ClubUML

CSYE7945 Fall 2013

Final Report

Contributors:

Kenneth Canaan, Thomas Cashavelly, Afshin Chaharmahalian, Xiang Cheng, Lauren DiCristofaro, Abdulkarim Egal, Ameya Joshi, Jeremy Lerch, Bing “Lee” Li, Chao Li, Vishal Patel, Sahil Patil, Yang Zheng

April 2014

Contents

Abstract 6

Getting Started 6

A. Quickstart 6

1. Required Software 6

2. Optional Software 6

3. Database 7

4. Code Setup 7

B. Tools 7

1. Java JRE 1.7 8

2. MYSQL community server 8

3. MYSQL workbench 9

5. Tomcat 7.0 10

6. Eclipse 12

7. StarUML 12

8. Github 13

9. Smartgit 13

10. GraphViz 14

C. Understanding the Project 15

Standards and Processes 16

A. Development Tools 16

1. JazzHub 16

2. Eclipse 18

3. MySQL 18

4. GitHub 19

5. SmartGit 19

6. Selenium 20

B. Development Processes 21

1. Overview 21

2. Work Breakdown Structure 22

3. Program Structure 22

4. Configuration Management 23

5. Summary 23

C. Best Practices 23

Full System Description 25

A. Navigation 25

B. Dependencies 29

C. Classes 30

D. Database 30

Spring 2014 Contributions 32

A. Rationale Management 32

1. Use Case RM1 32

2. Use Case RM2 35

B. Validation 36

1. Overview 36

2. Areas with need for Validation 37

C. Navigation 40

1. Overview 40

2. GUI Layout 40

3. Ported Tar Bar/Header 40

4. External Library Migration 41

5. Tab Modernization Research 41

6. Tab Modernization Prototype 44

D. Use Case Diagram Management 45

1. Overview 45

2. Design 45

3. Testing 49

E. Resolved Bugs 50

F. Database Updates 50

Maintenance Recommendations 52

A. Regression Testing 52

B. Partially Completed Functionality 54

1. Overview 54

2. Use Case Diagram 54

3. Rationale Management 55

4. Navigation 56

C. Suggested Development 57

D. Caveats and Pitfalls 59

1. Navigation Team 59

2. Use Case Diagram Team 59

3. Rationale Management Team 60

4. Validation Team 60

5. Bug Fix Team 60

Appendices 61

Appendix A: Lessons Learned Documents 61

Appendix B: Rationale Management Documents 61

Appendix C: Validation Documents 62

Appendix D: Navigation Documents 62

Appendix E: Use Case Diagram Management Documents 62

Appendix F: Bug Fix Documents 62

Appendix G: Database scripts 62

# Abstract

(Xiang)

ClubUML is has been an ongoing graduate school project by the CSYE7945 Software Engineering Project class under the order of Northeastern University Graduate School of Engineering. The effort of the Spring 2014 team is to provide sustainment to the system by providing upgrades and modernization to ClubUML.

The purpose of the project is to develop an application, which provides a way to compare different versions of UML diagrams in order to decide on a final version of the diagram. The initial version of the ClubUML software was developed in Fall 2012 and extended by Spring2013 and Fall 2013. The Fall 2013 version of the software supports uploading UML 2.0 class or sequence diagrams and the ability to give suggested promotion when comparing two diagrams based on some built-in metrics, among other enhancements. So the work done by this semester, which is shown in the document and appendices to follow consists of the functional extension and maintenance tasks. The functional extensions include Parsing and Evaluation of use case diagrams and Rationale Management functionality. Maintenance tasks included Navigation Redesign, Bug Fixes, and Validation.

In this report, you will see the detailed procedures to get started with this project, the standards and processes involved in the development, and the full system description. You will also find a more detailed description of the changes made during spring 2014, and there are maintenance recommendations for future development in the last section of this document.

\*Note\*: For links to documents on github, you can download the document by clicking the “Raw” button, or you can use the path elements to find it in your local repository.

# Getting Started

## A. Quickstart

(Lauren)

### 1. Required Software

-[Java 1.7](http://www.oracle.com/technetwork/java/javase/downloads/index.html) for compilation

-[Apache Tomcat 7.0.41](http://archive.apache.org/dist/tomcat/tomcat-7/) or 7.0.0 or 7.0.1,

-[MySQL 5.6.13 Community Server](http://dev.mysql.com/downloads/mysql/)

-[Eclipse](http://www.eclipse.org/downloads/moreinfo/jee.php) for Java EE

-[Git](http://git-scm.com/downloads)

-[Graphviz](http://www.graphviz.org/Download..php)

### 2. Optional Software

* .Ecore files can be generated using [Eclipse Modeling Tool](http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplerr) .
* Papyrus diagrams can be exported from the [Papyrus UML tool](http://www.eclipse.org/papyrus/downloads/index.php) .
* Use case diagrams can be exported as xml using [StarUML 5.0](http://staruml.sourceforge.net/en/download.php). (Windows only)
* Useful Database GUI [MySQL Workbench](http://dev.mysql.com/downloads/tools/workbench/)
* Git GUI client [Smartgit](http://www.syntevo.com/smartgithg/)

### 3. Database

* After installing MySQL, the default configuration will likely match:
  + host: localhost
  + port: 3306
  + user: root
  + password: “”
* Run the database server (“mysql.server start” on the command line or start from MySQL workbench)
* Run the script clubuml2014spring.sql as found in Appendix G
* Change the database password using the following:
  + “UPDATE mysql.user SET Password=PASSWORD('1234')

WHERE User='root' AND Host='localhost'; FLUSH PRIVILEGES;”

### 4. Code Setup

* (Optional) One person may fork the repository to set up a new repository and URL
  + Sign in to github and go to <https://github.com/lmd59/ClubUMLSpring2014>
  + Click “Fork repository” in the top right and choose location
* Use your git client of choice to clone the repository using clone url:<https://github.com/lmd59/ClubUMLSpring2014.git>
  + Using Smartgit: Go to Project>Clone>Choose the above url as the “Repository URL”>Continue>Continue>Choose local directory>Finish
* In eclipse, import the project using general->existing projects-> select ClubUML folder
* Add new server: File-> new-> server-> server->apache-> Tomcat 7->browse to apache tomcat location
* Click next, Move ClubUML from Available group to Configured, Finish
* Edit classpath: In the bottom tabs in eclipse, choose the server tab.
  + Double click on the Tomcat 7 server.
  + Click open launch configuration link>Classpath>User Entries>Add JARs>Choose file at:

ClubUMLSpring2013>WebContent>lib>dom4j-2.0.0-ALPHA-2.jar

* + Click Okay>Apply>Okay
* Click on ClubUML project. Go to run->run as->run on server, Choose Tomcat v7.0
* Go to<http://localhost:8080/ClubUMLSpring2013/> to view the application

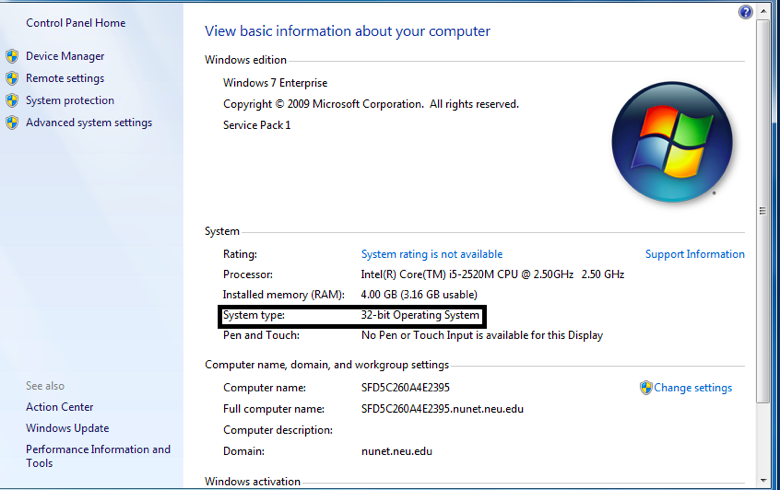
## B. Tools

(Sahil)

Here is list of tools used for ClubUML spring semester team for the project.

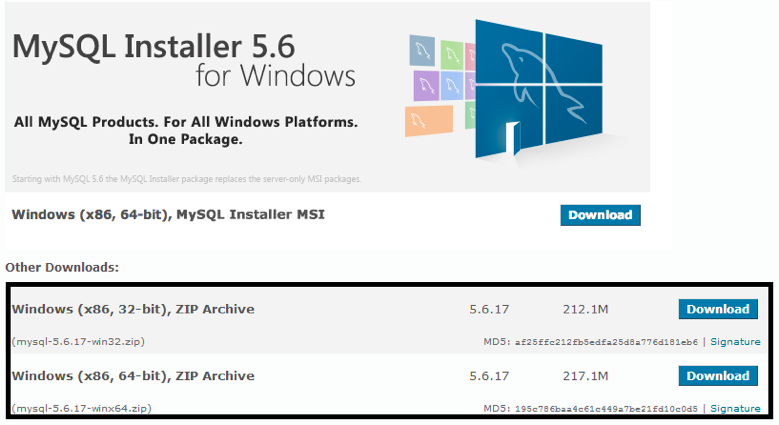
### 1. Java JRE 1.7

1. Go to the Oracle website (<http://www.oracle.com/technetwork/java/javase/downloads/java-se-jre-7-download-432155.html>)
2. Then click on accept license agreement and download the version compatible to your system like windows x64 or windows x32 etc.
3. In order to check your version of windows you can right click on computer icon in your computer and click on properties. For Mac system type, follow the instructions at <http://support.apple.com/kb/ht3696>.



### 2. MYSQL community server

1.     To install MySQL community server, go to (<http://dev.mysql.com/downloads/mysql/>) and then click on one of the installations according to the compatibility of your system.



2.     Once you install the Community Server, it'll take you through a MySQL Server Instance Configuration Wizard to setup a MySQL server. If this is the first time installing MySQL Server then:

-Use Standard Configuration.

-Install As Windows Service

-Set root password: 1234

### 3. MYSQL workbench

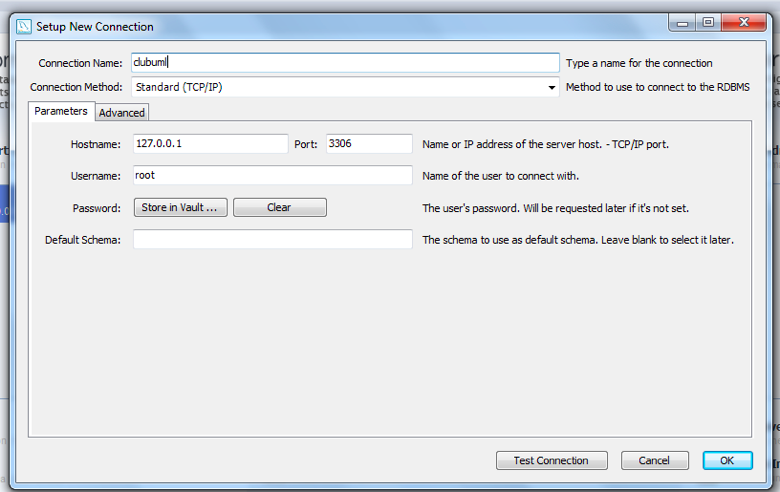
             1. Install MySQL Workbench.

             2. Launch MySQL workbench.

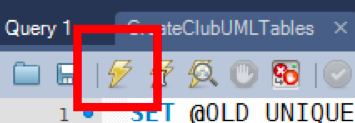
             3. Under the “Open Connection to Start Querying” section, select 'New Connection' at the bottom.



1. Give the new connection a name (ex: clubuml) and press **OK**. Enter ‘1234’ or any password when the password prompt appears. (This password was set in **MySQL Community Server section**)



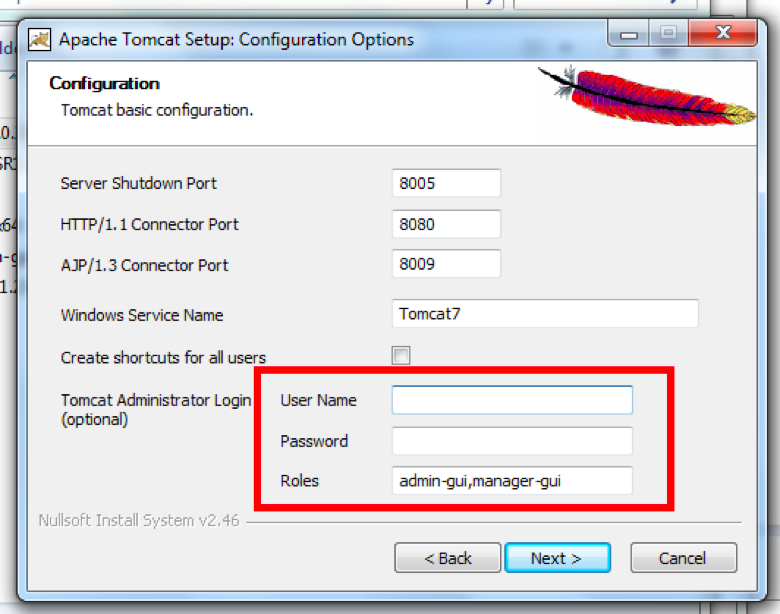
1. Go to **File->Open SQL Script,** open the clubuml2014spring.sql script (<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/clubuml2014spring.sql>), and run the script using the lightning icon.



6.Right-clickon Schema area in the Object Browser and **Refresh All**. Tables should be available under ClubUML.

### 5. Tomcat 7.0

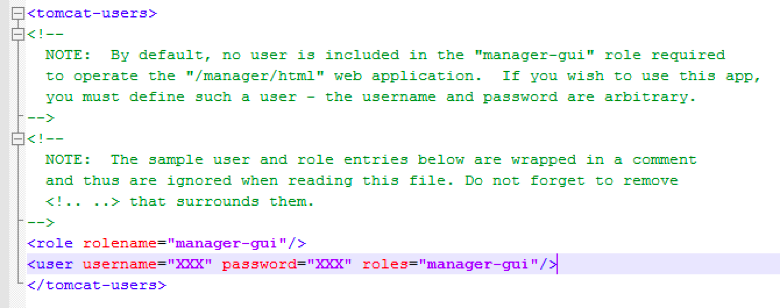
1. Download Tomcat’s 32-bit/64-bit Windows Service Installer or Mac version <http://tomcat.apache.org/download-70.cgi>)
   1. Make sure it is Tomcat 7.0.0 or 7.0.1. Newer versions (in particular 7.0.39) have caused errors – but could be fixed in the future.
2. Installing Tomcat 7.0 should be straightforward. Use default settings and enter the optional Administer user name and password to avoid editing the user access XML file later.



1. If you didn’t follow step 2, then you have to edit the user access XML file to set a “User Name” and “Password” to enter http://localhost:8080/manager/html
   1. **Open “Tomcat\conf\tomcat-users.xml”. Add two lines into the file:**

**<role rolename="manager-gui"/>**

**<user username="XXX" password="XXX" roles="manager-gui"/>**



### 6. Eclipse

(<https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplersr2>)

1. For setting up eclipse go on the above link to download java-ee version.
2. Once the file is downloaded, uncompress it using WinRAR or any other tool and do the following steps:

a . Open Eclipse EE and in the menu bar: **File->Import…**

* 1. Select **General->Existing Projects into Workspace…**
  2. Navigate to your ClubUML project folder cloned using Git and hit **OK**
  3. Select the ClubUML project and hit **OK**.

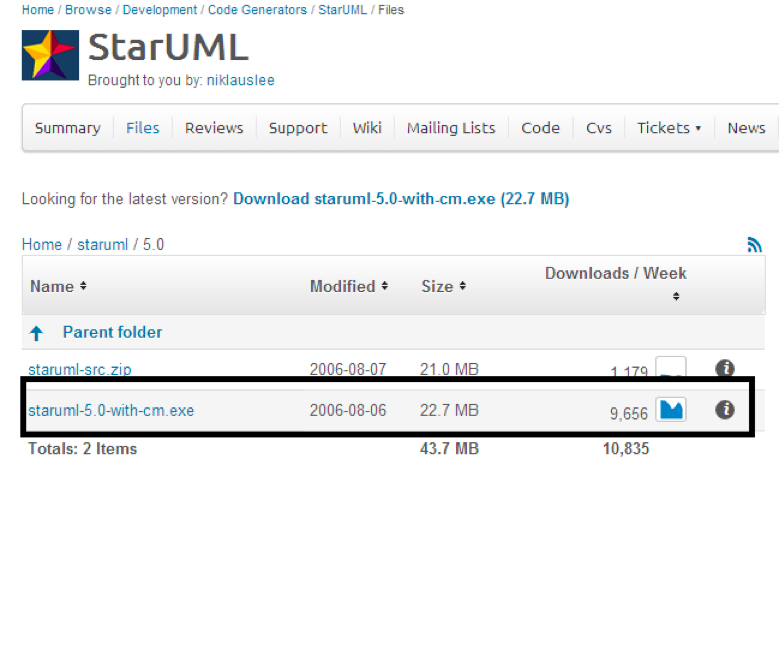
### 7. StarUML

(<http://staruml.sourceforge.net/en/download.php>.)

(Note: this is only available for Windows)

1.     Go on the above link to install StarUML

2.     Go to download page click on the appropriate link to download the tool



3.     After downloading .exe file click on setup and follow the standard procedure for installation.

4.     Then click on the icon to start StarUML. You will get a window like this



5.     In this window, you can go to file and click on the new project to start making class and use case diagrams.

### 8. Github

1. Go to link(<https://github.com/>) to create a github profile. You may also download the github software from here for windows or MAC, but it is not necessary if you use smartgit.

2. If you choose to use github software, after downloading the setup file, click on the file and follow the steps for installations.

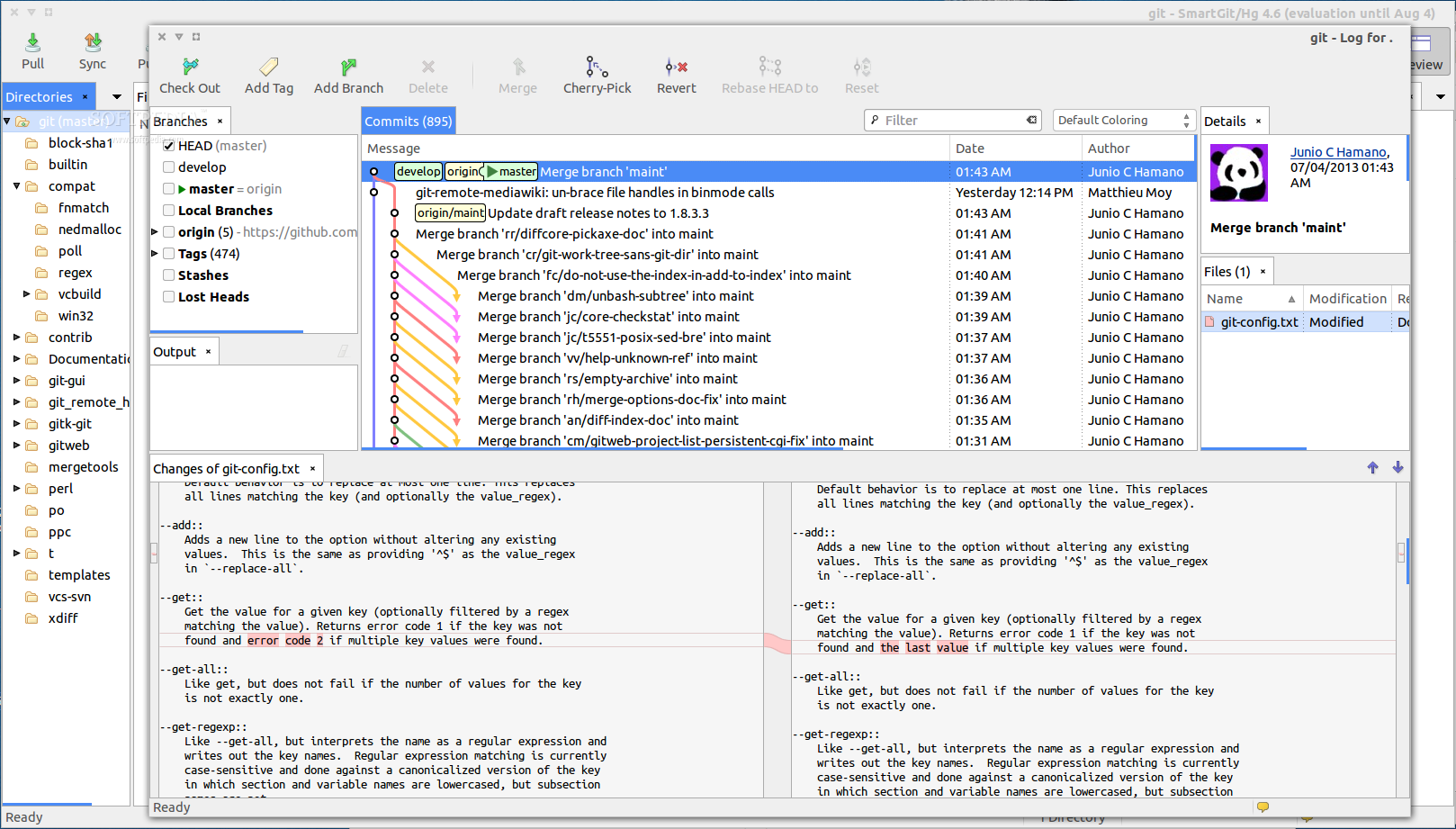
### 9. Smartgit

                1. Go to the website (<http://www.syntevo.com/smartgithg/download>) to install SmartGit.

                2.Choose the download link from the website as follows:



1. Follow the procedure to install smartgit into your system
2. 4. After installation you can use smartgit to clone the repository after which you will be able to see the following screens



### 10. GraphViz

(<http://www.graphviz.org/Download.php>)

1. For Mac, download the appropriate .pkg and follow installation instructions keeping all defaults

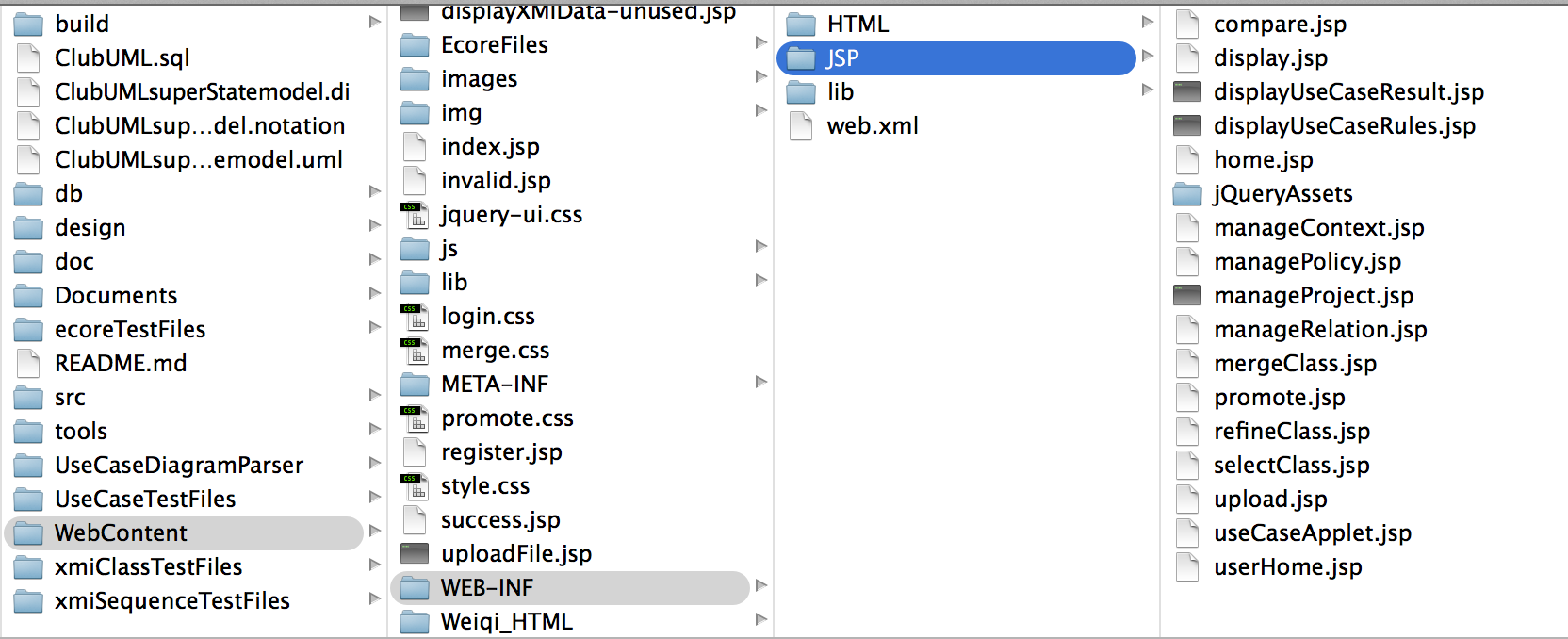
2. For Windows, install GraphViz to 'C:\Graphviz'. (The code will reference this directory when it attempts to generate UML class diagram images.)

3. For Windows, after installing GraphViz, copy ‘cgraph.dll’ to ‘graph.dll’ under C:\Graphviz\bin folder. Don’t remove ‘cgraph.dll’ file. Both files are needed to create a png file.

## C. Understanding the Project

(Ken)

ClubUML is an application developed and maintained by the CSYE7945 Software Engineering Project class. Every semester students are required to add new functionality, update or optimize existing functionality, or simply fix defects that naturally occur through each upgrade cycle of the application.



In order to facilitate understanding on handling the ClubUML code, the key folders needed to manage and update the application are structured as follows:

* **build**
  + **Classes**: contains all ‘.class’ files used in the application. These files will be updated automatically by the compiler.Unless creating new classes or adding functionality to the application, these classes will not change.
* **db**: contains all files and folders updating the database
* **src**: contains ALL the java files in the application
  + **controller**: Main controller folder and contains all the application’s subsystem java files
  + **repository:** java files that store and interact with the database
* **WebContent**: contains all the html, css, and jsp files
  + **WEB-INF**: JSP and HTML folders contains all the files relevant to the  GUI display of the application
* **Documents:** Includes diagrams and relevant documentation from the Spring 2014 and Fall 2013 class.

**Walkthrough Example**

If the development team wants to update the Context functionality, the team would follow these steps:

* If the team is just trying to switch up the layout/design of the Context page they would need to update the manageContext.jsp file (WebContent > WEB-INF > JSP > manageContext.jsp)
  + The manageContext.jsp contains the css, html, and JavaScript code needed to update this
* If updates are required to update functionality the team will have to update the ManageContext.java (src > controller > ManageContext.java).
  + The ManageContext.java calls the ContextDAO.java (src > repository > ContextDAO.java), which updates the database.
* Context values for a new context are taken from the front end and persists it in the database via CreateContext.java (src > controller > CreateContext.java), but this files only extends the HttpServlet
* In summary, the files the team would have to touch in order to update the context functionality would be the ManageContext.java, ContextDAO.java, and manageContext.jsp.

Most development teams will reside within the “src” and “WebContent” folders where most of the applications files reside.

A list of files and its dependencies within the application is provided under Full System Description section ‘B. Dependencies’

# Standards and Processes

## A. Development Tools

(AC)

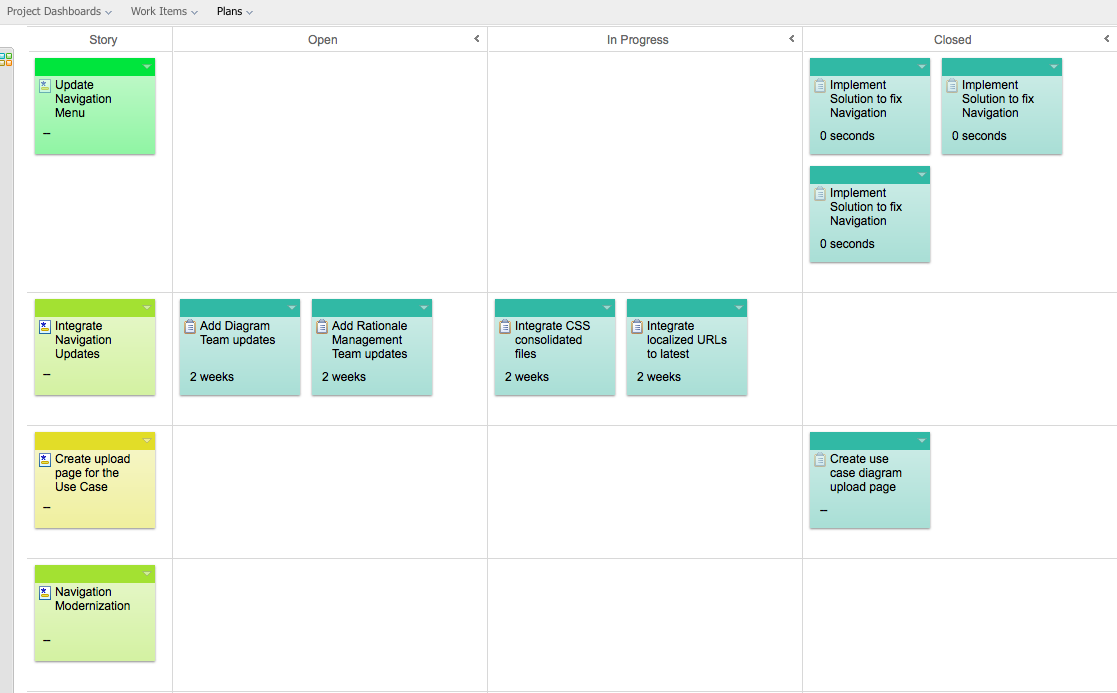
ClubUML’s functionality requires a certain set of tools to support development on multiple components of the application. These components combined with the development process required the use of a suite of tools. This section will highlight how these tools were used in the development of ClubUML.

### 1. JazzHub

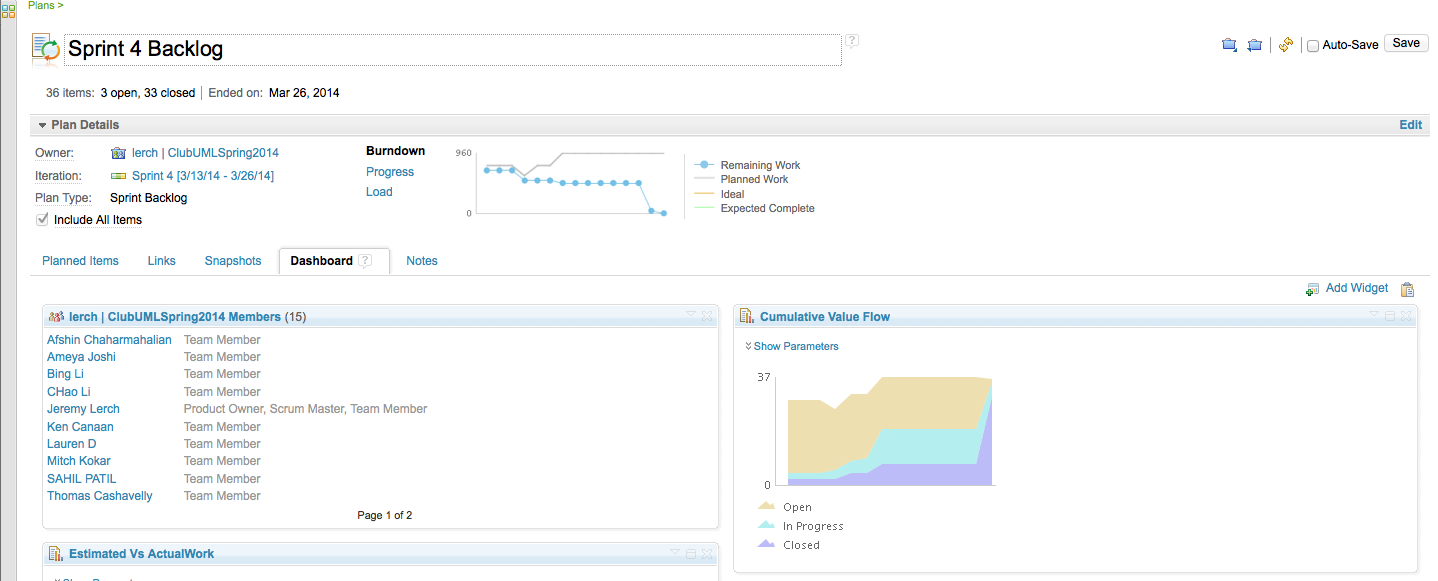
JazzHub was the program management tool in use in the development of the ClubUML application for the Spring 2014 semester. JazzHub is a cloud-based collaborative program tracking management tool produced by IBM. Its primary purpose is to track project progress while utilizing the Scrum development framework. It provides the capability to track the progress of individual work items. A list of work items on JazzHub includes but is not limited to:

* Epics
* User Stories
* Tasks
* Defects
* Retrospectives

These work items directly align with the scrum development process and is why this tool was chosen as the primary program management tool. The Taskboard is an important information radiator employed by the Scrum process. Its purpose is to allow the team to visualize the work that needs to be done, is in progress, and ultimately completed. JazzHub provides a virtual taskboard for tracking purposes. An example of this taskboard can be seen below.



JazzHub also provides tools to report Scrum artifacts. Another information radiator employed by Scrum is the burndown chart used to measure progress within a sprint. JazzHub provides this information in a “dashboard” shown below.

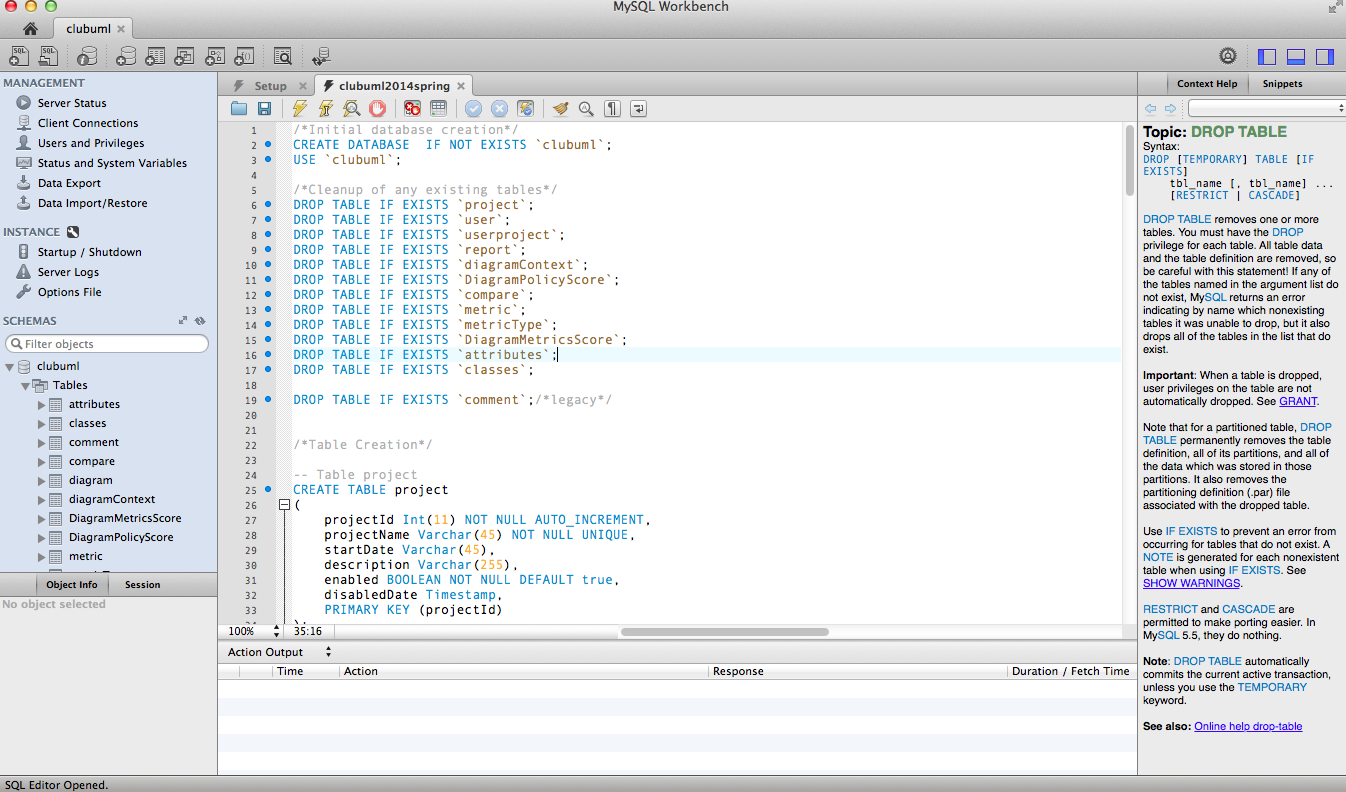


  In summary JazzHub provides the functionality necessary for a team to track its progress utilizing the Scrum development process.

### 2. Eclipse

Eclipse is one of the most commonly used Integrated Development Environments  (IDE) in both academia and industry today. It offers a common development environment among developers and has a substantial support community.  Eclipse is capable of supporting various development languages and complex software engineering environments through its use of plug-ins. This has been the teams primary IDE and has been used by other courses at Northeastern University.

### 3. MySQL

ClubUML requires data storage of various types. Some of these types include user, project, and diagram information. In order to support this a tool is necessary to manage the database tied to the application. MySQL workbench is a popular tool used to manage databases on complex software projects. It provides the ability to establish databases, monitor status, and create schema. An illustration of the working environment in MySQL workbench is included below. 

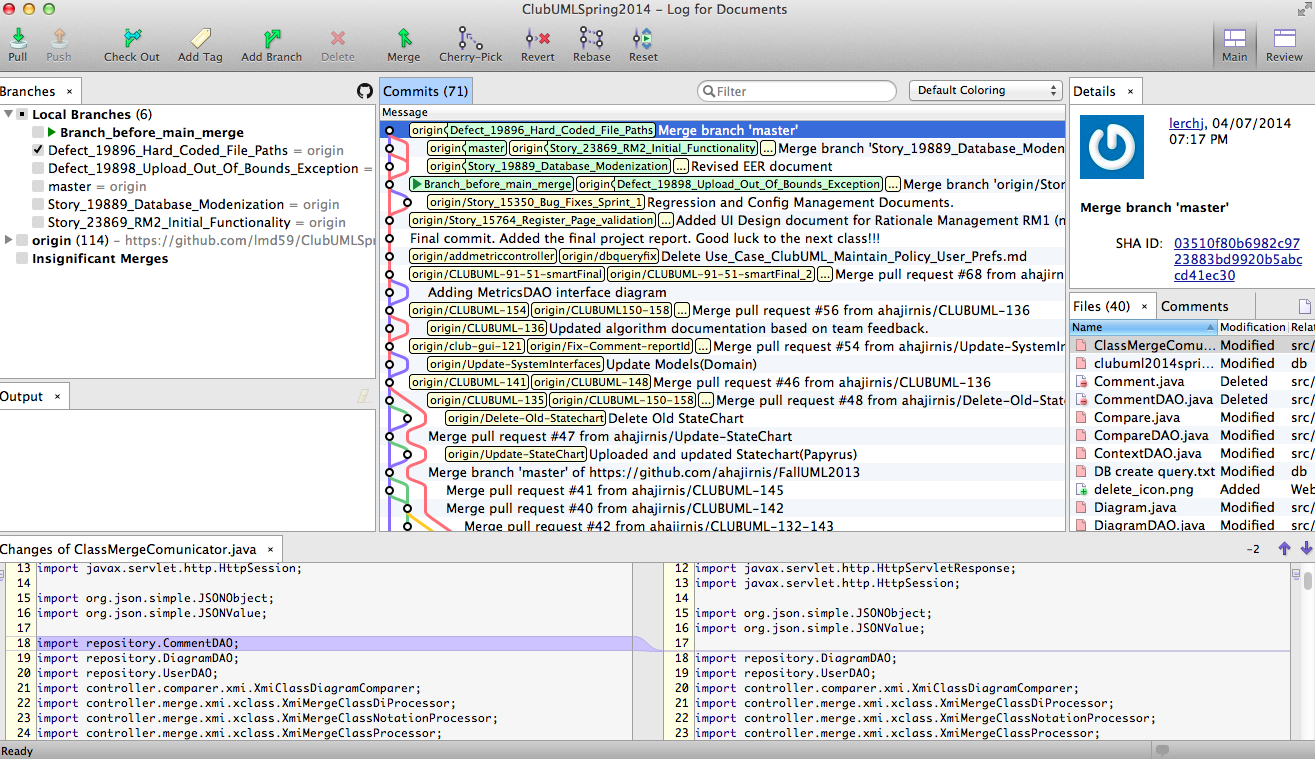
MySQL proved to be a valuable tool in managing the database information for  ClubUML.

### 4. GitHub

GitHub is the web hosting service used for the ClubUML project. GitHub was used by the application’s prior development team and proved to be a viable transition mechanism for starting development in the sprint of 2014. GitHub is free and employs the Git revision control system. This revision control system allows for concurrent development of an application and the ability for multiple users to edit source code and commit changes to a common baseline. GitHub also provides the ability to track committed changes, and branch information through a browser. In order to ease the transition of the project from the prior development team GitHub was used as one of the primary configuration management tools.

### 5. SmartGit

Another primary configuration management tool used in concert with GitHub was SmartGit. SmartGit is a native application that can be used on Windows and Macintosh platforms. SmartGit provides features that include branching, pull requests, commits, and differentiation reporting capabilities. These features allow for developers to work on source code in their own development branches prior to committing to a common software baseline. SmartGit also provides the capability to merge and track changes for development branches and main baselines. An illustration of this feature in SmartGit is included below, and basic instructions for its use in configuration management can be found in [Procedures for Using Smartgit.docx](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Miscellaneous/Procedures%20for%20Using%20Smartgit.docx). (Click “RAW” to download)

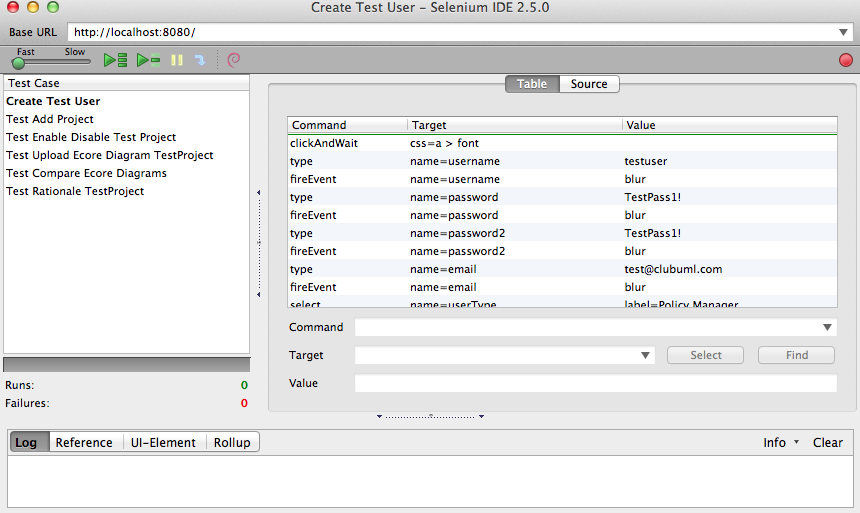


SmartGit proved to be an extremely useful configuration management tool that was easy to use. It is highly recommended that future development teams leverage this tool for managing the ClubUML software baseline.

### 6. Selenium

ClubUML is a web-based application relying on user inputs to exercise various pieces of functionality. Therefore a tool was needed to simulate these user inputs in order to provide a robust testing framework. Selenium is one such framework that allows developers and more importantly users of an application to author tests without having to learn a scripting language. For ClubUML’s testing configuration a browser add-on was used to deploy and use the Selenium IDE. This IDE allowed the user to navigate and record user actions throughout the ClubUML application.

Tests can be part of a suite, and the Spring 2014 development team used this practice. These tests were authored and updated over time to serve as a regression-testing suite prior to integration of source code changes. An illustration of the Selenium IDE can be found below.



It is recommended that future development teams leverage the regression-testing suite provided by the Spring 2014 ClubUML team. It will provide the ability to incorporate regression testing into the development process in a quick and easy fashion.

## B. Development Processes

(AC)

### 1. Overview

The Spring 2014 Team followed an Agile Development approach to support the development process throughout the semester. It is an iterative and incremental approach for development where requirements and solutions are evolved. Agile development promotes dynamic planning during development and frequent delivery of software functionality. Coming into the spring semester ClubUML had a software baseline established. The goal of this development team was to build upon the existing architecture and features that had already existed. The following sections will discuss the work breakdown and organizational structure of the project. An illustration of the scrum process can be found below.

## creen Shot 2014-04-12 at 4.09.28 PM.png

### 2. Work Breakdown Structure

At the onset of the semester the team took a couple of weeks to determine what features would provide value to the ClubUML product. All of the work was allotted into 5 distinct “features”. These features essentially served as buckets of work the team tracked throughout each sprint. These features were designated as epics and tracked respectively using JazzHub. These epics included:

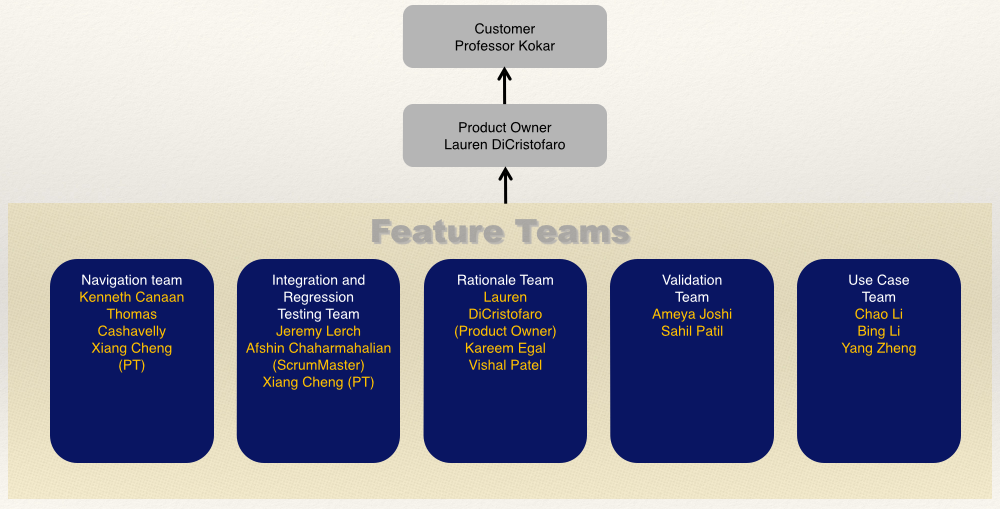
* Navigation Updates
* Rationale Updates
* Validation Updates
* Use Case Diagrams
* Integration and Regression

These epics are expounded upon later in this document.

### 3. Program Structure

The focus of any Agile framework is to ensure working software is delivered to the customer at the end of each sprint iteration. Once the work breakdown structure was established the leadership team worked to group individuals into separate teams to support the features being developed. The leadership team consisted of our customer and product owner. The customer for ClubUML was Professor Mitch Kokar. Lauren DiCristofaro served as the product owner by setting a vision for the project and ensuring stories were completed appropriately. Afshin Chaharmahalian served as the ScrumMaster and ensured scrum practices were being followed.

One of the benefits of scrum is the ability to group individuals of varying disciplines together in order to produce working features of a given product. These features typically encapsulate complete end-to-end functionality in a given product. That includes the user interface, service layer, and database portion of web applications. This proved to be the case for the spring 2014 development effort on a number of teams. For the spring semester there were 4 total development teams, which can be seen, in the organizational diagram below.



### 4. Configuration Management

As feature teams produced ClubUML functionality it became necessary to integrate the source code into the baseline. In order to do this the integration and regression testing team had to establish a method by which all of these teams would be able to commit their code to the software baseline. This process consisted of ensuring their functionality worked as described, the code was peer reviewed utilizing static methods, the code passed the test cases outlined by the development team, and ultimately passed all regression tests authored by the Integration team. These integration efforts highly leveraged the SmartGit and Selenium tools described previously in this document. The full details of this integration procedure can be found in the Configuration Management Checklist located [here](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Integration%20%26%20Regression/Software%20Config%20Management%20Checklist_RevA.pdf), and a layman’s explanation can be found in the repository [here](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Integration%20%26%20Regression/ConfigMgmtChecklistExplanation). (Click “RAW” to download)

### 5. Summary

The program organization and development process served the development teams well as they were able to produce working increments of functionality throughout the course of the semester. After each sprint, teams conducted retrospectives in order to gain insight into what they were doing well with regard to the features being produced. Lessons learned were also communicated and compiled into documents for each respective team.

## C. Best Practices

(AC)

Halfway through the semester each development team began to put together a lessons learned document for the work they had produced. These best practices and lessons learned are summarized below and can be found in the following location: https://github.com/lmd59/ClubUMLSpring2014/tree/master/Documents/2014Spring/LessonsLearned.

* Navigation
  + To have a prior understanding of the Scrum Development Process
  + To have a prior understanding of the JazzHub management tool
  + To have a prior understanding of the ClubUML application before development
* Integration & Regression Testing
  + Utilize static testing methods
  + Provide non “happy” path test cases
  + Utilize continuous integration techniques
* Agile Development Process
  + Establish Leadership Team & Process prior to development
  + Provide Scrum Training for development teams
  + Exercise Early Estimation (Sprint Planning) prior to development
  + Specify a common toolset to be used by the development teams.
  + Utilize better tools for communication in lieu of co-location
* Rationale Management
  + Requirements, design, and testing allow for understanding of new functionalities and their implementation.
  + Reserve time during planning to learn new technologies, understand the codebase, and how good implementations will tie parts of the application together.
  + Design documentation updates should be included as part of the definition for each user story.
  + UMLlab and Visio are better products to use than papyrus for design documentation.
  + Firefox and Chrome provides tools for debugging jsp issues, while eclipse is useful for debugging java application code and local testing.
* Validation
  + JSTL/Javascript implementation techniques
  + Coordinate with team members frequently on tasks
* Use Case Diagrams
  + Use Abstraction and Inheritance to reduce complexity.
  + Keep code D.R.Y. (Don’t repeat yourself)
  + Utilize existing frameworks (Spring MVC, Hibernate)
  + JSON allows UI and Database engineers to work in parallel
  + Define test cases in JUNIT prior to development (Test Driven Development)
  + Communicate technical needs amongst your team members

# Full System Description

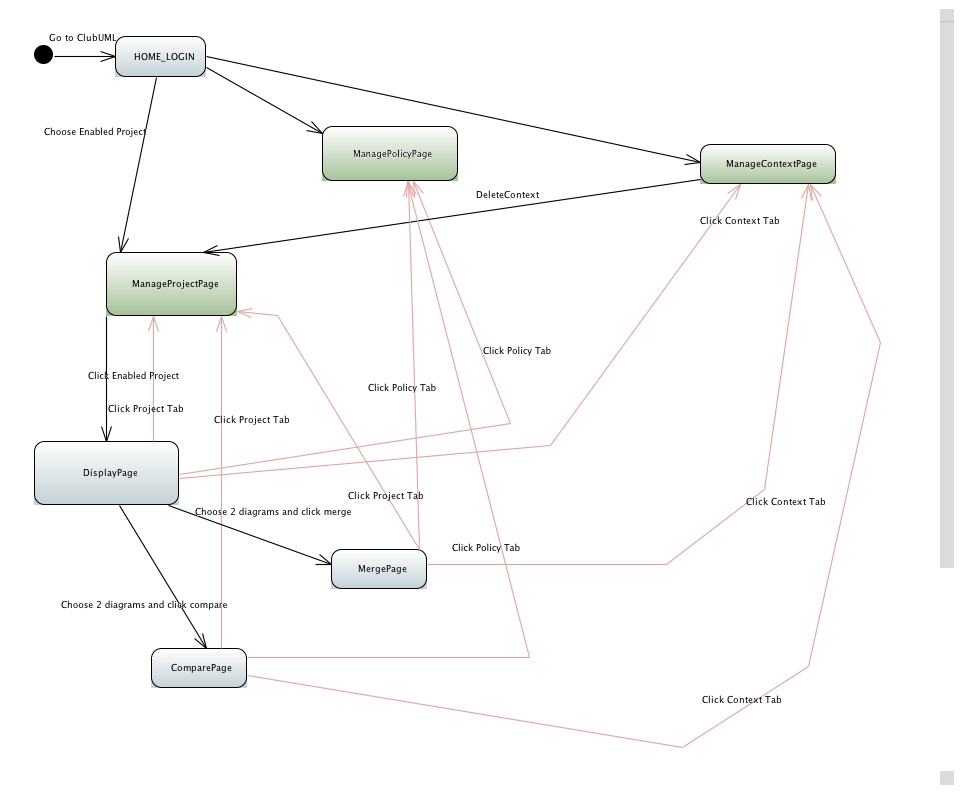
## A. Navigation

(Ken)

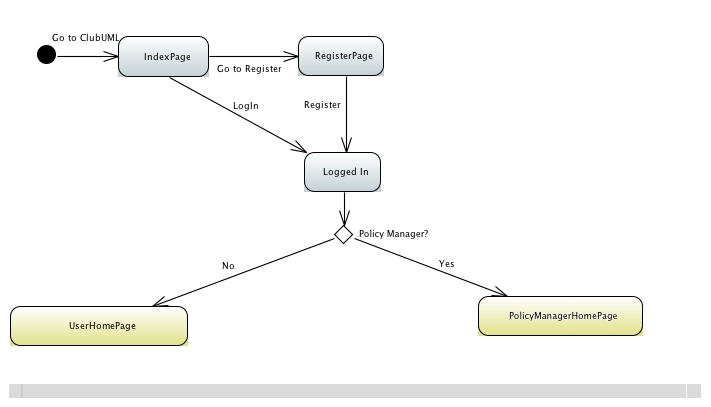
The Navigation Spring 2014 team identified and prioritized navigation access across all functionality of the application. Navigation through ClubUML is very simple and is composed of the following main states:

* Login
  + Registration
* Home
* Project
  + Display
    - Compare
    - Merge
* Context
* Policy

The following State Diagrams displays the navigation flow for the entire ClubUML system:



The first state when you start the application is Login State. Username and Password is required to proceed. If you do not have username and password you must register.



Registration options include registering as a User or Policy Manager. Based on your registration option you will have greater access available for managing projects. Upon login, the landing page is the home page. It provides an overview, history and “about” of the system.

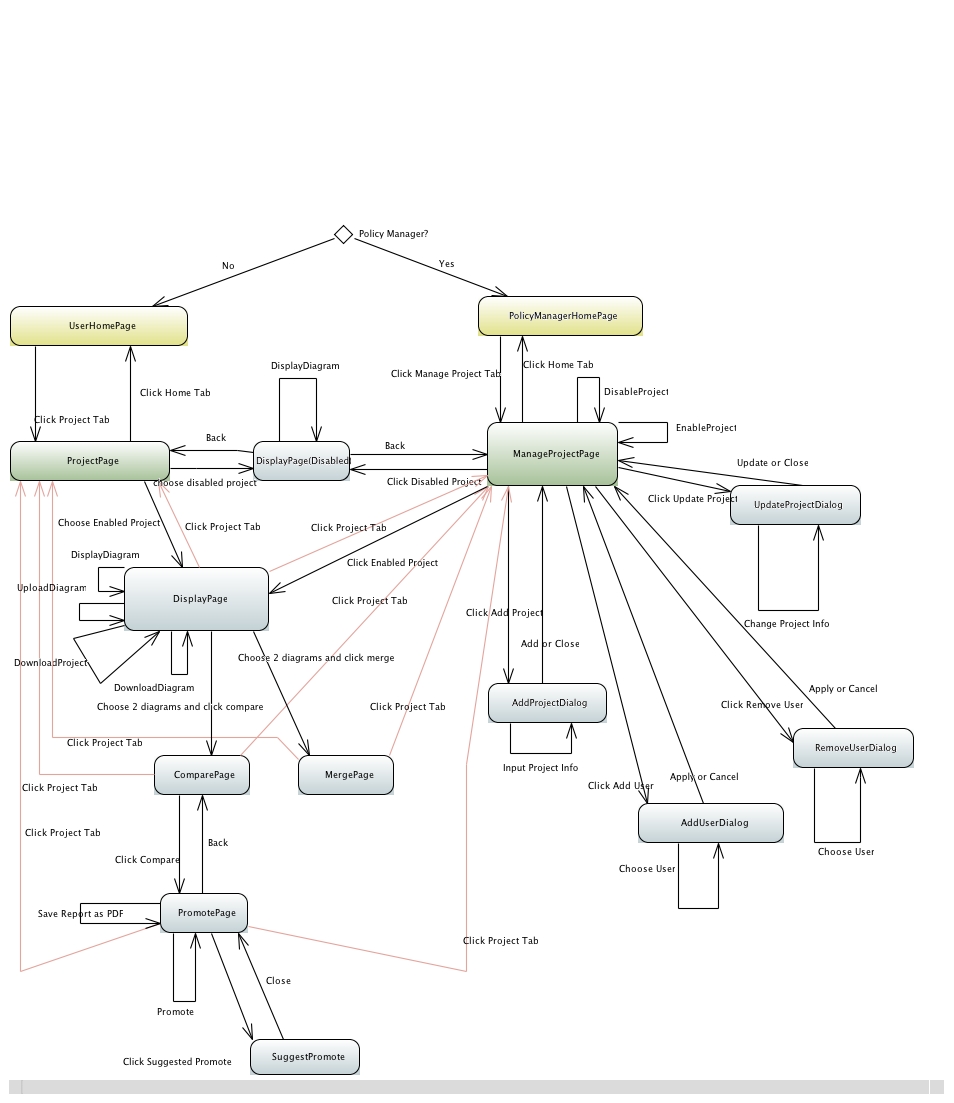
The next step is to open up the Project tab. The project page provides different functionalities to manage projects. It allows the user to Add, update, enable or disable the project. Once a project is added, the GUI brings the user to a new interface to work on the project

The following diagram depicts navigation through all the various functions within each project. This includes access to the multiple functionalities within the ClubUML such as Comparing and Merging diagrams.

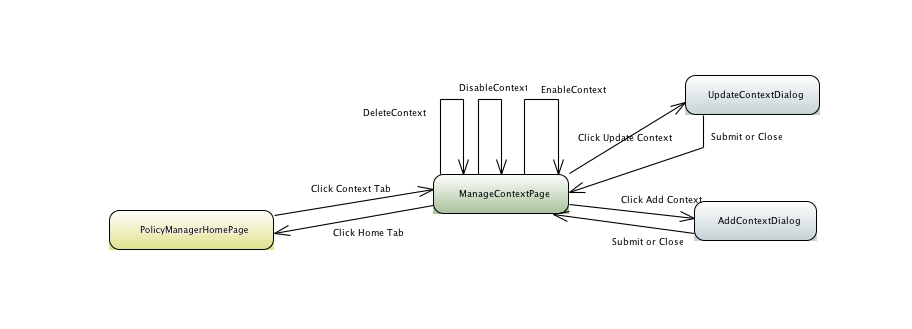
Once in a project the Display page allows you to upload xmi type diagrams to either Compare or Merge, as well as, Ecore type diagrams just to Compare.

Compare allows you to display two diagrams side by side and apply rationale management (see Spring 2014 Contributions Rationale Management for additional details)

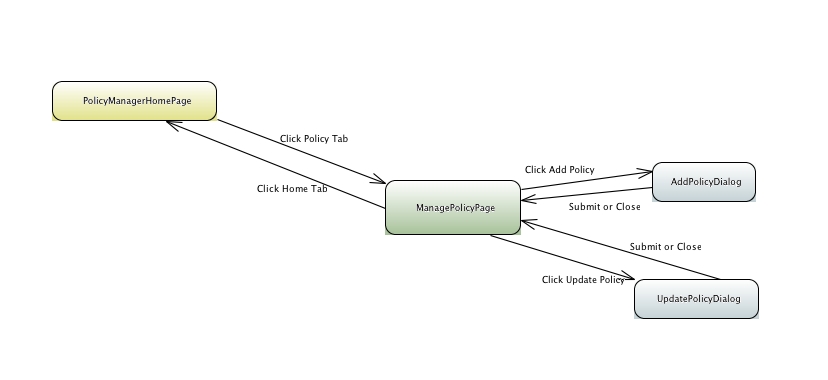
Merge allows you to add, break, and establish relationships among the attributes of two class diagrams.

Additionally, the Spring 2014 Use Case Diagram team added new functionality capable of parsing out uploaded Use Case Diagrams. The new functionality is also available on the Project tab via the Use Case button on the top right of the page (next to Disable Project)

The Context function allows to Add or Update Context dialogue. Existing Context dialogues could be enabled or disabled as well. The purpose of the context is to organize projects, similar to a folder/file system.



The Policy function allows to Add or Update Policies that could be used. The purpose is to support roles/permissions within the system.



Complete depiction of the ClubUML application and its navigation paths are displayed below:



## B. Dependencies

(Tom)

ClubUML has a layered architecture with multiple dependencies within each layer and also between the layers. In order for the team to understand and engineer a final system design solution, a dependency system diagram was implemented to understand the complexity of the ClubUML software. It was identified that there are 4 main modules (User Interface, Servlet, Database and Algorithms). There are a number of classes and implementations within each module where there are multiple dependencies.

(Click “RAW” to download:)

Reference to Dependency Matrix: [ClubUML File Dependencies.xlsx](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/ClubUML%20File%20Dependencies.xlsx)

Reference to System Dependency Diagram: [ClubUML System Dependencies.pdf](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/ClubUML%20System%20Dependencies.pdf)

## C. Classes

(Kareem)

CLUBUML JavaDoc ([LINK](http://kareemegal.github.io/javadoc/index.html)) gives a fair description of all packages and their classes in the ClubUML Project. The JavaDoc can also be viewed locally at "ClubUML2014/javadoc/". Please use the github pages link (<http://kareemegal.github.io/javadoc/index.html>) to browse full documentation of the packages and classes for the CLUBUML Codebase.

Using UML LAB (<http://www.uml-lab.com/en/uml-lab/>), Spring 2014 Team was able to create and generate Class Diagrams for entire ClubUML Project as well as for each individual package in the CLUBUML Project. The Class Diagrams are all stored in the code under ClubUML2014/Documents/2014Spring/Class Diagrams". Here is the list of the packages in the CLUBUML Project and the class diagrams associated with each package:

**Packages Class Diagram**

CLUBUML Project <[Link](http://kareemegal.github.io/Class%20Diagrams/ClubUMLSpring2013%20Class%20Diagram.svg)>  

compareAlgorithm <[Link](http://kareemegal.github.io/Class%20Diagrams/compareAlgorithm_package.svg)>

compareAlgorithm.smartPolicy <[Link](http://kareemegal.github.io/Class%20Diagrams/CompareAlgorithm_SmartPolicy.svg)>

controller <[Link](http://kareemegal.github.io/Class%20Diagrams/controller.svg)>

controller.compare <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_Compare.svg)>

controller.comparer.xmi <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_Compare_xmi.svg)>

controller.comparer.xmi.request <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_comparer_xmi_request.svg)>

controller.diagramparser <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_diagramparser.svg)>

controller.download <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_download.svg)>

controller.merge.xmi.xclass <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_merge_xmi_xclass.svg)>

controller.similaritycheck <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_similaritycheck.svg)>

controller.upload <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_upload.svg)>

controller.util <[Link](http://kareemegal.github.io/Class%20Diagrams/controller_util.svg)>

domain <[Link](http://kareemegal.github.io/Class%20Diagrams/Domain.svg)>

logging <[Link](http://kareemegal.github.io/Class%20Diagrams/Logging.svg)>

repository <[Link](http://kareemegal.github.io/Class%20Diagrams/repository.svg)>

uml2parser <[Link](http://kareemegal.github.io/Class%20Diagrams/uml2parser.svg)>

parser <[Link](http://kareemegal.github.io/Class%20Diagrams/parser.svg)>

policy <[Link](http://kareemegal.github.io/Class%20Diagrams/policy.svg)>

## D. Database

(Lauren)

The ClubUML application has a lot of entities, but how they’re related is relatively simple. There aren’t any very complex relationships or database tricks. The EER diagram for the full ClubUML database is shown below. The top-level entities are the user, project, context, and policy. The user is looked up at initial login time and then retained as part of the session. The projects are listed in the project tab, the contexts in the context tab, and the policies in the policy tab. From the project tab, each project may be selected and in turn have multiple diagram entities where diagrams are the main entities involved in user functionality. Diagrams may be compared, merged, rationalized etc., so they are referenced in several other entities. Comparing two diagrams creates a compare entity and a comparison report entity. From this comparison, the user can then promote one diagram or the other and create a rationale entity as part of their promotion. The user may also decide on a diagram by creating a decision entity, which may capture a number of rationales associated with that decision. The policy can be used to evaluate the diagram within a context. A policy is created by adding any number of metrics, which have extending entities (metricsType, attributes, classes) to more specifically describe the metric. The evaluation then becomes a combination of diagramMetricsScore entities to create the diagramPolicyScore. The rulesjsonstring and useCaseDiagram entities are mostly separate because they are not get rigorously integrated into the application. The rulesjsonstring has a simple string representation of the use case diagram rules, and the useCaseDiagram entity simply stores an access point for use case xml files uploaded to the application.



Link to github [EER](https://github.com/lmd59/ClubUMLSpring2014/blob/master/db/FinalEERSpring2014.mwb) (MySQL workbench editable .mwb-> Click “RAW” to download)

# Spring 2014 Contributions

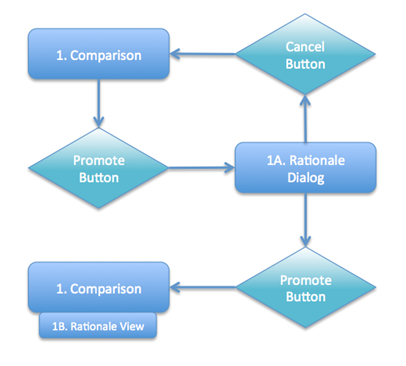
## A. Rationale Management

(Vishal)

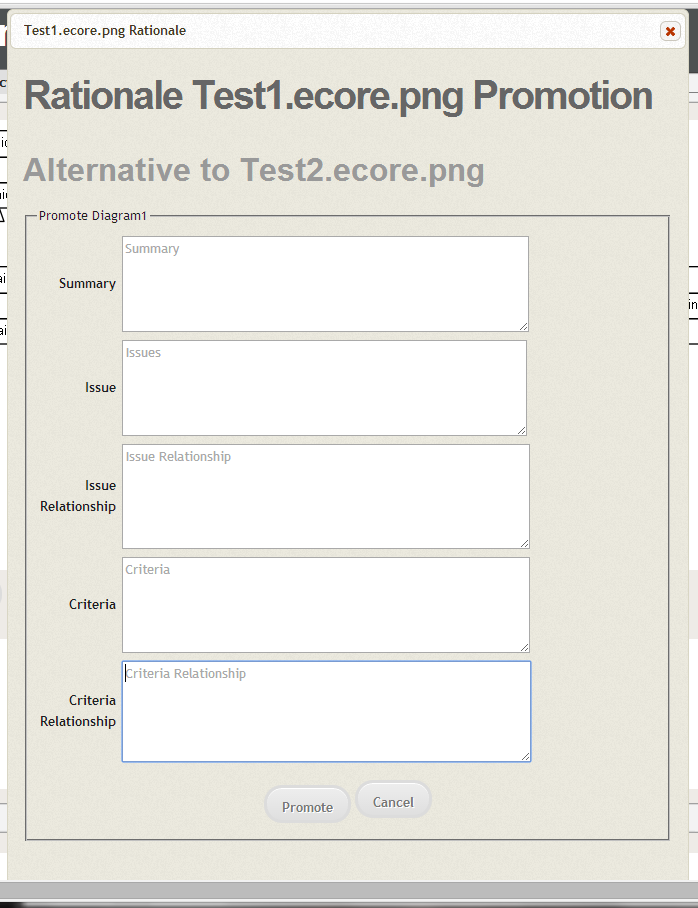
An added extension to the ClubUML project is Rationale Management. This new extension gives the user the ability to add, modify, and delete rationales associated with a particular diagram. With this added feature, it allows the users to see as well as explain in more detail the various reasons why a diagram was chosen or promoted compared to others. The extension was initially proposed as 3 separate use cases. See Rationale Management Use Cases in Appendix B for further details pertaining to the extension proposal. The first use case (RM1) is fully implemented, the second (RM2) is partially implemented, and third (RM3) is not implemented. The third use case is described in a further section for potential future expansions.

### 1. Use Case RM1

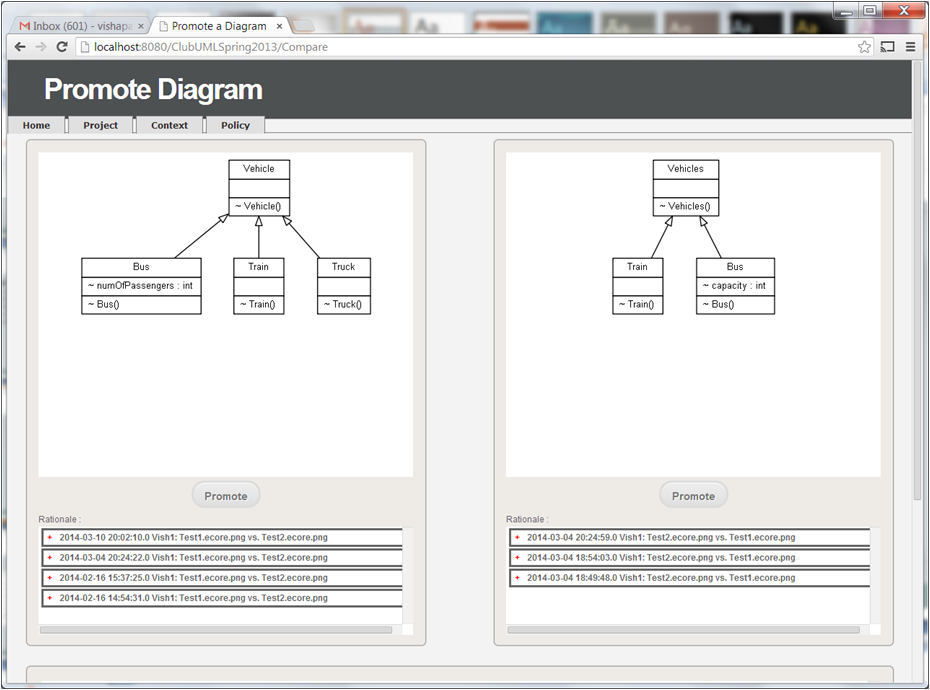
As mentioned earlier, the first use case is fully implemented and consists of giving the user the ability to add, modify, and delete rationales when promoting a diagram. The user can add a rationale when comparing two diagrams in the Promote page.



As the flow chart, taken from RM1 UI Design Document in Appendix B, highlights, once the user clicks on the Promote button for a diagram, a dialog box will display with multiple text boxes. The text boxes for input include a summary, issue, issue relationship, criteria, and criteria relationship, as shown below.



Once filled in the user has the option to promote or cancel. Once promoted, below the diagram, the rationale added will be displayed along with any other rationales added by other users for the same promoted/compared diagram.



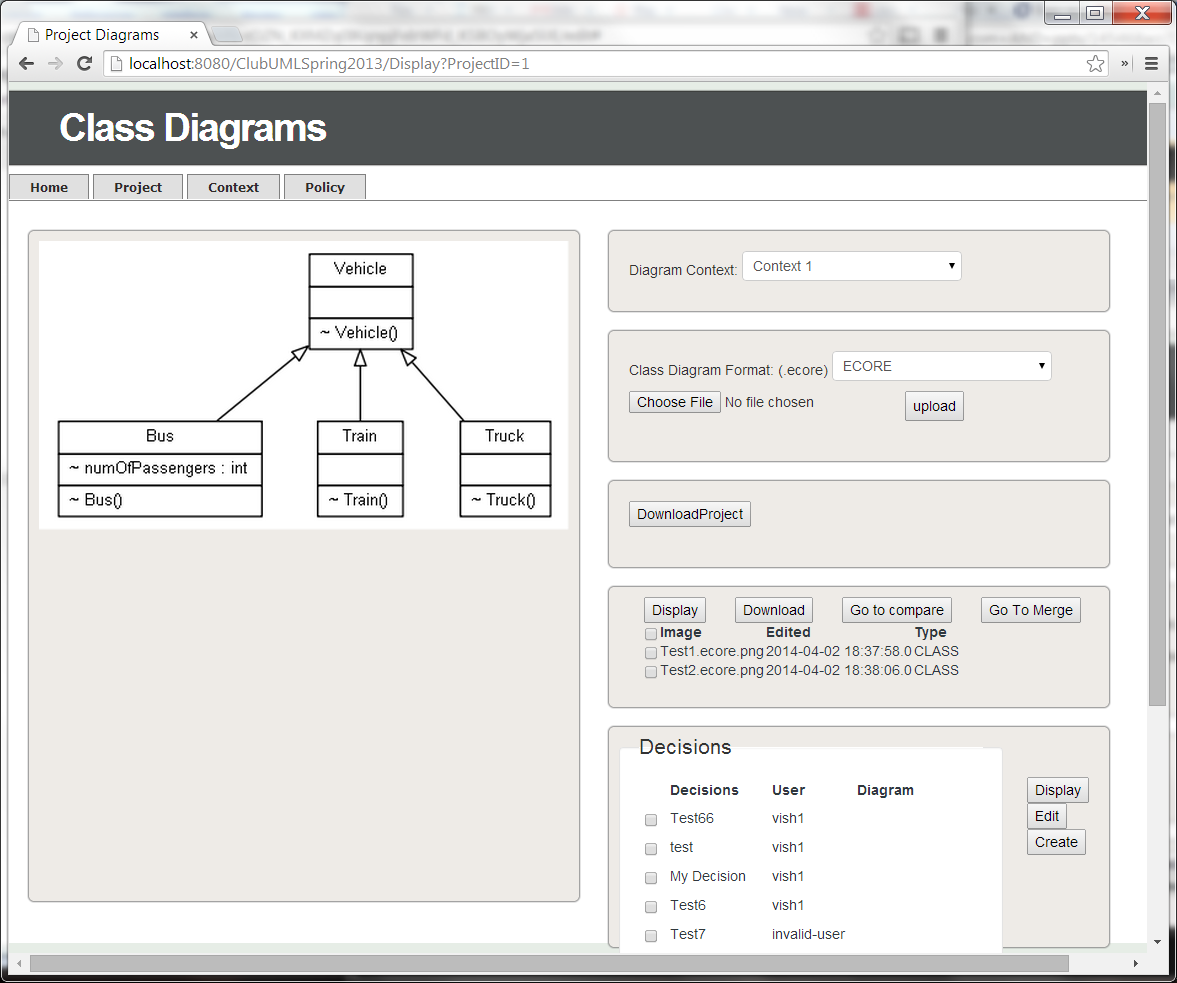
The rationales are grouped together for expanding and collapsing by the date, time, and user. When expanded, all the detailed information is displayed. Along with it, the user also has the ability to modify or delete that particular rationale.



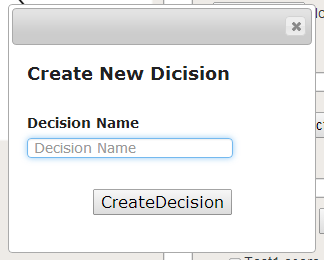
In order for these changes to be implemented, it required frontend JSP and CSS changes and backend Java and database changes. The files that were modified: promote.jsp, promote\_style.css, Compare.java, CompareDAO.java, Promote.java, RationaleDAO.java, Rationale.java

### 2. Use Case RM2

The second use case for Rationale Management gives the user the ability to create a decision, select a diagram associated with the decision, and select as many rationales as desired for why the diagram was chosen. The different navigation sequence and UI components can be seen as part of the Design Document RM2 in Appendix B.  Unlike the first use case, for this particular use case, a very limited implementation was created. Due to time constraints, we only implemented the piece for creating a decision name and displaying the decision names associated with the project. With the RM2 partial implementation, once the user clicks on a particular Project, in the bottom right, all decisions associated with the project are displayed.



Along with the display of all the decisions, the user also has 3 potential buttons that can be clicked: one to display the decision, second to edit a decision, or last to create the decision. The first two buttons are unimplemented. If the user does click on Create, a dialog box will appear giving the user the ability to create a decision name.



Once created, the decision gets added into the database and will be used for displaying all decisions associated with a particular project. Similar to the first use case implemented, this use case required both frontend and backend changes. The files modified included: Display.jsp, display.css, Style.css, DisplayDiagram.java, Display.java, Decision.java, and DecisionDAO.java.

## B. Validation

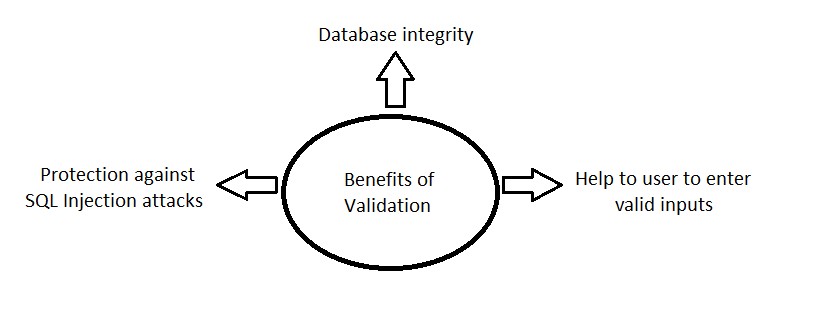
(Ameya)

### 1. Overview

Validation is considered an essential part of any application that deals with databases (data-driven applications). This is because the data that is entered by the user need not be completely accurate each time and is prone to human error. Any application which deals with sensitive data should not be built in a manner which would rely upon just good fortune and the developer shouldn’t have blind faith that there would not be any mistake during data entry. Hence, validation of data at entry is a very important part of an application.

Furthermore, any data-driven application which might contain sensitive data which can be harmful if leaked such as customer data in banks and credit card companies, patient records of hospitals, etc. should be secure. There are many ways in which a database can be hacked. One of them is SQL Injection.

SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.



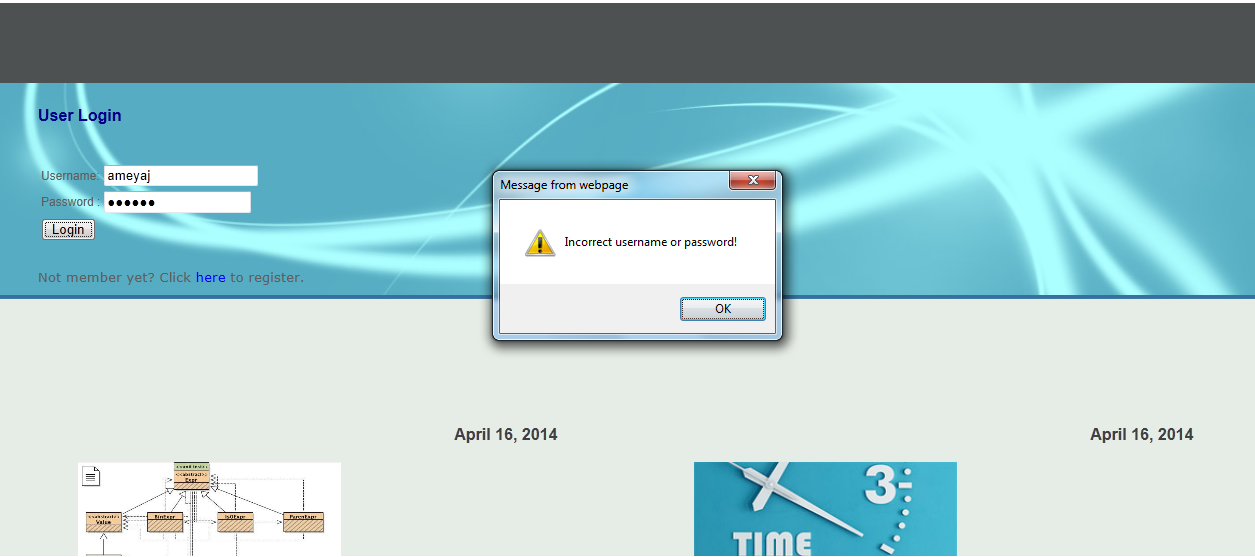
Our project, ClubUML, contains sensitive data. Data related to projects and its details in a company, the procedures followed by the company and sensitive discussions are things a company would not want to go in wrong hands. And hence, validation of the data being entered is very important for our project.

We have integrated validation into our project this semester. Most of the validation has been at the front-end using JavaScript. Validation has been included at every place which takes data as input from the user.

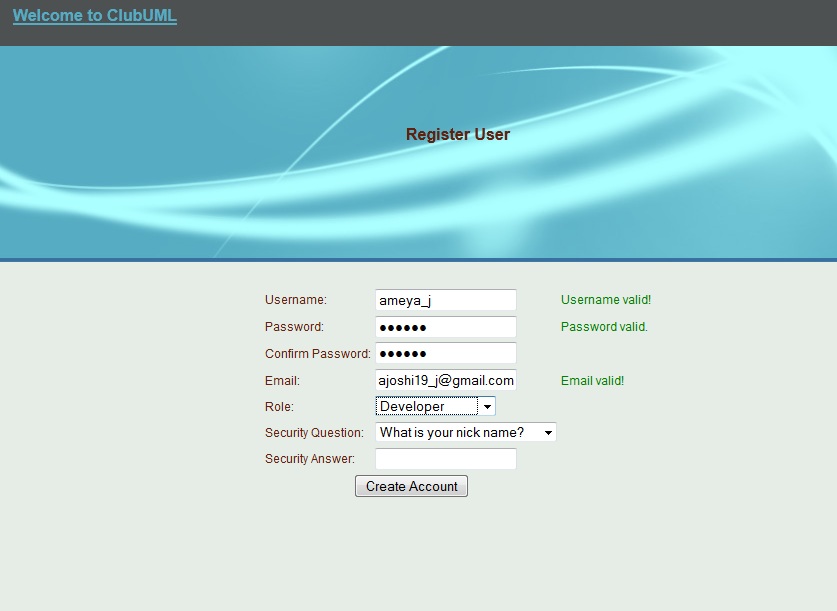
### 2. Areas with need for Validation

There are several areas in our project where validation is a must. The integrity of our project depends on the data being entered in by the user. Some of the major areas of validation are:

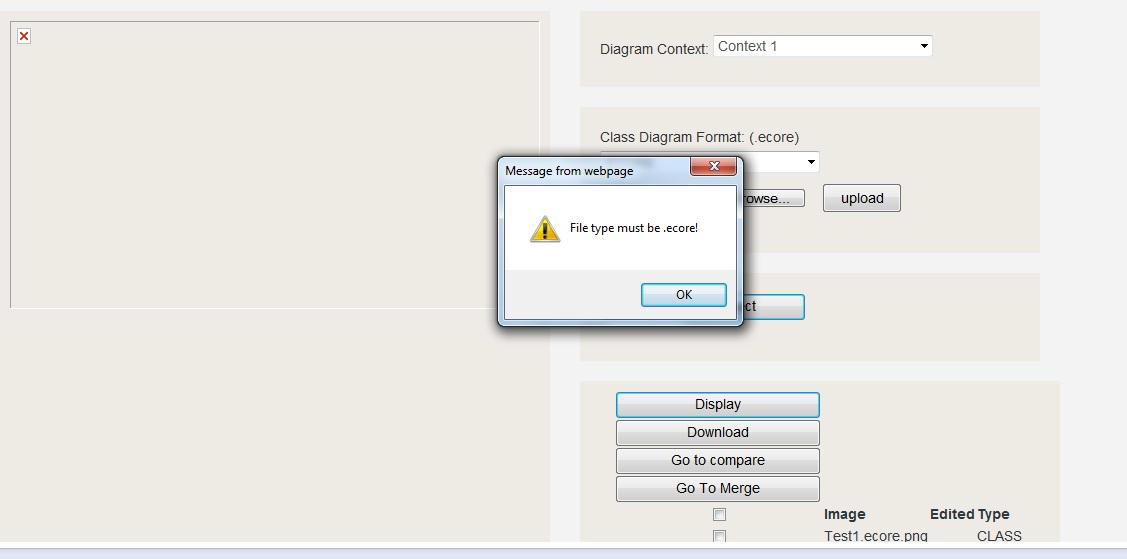
* **Login:** During login, we need to check that the input is safe to be used in a query and would not harm the data or the database itself. We need to make sure that it is free from escape characters and some special characters that have greater possibility of occurring in a query. In short, we need to make sure that the input does not have a query hidden in it. To do this, we can set some constraints on the username and password; and directly check these strings for the acceptable inputs. This will make sure that it is valid and safe.



* **Account registration:** While creating an account, we implement the same constraints on username and password as implemented during login. This makes sure that the username and password fit a certain rule (constraint) that we imply. Apart from this, we need to make sure that the username is not already in use by any other user, if so we need to alert the user that the username is already taken. Similar function for email will check if the email address being entered was already used to create an account. We do not want multiple accounts with the same email id.



* **File upload:** The user can upload diagrams that he can later use within a project to compare with one another or to further use the UseCase App. The project supports ‘ecore’, ‘xml’, ‘uml’, ‘di’ and ‘notation’ diagram files. Hence, here we need to make sure that the user is uploading the right type of file. We need to alert the user if he selects an incorrect file to upload.



* **Updating:** The user can update several details in the project. He can update existing policies and change their metrics. In doing so, the user can by error set the fields to some unexpected input types. Hence to make sure that the user is using correct type of input, we put few validation checks that make this sure for us. The user can also change the details of existing project, add context and update context. All these inputs need validation to check that the user has entered a valid input.

* **Database constraints:** The database that the product uses has specific constraints pertaining to its attributes. Validation of the input is a must to make sure that the constraints on the database are not violated, resulting in failure of the SQL query. Thus, these validations maintain the integrity of the database and give user a fair idea of how the input should look like.

## C. Navigation

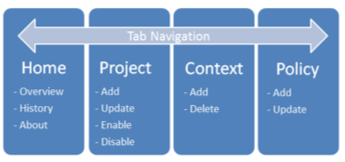
(Tom and Ken)

### 1. Overview

The Spring 2014 Navigation team was created to support the sustainment of the Navigation/Graphical User Interface (GUI) which supports the functionality and visualization to the user. The Navigation Spring 2014 team identified and prioritized the tasks’ level of effort during the course of the semester.

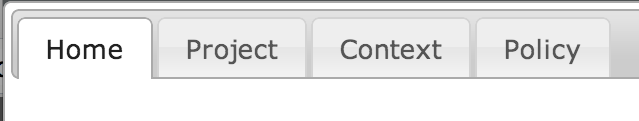
### 2. GUI Layout

The user is presented with a series of different functionalities when first logged into the system. Prior to the main compare/merge capability of the system, the user is able to navigate through different options of projects, contexts and policies. The high level navigation can be seen below.



### 3. Ported Tar Bar/Header

ClubUML is supported by a tab display to navigate throughout the entire application for easy access to the user.



Additional Headers were added in the Compare and Merge pages to help the user clearly understand it’s current location within the sub functions of the application. These headers also share a similar name with their governing java/jsp files.

The introduction of an additional HOME button at the bottom of the Project, Context, and Policy page was also required.



In the event of navigating out of either Display, Compare, and Merge functions into the Project, Context, or Policy page via Tab Menu, the current version of the application does not support call back of the jquery to load the initial tab menu present upon login. The home button will navigate you back home and will call back the main tab menu (this is only required when navigating to the Context or Policy page).

### 4. External Library Migration

#### After the effort of finding the dependencies within the entire ClubUML System, it was identified that there were multiple libraries, both CSS and JS that were on the internet. The problem with this architecture approach is that if there was a disconnection (such as a DIL network), ClubUML will not work correctly.

The Navigation team identified the effort to migrate the external libraries to the local system. This will allow the application to work no dependency of an outside connection. There was a series of implementation to support this effort.

* Migration

The team had to download and verify all of the external libraries and drop them into the ClubUML implementation.

* URL Localization

The team had to redirect the URLs in each file dependency to point to the new location of each library.

* Integration/Test

The team had to integrate the effort into the baseline and test the new implementation.

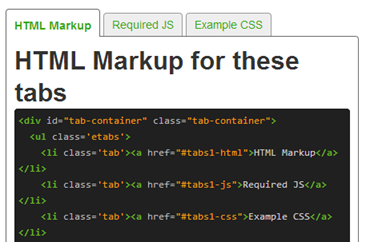
### 5. Tab Modernization Research

The GUI team initiated the analysis to look into different modern tab navigation components. The purpose of providing a new tab view is to give the application a new look and feel to the user. Instead of the team diving into creating a new tab navigation, the team did research for existing implementations that could be reused and ported to ClubUML in an easy low cost effort.

The Team looked into multiple different options that are both open source and licensed. The team found the following:

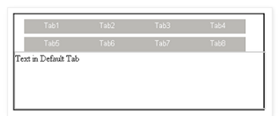
* ***Easy tabs*** (<http://os.alfajango.com/easytabs/>)

The easy tabs implementation provides the ability for a developer to apply a tab to multiple different CSS instances giving a different look and feel for multiple applications.



* ***Open Source Programmer*** (<http://opensourceprogrammer.blogspot.com/2010/01/tab-menu-in-javascript.html?m=1>)

The Open source programmer provides a library to create a simple tab menu within javascript with default actions and events to support the user.



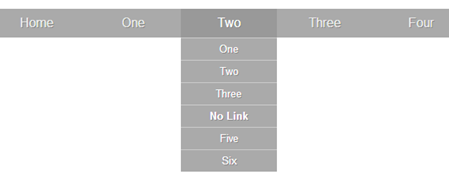
* ***Tabber*** (<http://www.barelyfitz.com/projects/tabber/>)

Tabber is an easy to use javascript tab library to be implemented within a web application. It automatically creates an HTML tab interface using the plug and play javascript.



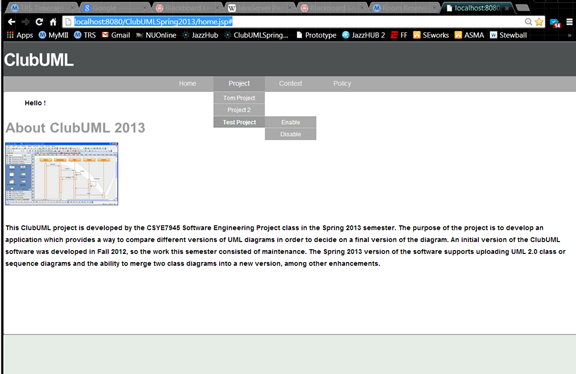
* ***TinyDropdown 2*** (<http://sandbox.scriptiny.com/tinydropdown2/>)

TinyDropdown provides a modern easy to use tabbing interface. It gives the user the ability to see an animated tab bar for easy access to multiple different GUI panels and implementations. It is easy for a developer to integrate and implement against their application.



### 6. Tab Modernization Prototype

The GUI team created a prototype to give the rest of the team a vision of a new implementation of a modern interface. The decision was made to utilize the TinyDropDown 2 library to use within the prototype. Changes were made to the home.jsp and also adding the set of the libraries used in referenced. A demonstration was conducted to show the potential for TinyDropDown 2 so that the team could make decisions about the next steps to take for a new tab style. A screen shot of the prototype is shown below.



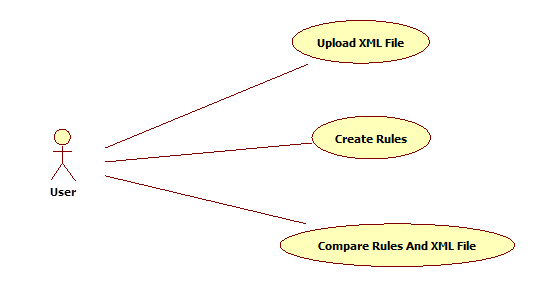
## D. Use Case Diagram Management

(Chao, Yang, Bing)

### 1. Overview

The Use Case Diagram team used the Spring 2014 semester to add another kind of diagram to the ClubUML Application. They investigated a couple of different diagram types and chose the UML Use Case Diagram because they felt that they could reasonably implement parsing and management functionality over the course of the semester. The team created an XMI parser for the Use Case XML files, developed Use Case evaluation rules, developed a front end to input rules, and a backend for evaluating parsed Use Case diagrams against those rules. Below is the use case diagram to describe the functionality that this team implemented and the class diagram describing the classes used to implement this functionality.

**Use Case Diagram:**



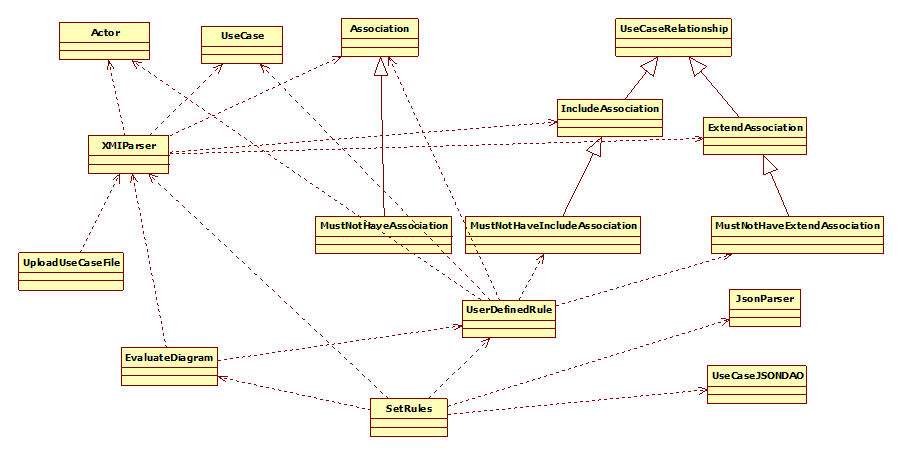
1. User could upload xml file and select the file they need.

2. User could create rules to evaluate the xml file they upload.

3. Click compare button, user could compare rules and xml file. And check the evaluate result.

### 2. Design

**Class Diagram:**



**Class diagram explanation:**

1. Actor, UseCase, Association, IncludeAssociation and ExtendAssociation are used to describe use case diagram.

2. IncludeAssociation and ExtendAssociation extend from usecaseRelationship.

3. XMIParser is used for parser the xml file and get the data in it.

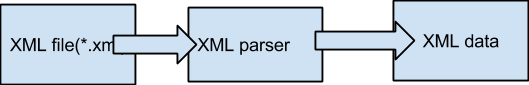
4. SetRules is a servlet for get rules from User Input.

5. UserDefinedRule is used to store the data which get from User Input.

6. UseCaseJSONDao is used to insert and get json string from database.

The team developed a structure for handling complex data flows. The use case metadata is located in a separate XML file as created by StarUML, so the XML parser is used to turn that information into internal XML data. Similarly, the Use Case rules are input by the user and transferred using json formatted data. This data then needs to be parsed and applied to the internal XML data to evaluate whether the Use Case in question complies with the rules.

**XMIParser**

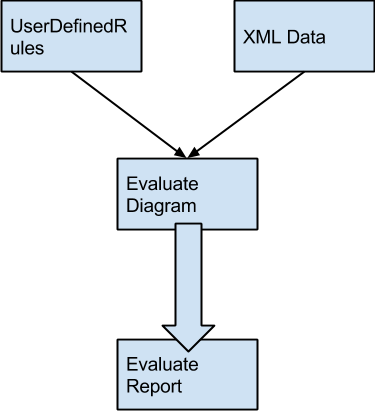


**Evaluate Diagram:**

User Defined Rules are stored in UserDefinedRules Object.

XML data is stored in XMIData

.Evaluate Diagram function could get data from these two objects and compare them.



**SetRules:**

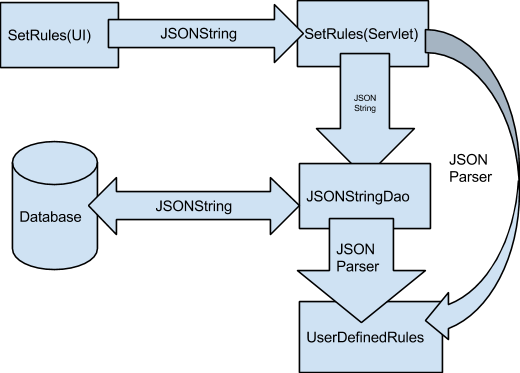
1. User create new rules by the UI.

2. After they click the submit button. All the rules pass to a JSON string and transit to

setRules servlet.

3. Insert the json string to the database directly.

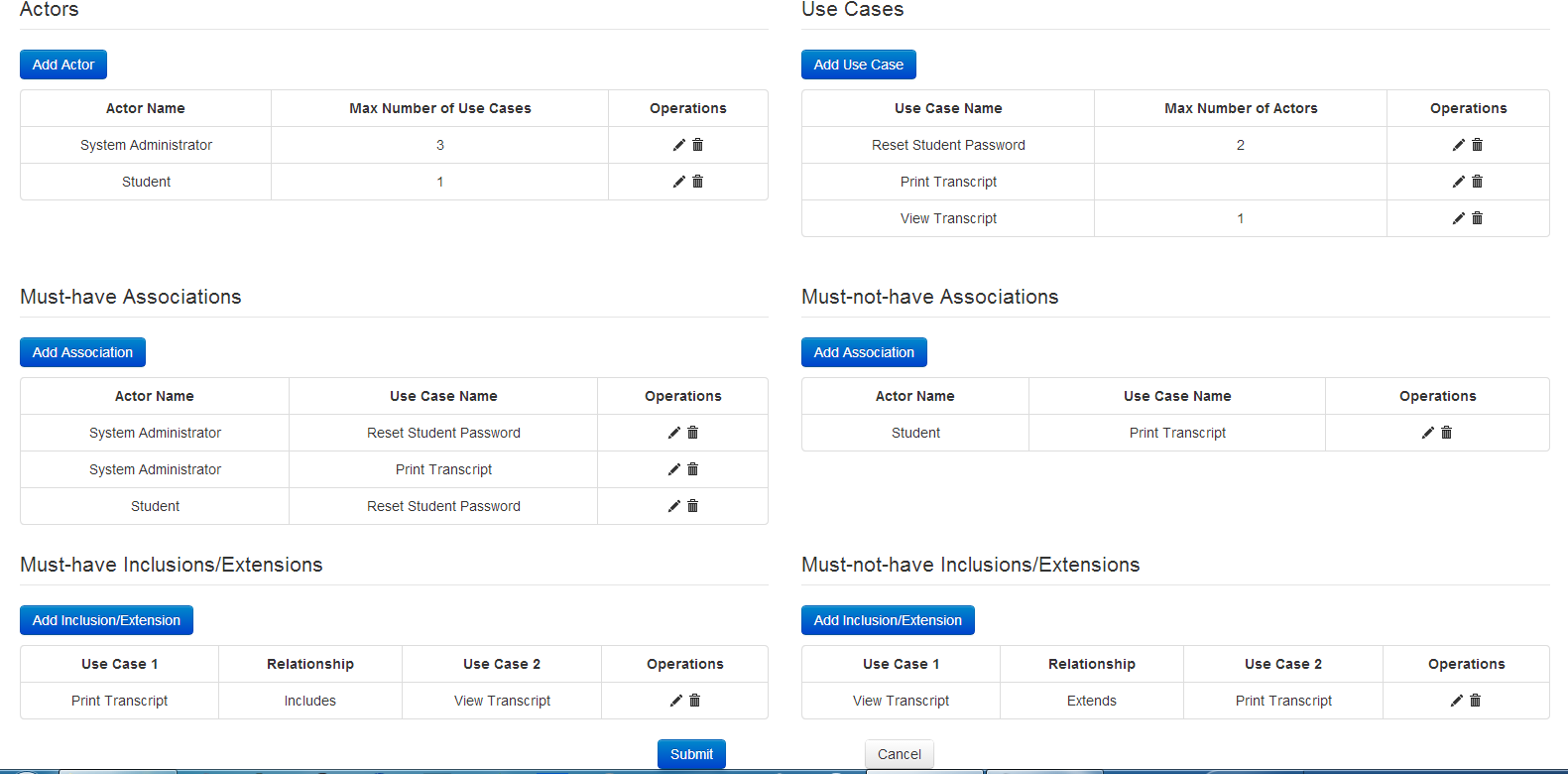
4. setRules servlet could store user-defined rules to the database via JSON parser.



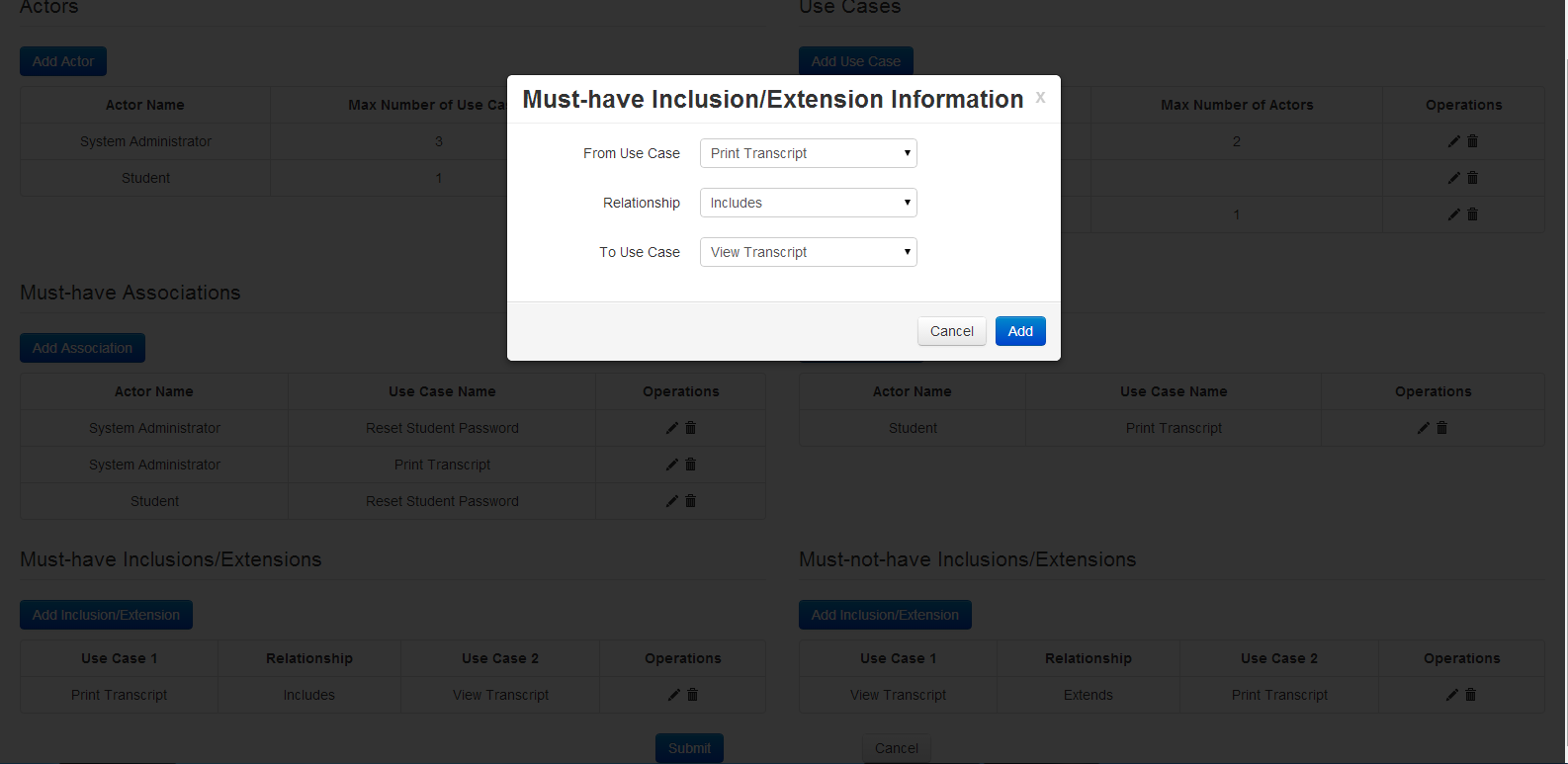
**User Interface:**

The UI is implemented using Bootstrap and JQuery. The reason for using Bootstrap is that Bootstrap can help us create very good look and feel without writing too much CSS code. But In order to have a very interactive UI we still need to write a lot a JavaScript code using JQuery. The front-end transform the data input by the user to JSON and use ajax to send them to the back-end. This is a good way to facilitate the collaboration between front-end developer and back-end developer.

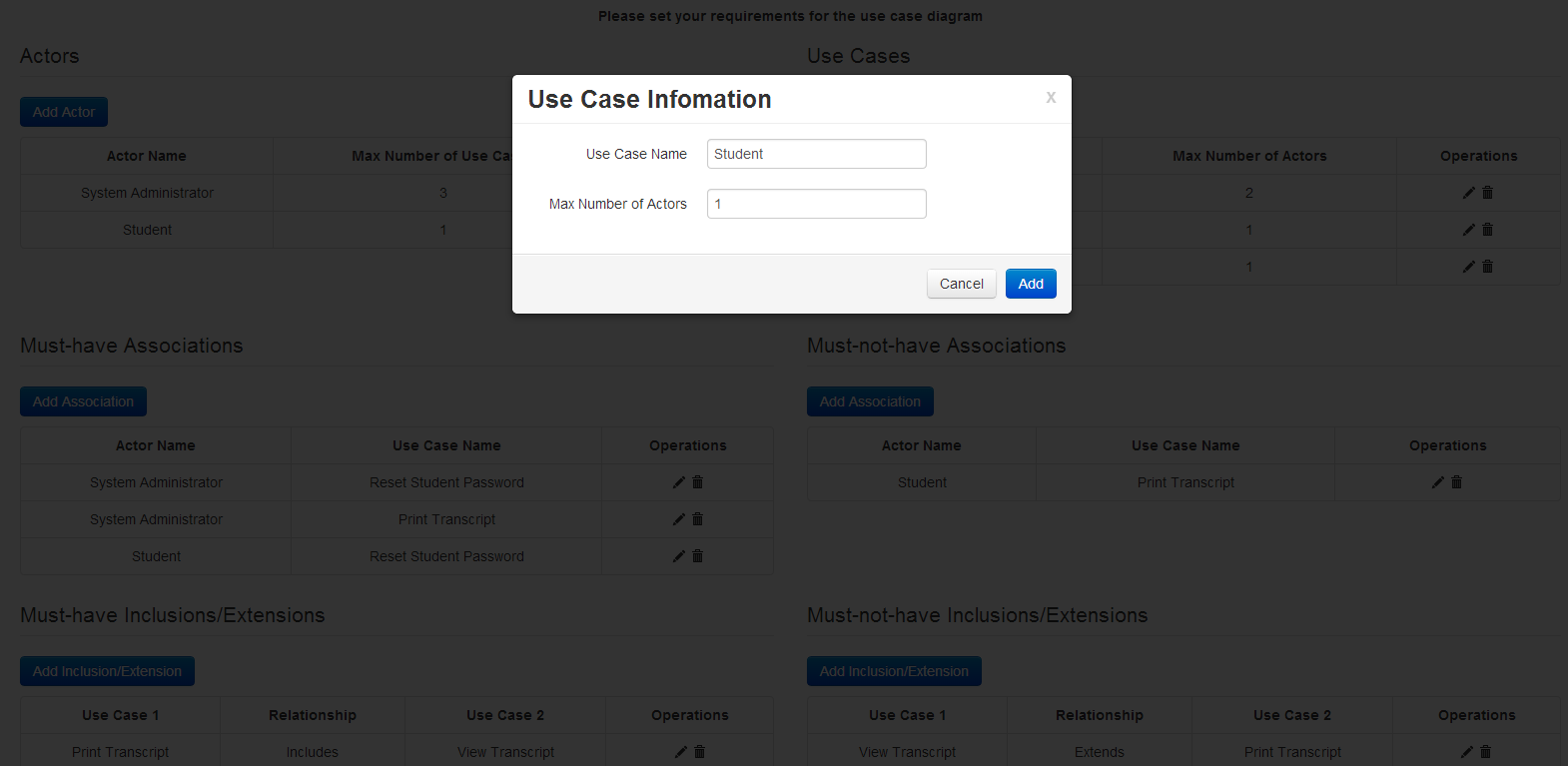
The UI for setting rules:



Add an inclusion/extension relationship



Edit the information of a use case



### 3. Testing

**Use Case Diagram Testing:**

Please refer to the testing document below:

[**Use Case Diagram Testing Document**](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/UseCaseDiagramFunctionality/UseCaseDiagramTesting.docx)(click “RAW” to download)

## E. Resolved Bugs

(Jeremy)

In the beginning of the semester and throughout development, many bugs were discovered. In order to track and report bugs, the team used jazzhub and followed the [Bug Report Procedures](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Miscellaneous/ClubUML2014%20Method%20Sheet%20-%20Bug%20Tracking%2C%20rev%20B.pdf). Throughout the semester, the bug team was able to resolve a number of bugs as listed below.

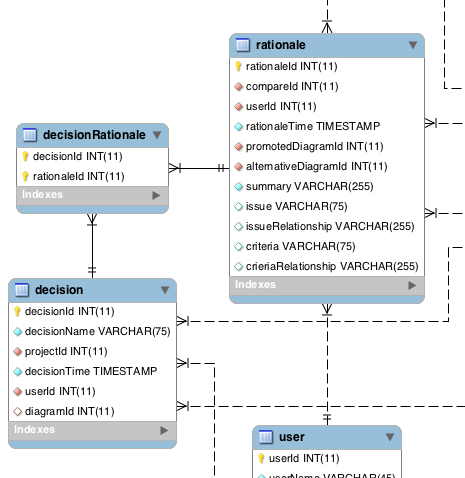
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Summary** | **Status** | **Priority** | **Severity** |
| 19898 | Out of bounds exception when uploading diagrams | Resolved | Unassigned | Normal |
| 16976 | Unable to download class diagrams | Verified | Medium | Normal |
| 15404 | Project Diagram Page - Click Here to go back doesn't function | Resolved | Low | Minor |
| 15416 | Create Project submission returns user back to home page | Resolved | Medium | Minor |
| 15788 | Cannot display diagrams for comparison on Class Diagrams Page | Verified | Medium | Normal |
| 19533 | Receiving Multiple Annoying emails about the status change of a ClubUML defect | Resolved | High | Critical |
| 16580 | Unable to enable a project | Verified | High | Major |
| 16581 | After creating a project, it goes back to the Home page | Resolved | Low | Minor |
| 16484 | Create Policy returns to home page | Resolved | Medium | Minor |

## F. Database Updates

(Lauren)

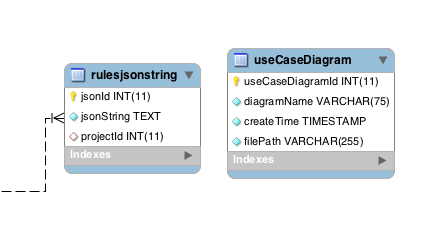
There were several database updates made to standardize the attributes used in the database tables, database access layer, and application domain. Across these layers, the entities different numbers of attributes (some more, some less), and the attributes sometimes had varying and inappropriate types. In the Spring 2014 semester, we made a brief concentrated effort toward making these consistent and appropriate for the data involved. This solved some immediate issues by populating more dates and names in tables across the application and eliciting more generally reliable behavior. Now domain objects are properly populated when they are pulled from the database, and database tables are properly populated when a domain object is stored. This makes development significantly easier as database access calls behave as would be naturally expected based on the properties available to the developer.

        There were also a number of larger chunk changes made to the database in order to support new functionality. These new entities are zoomed in upon in the images below. The rationale, decisionRationale, and decision entities were added to support Rationale Management use cases RM1 and RM2. The rationale entity is identified by its rationaleId, in the context of a diagram comparison (compareId) where the user (userId) chooses a diagram A (promotedDiagramId) to promote over a diagram B (alternativeDiagramId) at a specific time (rationaleTime). The user can then specify their rationale by inputting the summary, issue, issue relationship, criteria, and criteria relationship. This information is directly input by the user via the dialog box before it is stored in the database. The user can then create a decision identified by its decisionId. This decision should be named to specify what value the chosen diagram represents, but multiple decisions can be created with the same name so that the database may keep track of the decision history. This decision is created in the context of a project (projectId) by a user (userId) at a specific time (decisionTime), and that user should select a diagram (diagramId) that has been decided upon. The diagramId is currently not required because there is as of yet, no functionality to choose a diagram on the front end of the application. Similarly, the many-to-many decision-rationale relationship is specified so that a user may identify any number of rationales that contributed to some decision, but as of yet, there is no front-end method for choosing the associated rationales.



Added RM1 and RM2 entities for Rationale Management

The Use Case Diagram entities are shown in the image below. They are a fairly basic database implementation in order to expediently provide persistence for uploaded Use Case Diagrams and created Use Case rules. The rulesjsonstring entity simply stores the json string that was created from the Use Case Rules input page. The jsonId is used to identify the json string, and the jsonString entity contains the actual json text. The projectId is not currently used at all, and only the most recent rulesjsonstring is actually accessed at the moment until a more sophisticated front end is available to choose from a number of created rule sets. The useCaseDiagram entity is meant as a simple access point for tracking where Use Case Diagram xml files are located on the server. When a file is uploaded, the useCaseDiagram entity is created with a useCaseDiagramId to identify it. The diagramName is stores the file name, the diagram path stores the serverside file path (does not include file name), and the createTime stores the time at which the file was uploaded. This database entity structure could further be improved by further breaking down the rules and use cases into their data components after they have each been parsed by the json parser and the xmi parser respectively.



Added entities for Use Case Diagram upload and Use Case Rule creation

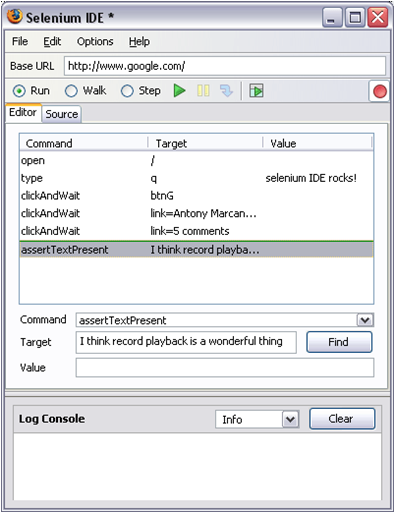
# Maintenance Recommendations

## A. Regression Testing

(Jeremy)

Regression testing for ClubUML was scripted using the Selenium web automation framework, specifically the Selenium IDE plugin for Mozilla Firefox. The Selenium IDE provides a simple method to automate tasks in a web browser. Scripts can be developed either by recording actions within a web browser or by manually editing the test scripts. The test scripts can then be played back individually or all at once as part of a test suite. The Selenium IDE keeps a running total of the number of test cases that have passed and have failed when running a test suite. An overview of the Selenium framework is located at:

[https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Integration %26 Regression/Automated\_Regression\_Testing.pptx](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Integration%20%26%20Regression/Automated_Regression_Testing.pptx)



**Selenium IDE Screenshot**

We found the best way to generate ClubUML test scripts was to use the record function to capture the user functionality under test and then manually make any necessary modifications to the script to ensure that it correctly tests the functionality. Individual commands in the script can be executed from the IDE editor to validate that they work correctly when manually editing the test script. Certain inputs such as file locations on the machine under test will need to be modified for each system that will run the test scripts. The current regression test suite is stored in the Git repository for ClubUML located at: <https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Integration%20%26%20Regression/Test%20Suite>

The best practices for integration and regression testing are located at:

<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/LessonsLearned/Integration_and_Regression_Testing_Lessons_Learned.revA.docx>

The current regression test suite only focuses on user functionality testing. A future enhancement to the regression testing would be to incorporate an automated test framework that could perform verifications on the server side as well as the client side of the ClubUML application. This would be a beneficial step forward in getting the ClubUML development to utilize continuous integration moving forward.

## B. Partially Completed Functionality

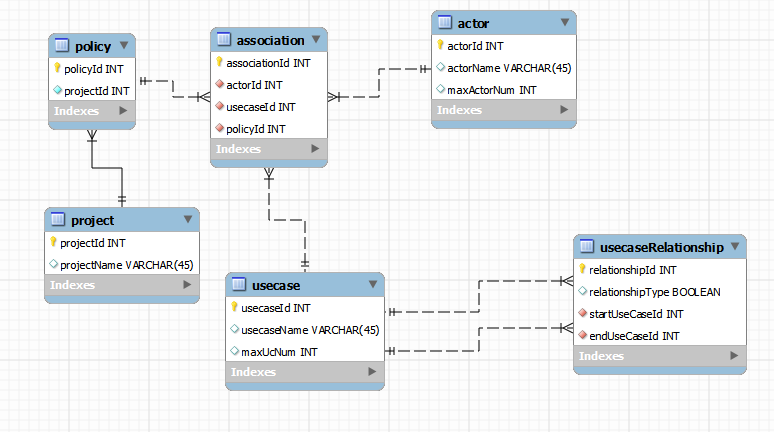
(Ameya)

### 1. Overview

There are a few functionalities that were discussed before the start of the project. These were either discarded completely or implemented partially due to time constraints of the course. These functionalities could be finished later to further enhance the product. Some of these functionalities are mentioned below.

### 2. Use Case Diagram

The use case diagram was originally a more complex functionality which had to be limited a certain amount due to time constraints. The following diagram shows the originally planned Use Case Diagram database structure:



But, in implementing this, the team had particular difficulties in retrieving the ids for Use Case and the Actor, so the temporarily discarded this more detailed structure. Instead, a more limited solution was implemented as a stopgap.

Additionally, due to time constraints, the more limited database structure does not include or track references to the project or user currently creating rules and uploading diagrams. As such, the Use Case App if still kind of a standalone app that could use further integration into the ClubUML application. Diagrams should be uploaded and evaluated in the context of the project, and the application should have a smoother integration between class, sequence, and use case diagrams since they each have some related data types and functionalities.

### 3. Rationale Management

In Rational Management RM2 Use Case, there were three main features that were decided to be implemented:

**Display Decision**

* Displays all the following:
  + The Diagram and the Diagram name selected by the team lead for the project.
  + List of aggregated Rationale or Rationales leading up to the decision-making.

**Create Decision**

* Create decision functionality allows the team leads to first pick a Diagram from currently uploaded diagrams and select best rationale from the list of rationales.

**Update Decision**

* The update decision allows the team leads to update the following:
  + Change the selected Diagram for the Decision and select a new diagram from the currently uploaded diagrams.
  + Change the rationale or rationales selected for the currently selected Diagram.
* Pick a diagram from the currently uploaded diagrams (excluding the currently selected decision diagram).

But due to limited time, this functionality could be only partially implemented. The functionalities that have been implemented are:

* A decision can be created.
* All the decisions for a particular project can be displayed.

The functionalities that have been left out are:

* Being able to associate a diagram to the decision.
* Being able to select from a list of rationales associated with the decision diagram.
* Edit a decision.
* View a particular decision.

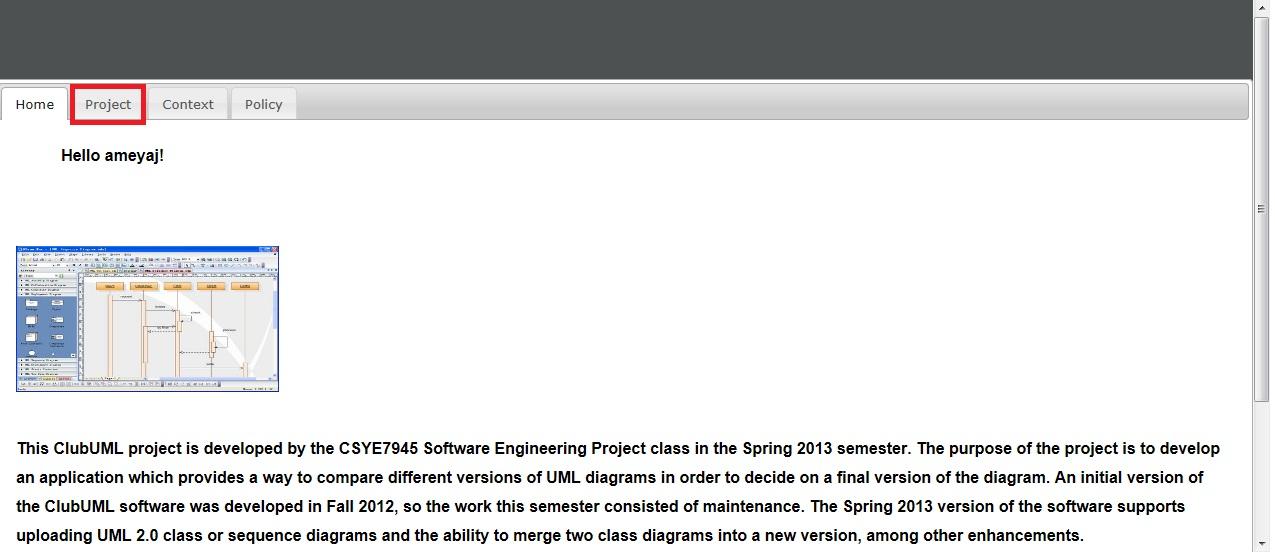
*(Refer to Design Document RM2 in Appendix B for further details)*

### 4. Navigation

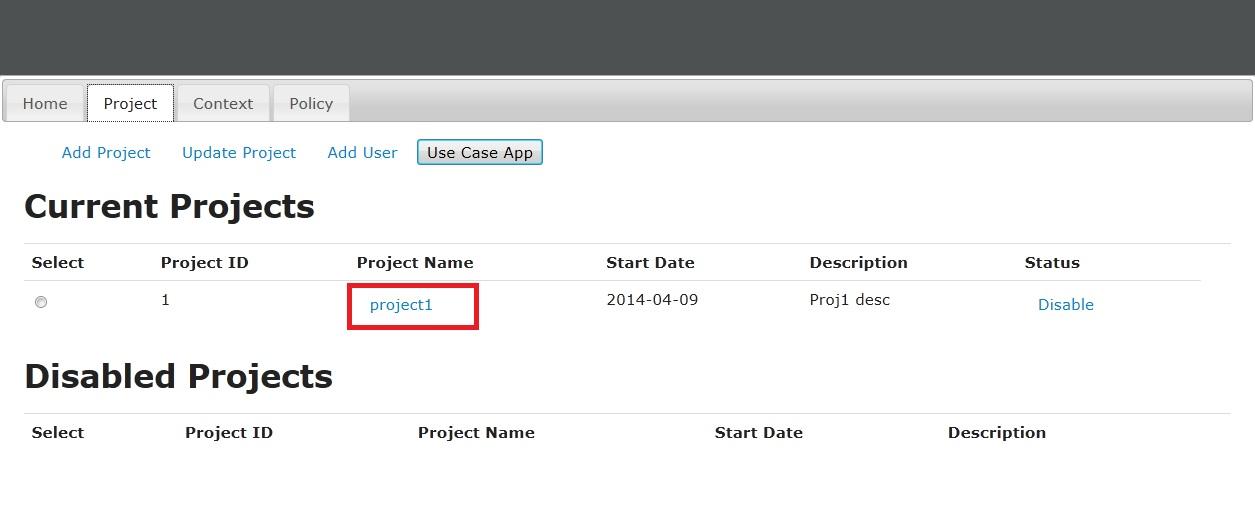
One of the most frustrating incomplete functionality is that the navigation bar tabs are not fully functional. The main reason behind this is that the servlets are loaded again, and some of the links just open up the JSP pages instead of reloading the servlet with the right JSP. This takes you to a JSP from where it’s impossible to navigate out or reload the servlet. The only way to get out is hit the back button on the browser.

Example:

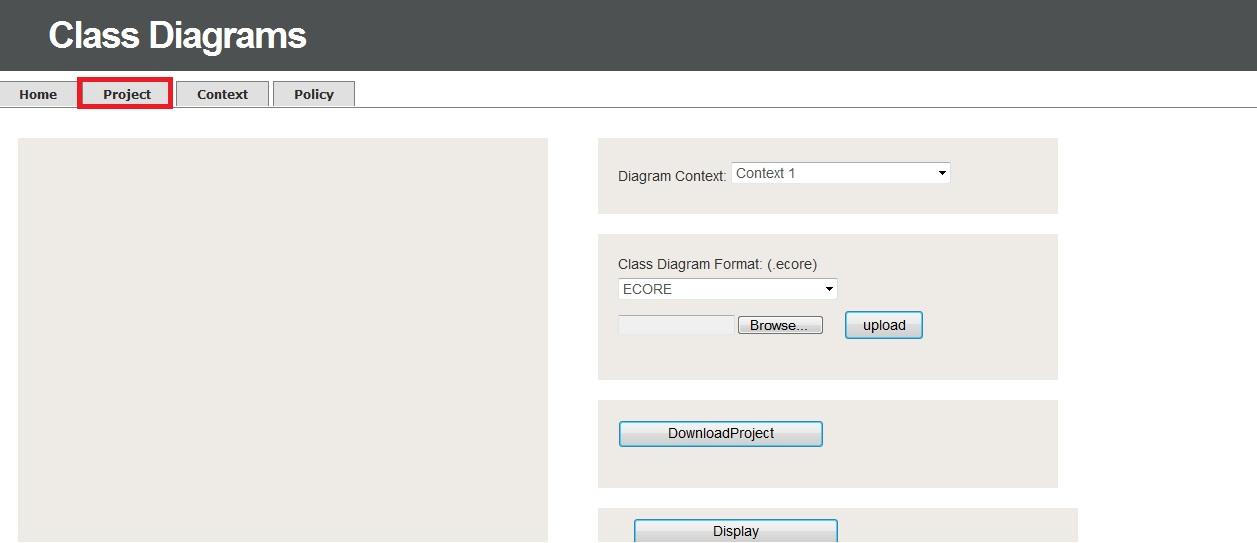
1) Log into the application. Hit the “Project” tab.



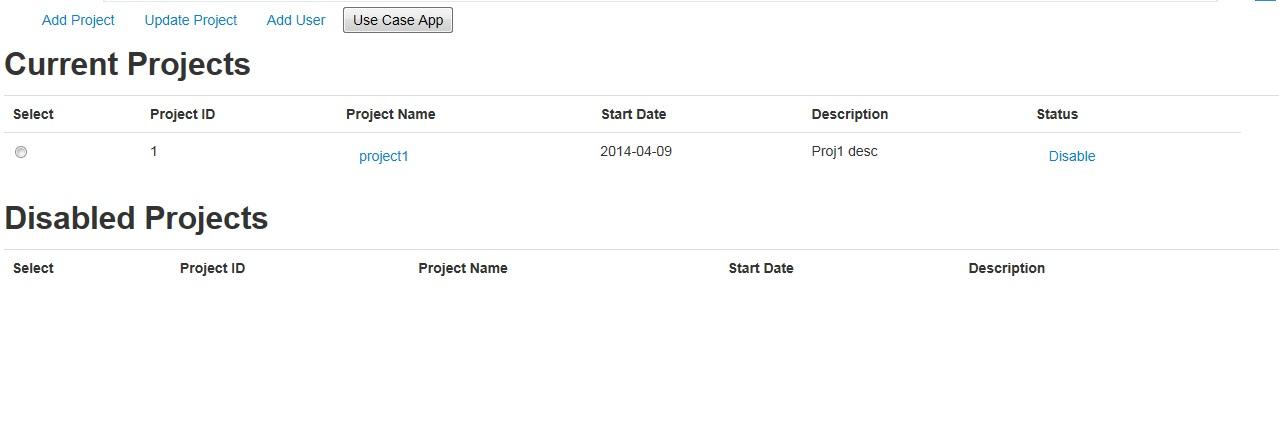
2) Now, select a project from the list of Projects.



3) Once in a project, hitting the “Project” tab should technically take you back to the list of projects page, but it doesn’t do this. Hit the “Project” tab again.



4) You land up at a page displaying just the JSP instead of reloading the servlet.



The temporary solution that has been implemented is, we have put a *Home* button on the bottom of such pages. A more sophisticated solution would be to redesign it, so that the servlets are reloaded with the right JSP.

## C. Suggested Development

(Yang)

1. Using Hibernate as the data persistence layer.

Hibernate is an ORM framework. So we can manipulate the database in an Object-Oriented way. The reasons for this suggestion are as follows:

By using Hibernate:

1. We don’t really have to care about creating the database.
2. We don’t need to deal with  the JDBC details such setting and closing  the     connection, writing code to get every field value from the domain object and passing them to the PreparedStatement object as SQL parameter manually.

When we  want to add a table to the database, we only need to add an entity class representing that table.

1. Use Hsqldb as the database server so we can embed the database server into the project. Anyone who wants to run the project won’t have to bother setting up the local database server.
2. Find a library or build one to convert the XMI file for the use case diagram to an image. So the system can display the use case diagram uploaded by the user.
3. Find ways to check or evaluate other types of UML diagram in addition to the currently supported Class, Sequence, and Use Case diagrams.
4. Try to find out how to support generic XMI files as generated by all tools. Now the system only supports specific UML tools.
5. Make the Front-end a template, and try to make it easier to add tabs or buttons to the front-end so that we don’t have to write JSP and HTML files from scratch.
6. Rationale History Documentation Generation (Use Case RM3):

Rationale *History* and Documentation shows a user-friendly linked history diagram of the *Decisions* that led to the current *Decision*. This history will include the *Rationale* for each *Decision*, so it can link the related *Issues, Criteria,* and *Alternatives* to each *Decision*. More details about this can be found in the [Rationale Use Case documentation](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/RationaleManagement/Rationale%20Management%20Use%20Cases.docx).

1. User groups should be extended to allow separate organizations to work on different groups of projects.
2. Authentication should be enforced so that a user cannot view any database details unless they are a member of the organization that owns those details.
3. The application should store cookies to avoid the necessity of re-login at the login page, and those cookies should expire after a certain idle period.
4. Roles should be extended to include at least administrators vs. users. Extensions could possibly include read/write privileges or different team member roles.
5. A new tab look and feel could be implemented using the TinyDropdown2 tabs as discussed in the Tab Modernization Research section of the Spring2014 Contributions>Navigation discussion above.
6. Validation of use case input could be added to check if it’s using the correct part of speech. For instance, a verb must be used for the action part of the use case. Also check the name of the use cases in the use case diagram that the user uploaded.

## D. Caveats and Pitfalls

(Sahil)

In the spring2014 semester, we made a lot of changes in the ClubUML project. These changes were not an easy task as there were many difficulties and it took a lot of trial and error to implement improvements. This section deals with the difficulties faced during the 2014 semester, and it discusses solutions and advice for handling these problems in the future.

### 1. Navigation Team

**Integration Issues:** One item the navigation team ran into issues with was under an integration effort with Smart Git. Sometimes when integrating branches that have diverged for quite some time, it can cause major conflicts. A big disconnect between two branches often causes merge conflicts in git if the same files have been edited in similar places. This can be resolved by double clicking on the merge conflict file in smart git. This brings up a comparison tool where the files from each branch are on the left and right, and you can use the tool to merge the appropriate changes into the file in the center. Once you are finished with this process, you can save the file and stage it to resolve the merge conflict. See [smartgit documentation](http://www.syntevo.com/smartgithg/documentation/5/show?page=commands-branch#resolve-conflicts) for more information.

### 2. Use Case Diagram Team

**File Formats:** The Use Case Diagram team encountered difficulty in dealing with differing file formats for UML diagrams that are exported from different tools. They responded to this by focusing on one particular file format as created by the StarUML tool. They would advise future teams to do initial diagram parsing work with one file type but construct your code in such a way that support for new file types can easily be added to the codebase.

**Testing Without UI:** The team found that it is more difficult to do testing without a proper user interface. Without this user interface, the team must construct stubs and manually create data that can input into backend portions of the codebase. Another way to avoid this is to create partial pieces of functionality that span from the front end to the backend rather than creating all of the backend pieces first.

**Algorithm Testing:** The Use case team found difficulty in implementing the evaluation Algorithm. During the start of the project, their algorithm was not supportive for evaluating use case diagram and it was deleting use case related inclusion/extension relationship, so they focused on implementing a new algorithm. They also set up a method, which will ask users to delete the related inclusion/extension relationship manually. Their advice for the future team is try to understand the theory of what you are going to build first, and then design a cautious architecture or algorithm for the program.

### 3. Rationale Management Team

**Synchronization:** Initially the Rationale management team faced problems of communication between team members which caused difficulties in synchronizing on a task. But as the semester progressed, the team implemented lightweight process and agile techniques to achieve their goals. They started the practice of having a group meeting in person in addition to informal communications. This practice helped their team members to get everyone on the same page and sort out any integration issues by assigning either individual work or working collaboratively. They starting breaking up their stories into smaller task and sub-tasks for easier integration. Their advice for future team is to have more in person meeting and break the larger task into smaller task.

**Code diving:** The Rationale team encountered the difficulties with JSP and CSS pages and found them to be very unorganized. For this problem, the team tried to cleanup the organization wherever it was possible. But where that wasn’t possible, the team used more focus and took their time to trace through the code and have a better understanding about the file relationships. Their advice for the future team is to do a deep dive in understanding for the code base and its organization at the start of the semester so that you have a better idea of the difficulty involved in planned tasks.

### 4. Validation Team

**Testing conditions:** The Validation Team encountered problems with javascript. In order to apply conditions to the fields, the team had to become familiar with the Javascript language. In previous iterations of the ClubUML project, there were no check conditions on various fields like password, username, etc. The team had to do a deep code dive in order to understand which field requires a conditional check. The team first tested the condition before implementing them in base line. The future recommendations for the teams will be to learn and get familiar with javascript and test condition before implementing them.

**Synchronization:** The Validation Team faced synchronization issues similar to those of the Rationale Management Team and would recommend similar actions in handling those problems.

### 5. Bug Fix Team

**Commenting:** During the process of bug fixing, it was often difficult to understand code that was not well commented. It’s much easier for to resolve the defects or create new extensions with detailed notes on the code. The notes do not need to be extremely detailed, but it would be useful to include some introductions such as what kind of library it uses, which servlet handles the page and how it is called, etc. For instance, because some servlets of this application were missing, the bug fix team had to create new servlets. Defects could be caught earlier and this type of problem could be avoided if the code was commented with some of these details. As such, good commenting should be kept in mind while creating new functionality, and it should be remedied where possible as legacy functionality is updated.

# Appendices

## Appendix A: Lessons Learned Documents

All lessons learned documents can be found in the repository at this folder:

<https://github.com/lmd59/ClubUMLSpring2014/tree/master/Documents/2014Spring/LessonsLearned>

## Appendix B: Rationale Management Documents

ClubUML Extension Proposal

<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/Miscellaneous/Extension%20Proposals/ClubUML%20Extension%20Proposal-LaurenD.docx> (click “RAW” to download)

Rationale Management Use Cases

<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/RationaleManagement/Rationale%20Management%20Use%20Cases.docx> (click “RAW” to download)

RM1 UI Design Document

<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/RationaleManagement/ClubUML-RM1%20UI%20Design%20Document.docx> (click “RAW” to download)

Design Document RM2

<https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/RationaleManagement/Design%20Document%20RM2.docx> (click “RAW” to download)

## Appendix C: Validation Documents

## Appendix D: Navigation Documents

[ClubUML File Dependencies.xlsx](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/ClubUML%20File%20Dependencies.xlsx) (Click to “RAW” download)

[ClubUML System Dependencies.pdf](https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/ClubUML%20System%20Dependencies.pdf) (Click to “RAW” download)

## Appendix E: Use Case Diagram Management Documents

## Appendix F: Bug Fix Documents

## Appendix G: Database scripts

Spring 2014 final database script can be found in the repository at <https://github.com/lmd59/ClubUMLSpring2014/blob/master/Documents/2014Spring/FullSystem/clubuml2014spring.sql> (Click to “RAW” download)