Creating a Methodology and Tool to Capture and Resolve Conflicts in Developing Software Requirements: Requirement Lifecycle Modeling Views Manager (RLMV).

A Senior Thesis

By

LEEHA RAE-LYN HERRERA

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
In partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

April 2000

Group:
Computer Science

Creating a Methodology and Tool to Capture and Resolve Conflicts in Developing Software Requirements: Requirement Lifecycle Modeling Views Manager (RLMV).

A Senior Thesis

By

LEEHA RAE-LYN HERRERA

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
In partial fulfillment of the requirements of the
For the Designation of

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

Approved as to style and content by:

William Lively
(Fellows Advisor)

Edward A. Funkhouser
(Executive Director)

April 2000

Group: Computer Science

ABSTRACT

Creating a Methodology and Tool to Capture and Resolve Conflicts

in Developing Software Requirements.

Requirement Lifecycle Modeling Views (RLMV). (April 2000)

Leeha Rae-Lyn Herrera
Department of Computer Science
Texas A&M University

Fellows Advisor: Dr. William Lively
Department of Computer Science

Requirements management has been a traditionally overlooked aspect in designing software based systems. This lack of emphasis on managing requirements has lead to a large percent of projects either failing to meet all the needs of the customer, or in extreme cases, being cancelled when budgets or schedules have been exceeded. Companies could potentially save time and money by ensuring that requirements are accurately represented in each phase of development.

The purpose of my research is to design a tool that will aid in tracing requirements throughout the software development lifecycle. The tool, named Requirement Lifecycle Modeling Views (RLMV), follows the architecture, as defined in *The Unified Modeling Language Users Guide,* for modeling software-intensive systems. This architecture is based on five views which are the use case view, design view, process view, implementation view, and deployment view. These views work together to define the modeling of a system by representing different aspects of the system, as it is developed. RLMV works with existing software tools created by a corporation named Rational. The tool itself is implemented using Java and Oracle.

RLMV is designed to trace pre-defined requirements to modeling diagrams created for each of the five views. Though the tool was designed to work with Rational RequisitePro and Rational Rose, it is generalized enough to work with most software designing tools. In this manner, a user can select a requirement and display the names of diagrams and files, for each phase of development, associated with that requirement. The benefit to RLMV is that a user can actively trace the requirement through development to ensure that each requirement is being satisfied and prevent deviations.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The software industry was grown drastically within the last few decades to the point where billions of dollars a year are spent on software projects around the world (Krishnan and Kellner, 1999). The competition between companies is ever present and the ability to produce a quality product in a timely manner can determine the overall fate of a company.

Despite the pressure to perform, the software industry has been reluctant to adjust to the growing complexity of projects and implement better practices of managing project requirements and resources. It was estimated that a majority of the most expensive projects will "eventually be cancelled for being out of control" (Willerton, 1999). According to the Chaos Study published by the Standish Group, the trend for complete project failure has been reduced from 40% in 1997 to 26% in 1999 (Reel, 1999). However, the number of projects that exceed cost and schedule or fail to satisfy customer needs has increased from 33% to 40%. The study indicates that the completion rate has increased due to companies producing smaller more manageable projects, not because management practices have improved. This can be seen by the lack of resource management that has caused an increase in failure to meet budget or schedule.

The Chaos Study pointed out a glaring problem in the software industry, failure to implement all functionality. During the creation of a project, all stakeholders (parties that have interests tied to the success of the project) must work together to establish the conditions that the project must meet. The engineers take the information gathered from such meetings and produce the specific requirements for a project.

Requirements

Every project has requirements that it must conform to. A requirement is a concise statement that defines what a system will do. According to *The IEEE Standard Glossary of Software Engineering Terminology* (1997), the definition of a requirement is as follows:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

Other definitions of a requirement focus on the fact that a requirement specifies what a system should do without referring to how it will be implemented (Sommerville and Sawyer 1997). A requirement serves as a guiding post or general blue print for which the development of a system must follow.

The problem with finding one universally accepted definition of a requirement is that no such definition exists. "The real requirements actually reside in people's minds" and documenting requirements is an attempt to model or represent their ideas (Wiegers, 1999). This is why it is paramount "that all project stakeholders arrive at a shared understanding of the terms used to describe" a requirement.

Requirements Management

Once a project's requirements have been stated, some system of managing those requirements must be implemented. Project failure to produce all functionality is usually caused by poor management of the project's requirements. Requirements management is a term to describe how a team handles the requirements of a project during the lifecycle of the project. It can be defined as a two-part definition. Requirements management is "a systematic approach to eliciting, organizing, and documenting the requirements of the system," as well as, "a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system" (Oberg, Probasco, and Ericsson 1998).

The overall success of a project largely depends on how the initial stages of a project are carried out. If care is taken to ensure requirements are clearly stated, documented, and managed the success rate of a project will be increased. It is rare to have a project succeed where the requirements were poorly defined despite almost flawless execution of design, implementation, and testing. Engineers must not only build quality projects but must build quality projects that satisfy the customer's need.

PROBLEM

The initial phases of a project directly determine the success of a project. As Fredrick Brooks stated in "No Silver Bullet: Essence and Accidents of Software Engineering" (1987):

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. Not other part is more difficult to rectify later.

As a project progresses in development the cost to correct a problem or defect increases dramatically (Arthur, Groner, Hayhurst, and Holloway, 1999). It is therefore beneficial to carefully manage requirements during the initial phases of development and screen out errors. This would prevent time and money from being wasted due to problems compounding as the project grows in complexity.

Several problems arise in managing requirements. Organizations tend to "struggle with the elicitation, specification, and management of requirements" (Weiger and Card, 1999). Problems like these can cause a vague understanding of the requirements, which leads to failure to implement all functionality. This also goes hand and hand with managers not accurately understanding the system, which causes poor estimations for resources and leads to projects going over budget and schedule.

The industry has been slow to change and at best may only implement partial strategies (Moitra, 1999). Any solution to better handle system requirements has to be easily adapted to a wide variety of projects and work environments, easily implemented without high demands of time to learn or interface with, and cost effective. As with any aspect of life, human nature usually dictates that if an action or process is too hard or complicated it will not be used. The goal of designing software development tools is to find a balance between providing a tool that provides enough functionality to be useful and not overwhelming the user and making the tool burdensome.

SOLUTION

Overview

The focus of this research is concentrated on tracing requirements through the development of a project by documenting the associations of requirements with the phases of project modeling. This will help engineers check their designs to ensure that all requirements are satisfied and avoid costly deviations. By being able to follow the development of a project, managers could have more documented information to base decisions on managing resources and make better estimations.

The tool implemented in this research is named Requirement Lifecycle Modeling Views Manager (RLMV). RLMV follows the architecture, as defined in *The Unified Modeling Language Users Guide*, for modeling software-intensive systems. This architecture is based on five views: a use case view, a design view, a process view, a implementation view, and a deployment view. In addition to these five views, I added two additional views a requirement view and a revision view. All these were used together to define the modeling of a system by representing different aspects of the system, as it is developed.

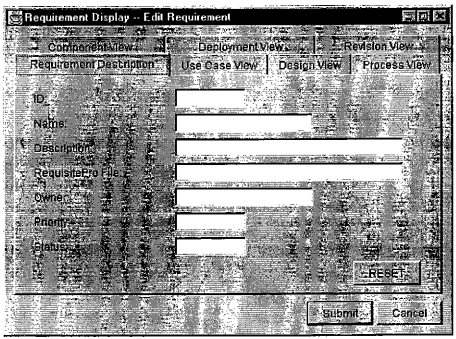

Fig. 1. The Requirement View implemented in RLMV

The Requirement View was created to contain fields that store the basic information of a requirement such as identification number, name, description, file location, owner, priority, and status (Fig. 1). The purpose of this view is to store the general characteristics of requirement in one easy to access location.

The Revision View was added to provide some revision control. Most projects go through revision or changing requirements (Fig. 2). Carefully controlled and managed revision can be healthy as the project adjusts to better meet customer needs. However changes in requirements that is unmanaged can lead to project failure though the requirements had been clearly defined in the beginning. One way to avoid run away feature creep, or prevent the loss of crucial requirements is to document changes and have it controlled by a small group. This view allows old requirements to be linked to revised requirements so that no requirement is deleted after it has been specified. The fields in this view include the identification of the revised requirement, who is responsible for making the decision, the status of the decision, and an area provide to record related notes. By doing this, users can refer to the history of a revision if problems and review the rationale for change.

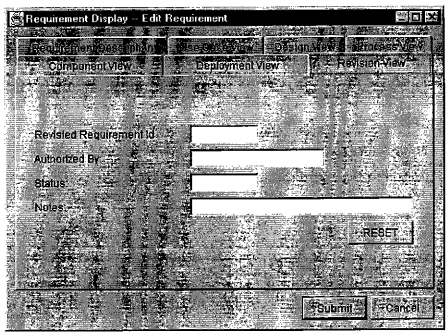

Fig. 2. The Revision View in RLMV

The purpose of the Use Case View is to describe "the behavior of the system as seen by its end users, analysts, and testers" (Booch, Jacobson, and Rumbaugh, 1999). It does not specify the organization of the

project but instead "specifies the forces that shape the system's architecture." To trace a requirement through this view associations between specific use case diagrams, sequence diagrams, and collaboration diagrams should be made (Fig. 3).

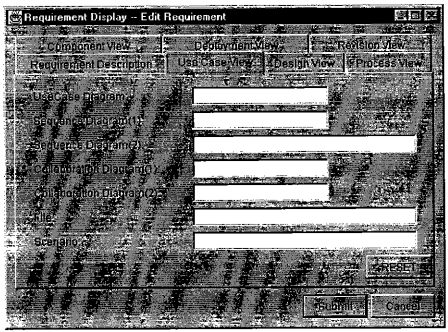

Fig. 3. The Use Case View in RLMV

The Design View helps to form the vocabulary of the project by defining the names of classes to be implemented (Booch, Jacobson, and Rumbaugh, 1999). This view addresses the functionality of the project through the services it provides the users. The fields in the design view include the package the requirement is implemented in, class diagrams, object diagrams, statechart diagrams, and the file that these diagrams are found in (Fig 4).

The Process View is identical to the Design View in the fields that it contains. The purpose of the process view differs, though, in that it addresses the dynamic functions of a project. This functions include "...performance, scalability, and throughput" of the project (Booch, Jacobson, and Rumbaugh, 1999).

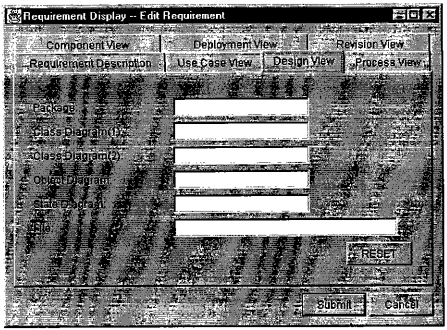

Fig. 4. The Design View in RLMV

The Component view, which can be referred to as the implementation view, "encompasses the components and files that are used to assemble and release the physical system" (Booch, Jacobson, and Rumbaugh, 1999). The focus of this view is on the "...configuration management of the system's releases." The diagrams associated with each requirement for this view are the component diagrams (Fig 5).

The Deployment view addresses the "nodes that form the system's hardware topology on which the system executes" (Booch, Jacobson, and Rumbaugh, 1999). It describes the "distribution, delivery, and installation of parts that make up the physical system." The diagrams that relate to this view are deployment diagrams (Fig 6). It is possible that multiple requirements share the same diagrams, especially in the deployment view.

By associating each requirement to elements in each view, the development of a project is forced to strictly adhere to the specified requirements. A more in-depth description of each field in the views can be found it the Relational Database subheading where fields are represented by columns in database tables.

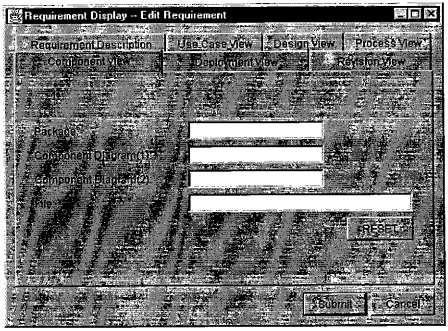

Fig. 5. The Component View in RLMV

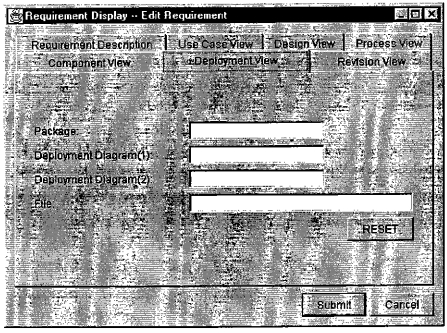

Fig. 6. The Deployment View in RLMV

Relational Database Design

The database backbone for RLMV was designed in Oracle. A relational database was created following the structure of the seven views chosen to represent the development of a requirement, the Requirement View, UseCase View, Design View, Process View, Component View, Deployment View, and Revision View. All tables use the ID column as the key, which represents the requirement identification. The ID must be a unique combination of letters and numbers that is expected to be assigned by Rational RequisitePro or by other similar tools during the documentation of the requirements.

The Requirement View represents the general information of a requirement gathered at the analysis phase. The columns defined for the table are the ID, Name, Description, File_Name, Owner, Priority, and Status, as seen in Table 1. The columns used in the table are fairly generalized and self-explanatory. For the Priority and Status column, no range of values is enforced by the database. Appropriate values for the Priority column would include a number range and for the Status column values such as, opened, revised, and closed. It is ultimately the responsibility of the project team to establish appropriate values and to adhere to them.

TABLE 1 Description of the Requirement_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	Requirement id specified by RequisitePro
NAME	Null	VARCHAR2(32)	any string up to 24 characters
DESCRIPTION	Null	VARCHAR2(64)	any string up to 64 characters
FILE_NAME	Null	VARCHAR2(64)	name and or path of the RequisitePro File
OWNER	Null	VARCHAR2(16)	name of individual responsible for requirement
PRIORITY	Null	VARCHAR2(8)	priority of requirement, no default range of values
STATUS	Null	VARCHAR2(16)	stage or status of the requirement no default range of values

The Use Case View table contains columns to record the names and location of use case, sequence, and collaboration diagrams, shown in Table 2. A use case diagram "shows a system in terms of the external users of the system, known as actors" (Pooley and King 1999). They address the static view and help to organize and model the system's behaviors (Booch, Jacobson, and Rumbaugh, 1999).

Sequence and collaboration diagrams are interaction diagrams, which describe the dynamic view. A sequence diagram "emphasizes the time-ordering of messages" and a collaboration diagram "emphasizes the structural organization of the objects that send and receive messages." These two types of diagrams can be transformed into other giving them isomorphic properties.

TABLE 2 Description of the UseCase_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
USE_DIAGRAM	Null	VARCHAR2(32)	name of Use Case Diagram
SEQUENCE_DIAGRAM1	Null	VARCHAR2(32)	name of Sequence Diagram
SEQUENCE_DIAGRAM2	Null	VARCHAR2(32)	name of Sequence Diagram
COLLABORATION_DIAGRAM1	Null	VARCHAR2(32)	name of Collaboration Diagram
COLLABORATION_DIAGRAM2	Null	VARCHAR2(32)	name of Collaboration Diagram
FILE_NAME	Null	VARCHAR2(64)	name and or path of the file containing the diagrams

The Design View and Process View tables contain columns to record the names and locations of the package the requirement is found in and class, object, and state diagrams, shown in Table 3 and Table 4. The Package column represents the name of the collection that the "model elements may be grouped into, representing modules or libraries" (Pooley and King, 1999). In Rational Rose the packages designated are often User Services, Business Services, and Data Services. The class diagram columns represent diagrams that describe "a set of classes, interfaces, and collaborations and their relationships" (Booch, Jacobson, and Rumbaugh, 1999). Object diagrams show "static snapshots of instances of the things found in class diagrams" from the "perspective of real or prototypical cases." The statechart diagrams of a *requirement* represent a state machine that consists of "states, transitions, events, and activities" and emphasizes the "event-ordered behavior of an object." The difference in diagrams between the two views is that the diagrams in Design View represent the static aspects of the system and those in the Process View represent the dynamic aspects.

TABLE 3 Description of the Design_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
PACKAGE	Null	VARCHAR2(32)	name of the Package or Tier the requirement is grouped into
CLASS_DIAGRAM1	Null	VARCHAR2(32)	name of a Class Diagram
CLASS_DIAGRAM2	Null	VARCHAR2(32)	name of a Class Diagram
OBJECT_DIAGRAM	Null	VARCHAR2(32)	name of a Object Diagram
STATE_DIAGRAM	Null	VARCHAR2(32)	name of a State Diagram
FILE_NAME	Null	VARCHAR2(64)	name and or path of the file containing the diagrams

TABLE 4 Description of the Process_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
PACKAGE	Null	VARCHAR2(32)	name of the Package or Tier the requirement is grouped into
CLASS_DIAGRAM1	Null	VARCHAR2(32)	name of a Class Diagram
CLASS_DIAGRAM2	Null	VARCHAR2(32)	name of a Class Diagram
OBJECT_DIAGRAM	Null	VARCHAR2(32)	name of a Object Diagram
STATE_DIAGRAM	Null	VARCHAR2(32)	name of a State Diagram
FILE_NAME	Null	VARCHAR2(64)	name and or path of the file containing the diagrams

The Component View table contains columns to record the name and location of the package and component diagrams of a requirement, as seen in Table 5. The package column in this view is the same as those in the Design View. Component diagrams describe the "organization and dependencies among a set of components" that show "the static implementation view of a system" (Booch). Component diagrams usually contain one or more classes, interfaces, or collaborations. The Deployment View tables, seen in Table 6, builds upon the Component View in that deployment diagrams show the "configuration of run-time processing nodes and the components that live on them." As components contain one or more classes, nodes in deployment diagrams contain one or more components.

TABLE 5 Description of the Componet_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
PACKAGE	Null	VARCHAR2(32)	name of the Package or Tier the requirement is grouped into
COMPONENT_DIAGRAM1	Null	VARCHAR2(32)	name of a Component Diagram
COMPONENT_DIAGRAM2	Null	VARCHAR2(32)	name of a Component Diagram
FILE_NAME	Null	VARCHAR2(64)	name and or path of the file that contains the diagrams

TABLE 6 Description of the Deployment_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
PACKAGE	Null	VARCHAR2(32)	name of the Package or Tier the requirement is grouped into
DEPLOYMENT_DIAGRAM1	Null	VARCHAR2(32)	name of a Deployment Diagram
DEPLOYMENT_DIAGRAM2	Null	VARCHAR2(32)	name of a Deployment Diagram
FILE_NAME	Null	VARCHAR2(64)	name and or path of the file containing the diagrams

The Revision View table contains columns that link a requirement, *to a requirement* that has replaced it, Table 7. The columns contain information on the revised requirement identification number, the party responsible for allowing the revision, and the status of the request for revision. The Notes column is available for team members to document comments associated with the history of the revision.

TABLE 7 Description of the Revision_View Database Table

Name	Null?	Type	Description
ID	Not Null	VARCHAR2(16)	requirement id specified by RequisitePro
REVISED_ID	Null	VARCHAR2(16)	requirement id specified by RequisitePro that reflects the changes made
AUTHORIZED_BY	Null	VARCHAR2(32)	name of a party responsible for revision control
STATUS	Null	VARCHAR2(16)	status or stage of revision
NOTES	Null	VARCHAR2(128)	related comments

Code Implementation

The tool was implemented in Java using Swing and JBDC (Appendix A contains a complete copy of the code). Most of the interface structure was built with JFrames, JDialog, JTabbedPages, and JPanel. In designing the different views a base class was used, Views, were the name of the table in the database associated with the view along with its column names and values were defined when the object is instantiated. A class named RequirementInfo then called a View for each of the views used in the interface. When a RequirementInfo object is created it creates each view and specifies the table and column names needed through private strings within the class.

All connections to the Oracle database using JDBC where handled in the DbActions class (designed after the DbActions classes used by Chris Wurts and Tom Woods in their implementation of LCAM, 1998). This class is called each time a connection is made. The View class calls the DbActions class in its methods that make changes to a requirement's information in the database and to retrieve information to display.

To create the views that a user interfaces with a JTabbedPages was loaded into a JPanel. Each view was created by a JPanel that the JTabbedPages displayed. This area of code still lacks completion but the basic framework has implemented. Currently each panel has a separate class to deal with the specific names of fields to be displayed and retrieved. This is awkward and a design for a basic JPanel class, which is given the specific information for each view at creation, has been designed but not full implemented.

User Interface

To start the tool, the user must go to a command prompt and start the program and type the Java command for execution. A window then appears with a menu bar with 3 options, File, Requirement, and Help (FIG. 7).

To add a requirement, the New option must be selected under the Requirement menu. This will open a new JPanel with a JTabbedDialog inside it. Each tab contains one of the seven views that I implemented. The only field that must be correctly completed to add a requirement to the database is the ID field, which serves as a key to all the database tables. The user fills the desired information out and selects the OK option on the panel. This then opens a connection to the database and adds the requirement to all the necessary tables.

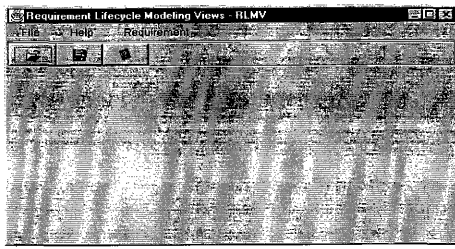

Fig. 7. RLMV when first started

To edit a requirement, the Edit option must be selected under the Requirement menu. This will open a JDialog, which will prompt the user to enter an ID for an existing requirement (Fig. 8). When the OK button is selected the tool opens a connection to the database and runs a query on the Requirement_View table. If a requirement by the given ID exists the tool continues by opening a new JPanel with a JTabbedPages with the seven views. The same JPanel and JTabbedPages class used in adding a new requirement is called and appears the same except in two main differences. The ID field is non-editable since it serves as the key value for the database tables and once a requirement is added it cannot be removed, only links to revised requirements can be made, and the values for fields that been pre-defined

are displayed. While editing the values the user can select the Reset button which will clear all field values for that page, except for ID value. When all the necessary changes are made the user then selects the Submit button and a connection to the database is opened and SQL update commands are made to change the values of the fields. The frame is then closed and the user is returned to the initial window.

To display a requirement, the Display option must be selected under the Requirement menu. This will open a new JDialog, which will prompt the user to enter an ID for an existing requirement. When the OK button is selected the tool opens a connection to the database and runs a query on the *Requirement_View* table. If a requirement by the given Id exists the tool continues by opening a new JPanel with a JTabbedPages with the 8 views. The same JPanel and JTabbedPages class used in adding a new requirement is called and appears the same except in one main difference. Though the fields look editable, no changes are made to the requirement in the database. This option is for viewing information in a read only manner. To exit this option the user selects OK and the frame is then closed and the user is returned to the initial window.

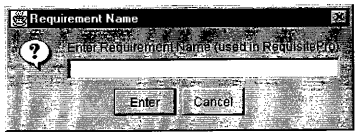

Fig. 8. Dialog Box prompting user to enter a requirement ID

To exit the application the user can either select the Exit option in the File menu or double click on the corner of the window in the standard MS Windows manner.

FUTURE IMPROVEMENTS

With the limitations of time and resources, RLMV lacks considerable functionality that could allow it to be more user friendly and offer better requirements management. RLMV does not take advantage of the capability of automation between itself and Rational RequisitePro and Rose. The tool could gather information for requirements such as identification number, name, description, priority, and ownership by specifying a specific RequisitePro file. This would make the tool more attractive to engineers and more likely to be implemented by removing the tedious busy work of filling out information that should have already been documented. RLMV could also automate the ability to bring up diagrams for each view by one click instead of having the user open up the application and file to view the diagram.

RLMV could also provide graphs and charts given stakeholders a quick way of checking on the overall progress of a project. The graphs could include information on how many requirements are in a particular stage or have a particular priority. This information could be used to better manage schedule and budget.

The possibility of presenting information in a manner easier for users to access and manipulate must also be considered. Research would have to be conducted in how humans organize information so it could be replicated as much as possible, instead of humans having to conform to ways that computers store information.

The biggest area for improvement though, is not functionality but in testing. RLMV has not be formally reviewed or used by a team of software engineers. All the benefits it was designed to offer have not been verified. On the onset of this research, the tool had been planned to be used and tested by a Software Engineering class in the Department of Computer Science at Texas A&M University. The class would have been divided into two groups, one using RLMV to complete the class project and one control group that would not use the tool. Comparisons would have been conducted between the two groups, analyzing overall documentation, shared understanding of the stated requirements, time for completion, and number of defects within the project. Personal interviews would have also been conducted on the team members that worked with RLMV to document the functionality from a user's perspective. The results gathered would have then been reviewed and possible changes to RLMV would have been considered. Due to time constraints, these tests were not realized and it is crucial that before any further functionality is added, RLMV go through such tests.

CONCLUSION

The basic idea that RLMV was built on was that if engineers were given an easy to use way of documenting the progress of a requirement, they could ensure that the requirement was being fulfilled. Thought must be given in defining the association between requirements and the models of a project, if inconsistencies or holes arise they should be easily detected. RLMV displays all information for a requirement in a well-organized manner, which allows all stakeholders to easily follow the progress of a requirement.

The direct benefits of implementing RLMV in a software project include complete requirement traceability throughout the development of a project, limited revision control management, and the ability to work in conjuncture with software tools already in use. Indirectly the benefits lead to a more organized and documented project where all stakeholders have access to the same information, which can ensure agreement. More importantly, by directly relating requirements to the modeling of a project, team members can ensure that all requirements are accurately being satisfied. All these factors could lead to more successful projects by reducing resources spent on correcting errors in later, more costly, phases of development while at the same time producing projects that implement all required functionality. Software companies could potentially save large amounts of money and create a more professional industry that customers can rely on.

The bottom line is that "software still takes too long to develop, costs too much, and does not work well when eventually delivered" (Fitzgerald and O'Kane, 1999). A standard process needs to be implemented to give structure and discipline to a field that has been characterized as chaotic and unpredictable. Whether teams decide to implement RLMV or other software development tools, the need for tools that follow an accepted standard is evident. Until the development of software is seen as a process that can be controlled and monitored, the industry will lack true professionalism in delivering quality products in a timely manner (Krishman and Kellner, 1999).

REFERENCES

Arthur, James, Groner, Markus, Hayhurst, Kelly, and Holloway, C. 1999. Evaluating the effectiveness of independent verification and validation. *IEEE Computer.* (Oct.), 79-83.

Booch, Grady, Rumbaugh, James, and Jacobson, Ivar. 1999. *The Unified Modeling Language User Guide.* Addison Wesley Longman, Inc. Reading, MA.

Fitzgerald, Brian, and O'Kane, Tom. 1999. A longitudinal study of software process improvement. *IEEE Software.* (May/June), 37-45.

Krishnan, M.S and Kellner, Marck I. 1999. Measuring process consistency: implications for reducing software defects. *IEEE Transactions on Software Engineering.* 25, 6 (Nov/Dec.), 800-815.

Moitra, Deependra. 1999. Software Engineering in the small. *IEEE Computer.* (Oct.), 39-40.

Oberg, Roger, Probasco, Leslee, and Ericsson, Maria . 1998. Applying requirements management with use cases. Technical Paper TP505. Rational Software Corporation.

Pooley, R. and King, P. 1999. Unified modeling language and performance engineering. *IEEE Proceeding Software.* 146, 1 (Feb.), 2-11.

Reel, John S. 1999. Critical success factors in software projects. *IEEE Software.* (May/June), 18-23.

Sommerville, Ian, and Sawyer, Pete. 1997. *Requirements Engineering: A good Practice Guide.* John Wiley & Sons. Chichester, England.

Wiegers, Karl E. 1999. *Software Requirements.* Microsoft Press, Redmond, WA.

APPENDIX A

```
//Title:    ReqTool
//Version:
//Copyright: Copyright (c) 1999
//Author:    Leeha Herrrera
//Company:   Dept Computer Science
//Description: Requirements Lifecycle Modeling Views Manager
//          Creates the main appication by calling the BaseFrame

import tool.*;

import javax.swing.UIManager;
import java.awt.*;

public class RLMV {
 boolean packFrame = false;

 //Construct the application
 public RLMV() {
  BaseFrame frame = new BaseFrame();
  //Validate frames that have preset sizes
  //Pack frames that have useful preferred size info, e.g. from their layout
  if (packFrame)
   frame.pack();
  else
   frame.validate();
  //Center the window
  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize = frame.getSize();
  if (frameSize.height > screenSize.height)
   frameSize.height = screenSize.height;
  if (frameSize.width > screenSize.width)
   frameSize.width = screenSize.width;
  frame.setLocation((screenSize.width - frameSize.width) / 3 , (screenSize.height - frameSize.height) / 3);
  frame.setVisible(true);
 }

 //Main method
 public static void main(String[] args) {
  try {
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
  }
  catch(Exception e) {
  }
  new RLMV();
 }
}
```

```
//Title:    BaseFrame
//Description: Creates the main window with a menu bar, for the tool

package tool;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BaseFrame extends JFrame {
  //create all the items for the menu bar
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenu menuHelp = new JMenu();
  JMenuItem menuHelpAbout = new JMenuItem();
  JToolBar toolBar = new JToolBar();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JButton jButton3 = new JButton();
  ImageIcon image1;
  ImageIcon image2;
  ImageIcon image3;

  //Items specified for the Requirement option in the menu
  JMenu menuRequirement = new JMenu();
  JMenuItem menuReqNew = new JMenuItem();
  JMenuItem menuReqEdit = new JMenuItem();
  JMenuItem menuReqDisplay = new JMenuItem();
  //

  BorderLayout borderLayout1 = new BorderLayout();

  //Construct the frame
  public BaseFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new ImageIcon(tool.BaseFrame.class.getResource("openFile.gif"));
    image2 = new ImageIcon(tool.BaseFrame.class.getResource("closeFile.gif"));
    image3 = new ImageIcon(tool.BaseFrame.class.getResource("help.gif"));
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(500, 300));
    this.setTitle("Requirement Lifecycle Modeling Views Manager - RLMV");
```

```
//prepare items to be added to menu and assign listners to detect user selection
menuFile.setText("File");
menuFileExit.setText("Exit");
menuFileExit.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
   fileExit_actionPerformed(e);
 }
});

menuHelp.setText("Help");
menuHelpAbout.setText("About");
menuHelpAbout.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
   helpAbout_actionPerformed(e);
 }
});

menuRequirement.setText("Requirement");
menuReqNew.setText("New");
menuReqNew.addActionListener(new ActionListener () {
 public void actionPerformed (ActionEvent e) {
    reqNew_actionPerformed(e);
 }
});

menuReqEdit.setText("Edit");
menuReqEdit.addActionListener(new ActionListener () {
 public void actionPerformed (ActionEvent e) {
    reqEdit_actionPerformed(e);
 }
});

menuReqDisplay.setText("Display");
menuReqDisplay.addActionListener(new ActionListener () {
 public void actionPerformed (ActionEvent e) {
    reqDisplay_actionPerformed(e);
 }
});

//add items to menu
jButton1.setIcon(image1);
jButton1.setToolTipText("Open File");
jButton2.setIcon(image2);
jButton2.setToolTipText("Close File");
jButton3.setIcon(image3);
jButton3.setToolTipText("Help");
toolBar.add(jButton1);
toolBar.add(jButton2);
toolBar.add(jButton3);
menuFile.add(menuFileExit);
menuHelp.add(menuHelpAbout);
menuRequirement.add(menuReqNew);
menuRequirement.add(menuReqEdit);
```

```
  menuRequirement.add(menuReqDisplay);
  menuBar1.add(menuFile);
  menuBar1.add(menuHelp);
  menuBar1.add(menuRequirement);
  //add menu to frame
  this.setJMenuBar(menuBar1);
  this.getContentPane().add(toolBar, BorderLayout.NORTH);
 }

 //File | Exit action performed
 public void fileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
 }

 //Help | About action performed
 public void helpAbout_actionPerformed(ActionEvent e) {
 }

 //Requriement | New action performed
 public void reqNew_actionPerformed(ActionEvent e) {
   //create the panel to hold the tabbedpages
   PanelFrame panelFrame = new PanelFrame("Requirement Display -- Edit Requirement");
   //create the requirement contain group for the requirement that will be added
   ReqViews reqView = new ReqViews();
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = panelFrame.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
      frameSize.width = screenSize.width;
    panelFrame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height + 45) / 2);
    //Load the TabbedPages into the panel, send info about whether a new
    //requirement is being created or not
    panelFrame.getContentPane().add(new GetReqTabbedPages(panelFrame,false,reqViews),
                    BorderLayout.CENTER);
    panelFrame.setVisible(true);
    panelFrame.repaint();
 }

 //Requriement | Edit action performed
 public void reqEdit_actionPerformed(ActionEvent e) {
   //create the panel to hold the tabbedpages
   PanelFrame panelFrame = new PanelFrame("Requirement Display -- Edit Requirement");
   //create the dialog box to prompt user for requirement ID
   ReqNameDialog reqNameD = new ReqNameDialog(this,"Requirement Name", true);
   reqNameD.setLocationRelativeTo(this);
   reqNameD.setVisible(true);
   //create the requirement contain group for the requirement that will be added
   ReqViews reqView = new ReqViews();
   //if string is valid ID store ID value in reqView
   reqView = reqNameD.getValidatedReq();
```

```
    if (reqView != null) {  //If requirement exists display
      //Center the window
      //load all stored info from tables to reqView
      reqView.retrieveAll();
      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = panelFrame.getSize();
      if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
      if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width;
      panelFrame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height + 45) / 2);
      panelFrame.getContentPane().add(new GetReqTabbedPages(panelFrame,true,reqView),
                        BorderLayout.CENTER);
      panelFrame.setVisible(true);
      panelFrame.repaint();
    }
  }

  //Requriement | Display action performed
  public void reqDisplay_actionPerformed(ActionEvent e) {
    //create the panel to hold the tabbedpages
    PanelFrame panelFrame = new PanelFrame("Requirement Display -- DISPLAY ONLY NO
CHANGES");
    //create the dialog box to prompt user for requirement ID
    ReqNameDialog reqNameD = new ReqNameDialog(this,"Requirement Name", true);
    reqNameD.setLocationRelativeTo(this);
    reqNameD.setVisible(true);
    //create the requirement contain group for the requirement that will be added
    ReqViews reqView = new ReqViews();
    //if string is valid ID store ID value in reqView
    reqView = reqNameD.getValidatedReq();
    if (reqView != null) {
      //Center the window
      //Load all stored info from tables to reqView
      reqView.retrieveAll();
      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = panelFrame.getSize();
      if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
      if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width;
      panelFrame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height + 45) / 2);
      panelFrame.getContentPane().add(new GetReqTabbedPages(panelFrame,true,reqView),
                        BorderLayout.CENTER);
      panelFrame.setVisible(true);
      panelFrame.repaint();
    }
  }
  //Overridden so we can exit on System Close
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
```

```
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
      fileExit_actionPerformed(null);
    }
  }
}
```

```
//Title:       PanelFrame
//Version:
//Copyright:   Copyright (c) 1999
//Author:      Leeha Herrrera
//Company:     Dept Computer Science
//Description: Creates a basic frame for which the TabbedPages will be loaded
//                 into

package tool;

import java.awt.*;
import javax.swing.JFrame;

public class PanelFrame extends JFrame {

//  BorderLayout borderLayout1 = new BorderLayout();
  public PanelFrame(String s) {
   enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit(s);
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  //Component initialization
  private void jbInit(String s) throws Exception  {
  // this.getContentPane().setLayout(borderLayout1);
   this.setSize(new Dimension(500, 500));
   this.setTitle(s);
  }
}
```

```
//Title:  ReqNameDialog
//Description:  Creates a dialog box which is used to prompt a user for a
//              requirement ID.  If the ID is valid it will close and pass the
//              value on

package  tool;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.beans.*; //Property change stuff

public class ReqNameDialog extends JDialog {
  private String typedText = null;
  private JOptionPane optionPane;
  private ReqViews req = new ReqViews();


//Returns Requirement that was requested in dialong box
  public Requirement getValidatedReq() {
    return req;
  }

  public ReqNameDialog(Frame frame, String title, boolean modal) {
    super(frame, title, modal);
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }

    pack();
  }

  public ReqNameDialog(Frame frame, String title) {
    this(frame, title, false);
  }

  public ReqNameDialog(Frame frame) {
    this(frame, "", false);
  }

  private void jbInit() throws Exception {

    final String msgString1 = "Enter Requirement Name (used in RequisitePro)";
    final JTextField textField = new JTextField(10);
    Object[] array = {msgString1, textField};

    final String btnString1 = "Enter";
    final String btnString2 = "Cancel";
    Object[] options = {btnString1, btnString2};
```

```
    optionPane = new JOptionPane(array,
                        JOptionPane.QUESTION_MESSAGE,
                        JOptionPane.YES_NO_OPTION,
                        null,
                        options,
                        options[0]);
    setContentPane(optionPane);
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent we) {
     /*
      * Instead of directly closing the window,
      * we're going to change the JOptionPane's
      * value property.
      */
      optionPane.setValue(new Integer(
                    JOptionPane.CLOSED_OPTION));
      }
     });

    textField.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
      optionPane.setValue(btnString1);
     }
    });

    optionPane.addPropertyChangeListener(new PropertyChangeListener() {
         public void propertyChange(PropertyChangeEvent e) {
            String prop = e.getPropertyName();

            if (isVisible()
             && (e.getSource() == optionPane)
             && (prop.equals(JOptionPane.VALUE_PROPERTY) ||
               prop.equals(JOptionPane.INPUT_VALUE_PROPERTY))) {
              Object value = optionPane.getValue();

              if (value == JOptionPane.UNINITIALIZED_VALUE) {
                //ignore reset
                return;
              }

              // Reset the JOptionPane's value.
              // If you don't do this, then if the user
              // presses the same button next time, no
              // property change event will be fired.
              optionPane.setValue(
                    JOptionPane.UNINITIALIZED_VALUE);

              if (value.equals(btnString1)) {
                  typedText = textField.getText();
                String ucText = typedText.toLowerCase();
/*Search for Requriement ID*/
                boolean found = reqView.req.searchKey(ucText);
```

```
                if (found) {
                    // we're done; dismiss the dialog
                    setVisible(false);
                } else {
                    // text was invalid
                    textField.selectAll();
                    JOptionPane.showMessageDialog(
                                ReqNameDialog.this,
                                "Sorry, \"" + typedText + "\" "
                                + "is not a valid Requirement ID.\n",
                                 "Please enter requirement ID.",
                                JOptionPane.ERROR_MESSAGE);
                     typedText = null;
                }
            } else { // user closed dialog or clicked cancel
                typedText = null;
                setVisible(false);
            }
        }
    }
});

}
}
```

```
//Title:  GetReqTabbedPages
//Description:  Creates a new tabbedPages which is used to display the views
//          of a requirement.  If the requirement is new if creates blank
//          text fields, if it is old it creates uneditable field for the ID

package  tool;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class GetReqTabbedPages extends JPanel {

  JTabbedPane jTabbedPane1 = new JTabbedPane();
  JPanel panel1 = new JPanel();
  JButton ok = new JButton();
  JButton cancel = new JButton();

  //Strings used as Titles for each Panel
  String reqTitle ="Requirement Description";
  String useTitle ="Use Case View";
  String designTitle ="Design View";
  String processTitle ="Process View";
  String compTitle ="Component View";
  String deployTitle ="Deployment View";
  String revTitle = "Revision View" ;

  //creates panels for each view
  GetReqInfoPanel reqPanel;
  UseCasePanel usePanel;
  DesignPanel designPanel;
  ProcessPanel processPanel;
  ComponentPanel compPanel;
  DeployPanel deployPanel;
  RevisionPanel revPanel;

  JPanel jPanel4 = new JPanel();
  GridBagLayout gridBagLayout1 = new GridBagLayout();
  GridLayout gridLayout1 = new GridLayout();

//  private Requirement req = new Requirement();

  public GetReqTabbedPages(JFrame frame, boolean old, ReqViews reqView) {
    try {
      jbInit(frame, old, reqViews);

    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
private void jbInit(final JFrame frame, boolean old, final Requirement req) throws Exception {
  //add Ok button
  ok.setText("Submit");
  ok.addActionListener(new ActionListener () {
    public void actionPerformed (ActionEvent e) {
      ok_actionPerformed(e, frame, req);
     }
    });
  //add cancel button
  cancel.setText("Cancel");
  cancel.addActionListener(new ActionListener () {
    public void actionPerformed (ActionEvent e) {
      cancel_actionPerformed(e, frame);
    }
   });
  //set grid for frame
  gridLayout1.setHgap(4);
  jPanel4.setLayout(gridLayout1);
  this.setLayout(gridBagLayout1);
  this.add(jPanel4, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
        ,GridBagConstraints.EAST, GridBagConstraints.NONE, new Insets(8, 0, 8, 8), 0, 0));
  jPanel4.add(ok, null);
  jPanel4.add(cancel, null);
  this.add(jTabbedPane1, new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0
        ,GridBagConstraints.CENTER, GridBagConstraints.BOTH, new Insets(8, 8, 0, 8), 0, 0));
  //init tabbed panels
  reqPanel = new GetReqInfoPanel(old,reqView);
  usePanel = new UseCasePanel(old,reqView);
  designPanel = new DesignPanel(old,reqView);
  processPanel = new ProcessPanel(old,reqView);
  compPanel = new ComponentPanel(old,reqView);
  deployPanel = new DeployPanel(old,reqView);
  revPanel = new RevisionPanel(old,reqView);

  //add tabbed panels to frame
  jTabbedPane1.add(reqPanel, reqTitle);
  jTabbedPane1.add(usePanel, useTitle);
  jTabbedPane1.add(designPanel, designTitle);
  jTabbedPane1.add(processPanel, processTitle);
  jTabbedPane1.add(compPanel, compTitle);
  jTabbedPane1.add(deployPanel, deployTitle);
  jTabbedPane1.add(revPanel, revTitle);
  this.setVisible(true);
}

/* When the user selects OK store all information and close the Frame*/
void ok_actionPerformed(ActionEvent e, final JFrame frame, ReqViews reqView) {
  req.addAll(reqPanel.getKey(), reqPanel.getFields(), usePanel.getFields(),
        designPanel.getFields(), processPanel.getFields(),
        compPanel.getFields(), deployPanel.getFields(), revPanel.getFields());
  frame.setVisible(false);
  frame.dispose();
}
```

```
  void cancel_actionPerformed(ActionEvent e, final JFrame frame) {
    frame.setVisible(false);
    frame.dispose();
  }
}
```

```
//Title:       GetReqInfoPanel
//Version:
//Copyright:   Copyright (c) 1999
//Author:      Leeha Herrrera
//Company:     Dept Computer Science
//Description: Creates the individual panels that are called by GetReqTabbedPages

package tool;

import java.awt.*;
import java.awt.event.*;
import javax.swing.JPanel;
import javax.swing.*;

public class GetReqInfoPanel extends JPanel {
  GridBagLayout gridbag = new GridBagLayout();
  GridBagConstraints c = new GridBagConstraints();

  JButton reset = new JButton("RESET");
//name of fields to be displayed
 private String[] labels = {"ID:", "Name:", "Description:",  "RequisitePro File:", "Owner:", "Priority:",
                            "Status:" };
 //labels used to diplay the text name of field.
 private JLabel idLabel = new JLabel();

//text fields user enters information into
  private JTextField idField = new JTextField(16);
  private JTextField nameField = new JTextField(32);
  private JTextField descripField = new JTextField(64);
  private JTextField fileField = new JTextField(64);
  private JTextField ownerField = new JTextField(16);
  private JTextField priorityField = new JTextField(8);
  private JTextField statusField = new JTextField(16);

  private static int numFields = 7;

  JLabel label;
  public GetReqInfoPanel(boolean old, final ReqViews reqView) {
   try {
    jbInit(old, reqView);
   }
   catch(Exception ex) {
    ex.printStackTrace();
   }
  }

  private void jbInit(boolean old, final ReqViews reqView) throws Exception {
   this.setLayout(gridbag);
   c.weightx = 0.5;
   c.insets = new Insets(7,20,0,20);

   //ADD LABELS
   c.anchor = GridBagConstraints.WEST;
```

```
for (int i=0; i<numFields; i++) {
  label = new JLabel(labels[i]);
  addItem(label,0,i,0);
 }
 //ADD TEXTFIELDS
  if(old) {   //should Id field be editable--if old then no
   idLabel = new JLabel("test123");
   addItem(idLabel,1,0,0);
  }
  else {
   addItem(idField,1,0,75);
  }

  addItem(nameField,1,1,150);
  addItem(descripField,1,2,250);
  addItem(fileField,1,3,250);
  addItem(ownerField,1,4,150);
  addItem(priorityField,1,5,75);
  addItem(statusField,1,6,75);

  //ADD RESET BUTTON
  c.anchor = GridBagConstraints.SOUTHEAST;
  addItem(reset,1,numFields,0);

  reset.addActionListener(new ActionListener () {
   public void actionPerformed (ActionEvent e) {

      setId(null);
      setName(null);
      setDescrip(null);
      setFile(null);
      setOwner(null);
    }
   });


 this.setVisible(true);

 }

//returns all fields in a string[]
 public String[] getFields() {
  String[] values = new String[7];
 }


 public String getId() {
  String value;
  value = idField.getText();
  return value;
 }

 public void setId(String value) {
```

```
 idField.setText(value);
}

public String getName() {
 String value;
 value = nameField.getText();
 return value;
}

public void setName(String value) {
 nameField.setText(value);
}
public String getDescrip() {
 String value;
 value = descripField.getText();
 return value;
}

public void setDescrip(String value) {
 descripField.setText(value);
}
public String getFile() {
 String value;
 value = fileField.getText();
 return value;
}

public void setFile(String value) {
 fileField.setText(value);
}

public String getOwner() {
 String value;
 value = ownerField.getText();
 return value;
}

public void setOwner(String value) {
 ownerField.setText(value);
}

public String getPriority() {
 String value;
 value = priorityField.getText();
 return value;
}

public void setPriority(String value) {
 priorityField.setText(value);
}

public String getStatus() {
 String value;
```

```
    value = statusField.getText();
    return value;
  }


  public void setStatus(String value) {
    statusField.setText(value);
  }

  private void addItem (JComponent f, int x, int y, int padx) {
    c.gridx = x;
    c.gridy = y;
    c.ipadx = padx;
    gridbag.setConstraints(f, c);
    this.add(f);
  }

}
```

```
//Title:  ReqViews
//Description: A class that calls the Views class to create each
//          of the 7 views implemented by the tool.
//          The specific information for each view and its corresponding database
//          table is contained in private strings.

package tool;
import java.sql.*;

public class ReqViews {
  //create each of the 7 views
  public View req;
  public View usecase;
  public View design;
  public View process;
  public View comp;
  public View deploy;
  public View revision;

  /*
  Initiates each view with the appropriate information
  */
  public ReqViews() {
    req = new View(reqTable, reqKeyCol, reqCols);
    usecase = new View(useTable, useKeyCol, useCols);
    design = new View(desTable, desKeyCol, desCols);
    process = new View(procTable, procKeyCol, procCols);
    comp = new View(compTable, compKeyCol, compCols);
    deploy = new View(depTable, depKeyCol, depCols);
    revision = new View(revTable, revKeyCol, revCols);

  }

  /*Given a string that represents the requirement ID this method will search
  for that requirement in the database.  If found it will retrieve all available
  information and store it in the proper views. */
  public boolean searchAll(String key) {
    boolean found = false;
    found = this.req.searchKey(key);
    this.usecase.searchKey(key);
    this.design.searchKey(key);
    this.process.searchKey(key);
    this.comp.searchKey(key);
    this.deploy.searchKey(key);
    this.revision.searchKey(key);
    this.key = key;
    return found;
  }

  /*Given a string that represents the requirement ID this method will search
  for that requirement in the database.  If found it will retrieve all available
  information and store it in the proper views. */
  public boolean retrieveAll() {
```

```
 boolean found = false;
 found = this.req.searchKey(this.key);
 this.usecase.searchKey(this.key);
 this.design.searchKey(this.key);
 this.process.searchKey(this.key);
 this.comp.searchKey(this.key);
 this.deploy.searchKey(this.key);
 this.revision.searchKey(this.key);
 return found;
}

/*Given the information for each view in a string array this method will call
the appropriate views method to store the inforation in the database*/
public void addAll (String key, String[] u, String[] ds, String[] p ,
        String[] c, String[] dp, String[] r,) { }


private String key;

//Table names and Columns
private String reqTable = "Requirement_View";
private String reqKeyCol ="Id";
private String[] reqCols ={"Name", "Description", "File_Name", "Owner", "Priority", "Status"};

private String useTable = "Usecase_View";
private String useKeyCol ="Id";
private String[] useCols ={"Use_Diagram", "Sequence_Diagram1", "Sequence_Diagram2",
        "Collaboration_Diagram2", "Collaboration_Diagram1", "File_Name"};

private String desTable = "Design_View";
private String desKeyCol ="Id";
private String[] desCols ={"Package", "Class_Diagram1", "Class_Diagram2",
        "Object_Diagram", "State_Diagram", "File_Name"};

private String procTable = "Process_View";
private String procKeyCol ="Id";
private String[] procCols ={"Package", "Class_Diagram1", "Class_Diagram2",
        "Object_Diagram", "State_Diagram", "File_Name"};

private String compTable = "Component_View";
private String compKeyCol ="Id";
private String[] compCols ={"Package", "Component_Diagram1", "Componet_Diagram2",
        "File_Name"};

private String depTable = "Deployment_View";
private String depKeyCol ="Id";
private String[] depCols ={"Package", "Deployment_Diagram1", "Deployment_Diagram2",
        "File_Name"};

private String revTable = "Revision_View";
private String revKeyCol ="Id";
private String[] revCols ={"Revisied Id", "Authorized_By", "Status",
        "Notes"};
```

```
    private String docTable = "Documents";
    private String docKeyCol ="Id";
    private String[] docCols ={"Document1", "Document2", "Document3",
                "Document4"};

}
```

```
//Title: View
//Description: A class contains variables to be added to a database table and retrieved from

package tool;

import java.sql.*;

public class View {

 /*Creates a view with the stores the given strings to asscess the associated
   database table*/
 public View(String table, String keyCol, String[] cols) {
   this.table = table;
   this.num = cols.length;
   this.fields = new String[num];
   this.keyCol = keyCol;
   this.cols = new String[num];
   for (int i=0; i < num; i++) {
     this.cols[i] = cols[i];
   }
 }

 public View(String table, String keyCol, String[] cols, String key,
         String[] field) {
   this.table = table;
   this.num = cols.length;
   this.keyCol = keyCol;
   this.key = key;
   this.fields = new String[this.num];
   this.cols = new String[this.num];
   for (int i=0; i < num; i++) {
     this.cols[i] = cols[i];
     if (i >= field.length)
       this.fields[i] = "";
     else this.fields[i] = field[i];
     System.out.println(fields[i]);
   }
 }

 /*returns a string array containing the names of all the fields*/
 public String[] getFields() {
   return this.fields;
 }

 /* returns a string array containing all the column names of the table*/
 public String[] getCols() {
   return this.cols;
 }

 /*returns the key column name which for all views is the requirement ID */
 public String getKeyCol() {
   return this.keyCol;
 }
```

```
/* returns the key value which for all views is the requirement ID*/
public String getKey() {
 return this.key;
}

/*opens a connection to the database and adds the information already
 stored in the class*/
public void addViewDb() {
 DbActions job = new DbActions();
 Connection con = job.getConnection();
 String insert = "INSERT INTO " + this.table + " VALUES (";
 int i;
 insert +=" '" + this.key + "',";
 System.out.println(insert);
 for (i=0; i < num-1; i++) {
  insert += " '" + this.fields[i] + "',";
 }
 insert += " '" + this.fields[i] + "')";
 System.out.println(insert);

 try{
  job.update(con,insert);
  con.close();
 }
 catch(Exception excep){
  System.out.println("Error " + excep);
 }
}


/*opens a connection to the database and adds the information set to the
 method*/
public void addViewDb(String k, String[] field) {
 DbActions job = new DbActions();
 Connection con = job.getConnection();
 //copy values from field to this.fields, if value does not exist assign it ""
 for (int i=0; i < num; i++) {
   if (i >= field.length)
    this.fields[i] = "";
  else this.fields[i] = field[i];
 }
 this.key = k;

 //Create String to use for SQL command
 String insert = "INSERT INTO " + this.table + " VALUES (";
 insert +=" '" + this.key + "',";

 int i;
 for (i=0; i < num-1; i++) {
   insert += " '" + this.fields[i] + "',";
 }
 insert += " '" + this.fields[i] + "')";
```

```
  System.out.println(insert);
  try{
   job.update(con,insert);
   con.close();
  }
  catch(Exception excep){
   System.out.println("Error " + excep);
  }
 }

/*opens a connection to the database and updates the values to the given string[]*/
 public void updateViewDb(String[] f) {
  DbActions job = new DbActions();
  Connection con = job.getConnection();
  String update1 = "UPDATE " + this.table + " SET ";
  String update2 = " WHERE " + this.keyCol + " = '" + this.key + "'";
  String changes = "null";
  int c = 0;
  int s;
  for (int i=0; i<num; i++) {
   if((s = this.fields[i].compareTo(f[i])) != 0) {
    this.fields[i] = f[i];
    if (c == 0) {
     changes = this.cols[i] + " = ";
     changes += "'" + this.fields[i] + "'";
     c++;
    }
    else  {
     changes += ", " + this.cols[i] + " = ";
     changes += "'" + this.fields[i] + "'";
     c++;
    }
   }
  }
  //Send Command to DB
  try{
   if (!(changes.equals("null"))){
    System.out.println(update1 + changes + update2);
    job.update(con, update1 + changes + update2);
   }
   con.close();
  }

  catch(Exception excep){
   System.out.println("Error " + excep);
  }
 }

/*opens a connection to the database and executes a query for the given ID
 if found it will store appropriate values*/
 public boolean searchKey(String key) {
  boolean found = false;
  DbActions job = new DbActions();
```

```
Connection con = job.getConnection();
String q = "SELECT * from " + this.table + " WHERE " + this.keyCol + " = '" +
        key + "'";

try{
  Statement stmt = con.createStatement();
  System.out.println(q);
  ResultSet rs = stmt.executeQuery(q);
  rs.next();
  int c;
  System.out.println(rs.getString(this.table));
  System.out.println(rs.getString(keyCol));
  if ((c = key.compareTo(rs.getString(keyCol))) == 0) { //result set id = key
    this.key = key;
    for (int i = 1; i< num+1; i++) {
      this.fields[i-1] = rs.getString(i+1);
      System.out.println(rs.getString(i));
    }
    found = true;
    con.close();
  }
 }
 catch(Exception excep){
  System.out.println("Error " + excep);
  System.out.println("Error Requirement Name not found");
 }
 return found;
}



private String table;
private String keyCol;
private String key;
private String[] fields;
private String[] cols;
private int num;
}
```