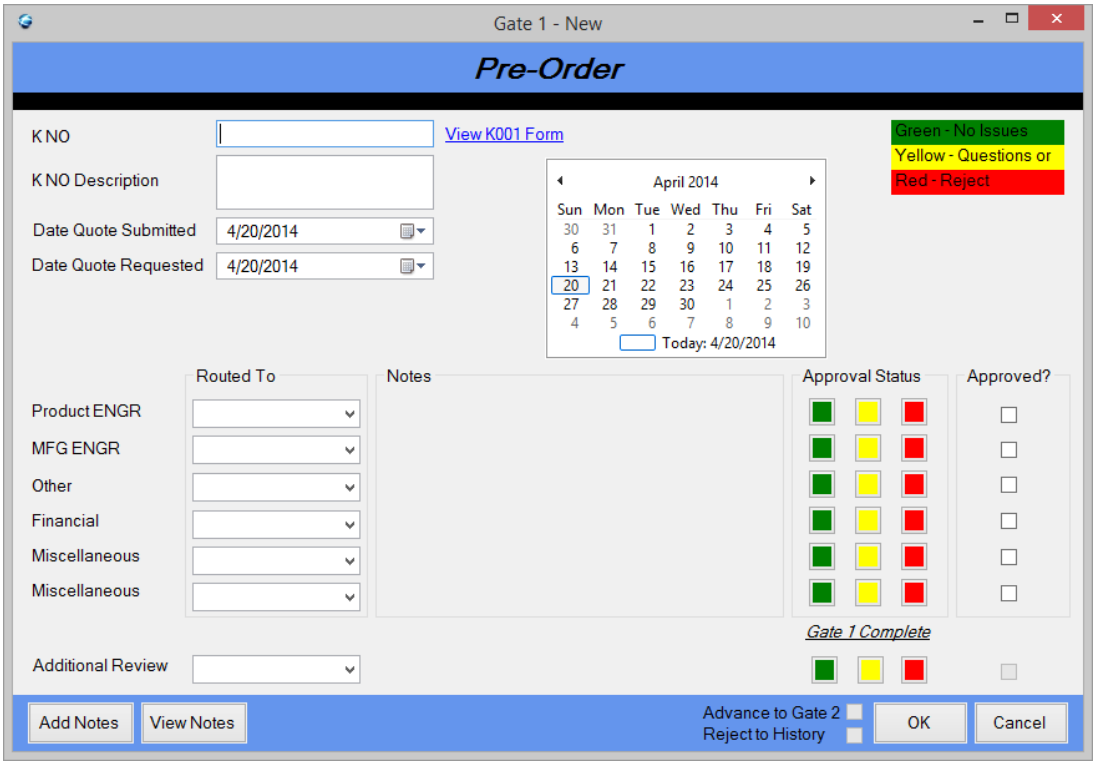
Entry Screen (Before)



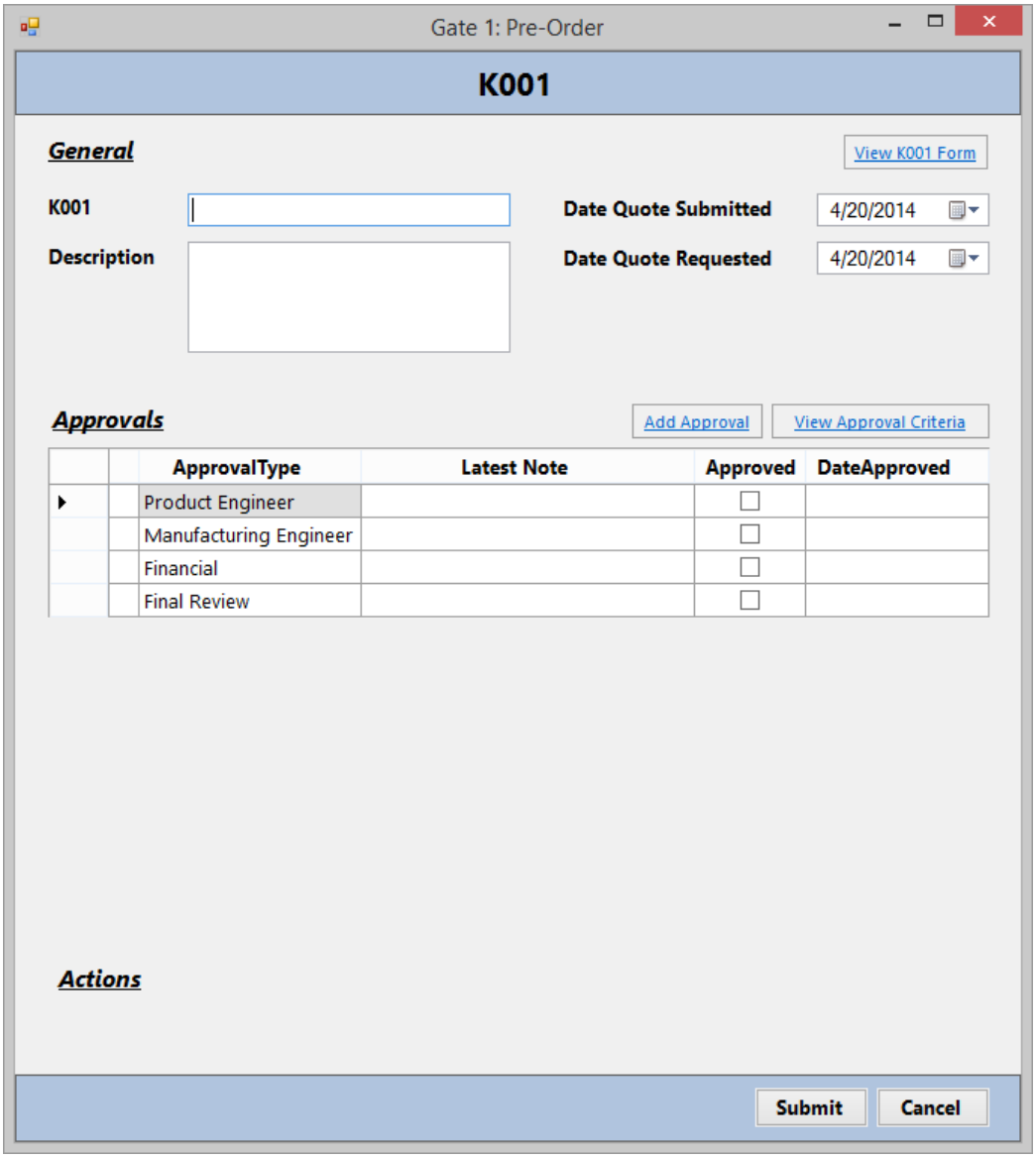
Entry Screen (After)

Main Screen (Before)

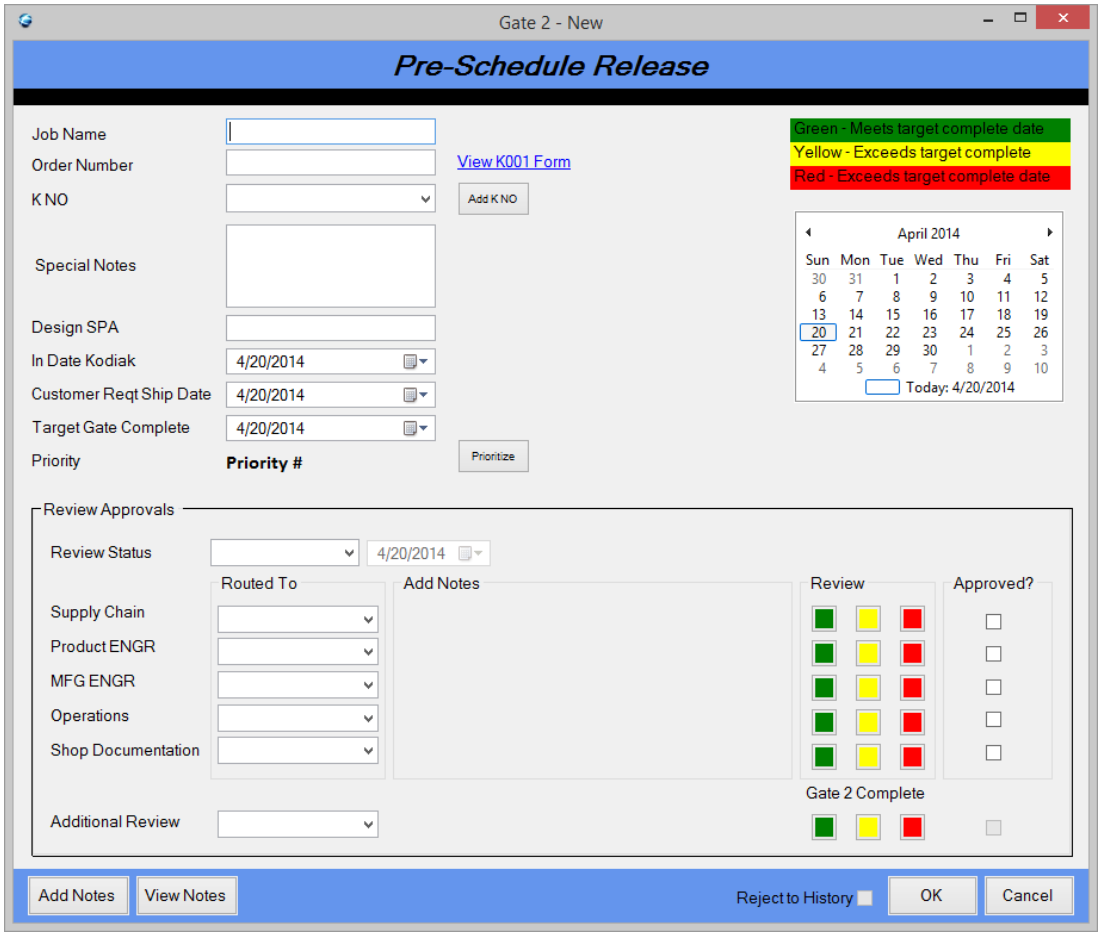
Main Screen (After)



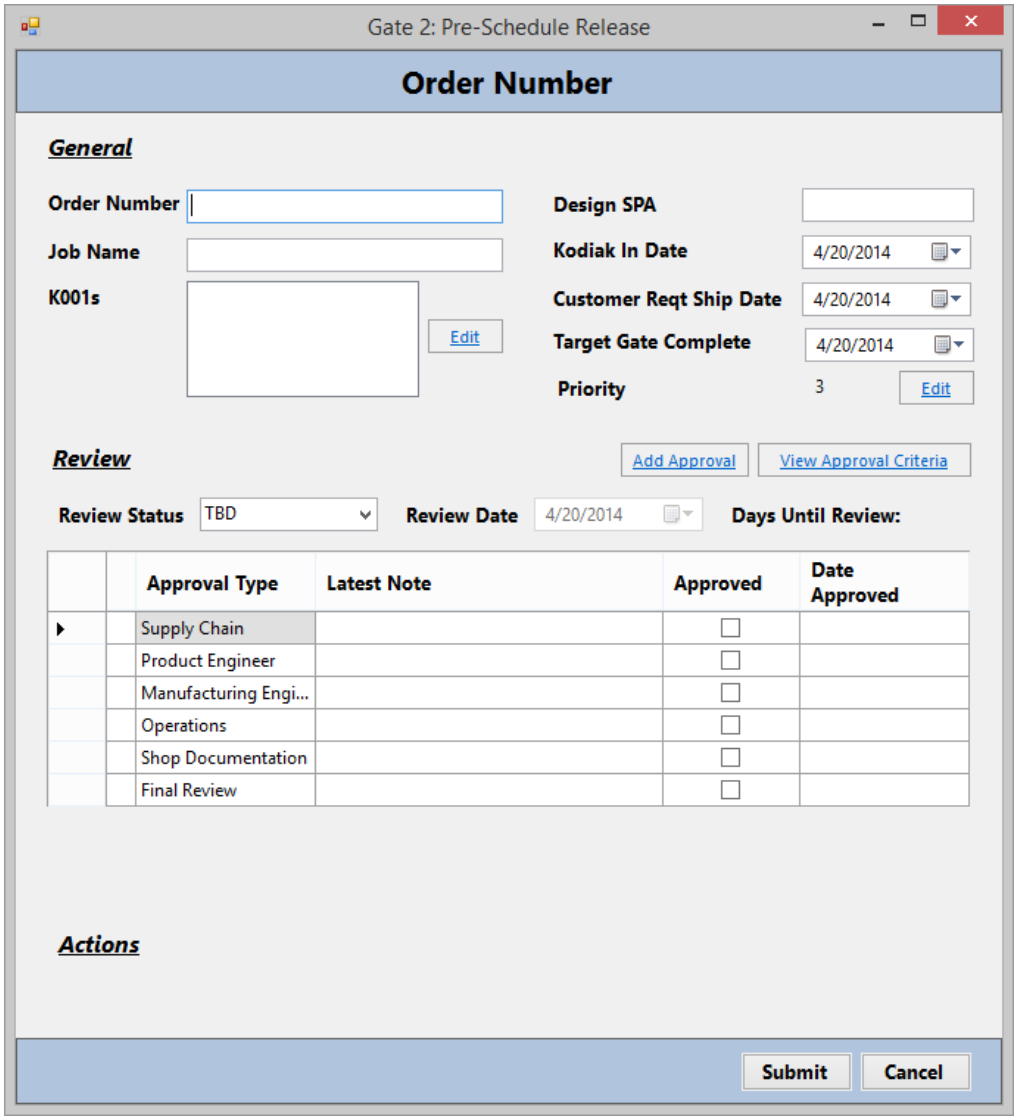
Gate 1 Add/Edit (Before)



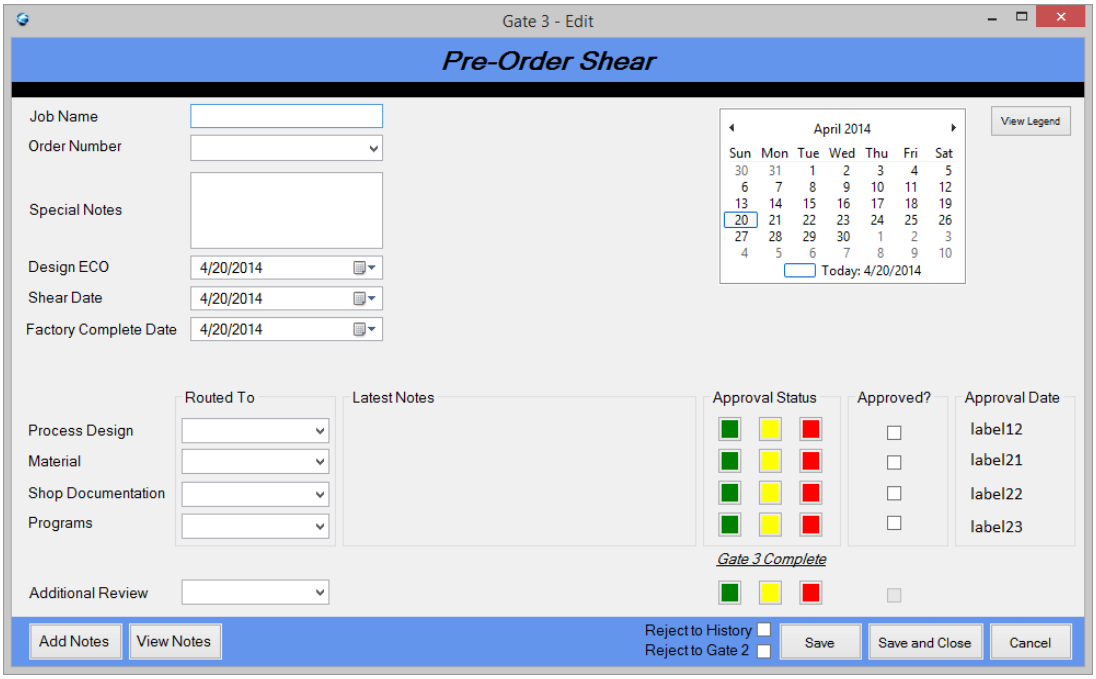
Gate 1 Add/Edit (After)



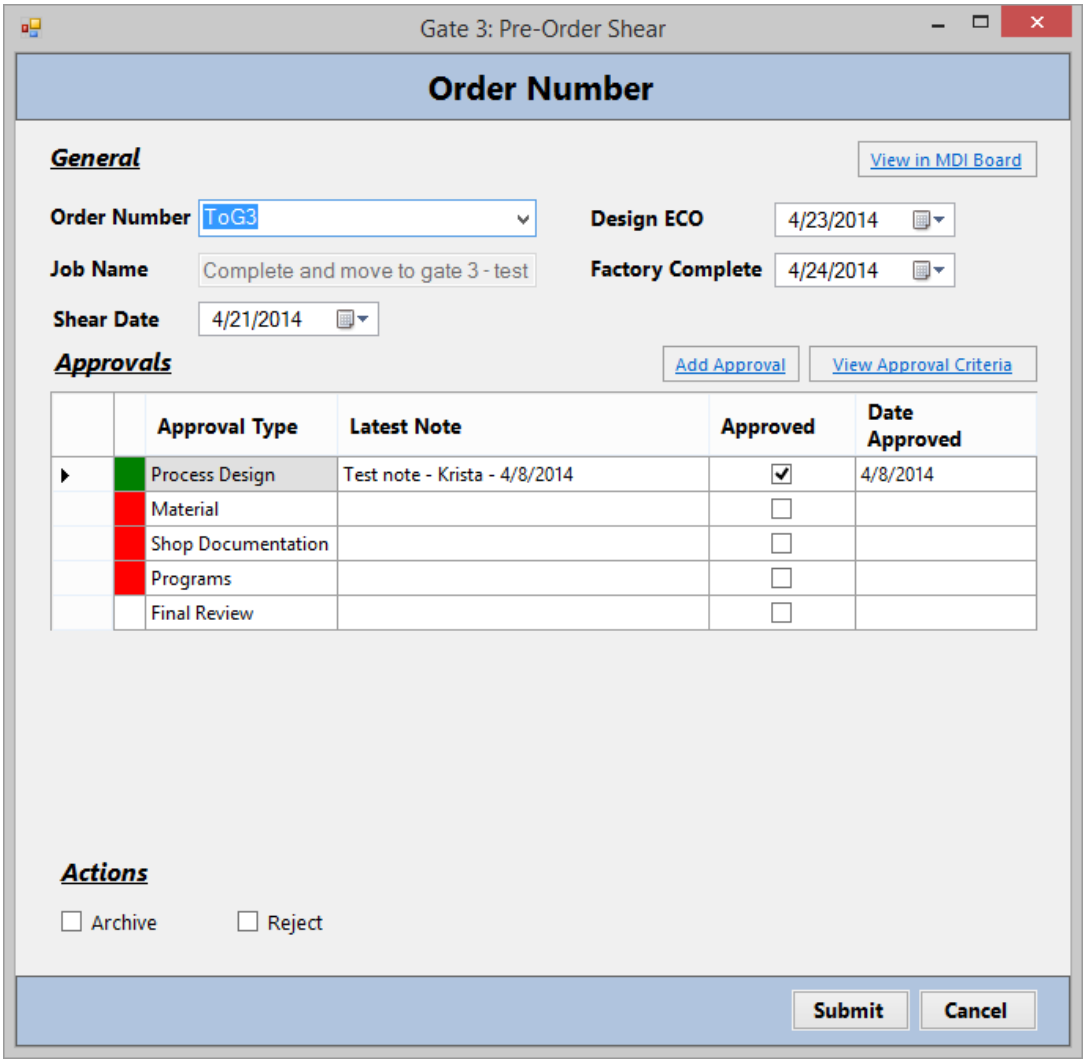
Gate 2 Add/Edit (Before)



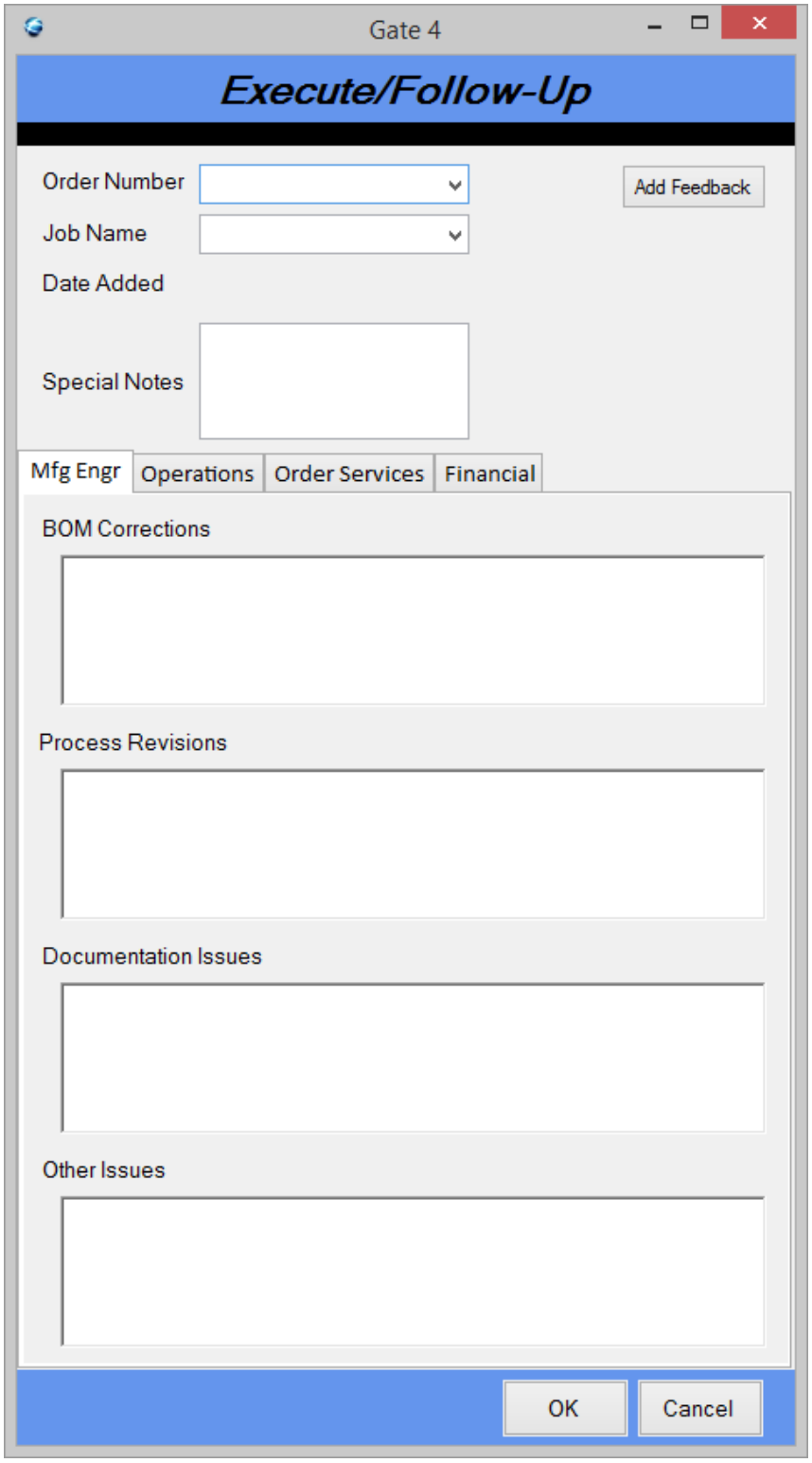
Gate 2 Add/Edit (After)



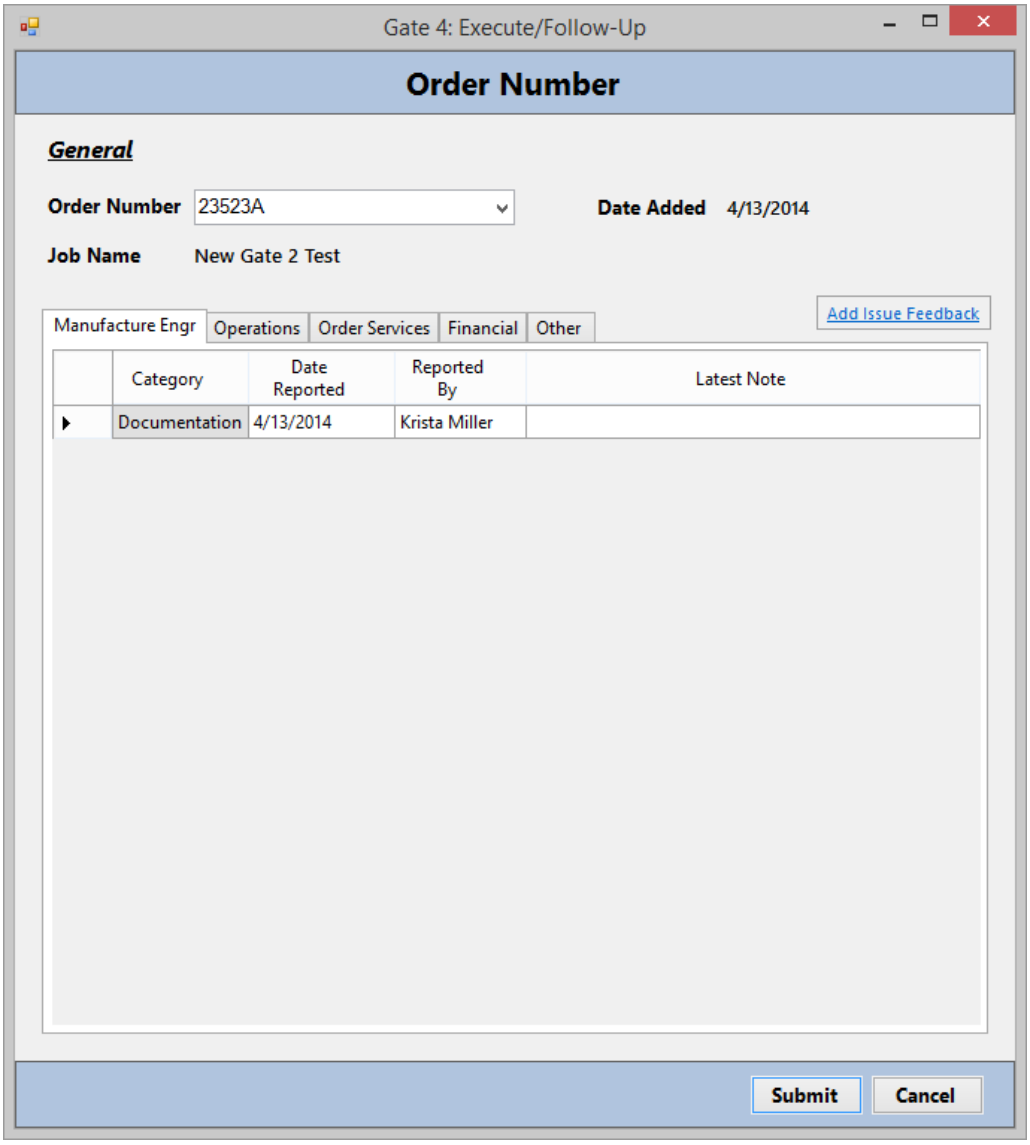
Gate 3 Edit (Before)



Gate 3 Edit (After)

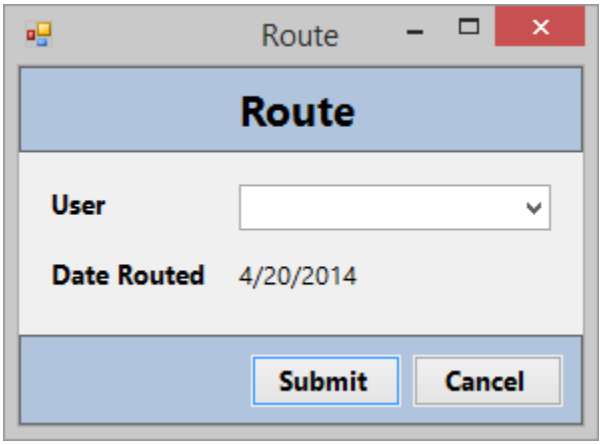


Gate 4 Edit (Before)

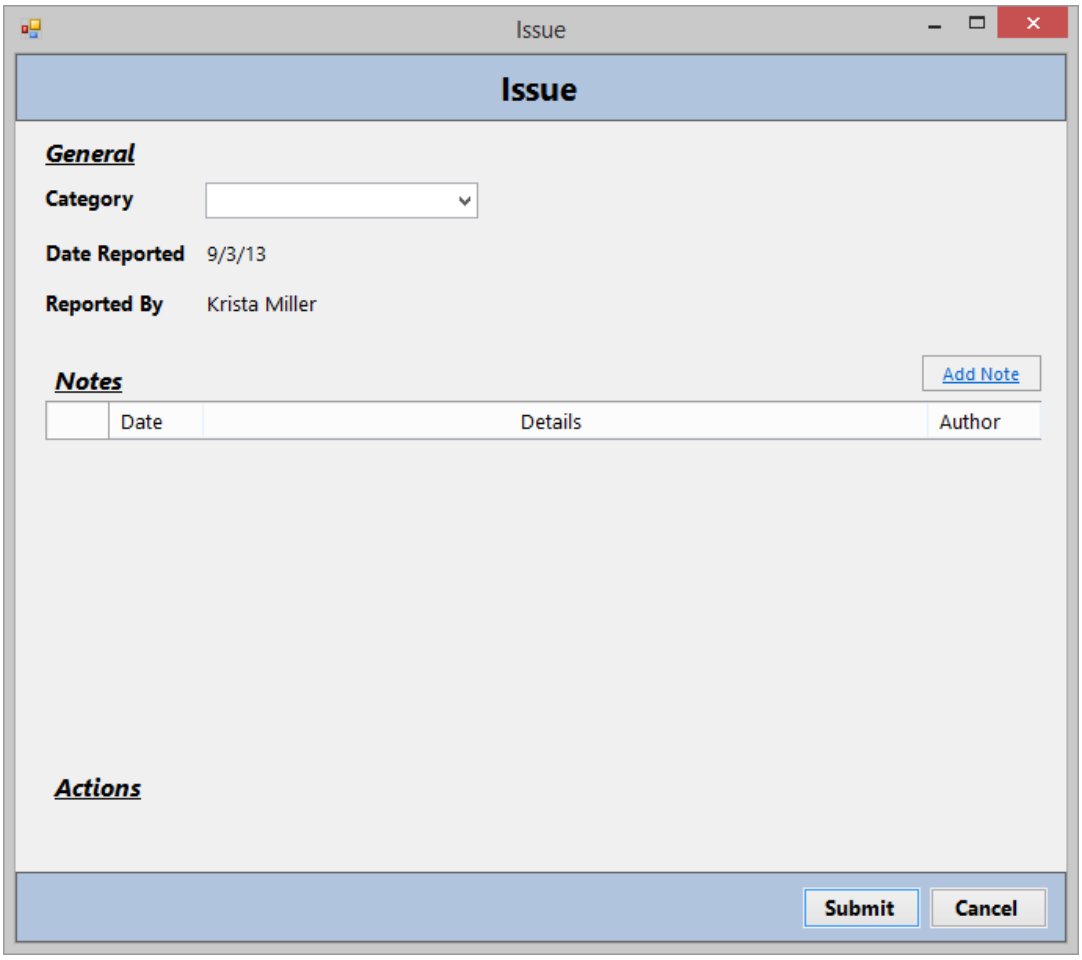


Gate 4 Edit (After)

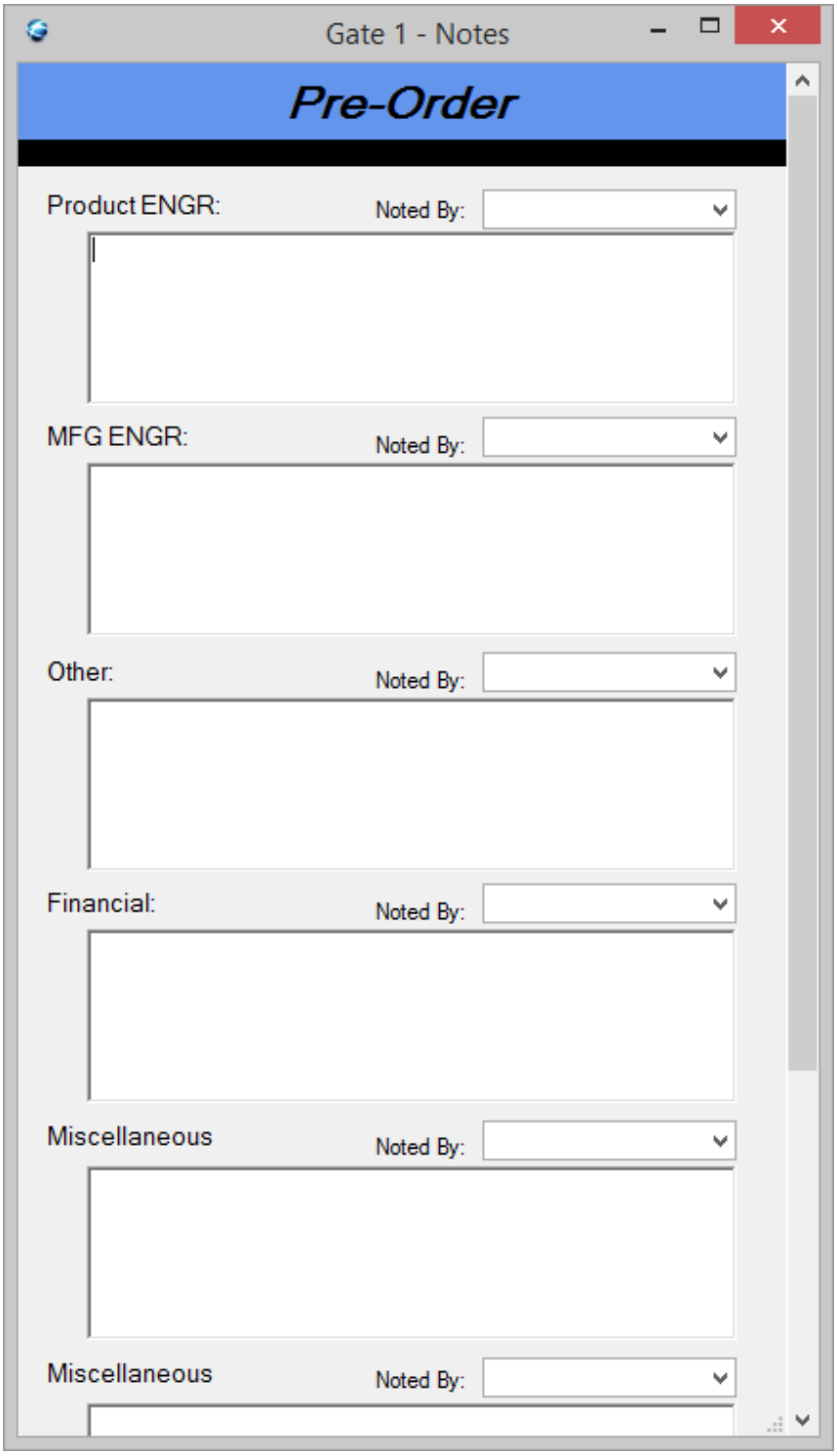
Approval (After – Previously on same screen as Gate 1-3 Add/Edit)



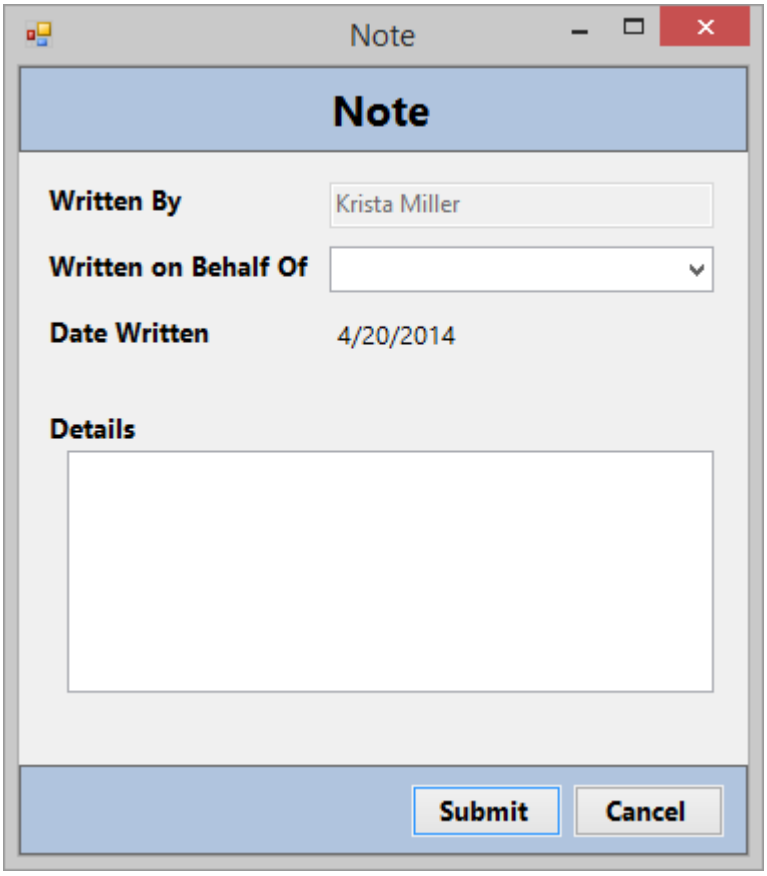
Route (After – Previously on same screen as Gate 1-3 Add/Edit)

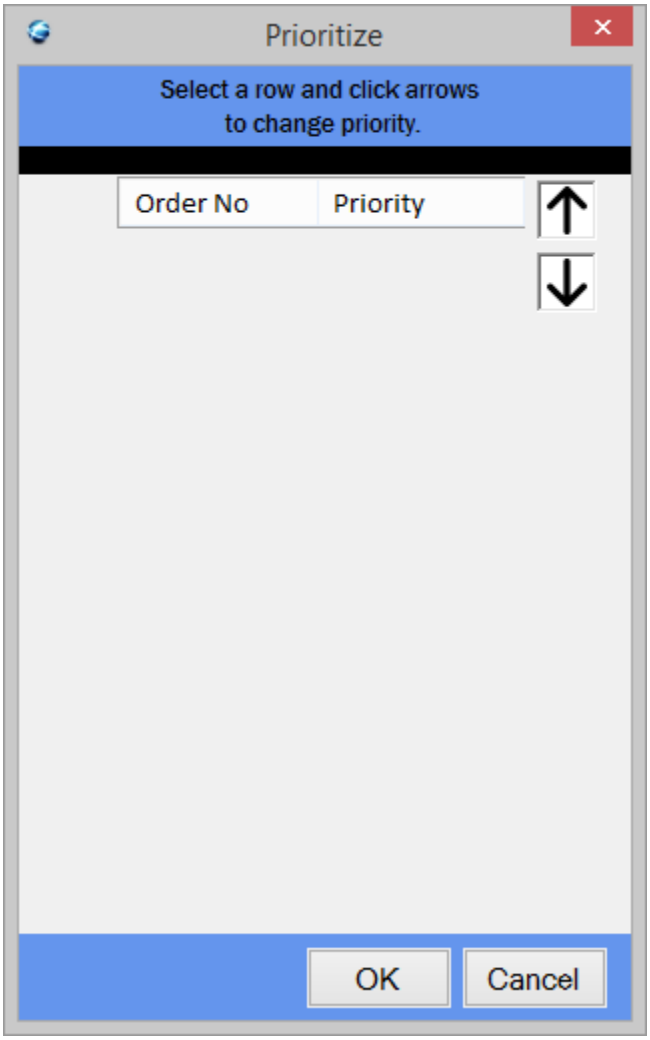


Issue (After – Previously on same screen as Gate 4 Edit)

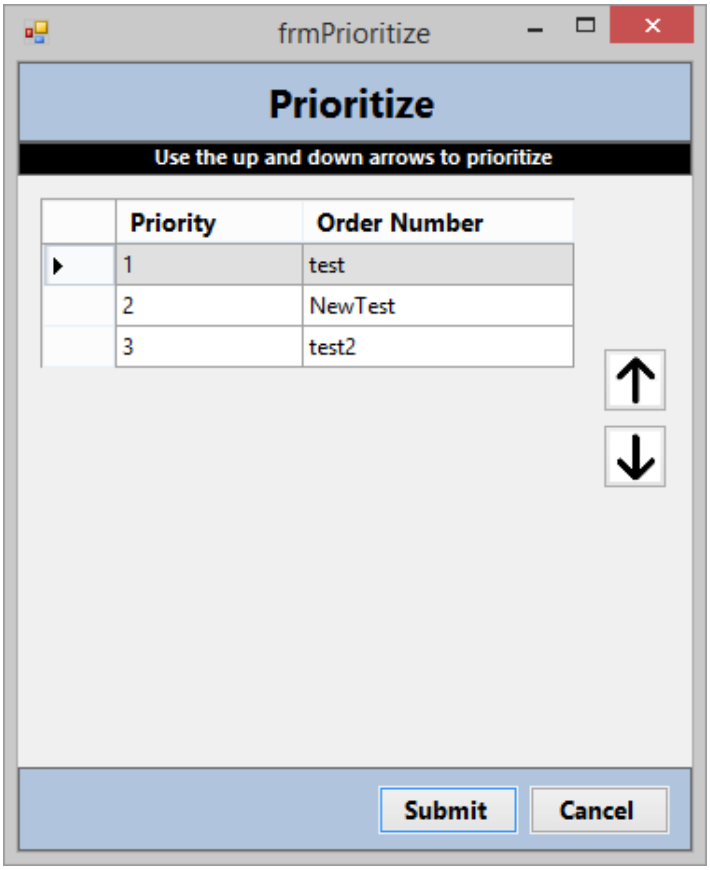


Notes (Before)

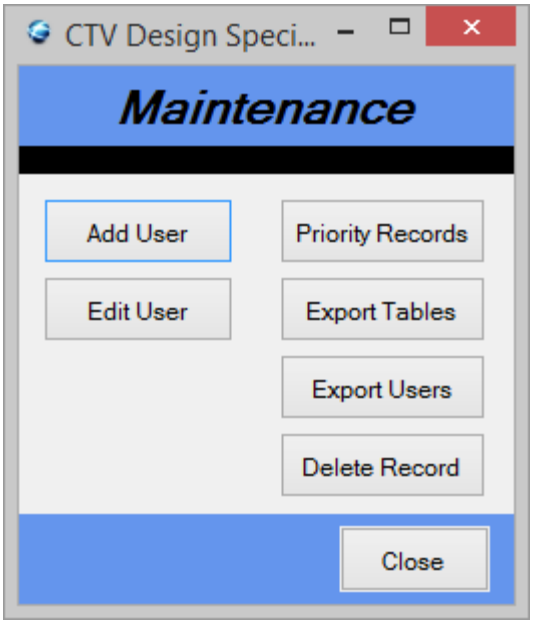

Note
Note
Written By
Krista Miller
Written on Behalf Of
Date Written
4/20/2014
Details
Submit
Cancel
Notes (After)



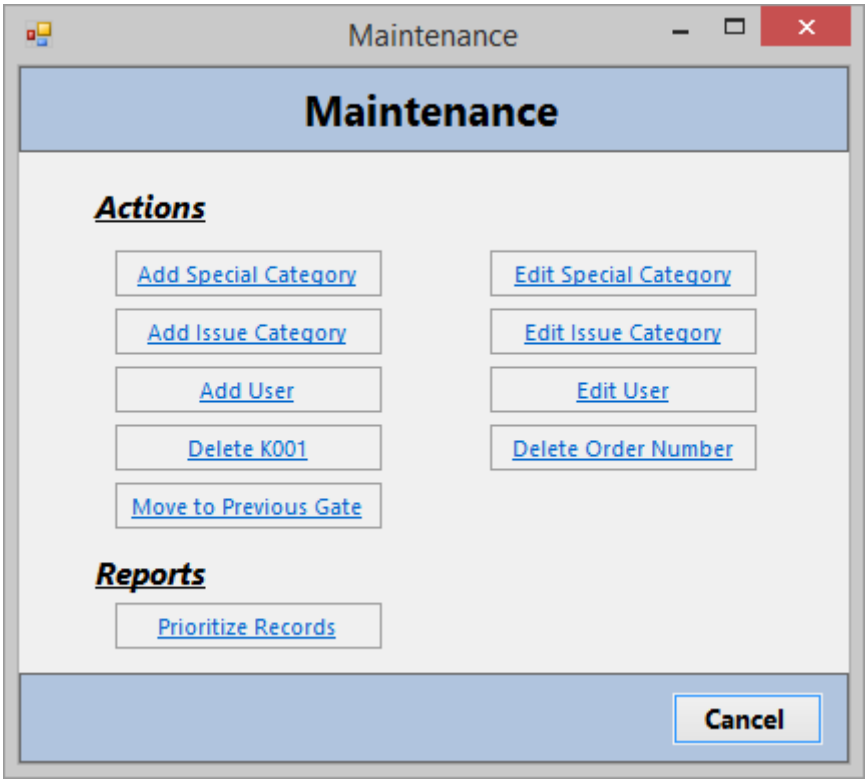
Prioritize (Before)



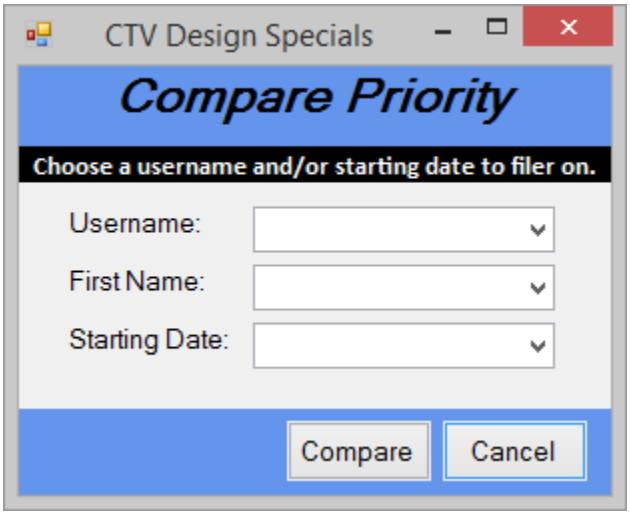
Prioritize (After)



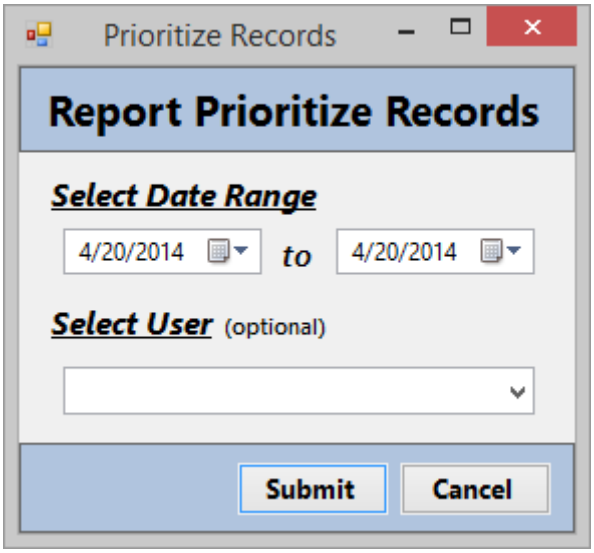
Maintenance (Before)



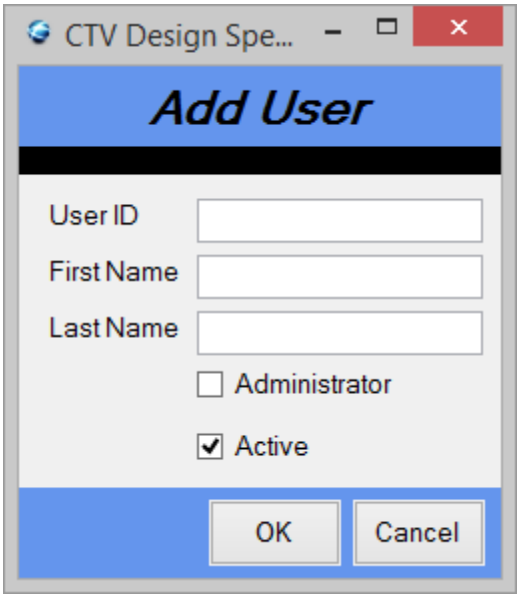
Maintenance (After)



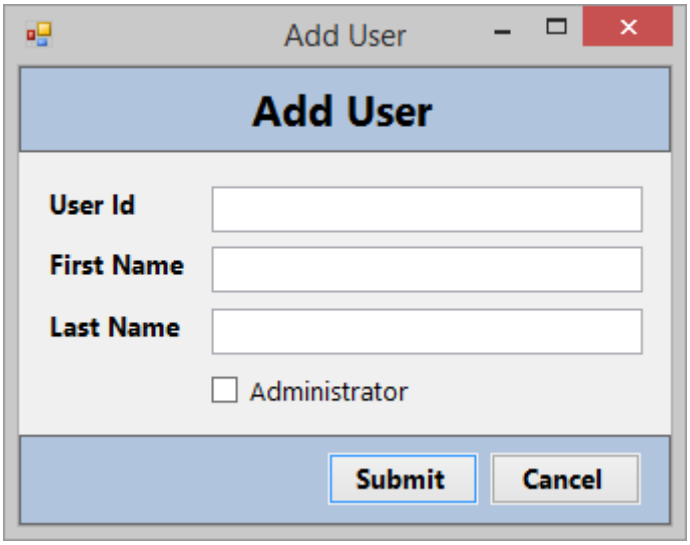
Create Priority Report (Before)



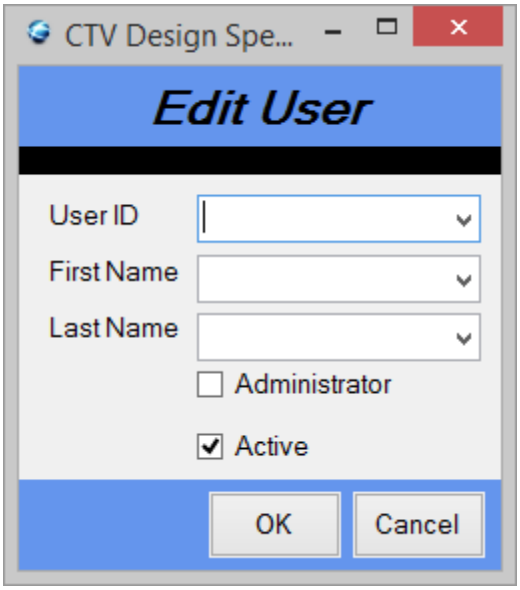
Create Priority Report (After)



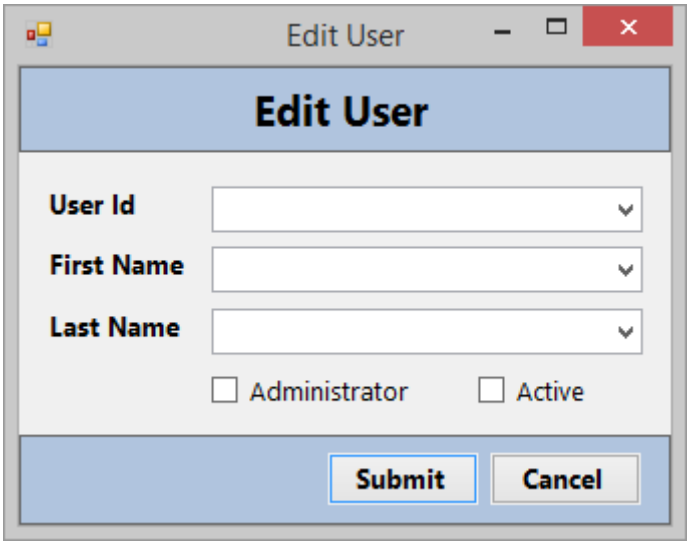
Add User (Before)



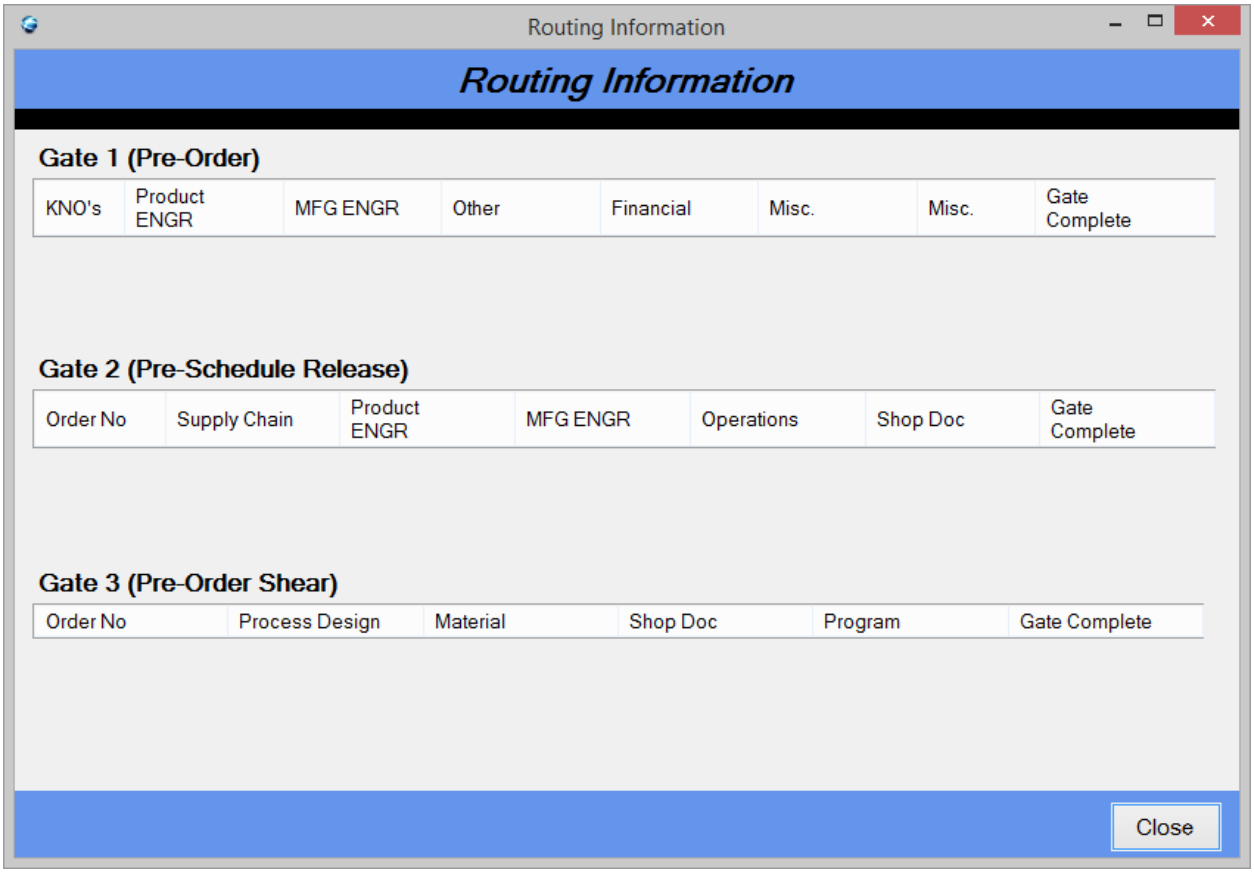
Add User (After)



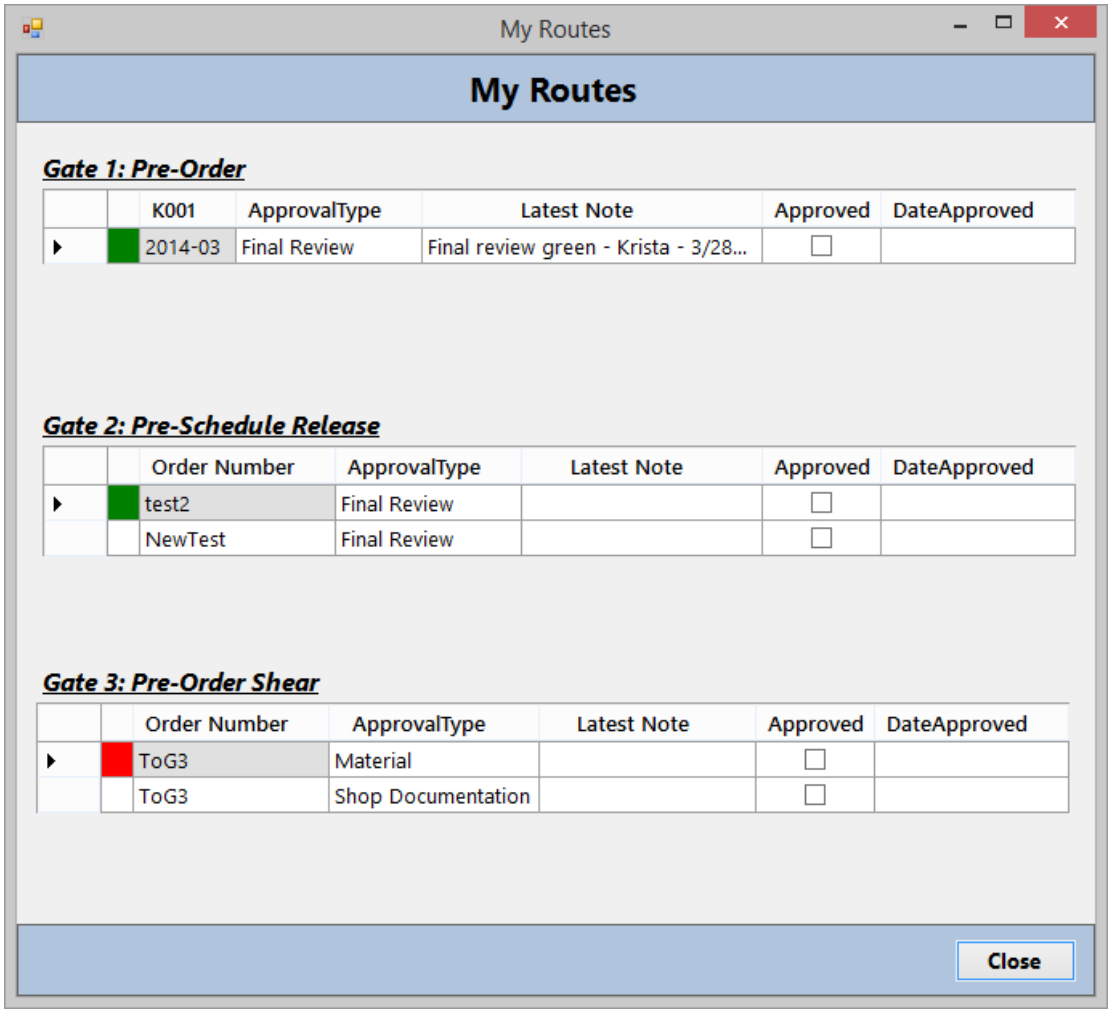
Edit User (Before)



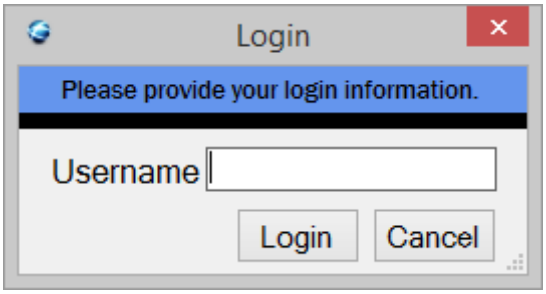
Edit User (After)



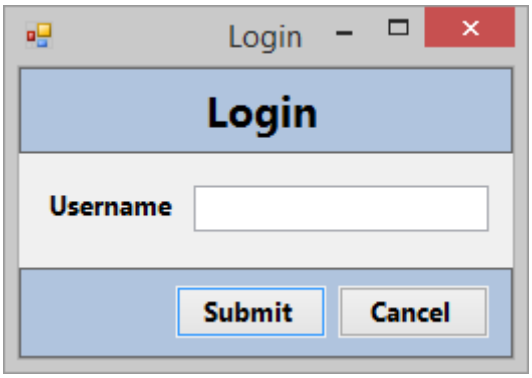
View Routes (Before)



View Routes (After)



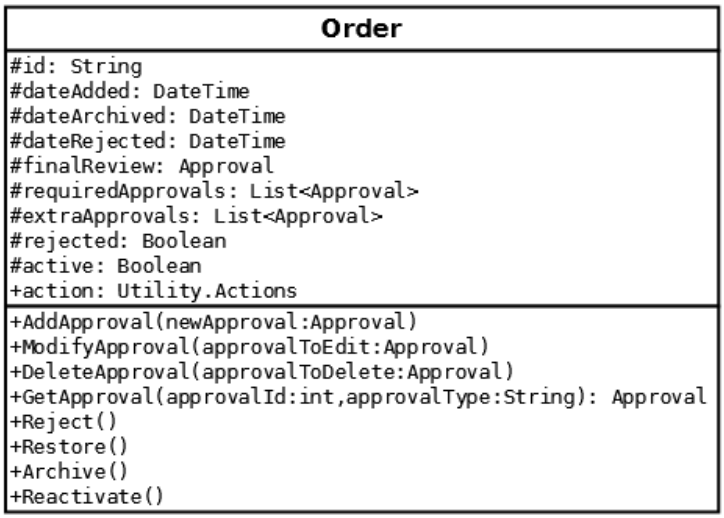
Login (Before)

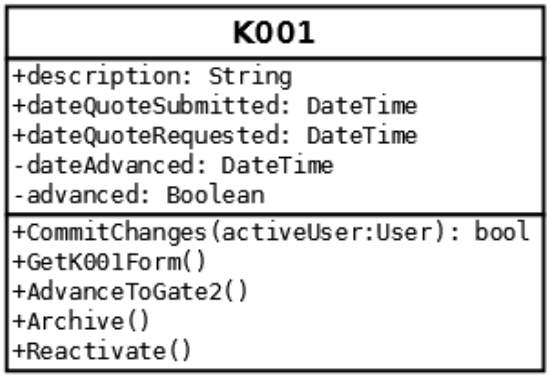


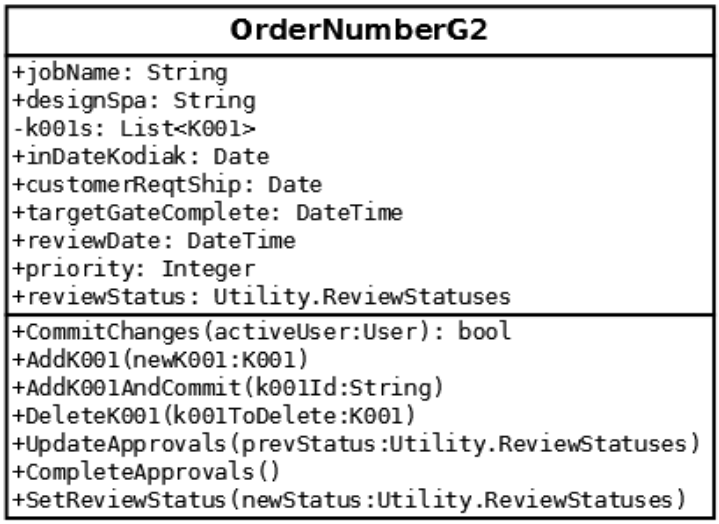
Login (After)

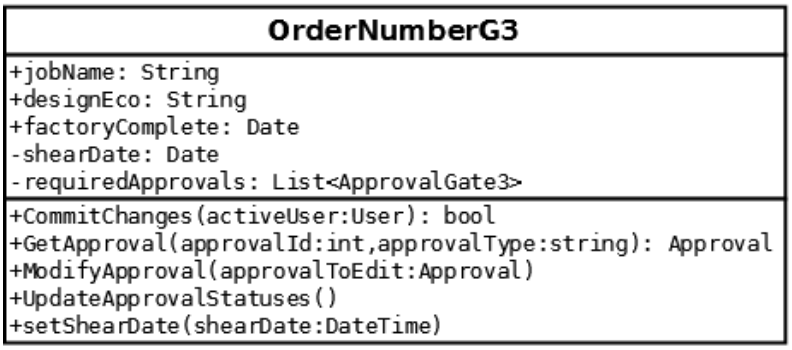
Appendix B: Internal Class Structure

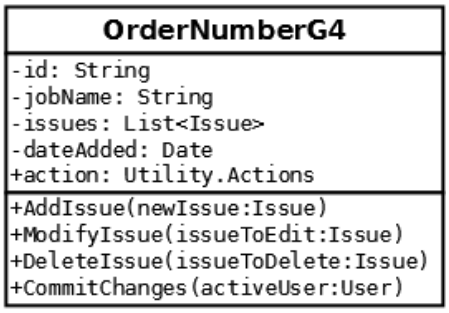
All classes have an implied constructor that takes initial inputs, set methods for private variables, and get methods for all variables.

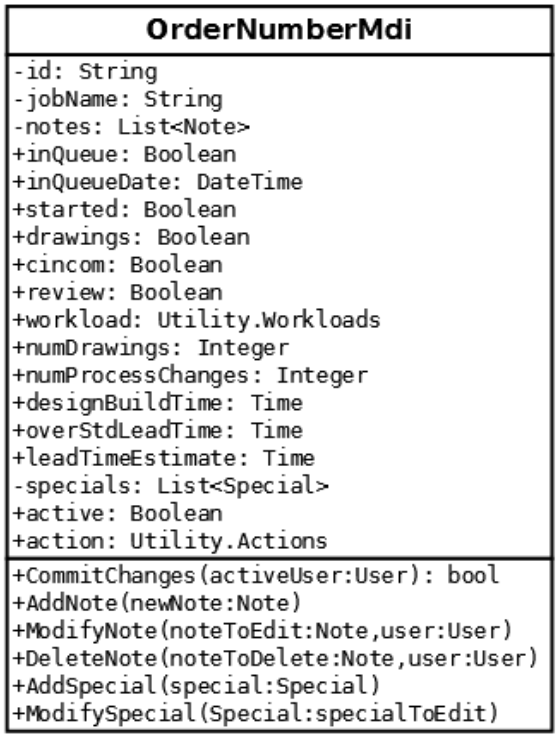


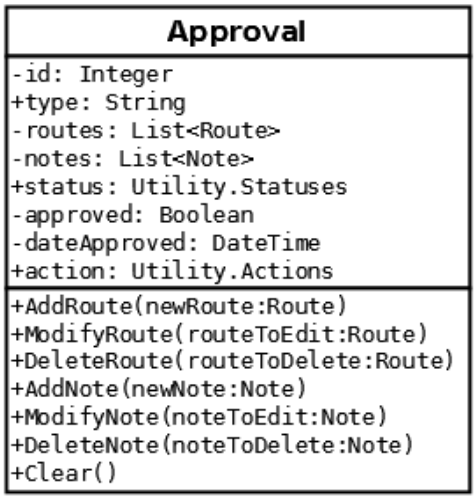


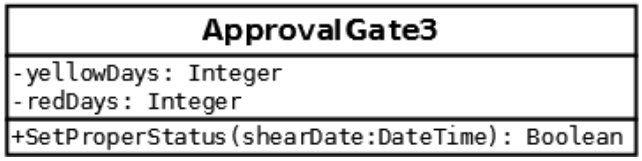


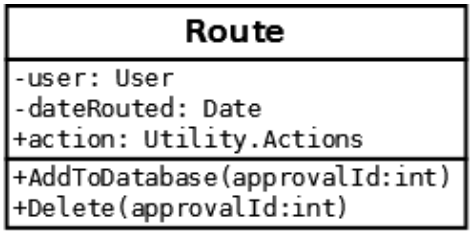

OrderNumberG3
+jobName: String
+designEco: String
+factoryComplete: Date
-shearDate: Date
-requiredApprovals: List<ApprovalGate3>
+CommitChanges(activeUser:User): bool
+GetApproval(approvalId:int,approvalType:string): Approval
+ModifyApproval(approvalToEdit:Approval)
+UpdateApprovalStatuses()
+setShearDate(shearDate:DateTime)

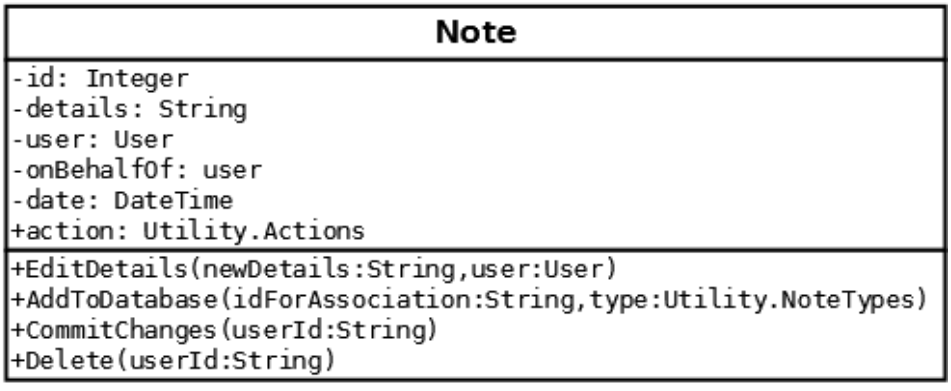

OrderNumberG4
-id: String
-jobName: String
-issues: List<Issue>
-dateAdded: Date
+action: Utility.Actions
+AddIssue(newIssue:Issue)
+ModifyIssue(issueToEdit:Issue)
+DeleteIssue(issueToDelete:Issue)
+CommitChanges(activeUser:User)

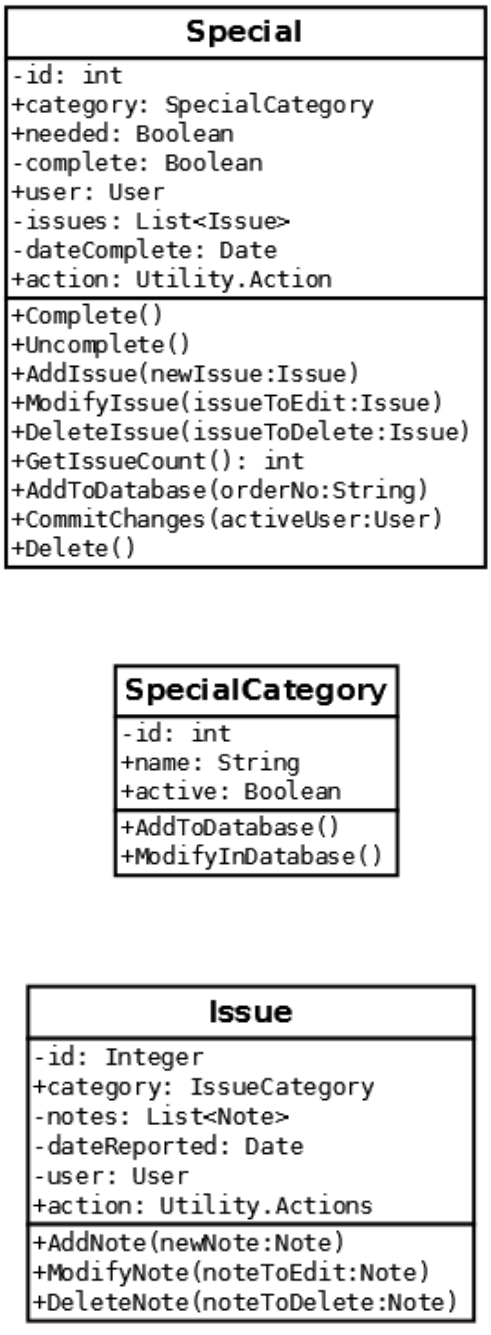

OrderNumberMdi
-id: String
-jobName: String
-notes: List<Note>
+inQueue: Boolean
+inQueueDate: DateTime
+started: Boolean
+drawings: Boolean
+cincom: Boolean
+review: Boolean
+workload: Utility.Workloads
+numDrawings: Integer
+numProcessChanges: Integer
+designBuildTime: Time
+overStdLeadTime: Time
+leadTimeEstimate: Time
-specials: List<Special>
+active: Boolean
+action: Utility.Actions
+CommitChanges(activeUser:User): bool
+AddNote(newNote:Note)
+ModifyNote(noteToEdit:Note,user:User)
+DeleteNote(noteToDelete:Note,user:User)
+AddSpecial(special:Special)
+ModifySpecial(Special:specialToEdit)

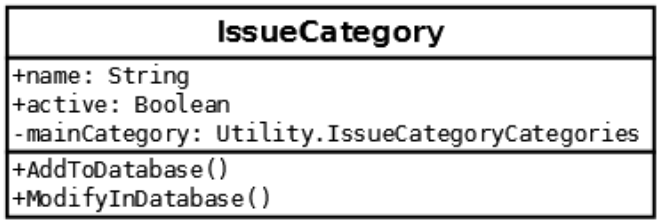

Approval
-id: Integer
+type: String
-routes: List<Route>
-notes: List<Note>
+status: Utility.Statuses
-approved: Boolean
-dateApproved: DateTime
+action: Utility.Actions
+AddRoute(newRoute:Route)
+ModifyRoute(routeToEdit:Route)
+DeleteRoute(routeToDelete:Route)
+AddNote(newNote:Note)
+ModifyNote(noteToEdit:Note)
+DeleteNote(noteToDelete:Note)
+Clear()

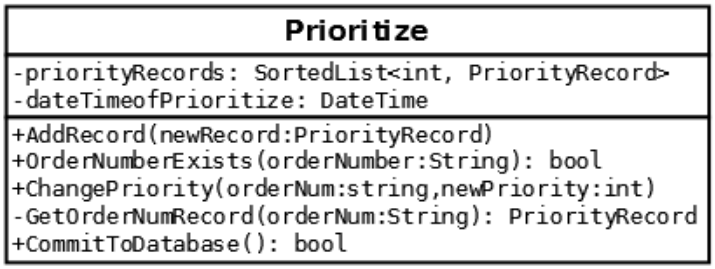

ApprovalGate3
-yellowDays: Integer
-redDays: Integer
+SetProperStatus(shearDate:DateTime): Boolean

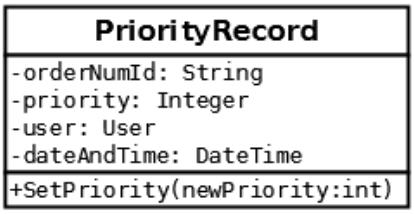

Route
-user: User
-dateRouted: Date
+action: Utility.Actions
+AddToDatabase(approvalId:int)
+Delete(approvalId:int)

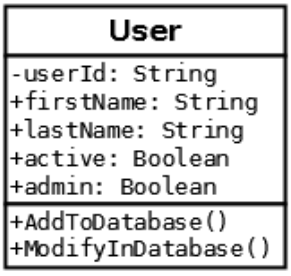

Note
-id: Integer
-details: String
-user: User
-onBehalfOf: user
-date: DateTime
+action: Utility.Actions
+EditDetails(newDetails:String,user:User)
+AddToDatabase(idForAssociation:String,type:Utility.NoteTypes)
+CommitChanges(userId:String)
+Delete(userId:String)


Special
-id: int
+category: SpecialCategory
+needed: Boolean
-complete: Boolean
+user: User
-issues: List<Issue>
-dateComplete: Date
+action: Utility.Action
+Complete()
+Uncomplete()
+AddIssue(newIssue:Issue)
+ModifyIssue(issueToEdit:Issue)
+DeleteIssue(issueToDelete:Issue)
+GetIssueCount(): int
+AddToDatabase(orderNo:String)
+CommitChanges(activeUser:User)
+Delete()
SpecialCategory
-id: int
+name: String
+active: Boolean
+AddToDatabase()
+ModifyInDatabase()
Issue
-id: Integer
+category: IssueCategory
-notes: List<Note>
-dateReported: Date
-user: User
+action: Utility.Actions
+AddNote(newNote:Note)
+ModifyNote(noteToEdit:Note)
+DeleteNote(noteToDelete:Note)


IssueCategory
+name: String
+active: Boolean
-mainCategory: Utility.IssueCategoryCategories
+AddToDatabase()
+ModifyInDatabase()


Prioritize
-priorityRecords: SortedList<int, PriorityRecord>
-dateTimeofPrioritize: DateTime
+AddRecord(newRecord:PriorityRecord)
+OrderNumberExists(orderNumber:String): bool
+ChangePriority(orderNum:string,newPriority:int)
-GetOrderNumRecord(orderNum:String): PriorityRecord
+CommitToDatabase(): bool


PriorityRecord
-orderNumId: String
-priority: Integer
-user: User
-dateAndTime: DateTime
+SetPriority(newPriority:int)


User
-userId: String
+firstName: String
+lastName: String
+active: Boolean
+admin: Boolean
+AddToDatabase()
+ModifyInDatabase()