ClubUML

CSYE7945 Fall 2013

Final Report

Gan, Ying; Hajirnis, Aniket; Lee Seth; Misra, Aindra; Srinivasan, Aishwarya, Prasanna, Gautam;

Jaiswal, Siddhesh;Kong Weiqi; Jiayuan Song; Indrajit Kulkarni

December 2013

Contents

Revision History 5

Abstract 6

Project Overview 7

Problem Definition 7

Requirements 9

Project Guidelines 10

*Scrum:* 10

*Roles* 10

*Blackboard forum usage & participation, subscriptions:* 11

*Issue transparency and communication:* 11

*Meeting agendas & minutes:* 11

*Jira*: 12

*Development Environment Configuration:* 13

*Software Configuration Management:* 13

*Test cases track to requirements:* 14

Use Cases 15

Register New Account Use Case 15

Add Project Use Case 18

Basic Flow of Events 19

Alternative Flows 19

Maintain Project Use Case 20

Add Context Use Case 22

Maintain Context Use Case 24

Enable or Disable Context Use Case 27

Maintain Project Users Use Case 29

Upload Diagram Use Case for ClubUML 32

Display Diagram Use Case 34

Use Case - ClubUMLv3 – Compare Diagram 36

Manual Merge Diagram Use Case 39

Auto Merge Diagram Use Case 42

System Analysis 45

Overview of Previous System 45

Enhancement Overview 45

System Architecture 46

Assumptions and dependencies 46

Decisions, Constraints, and Justifications 46

Architectural Mechanisms 50

Layers of Architectural Framework 52

Architectural and Design Overview *(Gautam)* 53

Design and Implementation 57

Client-Side 58

Server-Side 63

Client-Server Side 69

Class diagrams 79

DB Classes 79

Domain classes 79

Servlet classes 81

Database 83

Test 88

Test Cases : 88

Test Results 94

Conclusion 98

Appendix 99

A. Smart Policy Comparison Algorithm Overview 99

B. Glossary: 104

C. Guidelines for future batches: 106

D. Guidelines and suggestions for future batches (Controller side) 106

E. Database Creation Scripts 107

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author(s)** | **Notes** |
| **2.0** | **12/11/2013** | **Ying Gan** | Finalize format |
| **1.0** | **12/02/2013** | **Ying Gan** | Detailed outline with some existed documentations |

# Abstract

This report documents the ClubUML project developed by the CSYE7945 Software Engineering Project class in the Fall 2013 semester. 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. An initial version of the ClubUML software was developed in Fall 2012 and extended by Sprint 2013, so the work this semester consisted of maintenance. 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.

# Project Overview

## Problem Definition

The following is an excerpt from the initial project requirements document provided by Professor Kokar at the beginning of the semester:

Large software engineering project teams include many (sometimes even thousands) of developers. Communication among the developers is a very important activity; it is absolutely necessary to communicate in order to achieve the project goals! However, at the same time, if the communication is inefficient, it can lead to both delays and the increase of the cost of the project. One of the aspects of communication is ensuring that communication events are properly driven by the need to make rational decisions. This is not an easy task since what looks rational to one person may look irrational to another. When developers disagree and spend a lot of time discussing their disagreements, the whole project may suffer. However, at the same time, we need to make sure that communication is occurring; that developers express their personal views on the project related issues. Free communication is needed to ensure that various ideas are explored. A rationale driven process needs to ensure that the best ideas are chosen.

Rationale management, communication, issue resolution and conflict resolution are topics in software engineering. These topics were covered in the textbook used for the Software Engineering class. For instance, Chapters 3 and 12 cover some of these topics. This textbook describes some of the procedures on the management rationale and managing communication processes. However, the textbook does not mention any of the tools that would support such processes. The charge for the Fall 2013 Software Engineering Project team is to work on such a tool.

Although the customer does not want to impose too many constraints on the solution approach, the problem to be solved, at least partially, falls within the “social network” domain. After all, a large team of software developers constitutes a network (network of people, not communication nodes). Thus some inspiration for this project can be sought in the social networks domain.

The social networks most people are familiar with are typically associated with social interactions – sharing pictures, stories, cooking recipes and such. Often social networks fulfill the (hidden) goals of the owners or investors–providing information about the people involved that can then be utilized by the merchants or politicians. At the same time, there are many (hidden) goals of the social network participants (e.g., showing off). The goal of the network for this project needs to be the improvement of the convergence of making decisions in the area of software engineering.

While in the known social networks people use such documents as photographs, in the tool for this project you need to focus on such documents as UML diagrams. UML is a huge language, so the team will have to focus on a small subset of UML to deal with. E.g., the team might decide to focus on just the sequence diagrams and on the support for the convergence of discussions about such diagrams. In one such scenario, developers would discuss (on the Internet) different versions of a sequence diagram stored in a UML tool, providing pros and cons for each. The tool developed in this project might be able

to read the UML versions of the two sequence diagrams, identify the common part of the two solutions (i.e., the part that all parties agree on), identify the differences, propose compromise solutions, and possibly more. The sky is the limit on the possible approaches to this problem.

The Fall 2012 and Spring 2013 Software Engineering Project teams have started work on such a tool. So your work will fall under the label of “maintenance” rather than development from scratch. The focus of the previous work was to support the exchange of ideas on class diagrams, and partially on sequence diagrams. The main use case for the system developed by the teams was that a user would upload a class or sequence diagram to the system’s database and then request the system to compare his/her diagram with any of the diagrams that have been uploaded before. The system then would identify “the same” elements in the diagram, as well as the elements that are in one of the diagrams but not in the other. This would serve as the feedback to the user, which would (hopefully) cause the user to modify his/her class diagram, or accept one of the diagrams in the database as a better one. The expectation is that the users would eventually converge on a diagram that reflects the consensus the team achieves. The real expectation is that such a tool would result in faster conversion, i.e., in a higher productivity of the software development team.

The customer would like to have a tool that supports rationale based decision support for resolving differences of opinions on all of the UML diagrams – from use case to deployment. Thus the current project can extend the work of the previous teams in two directions:

1. Covering more UML diagrams, and

2. Adding some “smarts” to the tool, so that the tool not only displays the differences between two class diagrams, but also suggests the aspects that are “better” in one of the diagrams than in the other.

The team is free to make selections of the types of diagrams they want to work on and on the extent of the smarts for assessing the particular diagrams. These decisions should be made rather early in the project, preferably in the second meeting of the class.

# Requirements

The following are requirements for the development tools and process defined by the syllabus at the start of the semester:

* Version control: Subversion
* Development environment: Eclipse
* UML - we need to decide on one tool and then use it throughout the project. The 2012 project did not have a standard UML tool; this should be fixed this time.
* Language: Java
* Group interaction: Blackboard. The instructor will create specific Discussion Boards on Blackboard, as needed.
* Process representation: OpenUP, developed under the EPF (the Eclipse Process Framework) effort

# Project Guidelines

## *Scrum:*

Scrum, an iterative agile framework, became the software development process used for the majority of the semester. Since most of us did not really know each other and had different work/school schedules, Scrum allowed for us to achieve the end goal together. There was no one person or group choosing what had to be done like there can be with a more traditional development approach. We were all learning the process, application and tools together.

We chose to treat this semester like one big sprint. This took out the sprint planning and sprint completion events that happen before and after a sprint. There just was not enough time to learn it and do all of this with what we had. Even though this led to some confusion and organizational difficulty, the team did pick up this limited Scrum process well. Stories were written for every week and tracked as they got completed in the backlog. Stories were added mid-week too as we saw new work that could be done.

We had limited assigned roles from the typical Scrum process. A lack of expert knowledge in this area led to this decision. We only had the key roles: a product owner (Professor Kokar), ScrumMaster, the project manager, and the development team. Some of the development team focused on particular areas like server development, analysis, database development and GUI development (see Figure 1 for details). Other roles we did not include were just auxiliary, helping more with sprint-to-sprint planning that we did not do.

## *Roles*

**Table 1: Team Member Scrum Roles**

|  |  |
| --- | --- |
| Scrum Role (Focus) | Team member name |
| Project Manager (Analysis/DB/Controller) | Ying G. |
| Scrum Master (Controller/DB/Algorithms) | Aniket H. |
| Team member (GUI) | Weiqi K. |
| Team member (Analysis/Algorithms) | Seth L. |
| Team member (Architecture/Algorithms) | Gautam C. |
| Team member (DB) | Jiayuan S. |
| Team member (Controller/Test) | Aindra M. |
| Team member (Algorithms) | Prasanna S. |
| Team member (GUI/Controller) | Aishwarya S. |
| Team member (Controller/GUI) | Siddhesh J. |
| Team member (Controller/Analysis) | Indrajit K. |

As a small team, we decided to have multiple roles in different sprints. Table 1 is the detail of the roles.

## *Blackboard forum usage & participation, subscriptions:*

Blackboard Discussion Board is the primary collaboration and communication tool for the ClubUML project, as it allows team members to communicate broadly and transparently, so that all project-related topics are open to review and feedback.

Team members are expected to regularly review postings, which can be accomplished by setting up Discussion Board Forum subscriptions. Information on setting up subscriptions can be found [here](https://blackboard.neu.edu/webapps/portal/frameset.jsp?url=%2Fwebapps%2Fblackboard%2Fcontent%2FlaunchLink.jsp%3Fcourse_id%3D_1851669_1%26tool_id%3D_137_1%26tool_type%3DTOOL%26mode%3Dview%26mode%3Dreset).

Team members will communicate any project-related content in the appropriate Discussion Board forum.

## *Issue transparency and communication:*

A key component to the success of any project is the timely communication of any project-related issues that may impede the project’s momentum. Not only are issues (technical, logistical, experiential and otherwise) inevitable, they are expected in any project.

Team members should openly communicate any issues for discussion in Blackboard’s Discussion Forums, so that not only the Project Manager and Scrum Master are aware of them, but also so that the team can suggest pathways to resolution.

## *Meeting agendas & minutes:*

A standard meeting agenda template should be followed for all scheduled meetings, and adherence to planned agenda topics and timelines should be followed, given the brevity of weekly project team meetings.

Requests for meeting topics for the weekly team meeting agenda should be posted on Blackboard in the General Discussions Forum by Wednesday afternoon at 12:00 PM EST to be discussed in the Wednesday project meeting.

Detailed meeting minutes will be captured for all meeting topics, with the exception of in-class Scrum meeting topics, which will be tracked by the Scrum Master. Meeting minutes should summarize the topics discussed, team members present/absent, key areas of discussion and salient observations, issues raised and all agreed-upon action items (including the assignee information).

Meeting minutes will be posted to the Blackboard Agenda & Minutes forum by Thursday evening at 21:00 EST.

## *Jira*:

Jira is the primary tool that the project team is using to manage the project backlog, maintain the details of the project sprint/iteration plan, track project progress and generate project metrics, hence, it is imperative that team members regularly review [Jira](https://clubuml.atlassian.net/) for updates to the project.

Team leads (or in some cases individual team members) will be assigned user stories and/or tasks in Jira. Team leads will create ‘Sub-Tasks’ in Jira to track individual work items, and will assign them to the team member that will be responsible for completing the sub-task. Once a team member begins to work on a sub-task, he or she should choose the ‘Start Progress’ option on that sub-task to indicate that the sub-task is now ‘In Progress.’ As team members spend time on a particular sub-task, they will track the effort spent since the last tracking of effort using the ‘More/Log Work’ option within a particular task.

Team members will provide status updates in user stories and tasks. Where applicable, any source code or other project artifacts that have been changed in conjunction with a Jira item should be cited in the item’s comments section. NOTE: A systematic connection between Jira and Git may be forthcoming which will automate this reference.

Team members may create new Jira issues of type ‘Bug’ to track any software defects that are identified during the course of development and/or testing. Newly identified bugs will be reviewed by the Project Manager and the project team and will be prioritized as a component of the backlog for potential development in an upcoming sprint.

Jira sub-tasks or user stories which have been completed will be marked as ‘Resolved’ by choosing the ‘Resolve Issue’ option within the item. The Project Manager will review resolved issues to determine which ones may be marked as closed.

User stories that have been previously closed but are later determined to be incomplete by the project team may be re-opened in Jira. The Project Manager will assign the story to an upcoming sprint based on review and prioritization of the backlog, and will assign an appropriate resource accordingly. A new task or tasks will be created and linked to the user story for any additional work items that need to be completed to address the story. (NOTE: This approach may be reconsidered in lieu of opening new stories and tasks and linking them to the original story.)

Navigate to Agile dashboard for items that are resolved and drag them to the Code Review. Sub-tasks, ultimately user stories.

## *Development Environment Configuration:*

Team members will use [Eclipse IDE](http://www.eclipse.org/downloads) to develop and test the ClubUML application.

Team members will run Apache Tomcat version 7 locally on their computers to locally developer and test the ClubUML application. Tomcat can be downloaded [here](http://tomcat.apache.org/download-70.cgi).

Team members will run an instance of the MySQL database locally on their computers to locally develop and test the ClubUML application. The script to create the ClubUML schema can be found in the Appendix section of this document. The script should be run using an account in MySQL, such as ‘root’, that has appropriate DBA privileges.

UML diagrams will be generated for use by the ClubUML application in the .ecore and Papyrus exported formats. .Ecore files can be generated using Eclipse Modeling Tool (Available for download [here](http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplerr)). Papyrus diagrams can be exported from the Papyrus UML tool (Available for download [here](http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=114&L=EN&ITEMID=16)). Team members developing or testing on operating systems other than Microsoft Windows may use a virtual machine running the MS Windows operating system to use Papyrus. More information can be found regarding the VMWare Fusion virtual machine [here](https://blackboard.neu.edu/webapps/portal/frameset.jsp?url=%2Fwebapps%2Fblackboard%2Fcontent%2FlaunchLink.jsp%3Fcourse_id%3D_1851669_1%26tool_id%3D_137_1%26tool_type%3DTOOL%26mode%3Dview%26mode%3Dreset).

Version control of project artifacts will be managed by the [Git](http://git-scm.com/) version control system. The primary project repository storing the artifacts can be found [here](https://github.com/ahajirnis/FallUML2013)). A [tutorial](http://www.vogella.com/articles/EGit/article.html) is also available that describes how to use the Eclipse Git plug-in, or the Git [console](http://git-scm.com/docs/gittutorial).

Whenever possible, for text-based artifacts, Markdown (.md) formatting will be used to author documents. This format is dynamically supported by Github, such that any documents posted in .md format will be displayed in HTML format in Github. More information on Markdown format can be found [here](http://daringfireball.net/projects/markdown/syntax).

## *Software Configuration Management:*

As described above, version control will be managed using the Git version control system. Team members will modify and test changes locally before committing any changes to the shared Git repository.

Git will also be used to place non-source artifacts under version control (e.g. use cases, design diagrams, etc.). These artifacts will be placed in a Documents folder within the FallUML2013 repository.

All changes must be successfully unit tested in the developer’s local Git environment, and must produce a clean build locally before committing any changes to the shared Git repository. Each modification will be tested under a branch named according to the Jira task associated with the change.

Once changes have been successfully tested and a clean build performed, the developer will perform a Git pull request to produce a differences report. He or she will provide commentary for the pull request indicating the nature of the changes and the Jira user story(ies) and/or tasks associated with the change.

The developer will also post a new Blackboard Discussion Board thread in the Version Control and Change Management forum notifying the team of the pull request, and providing an opportunity for the team to perform a code review of the changes. A code review should be performed minimally by the team lead and team peers, comprised of at least 2 developers. Code review feedback from the general team is encouraged, time permitting. All code review feedback will be posted in the Blackboard Discussion Board version control thread associated with the change, and also is recommended in the Git differences review comment section.

With code reviews performed, changes may be committed with a merge into the main 'master' Git repository branch.

Additional information regarding Git best practices can be reviewed in the project's Github repository under the FallUML2013/Documents/2013Fall/Best Practices/ folder.

## *Test cases track to requirements:*

Test cases will reference the Jira user story to which the test case is related. All test cases will include a link to the URL for the Jira user story being tested.

# Use Cases

Note: Use cases can also be viewed in Github in the FallUML2013/Documents/2013Fall/Use Cases/ folder.

## Register New Account Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.1 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| Spring 2013 | 1.0 | Initial draft | Spring 2013 Team |
| 10.14.2013 | 1.1 | Updated for user roles. | Seth Lee |

Brief Description

This use case describes how a user registers a new account with ClubUML.

Actors

1 User

2 Relational Database

Preconditions

The User has a turned-on computer with internet access and a browser installed.

The User does not currently have an account with ClubUML.

Basic Flow of Events

1. The use case begins when the User opens a web browser, inputs the address of ClubUML, and hits enter.
2. The User clicks the “register’ link
3. The web page prompts the user for a user name, password, a repeat of the password, an email address, a security question and finally the security question answer. The user also selects a user role, either a developer (regular user) or policy manager (administrator).
4. The User fills in all of the fields, and then clicks submit. 5 app will provide three pre define question to the user and user needs to choose any of them from the drop down box

6.User need to enter a answer into the security answer field 7. The username and password are sent to the ClubUML server, verified, and stored in a database. 8 app will shore security question and the corresponding answer to the database 9. The web page tells the User that the account has been successfully created. 10. The web page prompts the User to log in. 11. The use case ends successfully.

Alternative Flows

Invalid Username

If in step 5 of the basic flow the username does not meet the username requirements (see WC-1), then

1. The web page prompts the user that the username does not meet a requirement, and which.
2. The web page clears all fields.
3. The use case ends with a failure condition.

User Name Already Exists

If in step 5 of the basic flow the username is found to already exist, then

1. The web page prompts the User that the username already exists.
2. The web page clears all fields.
3. The use case ends with a failure condition.

Invalid Password

If in step 5 of the basic flow the password does not meet the password requirements (see WC-2), then

1. The web page prompts the user that the password does not meet a requirement, and which.
2. The web page clears all fields.
3. The use case ends with a failure condition.

Passwords Don’t Match

If in step 5 of the basic flow the password and the repeat password do not match, then

1. The web page prompts the user that the password and the repeat password do not match.
2. The web page clears the password and repeat password.
3. The use case ends with a failure condition.

Cancel

If in any step before step 4 the User presses cancel, then

1. The web page will return to the home page.
2. The use case ends.

Empty Field

If in step 4, any of the fields are not filled in, then

1. The web page prompts the user that the offending field is empty, and is required to proceed.
2. The web page retains all other fields filled in. The submit button will not work until field is filled in.
3. The use case continues when the field is filled in.

Key Scenarios

1 User Name Already Exists

**Post**-conditions

**Successful Completion**

The user has created an account and is able to login.

**Failure Condition**

No new account has been created.

**Special Requirements**

[SpReq:WC-1] The username shall be 1-6 characters long, with only the following characters [a-zA-Z0-9] [SpReq2:WC-2] The password shall be 6-10 characters long, with only the following characters [a-zA-Z0-9].

## Add Project Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 10.20.2013 | 1.0 | Initial draft | Seth Lee |

**Brief Description**

This use case describes how the policy manager adds project information in the application.

**Actors**

Policy Manager : The policy manager is the actor who maintains project, context and policy information in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

User: The user is the actor who is assigned to a project(s), who may upload diagrams to a project, compare diagrams or merge diagrams in a project. Users may upload diagrams, etc. to a project to which they are assigned, but not to projects to which they are not assigned. Users may only upload diagrams, etc. to an enabled project.

**Preconditions**

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

## Basic Flow of Events

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default).
2. The policy manager clicks the 'Add Project' button to add a new project.
3. The policy manager is presented with an entry screen that includes input fields for the project detail.
4. The policy manager enters the project name, description, and other project details.
5. The policy manager chooses the project's enabled status.

5.1 The policy manager chooses the 'Enabled' option if the project is immediately enabled, allowing users to add diagrams to the project.

OR

5.2 The policy manager chooses the 'Disabled' option if the project is being added but is not being enabled to allow users to upload diagrams to the project.

1. The policy manager decides to save or abort the addition of the new project.

6.1 The policy manager clicks the 'Save' button to save the project. The new project is added to the database.

OR

6.2 The policy manager clicks the 'Cancel' button to abort saving the project. The new project is not added to the database.

1. The policy manager is taken back to the welcome page displaying the projects in the Project tab. The new project and its attributes are visible on the welcome page (e.g. project name, etc.).

## Alternative Flows

Key Scenarios

Post-conditions

Successful Completion

If the policy manager added a project and subsequently saved the new project, it appears in the project summary of the welcome page for the project that was added. The new project is saved to the database. Users and policy managers may now choose the new project from the welcome page to upload diagrams, merge diagrams, etc.

If the policy manager added a project and subsequently chose to cancel the request in the add project screen, the new project does not appear in the welcome page, nor is it saved in the database.

Failure Condition

No changes are made to the database or the welcome page.

Special Requirements

## Maintain Project Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 10.20.2013 | 1.0 | Initial draft | Seth Lee |

Brief Description

This use case describes how the policy manager maintains project information in the application.

Actors

Policy Manager : The policy manager is the actor who maintains project, context and policy information

in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

Preconditions

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

Basic Flow of Events

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default).
2. The policy manager highlights the project that he or she is maintaining and clicks the 'Maintain Project' button.
3. The policy manager is presented with a review screen that displays the project details for the project.
4. The policy manager edits the name or other project attributes.

4.1 The policy manager edits the attributes of the project.

OR

4.2 The policy manager disables an active project that is currently enabled.

OR

4.3 The policy manager enables a deactivated project that is currently disabled.

1. The policy manager commits or aborts changes to the project.

5.1 The policy manager clicks 'Save' to save his or her changes.

5.1.1 The changes are saved to the project table in the database. OR

5.2 The policy manager clicks 'Cancel' to cancel any changes to the project.

5.2.1 No changes are committed to the project table in the database.

1. The policy manager is taken back to the welcome page displaying the projects in the Project tab. Any changes that were made to project attributes that are visible in this screen are now visible on the welcome page (e.g. project name, etc.).

Alternative Flows

Key Scenarios

Post-conditions

Successful Completion

If the policy manager made changes to a project and subsequently saved them, they appear in the project summary of the welcome page for the project that was maintained. Any changes made to the project have also been applied in the database.

If the policy manager made changes to a project and subsequently chose to cancel the request in the project maintenance screen, the project appears in the welcome page as it was before the request, and no changes have been applied to the database.

Failure Condition

No changes are made to the database or the welcome page.

Special Requirements

## Add Context Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 10.20.2013 | 1.0 | Initial draft | Seth Lee |

Brief Description

This use case describes how the policy manager adds context information in the application.

Actors

Policy Manager : The policy manager is the actor who maintains project, context and policy information in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

User: The user is the actor who is assigned to a project(s), who may upload diagrams to a project, compare diagrams or merge diagrams in a project. Users may upload diagrams, etc. to a project to which they are assigned, but not to projects to which they are not assigned. Users may only upload diagrams, etc. to an enabled project.

Preconditions

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

Basic Flow of Events

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default).
2. The policy manager clicks the 'Context' tab to display a list of existing contexts.
3. The policy manager clicks the 'Add Context' button to add a new context.
4. The policy manager is presented with an entry screen that includes input fields for the context detail.
5. The policy manager enters the context name, description, and other context details.
6. The policy manager chooses the project with which the context will be associated.

6.1 The policy manager clicks the Project drop-down list and selects an **enabled** project.  
Only **enabled** projects are displayed in the list of projects to which the context may be assigned.

1. The policy manager decides to save or abort the addition of the new context.

6.1 The policy manager clicks the 'Save' button to save the context. The new context is added to the database.

OR

6.2 The policy manager clicks the 'Cancel' button to abort saving the context. The new context is not added to the database.

1. The policy manager is taken back to the welcome page displaying the contexts in the Context tab. The new context and its attributes are visible on the welcome page (e.g. context name, etc.).

Alternative Flows

Key Scenarios

Post-conditions

Successful Completion

If the policy manager added a context and subsequently saved the new context, it appears in the context summary of the welcome page for the context that was added. The new context is saved to the database. Users and policy managers may now choose the new context from the project screen as a drop-down option when they upload new diagrams.

If the policy manager added a context and subsequently chose to cancel the request in the add context screen, the new context does not appear in the welcome page in the Context, nor is it saved in the database. Users and policy managers do not see the context listed in the project screen as a drop-down option when uploading new diagrams.

Failure Condition

No changes are made to the database or the welcome page in the Context tab. The new context is not available in the project screen as a drop-down option when users or policy managers upload new diagrams.

Special Requirements

## Maintain Context Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 10.22.2013 | 1.0 | Initial draft | Seth Lee |
| 11.6.2013 | 1.1 | Updated for disabled context functionality. | Seth Lee |

Brief Description

This use case describes how the policy manager maintains context information in the application.

Actors

Policy Manager : The policy manager is the actor who maintains project, context and policy information in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

User: The user is the actor who is assigned to a project(s), who may upload diagrams to a project, compare diagrams or merge diagrams in a project. Users may upload diagrams, etc. to a project to which they are assigned, but not to projects to which they are not assigned. Users may only upload diagrams, etc. to an enabled project.

Preconditions

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

There is at least one enabled Context associated with a Project to be maintained.

Basic Flow of Events

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default). Contexts appear as either enabled or disabled.
2. The policy manager clicks the 'Context' tab to display a list of existing enabled contexts.
3. The policy manager highlights the Context he or she is maintaining and clicks the 'Maintain Context' button to maintain an existing context.
4. The policy manager is presented with an entry screen that displays the details for the Context being maintained.
5. The policy manager modifies the context information:

5.1 The policy manager modifies the context name, description, other other context details as desired.

5.2 The policy manager chooses the project with which the context will be associated.

5.2.1 The Project drop-down list defaults to the Project with which the context is currently associated. The policy manager clicks the Project drop-down list and selects an **enabled** project.  
Only **enabled** projects are displayed in the list of projects to which the context may be assigned.

1. The policy manager decides to save or abort the addition of the new context.

6.1 The policy manager clicks the 'Save' button to save the context modifications. The modified context information is updated in the database.

OR

6.2 The policy manager clicks the 'Cancel' button to abort saving the context modifications. The updated context is not applied to the database.

1. The policy manager is taken back to the welcome page displaying the contexts in the Context tab. The context and its attributes are visible on the welcome page (e.g. context name, etc.).

Alternative Flows

Key Scenarios

Post-conditions

Successful Completion

If the policy manager modified context information and subsequently saved the modified context, it appears in the context summary of the welcome page for the context that was modified. Any changes to the context such as name or description that are visible in the welcome page Context tab are reflected in the summary. The updated context information is saved to the database. If the context modification included a change to the project with which the context is associated, then users and policy managers see the context as a drop-down in the project screen for the newly associated project when uploading diagrams.

If the policy manager modified context information and subsequently chose to cancel the request in the maintain context screen, the updated context information does not appear in the welcome page in the Context tab, nor is it saved in the database.

Failure Condition

No changes are made to the database or the welcome page in the Context tab.

Special Requirements

## Enable or Disable Context Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 11.4.2013 | 1.0 | Initial draft | Seth Lee |

**Brief Description**

This use case describes how the policy manager enables disabled contexts in the application, or disables enabled contexts in the application.

**Actors**

Policy Manager : The policy manager is the actor who maintains project, context and policy information in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

User: The user is the actor who is assigned to a project(s), who may upload diagrams to a project, compare diagrams or merge diagrams in a project. Users may upload diagrams, etc. to a project to which they are assigned, but not to projects to which they are not assigned. Users may only upload diagrams, etc. to an enabled project.

**Preconditions**

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

There is at least one Context associated with a Project to be enabled or disabled.

**Basic Flow of Events**

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default). Contexts appear as either enabled or disabled. Enabled Contexts appear in the enabled Contexts (upper) portion of the screen. Disabled Contexts appear in the disabled Context (lower) portion of the screen.
2. The policy manager clicks the 'Context' tab to display a list of existing contexts.
3. The policy manager chooses an enabled Context to disable, or a disabled Context to enable:

3.1 The policy manager selects an enabled Context to disable:

3.1.1 The policy manager clicks the 'Disable' link on the row containing the desired Context:

* If there are no Diagrams associated with the Context, the Context is marked as disabled in the database.
* If there are Diagrams associated with the Context, the Context cannot be disabled. The policy manager receives an error message from the application indicating that the Context cannot be disabled because there are Diagrams associated with the Context.

3.1.2 The policy manager clicks the 'Enable' link on the row containing the desired Context:

* The Context is marked as enabled in the database.

1. The Context shows as either enabled or disabled in the Context panel. If the Context was enabled, it appears in the upper portion of the screen in the enabled Contexts list. If the Context was disabled (successfully), it appears in the lower portion of the screen in the disabled Contexts list.

**Alternative Flows**

**Key Scenarios**

**Post-conditions**

**Successful Completion**

If the Policy Manager disabled an enabled Context and there are no Diagrams associated with the Context, then the Context appears in the lower portion of the Context panel in the disabled Contexts list. The database is updated for the Context to reflect that it has been disabled.

If the Policy Manager enabled a disabled Context, it appears in the upper portion of the Context panel in the enabled Contexts list. The database is updated for the Context to reflect that it has been enabled.

**Failure Condition**

No changes are made to the database or the welcome page in the Context tab.

**Special Requirements**

## Maintain Project Users Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.0 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| 11.5.2013 | 1.0 | Initial draft | Seth Lee |

**Brief Description**

This use case describes how the policy manager maintains users associated with projects in the application. Users associated with a project will see the project listed on the welcome page, where they can proceed to the project to upload diagrams, compare diagrams, etc.

Once a user is removed from a project, he or she is no longer able to access that project in the application. Please note, however, that any diagrams, comments, etc. that are associated with the user in the project remain in the database.

**Actors**

Policy Manager : The policy manager is the actor who maintains project, context, policy and project user information in the application. The policy manager is a special type of user identified by the application by a policy manager role, not a typical user role.

User: The user is the actor who is assigned to a project(s), who may upload diagrams to a project, compare diagrams or merge diagrams in a project. Users may upload diagrams, etc. to a project to which they are assigned, but not to projects to which they are not assigned. Users may only upload diagrams, etc. to an enabled project.

**Preconditions**

The policy manager has successfully logged in to the application.

The policy manager has the appropriate user role designation in the database.

At least one user exists in the application.

**Basic Flow of Events**

1. The policy manager is directed to the application welcome page, which displays a list of the projects that have been defined in the application in the Project tab (default).
2. The policy manager identifies the project for which he or she wishes to maintain user associations. The policy manager clicks the 'Maintain Users' link for that project.
3. The policy manager is directed to the project user maintenance screen. A list of users that have been assigned to the project is displayed.
4. The policy manager identifies users to be added to or removed from the project:

* The policy manager removes existing project users:

\*\* The policy manager clicks the 'Remove' link for the user to be removed.  
\*\* The policy manager is presented with a confirmation dialogue to confirm the changes.

\*\*\* The policy manager clicks 'Ok' to save the changes. \*\*\*\* The user is removed from the list of users associated with the project. The database is updated to mark the project user association as disabled if any diagram or comment associations exist for the user in the project.  
The database is updated to delete the project user assocation if no diagram or comment associations exist for the user in the project.

\*\*\* The policy manager clicks 'Cancel' to cancel the changes. \*\*\*\* The user is not removed from the list of users associated with the project. No updates are performed in the database.

* The policy manager adds project users:

\*\* The policy manager clicks the 'Add User' link. \*\* The policy manager is presented with a list of registered users that are not already associated with the project.

\*\* The policy manager chooses the appropriate user and clicks 'OK':

\*\*\* The user is added to the list of users associated with the project.  
The database is updated to associate the user with the project.

\*\* The policy manager does not choose a user and clicks 'Cancel' to cancel any changes:

\*\*\* The user is not added to the list of users associated with the project.  
No database updated are performed.

The policy manager repeats step 4 as needed to add or remove users from the project.

1. The policy manager is taken back to the welcome page displaying the projects in the Project tab.

**Alternative Flows**

**Key Scenarios**

**Post-conditions**

The list of users associated with the project is updated to reflect the policy manager's modifications.

Users that have been added by the policy manager are able to access the project from the welcome page in the application, and may upload diagrams, compare diagrams, etc. for the project.

Users that have been removed from a project are not able to access the project from the welcome page in the application.

Diagrams, comments or other associations with the user in the project are retained in the database for the project.

**Successful Completion**

The list of users associated with the project is updated to reflect any users that have been added to or removed from the project.

The database is updated to add new associations between the project and any users that are added to the project.

The database is updated to update associations between the project and any users that are removed from the project to mark the user as disabled in the project if the user has uploaded diagrams or is associated with comments.

The database is updated to delete associations between the project and any users that are removed from the project if the user is not associated with any diagrams or comments for the project.

**Failure Condition**

No changes are made to the database.

**Special Requirements**

## Upload Diagram Use Case for ClubUML

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.1 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| Spring 2013 | 1.0 | Initial draft | Spring 2013 Team |
| 10.14.2013 | 1.1 | Updated for Context functionality. | Seth Lee |
| 10.23.2013 | 1.2 | Updated for soft delete functionality. | Seth Lee |
| 11.5.2013 | 1.3 | Updated for disabled context functionality. | Seth Lee |

**Brief Description**

This use case describes how a user uses ClubUML to upload a UML diagram encoded in Ecore data model

**Actors**

1. User
2. Database Server
3. File System on Server

**Preconditions**

1. The user is logged in to ClubUML.
2. The user has a UML diagram encoded in Ecore or Papyrus to upload.
3. The user has selected a project from the welcome page that is enabled.
4. At least one enabled Context has been defined for the Project in which the user is working.

**The Basic Flow of Events**

1. The use case begins when the user selects “Upload” button
2. ClubUML displays a dialog box for the user to browse for a file.
3. The user browses and selects a file.
4. The user selects a Context from the Context drop-down box for the diagram. Only enabled contexts will be displayed in the drop-down list.
5. ClubUML uploads the file to the File System on Server from the user’s computer and renames the file by using “userId\_timeStamp\_originalFileName”.
6. ClubUML sends full path to the Database Server.
7. ClubUML creates an image of the UML diagram through internal parser. If successful, the image is stored off to File System on Server, the image path is send to the Database Server. Context information for the diagram is also stored in the database.
8. ClubUML displays the “project page” to the user with the new created image for visual verification by the user. The newly uploaded diagram appears in the list of uploaded diagrams, along with the Context of the diagram.
9. The use case ends successfully.

**Alternative Flows**

**No UML File**

If in step 3 of the basic flow the user does not have a UML file:

1. The user selects “CANCEL” on the page to browse for a file.
2. ClubUML brings the user back to the project page (the page the user was on when he selected “Upload” button
3. The use case ends in a failed condition.

**User does not select a Context drop-down**

ClubUML issues an error message indicating that the user must select a Context for the diagram prior to uploading it.

**User selects a disabled project from the welcome page**

The user is presented with the list of diagrams that have been previously uploaded in read-only mode. The user is unable to upload diagrams, compare diagrams, etc.

**Wrong file extension**

If in step 3 of the basic flow the user chooses a file that does not have the correct extension, then

1. ClubUML indicates an error and reminds the user of the correct file extension.
2. The user clicks on “OK”
3. The use case resumes at step 2 of the basic use case.

**File is not able to be converted to a .PNG file**

If after step 6, the file is not able to be converted to a .PNG file, then

1. The user is prompted that the file is unable to be displayed, and is told that the ecore file cannot be saved.
2. The “ecore” file is removed from the database and does not show up on the list of files. However the history of this user trying to upload a file that is rejected is saved.
3. The use case resumes at step 2.

**No Response from Server**

If in step 6 of the basic there is no response from the server within 3 seconds, then

1. ClubUML will re-try, up to three times.
2. If there is still no response from the server ClubUML shall display the message "Network unavailable – try again later".
3. The use case ends with a failure condition.

**Key Scenarios**

The basic flow of events

**Post-conditions**

**Successful Completion**

## Display Diagram Use Case

|  |  |  |  |
| --- | --- | --- | --- |
| Document Type | | Use case | |
| Activity | | Analysis | |
| Document Version | | 1.1 | |
| Document Status | | Draft | |
| Date | **Version** | **Description** | **Author** |
| Spring 2013 | 1.0 | Initial draft | Spring 2013 Team |
| 10.14.2013 | 1.1 | Updated for Context functionality. | Seth Lee |
| 11.5.2013 | 1.2 | Updated to include removed users functionality | Seth Lee |

**Brief Description**

This use case describes how the application display diagrams to the user.

**Actors**

1 User : The user is the actor who reviews previously uploaded UML diagrams to be displayed.

**Preconditions**

The user has successfully logged in to the application.

The user is assigned to at least one project.

The user selects a project to which he or she is currently assigned, and may not select a project from which he or she has been removed.

The project must contain at least one diagram.

**Basic Flow of Events**

1. The app shows a 'Diagram List'. Along with the name of the diagrams that are available for display, each diagram will include the identifier of the Context with which the diagram is associated.
2. The user clicks one diagram from that list.
3. The app retrieves the path of png. file related to this diagram from Database.
4. The app finds the file on server according to the path.
5. The app displays the Diagram to the user.
6. The use case ends in successful completion.

**Alternative Flows**

**No diagram assigned to the project**

During 4.2, if the project has no diagrams, then

1. The user will see an empty 'Diagram List'.
2. The app prompts the user to upload a diagram first.
3. The use case will end in failure condition.

**The app fails to display the selected diagram**

If in basic step 5, the app fails to display the selected diagram:

1. The user refreshes the webpage or selects another diagram.
2. If the problem remains, the use case will end in failure condition and the user would start over the procedure.

**Key Scenarios**

**No Response from the app**

**Post-conditions**

**Successful Completion**

The user sees the selected diagram.

**Failure Condition**

The app fails to display the diagram.

**Special Requirements**

If the selected diagram has comments, the app should display comments as well.

The app should also display the information of the version and uploaded time.

## Use Case - ClubUMLv3 – Compare Diagram

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document Type | | | Use case | | |
| Activity | | | Analysis | | |
| Document Version | | | 3.2 | | |
| Document Status | | | Draft | | |
| Version history | **Date** | **Version** | | **Description** | **Author** |
| 09.17.2013 | 3.0 | Copied over from Spring2013 | | ClubUml |  |
| 09.18.2013 | 3.1 | Amended to have user choose between ClubUML recommendation promotion or manual and accommodate only manual compare. | | Seth |  |
| 09.24.2013 | 3.2 | Updated to accommodate both manual and auto promotion in one Use Case | | Gautam |  |
| 10.7.2013 | 3.3 | Updated to include Context functionality | | Seth |  |
| 10.23.2013 | 3.4 | Updated to include Project soft delete functionality. | | Seth |  |

**Brief Description**

This use case portrays the comparison of two diagrams selected by the user, with support for comments to aid in collaboration.

**Actors**

User: The user is the actor who invokes the comparison and reviews the report.

**Preconditions**

1. The user follows Login use case flow to access ClubUML.
2. The user has already uploaded two valid diagrams of the same type (following UploadDiagram flow of events), and also of the same Context.
3. The user has selected a project from the welcome page that is enabled.

**Flow of events**

1. The user follows Login use case flow to access ClubUML.
2. The user selects two diagrams from the list of diagrams in the project, which are the same type as well as Context, and have been output by the same piece of software to ensure compatibility.
3. The user clicks “Go to Compare” button for the selected diagram of the same type.
4. ClubUML generates a detailed comparison report in a display panel within the web page including the Smart/Policy Scores of each diagram.
5. If the User clicks the ‘Suggest Promote’ button to allow ClubUML to use smart policy to identify which diagram to promote, ClubUML highlights the diagram that is suggested and the display panel indicates why the recommendation was made including the refactoring score of each diagram.
6. The user has the option to click the “save” button to download a PDF report of the comparison including the details of the ClubUML recommendation if it was invoked.
7. The user decides which diagram is preferred and adds a comment under the preferred diagram.
8. The user then clicks the Promote button next to the comment box.
9. The user is returned to the main ClubUML page and the comment can be seen under the promoted diagram. The comment will have an indicator to notify if it was a ClubUML ‘smart policy’ recommendation or not.

**Alternate Flow**

**User selects less than two diagrams**

If the user selects less than two diagrams the system alerts with a pop-up saying, “Please select at least two diagrams”

**User selects more than two diagrams**

If the user selects more than two diagrams the system alerts with a pop-up saying, “Please select at most two diagrams”

**User selects a project from the welcome page that is disabled**

If the user selects a project from the welcome page that is disabled, then he or she will be presented with a list of diagrams that have been previously uploaded to the project in read-only mode. The user will be unable to compare any diagrams, merge diagrams, or upload any diagrams.

**No Response from Server**

If in process of preparing the comparison report there is no response from the server, then:

1. ClubUML application shall display the message "Network unavailable – try again".
2. The use case ends with a failure condition.

**Key Scenarios**

If the button “save” has not been clicked, no file would be saved.

**Post-conditions**

1. Successful Completion

A detailed comparison report will be generated.

1. Failure Condition

No comparison report is generated.

## Manual Merge Diagram Use Case

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document Type | | | Use case | | |
| Activity | | | Analysis | | |
| Document Version | | | 3.2 | | |
| Document Status | | | Draft | | |
| Version history | **Date** | **Version** | | **Description** | **Author** |
| 09.17.2013 | 3.0 | Copied over from Spring2013 , renamed it from ‘MergeDiagram’ to ‘Manual-Merge Diagram’ Use case | | ClumUml |  |
| 09.18.2013 | 3.1 | Updated to fit into the new auto ‘smart’ merge paradigm. | | Gautam Chongtham |  |
| 10.7.2013 | 3.2 | Updated to include Context functionality. | | Seth Lee |  |
| 10.23.2013 | 3.3 | Updated to include Project soft delete functionality. | | Seth Lee |  |

**Brief Description**

This use case describes a manual merge functionality, in which the user selects which elements to keep from two related diagrams.

**Actors**

User: The user selects the diagrams to merge and makes decisions on merging.

**Preconditions**

1. The user follows Login use case flow to access ClubUML.
2. The user has already uploaded two valid diagrams of the same type (following UploadDiagram flow of events) and Context.
3. The uploaded diagrams to be merged are from Papyrus (no Ecore merge support).
4. The user has selected a project from the welcome page that is currently enabled.

**Basic Flow of Events**

1. The user selects two diagrams of the same type (class or sequence) and Context, both Papyrus files, to be merged.
2. The user clicks the “merge” button.
3. The user is prompted to choose either ‘Auto-Merge’ or ‘Manual Merge’. The user chooses ‘Manual-Merge’.
4. The webpage shows the two diagrams side by side. Under each diagram, there is a list of: a. Classes in diagram A only. b. Classes in diagram B only. c. Classes in common with diagram A and diagram B.
5. The user can select any pair of classes to merge if they are not already in common with both diagrams, e.g. class Bike in diagram A and class Bicycle in diagram B.
6. The user makes the selection of which elements to be kept in the merged class by checking a box (or some similar type of input) next to each desired element.
7. The user clicks another “merge” button to merge the class.
8. The user will be prompted to resolve any conflicts with associations in the newly merged class by picking which type of association is appropriate in each situation.
9. The user can repeat steps 4-7 to merge other classes which are not in common.
10. When the user is finished merging classes, the user clicks another Merge button to finalize the process and generate the overall merged diagram.
11. The UI displays a new merged diagram with the versions of each element combined.
12. The user can save the merged diagram.
13. The user can export Papyrus files of the merged diagram (see DownloadProject use case).
14. The user can comment on the merged diagram.
15. The user can return to the main ClubUML page after doing any combination of saving the drawing, exporting the merged diagram, or commenting.
16. The merged diagram will appear on the main ClubUML page along with the other diagrams that have been uploaded.

**Alternative Flows**

**Incorrect type of diagrams**

If user selects two different types of diagrams, then:

1. An error dialog will be prompted and ask the user to select two diagrams with the same type.
2. The use case returns to basic flow step 1.

**Number of diagrams error**

If user selects more or less than two diagrams, then:

1. An error dialog will be prompted and ask the user to select more or less diagrams to be merged.
2. The use case returns to basic flow step 1.

**Two exact diagrams are selected**

If user selects two diagrams that are exactly the same, then

1. A dialog will be prompted notifying user that the two diagrams are the same. Neither selections of merging nor saving of the diagram is allowed.
2. The use case returns to basic flow step 1.

**The user selects a project that is currently disabled from the welcome page**

If the user selects a project from the welcome page which is currently disabled, then he or she is presented with a list of diagrams in the project display page that have been previously been uploaded in read-only mode. The user is unable to upload any additional diagrams, merge or compare diagrams, etc.

**Subflows**

**No merging**

1. The user doesn’t make the request to merge diagrams
2. The user can return to the main ClubUML page, and the use case ends.

**No saving diagram**

1. The user chooses not to save an image of the merged diagram.
2. No saving will be done.

**No exporting XMI**

1. The user chooses not to export an XMI file of the merged diagram.
2. No exporting will be done.

**No commenting**

1. The user chooses not to leave any comment.
2. No comment will be added to the diagram.

**Key Scenarios**

No diagrams will be saved unless the user makes the request.

**Post-conditions**

1. The merged diagram is saved with comments, if any.
2. No diagrams are lost.

**Special Requirements**

The merged diagram has the same type as the two original diagrams.

The merged diagram can be merged again with other diagrams, if the user makes the request.

## Auto Merge Diagram Use Case

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 9.18.2013 | 3.0 | Created by deriving it from the existing Use Case for MergeDiagram in Spring2013 version | Gautam Chongtham |
| 10.7.2013 | 3.1 | Updated to include Context functionality | Seth Lee |
| 10.23.2013 | 3.2 | Updated to include Project soft delete functionality | Seth Lee |

**Brief Description**

This use case describes a manual merge functionality, in which the user selects which elements to keep from two related diagrams.

**Actors**

User: The user selects the diagrams to merge and makes decisions on merging.

**Preconditions**

1. The user follows Login use case flow to access ClubUML.
2. The user has already uploaded two valid diagrams of the same type (following UploadDiagram flow of events), as well as of the same Context.
3. The uploaded diagrams to be merged are from Papyrus (no Ecore merge support).
4. The user has selected a project from the welcome page that is enabled.

**Basic Flow of Events**

1. The user selects two diagrams of the same type (class or sequence) and Context, both Papyrus files, to be merged.
2. The user clicks the “merge” button.
3. The user is prompted to choose either ‘Auto-Merge’ or ‘Manual Merge’. The user chooses ‘Auto-Merge’.
4. The user is given the option to change default settings and preferences for the auto-merge algorithms.
5. The user clicks another “merge” button to merge the class.
6. The UI displays a new merged diagram with the versions of each element combined.
7. The user can save the merged diagram.
8. The user can export Papyrus files of the merged diagram (see DownloadProject use case).
9. The user can comment on the merged diagram.
10. The user can return to the main ClubUML page after doing any combination of saving the drawing, exporting the merged diagram, or commenting.
11. The merged diagram will appear on the main ClubUML page along with the other diagrams that have been uploaded.

**Alternative Flows**

#### Auto-merge is not able to resolve conflicts

If auto-merge is not able to resolve some conflicts, then: 1. The user will be prompted any unresolved conflicts with associations, if any, in the newly merged. Also a message saying manual merge is necessary for such scenarios. 2. The use case ends here.

#### Incorrect type of diagrams

If user selects two different types of diagrams, then: 3. An error dialog will be prompted and ask the user to select two diagrams with the same type. 4. The use case returns to basic flow step 1.

#### Number of diagrams error

If user selects more or less than two diagrams, then: 1. An error dialog will be prompted and ask the user to select more or less diagrams to be merged. 2. The use case returns to basic flow step 1.

#### Two exact diagrams are selected

If user selects two diagrams that are exactly the same, then 1. A dialog will be prompted notifying user that the two diagrams are the same. Neither selections of merging nor saving of the diagram is allowed. 2. The use case returns to basic flow step 1.

**User selects project from the welcome page that is disabled**

If the user selects a project from the welcome page that is currently disabled, then he or she will be presented with a list of diagrams that have been previously uploaded to the project in read-only mode. The user will be unable to merge diagrams, upload diagrams or compare diagrams.

**Subflows**

#### No merging

1. The user doesn’t make the request to merge diagrams
2. The user can return to the main ClubUML page, and the use case ends.

#### No saving diagram

1. The user chooses not to save an image of the merged diagram.
2. No saving will be done.

#### No exporting XMI

1. The user chooses not to export an XMI file of the merged diagram.
2. No exporting will be done.

#### No commenting

1. The user chooses not to leave any comment.
2. No comment will be added to the diagram.

**Key Scenarios**

No diagrams will be saved unless the user makes the request.

**Post-conditions**

1. The merged diagram is saved with comments, if any.
2. No diagrams are lost.

**Special Requirements**

The merged diagram has the same type as the two original diagrams. The merged diagram can be merged again with other diagrams, if the user makes the request.

# System Analysis

## Overview of Previous System

In order to analyze what enhancements would be best to employ, an overview of the work done in the previous semester (Fall 2012 and Sprint 2013) is necessary. The previous project features consisted of:

* Register new user
* Login as an existed user
* Uploading Ecore class diagrams and XMI class diagrams
* Display Ecore class diagrams and XMI class diagrams
* Download image of an uploaded diagram
* Download project
* Compare two Ecore diagrams side-by-side
* Merge two XMI diagrams and save as a new diagram
* Generate a PDF report of differences between elements of two diagrams
* Add a comment after comparison of two diagrams (Not working, only shown on GUI)

## Enhancement Overview

Maintenance, including:

* Using Bootstrap.js for GUI development to make the pages rendering properly and more attractive
* Restructuring the code of algorithms part to make it more flexible and extendable.

Enhancement, including:

* Comparing any two diagram – any combination of XMI or Encore.
* Role: Policy Manager
* Accessing multiple projects
* Managing projects including add and update (Policy Manager)
* Accessing multiple diagram context in each project
* Managing diagram context including add and update(Policy Manager)
* Suggesting promotion when comparing two diagrams based on configurable policy (consist of metrics)
* Managing policy including add with metrics, update (Policy Manager)

These enhancements are described in more detail in the following sections.

# System Architecture

The goal of the architecture is to provide an extensible design and design roadmap for the development of the ClubUML software according to the use cases and requirement specifications. The purpose of the tool is to a provide a platform for users to interact on UML Diagrams and give the user the capability for compare, merge and download UML diagrams.

## Assumptions and dependencies

The architecture assumes the following:

1. A Linux server will be provided by Northeastern University.

2. Software packages may be remotely administered and installed on this server.

3. The team can program in Java.

4. The team can develop on a MySQL database.

5. The team has Java-based web service experience: JavaScript, JSP, etc.

6. The users will have Eclipse with the Eclipse Modeling Framework (or equivalent) to create UML class diagrams in .Ecore format.

7. The users will have Papyrus tool (an Eclipse plugin) for creation of different UML diagrams (XMI format).

## Decisions, Constraints, and Justifications

The following decisions were made to address the requirements:

***The language will be Java.***

Java is a cross-platform language that is not tied to specific chip architecture. This means that it can run on any machine, which is a goal of ClubUML. Also, Java is the most familiar language to the developers, which reduces any learning curve in the development cycle.

***The IDE will be Eclipse.***

Eclipse is a free, powerful Integrated Development Environment (IDE) that supports Java. Like Java, it is also available for any architecture. In addition, most of the developers are already familiar with Eclipse which will enable a quicker development cycle.

***The database will be MySQL.***

MySQL is a fully-functional relational database that is free and open-source and with which many on the development team are familiar. MySQL is a relational database that is open source and free. Oracle was considered, however, the licensing is very expensive. PostGREs was also considered, but MySQL was chosen as it met the requirements and the developers have more experience in it.

***The web server will be Apache Tomcat.***

Tomcat is an open-source Java-based web server. Again, many on the team have experience with Tomcat. Besides having built-in Java support, it is also standard in the field and current infrastructure in the deployment environment. No alternatives were considered.

***A Linux server provided by Northeastern will host the application.***

This will be free to ClubUML and will be accessible from anywhere using the internet.

***The diagrams will be uploaded in the XMI Format or Ecore format.***

The Ecore format for class diagrams will be supported for backward compatibility. The UML diagrams generated by Papyrus are stored in the XMI format. Papyrus model generates three files and the User Interface shall add support for uploading multiple files. The files uploaded will saved in a folder with a unique date and time field.

***Compare will be performed on diagrams generated using the different tool.***

The tool will support compare between ECORE class diagrams, XMI class diagrams, ECORE/XMI class diagrams and support compare between class diagrams generated using the Papyrus and compare between sequence diagrams between sequence diagrams generated using Papyrus.

***Merge functionality will be only performed on diagrams generated using the Papyrus UML tool***

The merge functionality will only supported for Class and Sequence diagrams generated using the Papyrus UML tool. Merging ECORE Class diagrams is out of the scope of this project.

***ClubUML will have multiple projects.***

Supporting multiple projects is necessary in a tool like clubUML for industrial use. Developers in a team or multiple teams need access different projects in same time.

***ClubUML will have diagram contexts.***

A project can have multiple diagram contexts. They have their own policy for comparison. This concept makes it more convenient to compare a small or functional part of a large diagram.

***ClubUML will have suggested promotion function.***

This is the smart part we decided to add into the project. After adding policy and metrics into diagram context, it will give user a suggestion on which diagram is better when comparing two diagrams.

***ClubUML will have a new role: policy manager.***

To obtain the previous two features, a new role – policy manager is needed. Policy Manager is the person who has permission to manage the whole project (including project, diagram context, policy, metrics).

***MySQL will only store links to the UML diagrams in the database, not the diagrams themselves.***

This will save on the size of the database, which should make it faster. In addition, a MySQL database doesn’t handle Binary Large Object (BLOB) data like some other database applications.

***Servlets will be used as the controller mechanism.***

This includes JSP (Java Server Pages), as well as generic web languages like HTML and CSS. These mesh well with Java, are handled well with Eclipse, and are more familiar to the developers than their Microsoft counterparts.

***SQL will be the database query language.***

This is the standard for modern relational databases, and is what MySQL requires. No alternatives were considered.

***The*** ***Java Database Connectivity (JDBC) API will interface between Java and MySQL.***

An abstraction layer/library was considered, such as MyBatis, however, the developers were familiar with Java Database Connectivity (JDBC) and did not want to introduce another learning curve by adding an additional library.

***The UML diagrams will be generated into .PNG files and will be saved to disk.***

This will prevent any redrawing of the same diagrams twice, which will give speed at the cost of disk space. This decision is mainly based on the first milestone of the software and subject to change.

***The architecture will be a layered architecture***

The basic model of the layered architecture is shown in Figure :

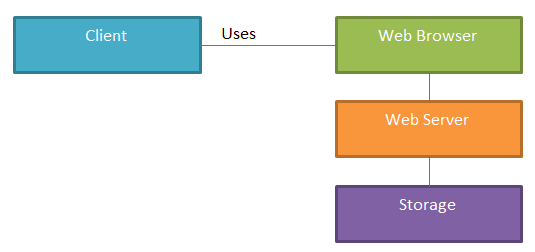


Figure : **Layered Architecture Diagram**

# Architectural Mechanisms

***1 – Web server***

This will drive the entire project. An Apache Tomcat web server instance will be running on a Northeastern-provided Linux server. Tomcat is the Apache web server that supports Java-based applications.

***2 - Database***

All users and project data will either be stored in the database, or have a reference to their location stored (as in the case of the UML files, both .core and .png). MySQL is the database, because it’s an open source relational database that is familiar to most of the development team.

***3 – UI/Front End (view)***

A combination of HTML/CSS will be used to design the webpage User Interface (UI), as most of the team has had some training in this as part of their academic program.

***4 – High-level Components***

This consists of a series of high-level components as discussed in the next section. The main goal is to provide a level of abstraction for the developer’s code to compare and possibly merge the UML diagrams. It also caters for facilitating any extension during the next iteration of the software. The other goal is to isolate the development teams’ tasks and promote a low coupled design mindset.

***5 – UML Diagram Parsing***

The diagrams will be parsed using a standard SAX Parser. The Elements and its corresponding attributes will be saved in a list. The UML Parser will then place each element based on its UML attribute.

***6 – Diagram Visualization***

UMLGraph and GraphViz are responsible for class diagram visualization. The upload code will parse the .ecore/.uml and .notation file(s) and create a simple .java file which will be the input to UMLGraph. The output of UMLGraph (known as dot format) will be processed through GraphViz (a generic graph visualization tool, not UML specific), which will produce the diagram as a .png file.

For the extension to integrate sequence diagram in the existing project, the new system will continue with UMLGraph to define the sequence diagram in .pic file format and then visualize by pic2plot graphing tool, which will produce the diagram as a .png or .gif file.

The current limitation of pic2plot support is that it only runs on Windows system. Any extension of support on Linux systems could be implemented by installation of pic2plot for Linux.

***7 – MVC Architecture (model-view-controller)***

The MVC architecture is used throughout the application. This allows a separation of business logic from data and from presentation code. This means that there will be separate code for the Controller (servlets), and that does not mix with our Presentation code (JavaScript/JSP). Similarly, the data is stored in JavaBeans, which are again separate.

***8 – Servlet***

Java servlets will provide our web mechanism. This is where all of the application logic is done. This is considered the ‘Controller’ portion of our MVC architecture.

***9 – Model***

The JavaBeans model is used in the software, which makes the code a system of reusable software components. This is done by writing classes in a particular convention. Using JavaBeans allows us to encapsulate the data and have a standard method of accessing it, which means it can be completely separate from the view or controller portions of the code. These JavaBeans are the only code pieces that directly manipulate data in the system.

## Layers of Architectural Framework

ClubUML will use the Layer framework. A high level view of the layers is given here:

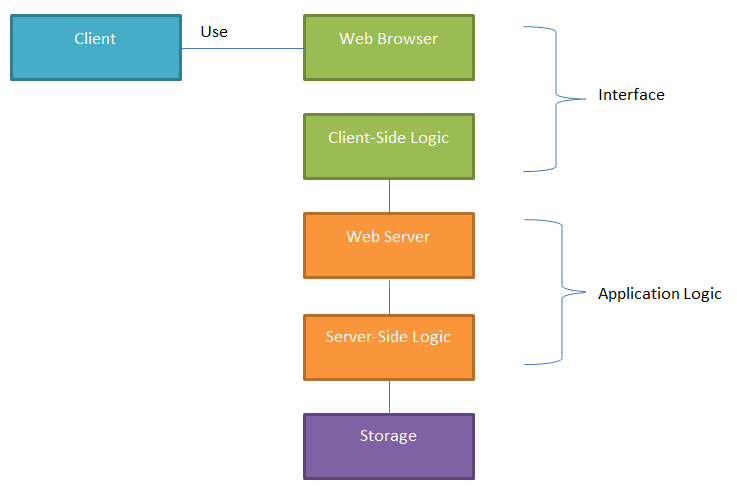


Figure: **Layer Framework**

# Architectural and Design Overview *(Gautam)*

**ClubUML Design:**

ClubUML is a solution meant to be used as a platform where groups of people working in a project can share their versions of UML diagrams and also give them the ability to compare or merge diagrams. ClubUML also gives the users ‘suggest’ feature during comparing diagrams. In this feature the “Smart Scoring Engine” kicks in for each diagram and generates scores for the diagrams and suggests which diagram is better as per the scoring policy set up for the context the diagrams belong to.

The usage of ClubUML can be better described in the setting described in the diagram below.

We can see ClubUML deployed in an enterprise where there are multiple projects running. In a **Project**, there will be multiple functional areas being developed. Since any UML diagram will be created to serve or address a particular functionality or requirement, we can safely generalize and say that the diagram is meant to serve a specific ‘Context’. This way we can group multiple diagrams serving the same purpose to be in one **Diagram Context**.

ClubUML provides a framework that handles multiple **Projects**. Each of these projects can have multiple Diagram Contexts and each diagram context can have multiple diagrams uploaded under them. The user can perform actions (compare and merge) on diagrams in the same context.

The Solution Architecture of ClubUML 2013 is a drive to accomplish the following three major things:

1. Improve and streamline the existing solution (the solution we inherited had many gaps, workarounds and hard coding to make things ‘work’; and also, the code was not in line with the documentation nor with the intent of the initial design).
2. Bridge the gap that was created between the implementation for handling ENCORE file types by the Fall2011 batch and the XMI file types that was implemented by the Summer2013 batch.
3. Design and implement the ‘Scoring’ of diagrams against a policy set up for the context those diagrams belong to. This helps in giving suggestions to the user during the comparison of two diagrams.

The fundamental element of the design was to create a very modular and non-coupled architecture. But due to the legacy code from previous batches, we were not able to de-couple some of the modules; we’ll see this later in the document.

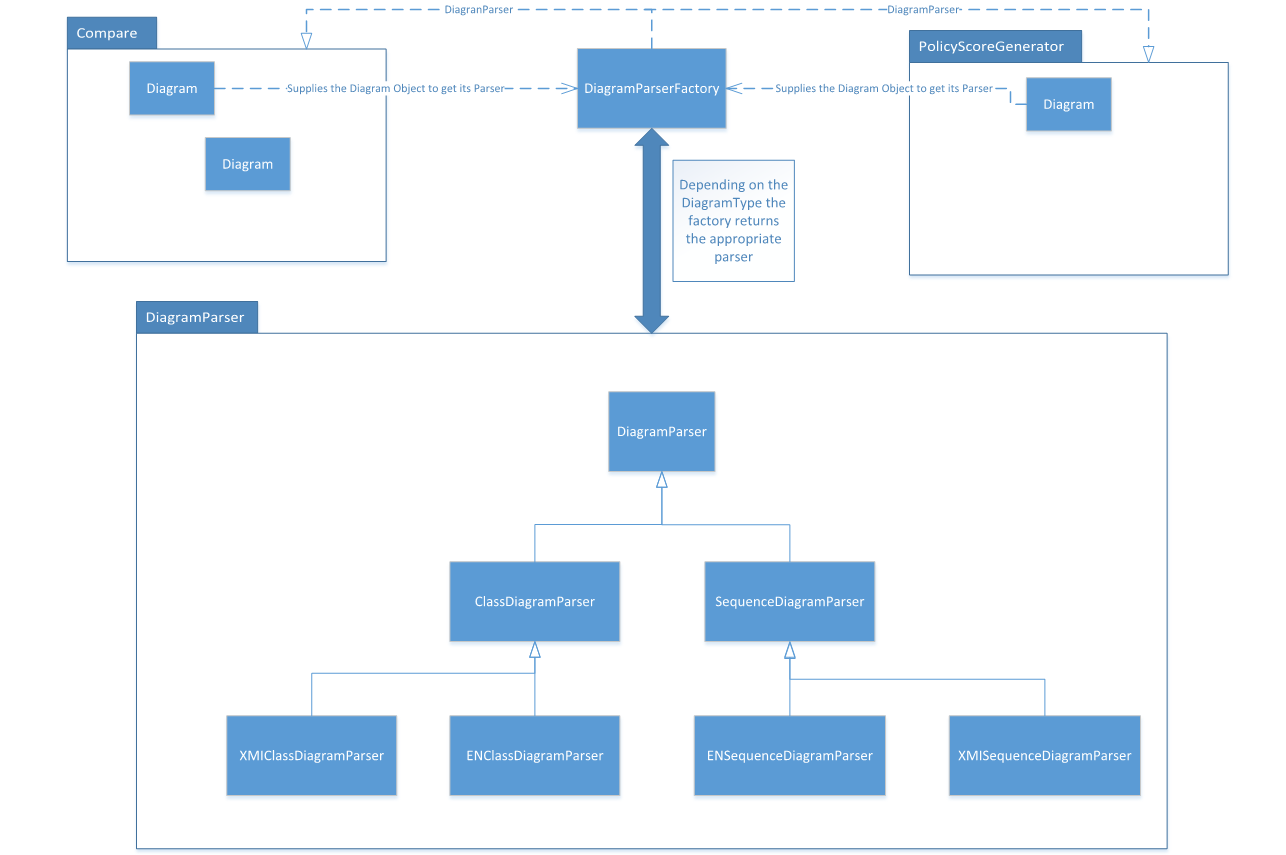
The various modules are as illustrated in the diagram below:

**Decoupling the Algorithms and the different file types**

In the code we inherited from the previous batch, compare algorithm only worked with Encore file type and the Merge algorithm only worked for XMI file type. This was because the Encore File Type was tightly coupled within the Compare Algorithm and the XMI file type was tightly coupled with the XMI file type.

The aim for ClubUML 2013 was to create a framework where we will be developing a framework which is independent of the file type being used. This will allow the Compare and the merge algorithms to work with both Encore and XMI file types. It can be accomplished by abstraction of the separate parsers for both file types to a more generic parser called – **DiagramParser**. So, the Merge and Compare engines will work with the DiagramParser directly instead of separate parsers for separate file types.

**Note**: We have the domain object: **Diagram**, which contains details of the file including the location of the file and its file type. The **DiagramParser** will take the **Diagram** object and it will expose operations specific to the diagram and the diagram type. This process is accomplished using a Factory pattern where we have a DiagramParserFactory, which takes a Diagram object and gives back a DiagramParser object relevant to the diagram type and the file type. Then the relevant engine (Compare, merge or Scoring) can use the DiagramParser to get the properties of the diagram. This way we have illuminated the dependency on the file type. The following illustration shows this architecture very well.



**The Engines/Algorithms**:

We have 3 core engines running for ClubUML –

1. Compare Algorithm
2. Policy Scoring
3. Merge

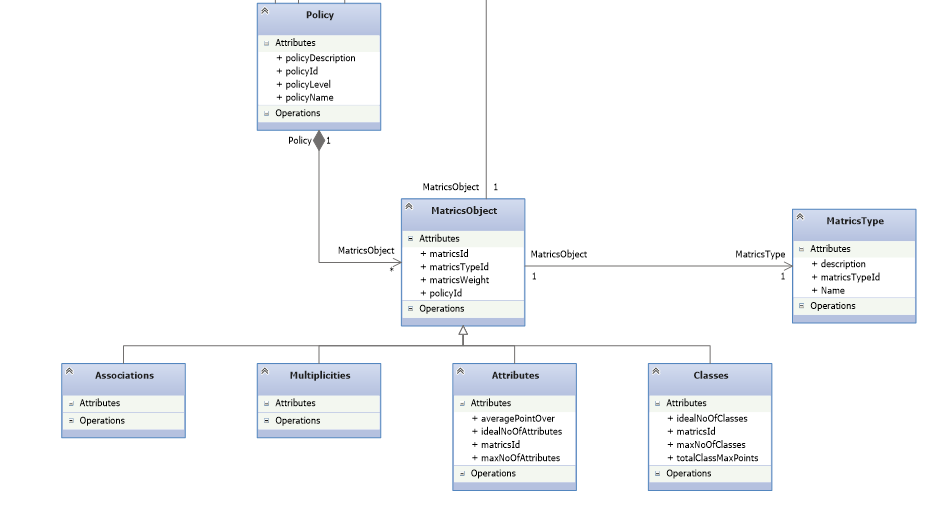
For Fall 2013, we managed to incorporate the de-coupling described for the Compare Algorithm, but the Merge engine is still in its legacy old state where it is tightly coupled with XMI file types.

Since the Policy Scoring engine is brand new, it was built from scratch using the new de-coupled architecture.

**Policies and Scores:**

The most important addition in this version of ClubUML is the introduction to ‘smart scoring’ for diagrams. The need for this arises when we want ClubUML to be smart enough to suggest to the users which diagram is ‘technically better’ during comparison. In order to do this, we need to set up standards for the diagrams that goes into a particular context. The **Policy** does exactly this.

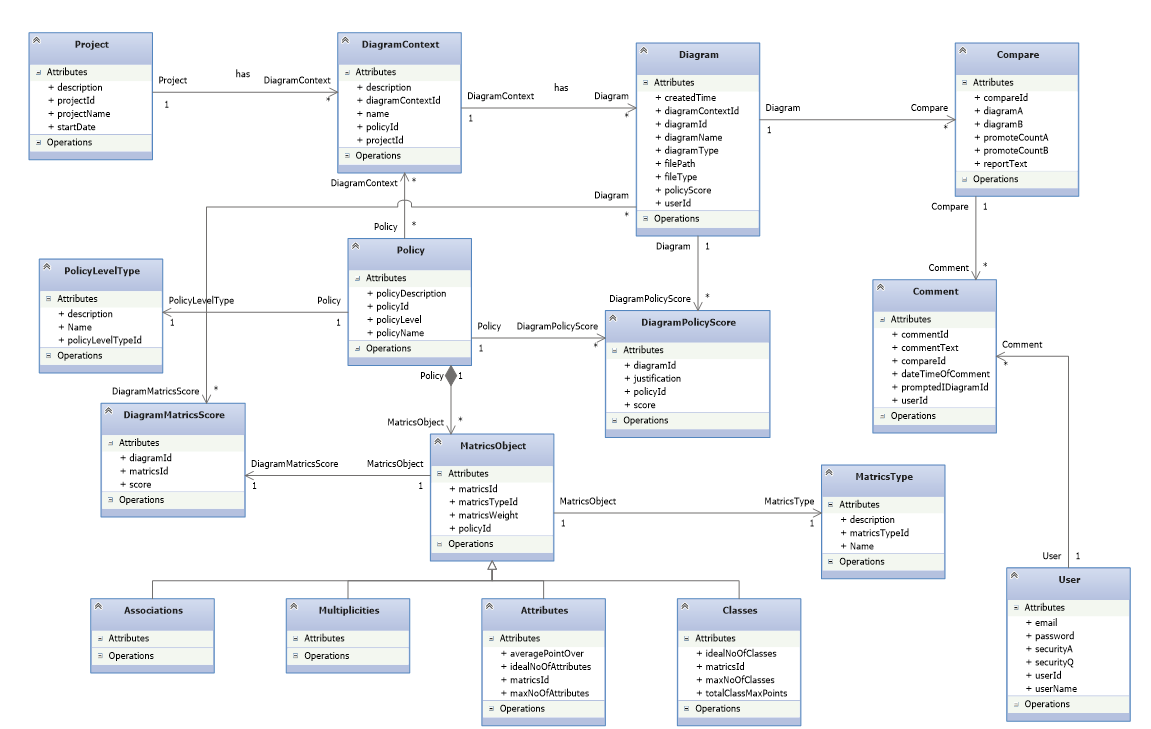
A Policy is a set of **Metrics** with pre-set values which we will use as standards to score the qualities of a diagram. For every Diagram Context, we can define ***one*** Policy and all diagrams being compared under that context will use the same Policy. The details of how the scoring happens is described in **APPENDIX X**.



As illustrated in the diagram above, we have designed this functionality in such a way that future batches can add different types of Metrics and have different properties without changing any of the existing solution. The framework being set up is flexible and allows different scoring patterns. For ClubUML Fall 2013, we have implemented scoring for “number of Classes” and “number of Attributes”. As an example, the diagram above shows other possible metrics – ***Multiplicity and Associations***.

**ClubUML Structure** (class diagram):

Below is the class diagram for ClubUML Fall 2013. It shows all the relationships and multiplicity between each relationships. Note that the Enum types in the Domain viz. DiagramType, PolicyLevelType etc. are represented as their own class and will have their own tables in the database.



# Design and Implementation

## Client-Side

GUI Changes (Aishwarya Srinivasan)

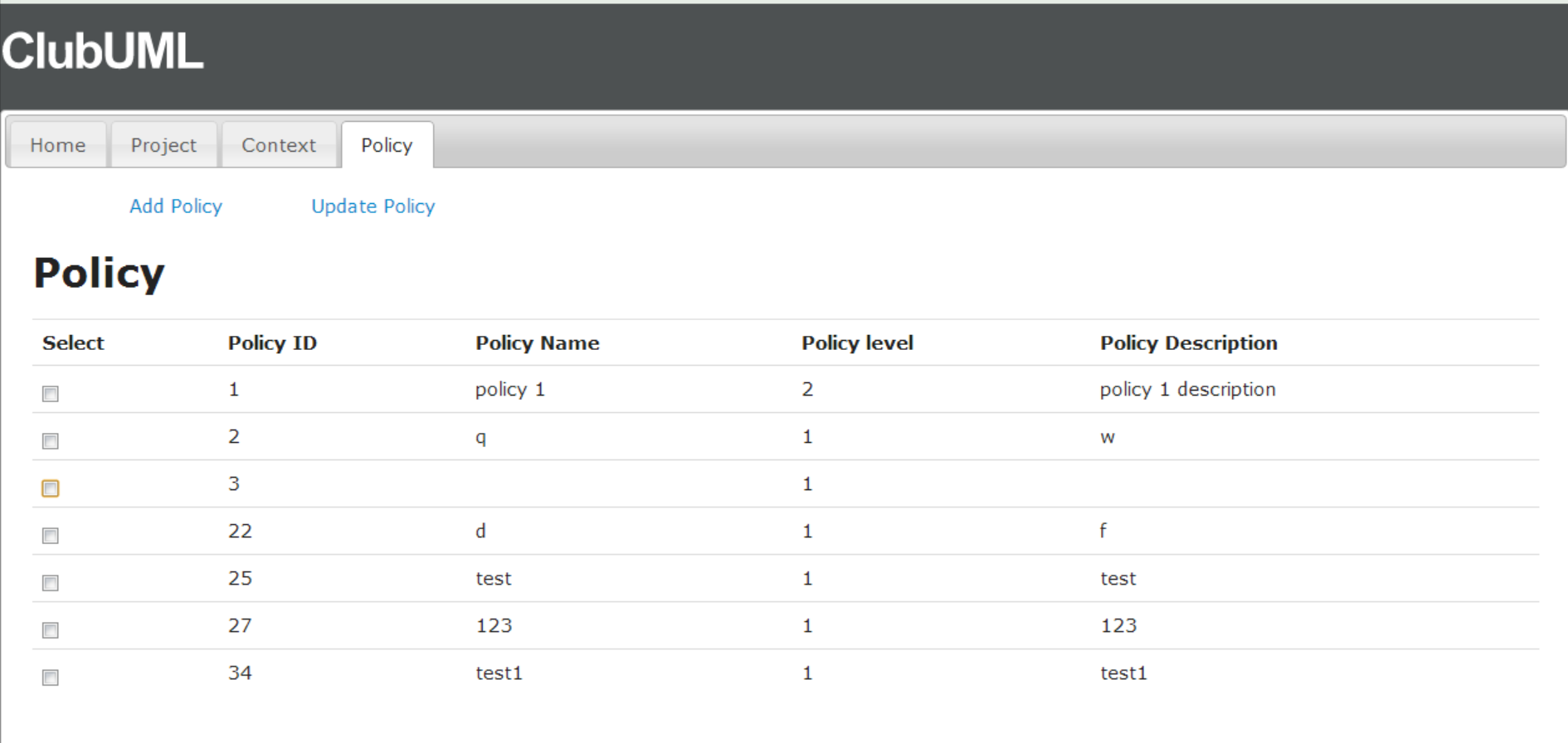
The major changes in the GUI component of the web application are as follows:

1. Twitter Bootstrap: Twitter Bootstrap has JavaScript and CSS components which can be rendered directly in a JSP. We have used it to create modular tables which highlights the hovered row. All the input components are overridden by the Bootstrap CSS. The buttons and links used use the bootstrap CSS classes.
2. JQuery UI: JQuery UI template was used to create the different tabs that is the Home, Project, Context and Policy.
3. JQuery: JQuery was used to display contents using AJAX to render the contents of all the different tabs.

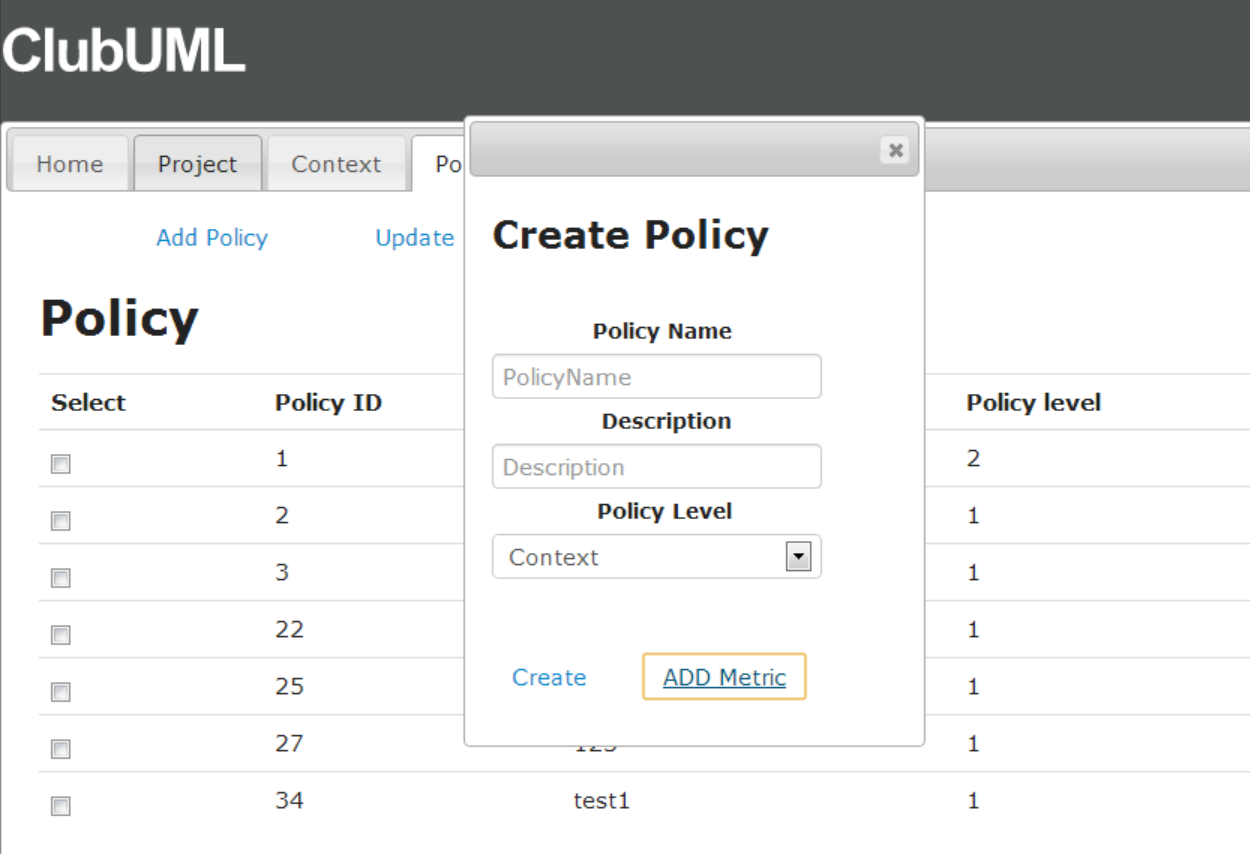
The above mentioned changes are with respect to the view and layout. Apart from that, a number of JavaScript functions have been defined to enforce the application features to display pop-ups, submit forms to add user, app and update project and context and policy, enforce rules for entering values in the input forms, etc. This was done using simple JavaScript along with JQuery.

Policy.jsp:(Weiqi Kong)

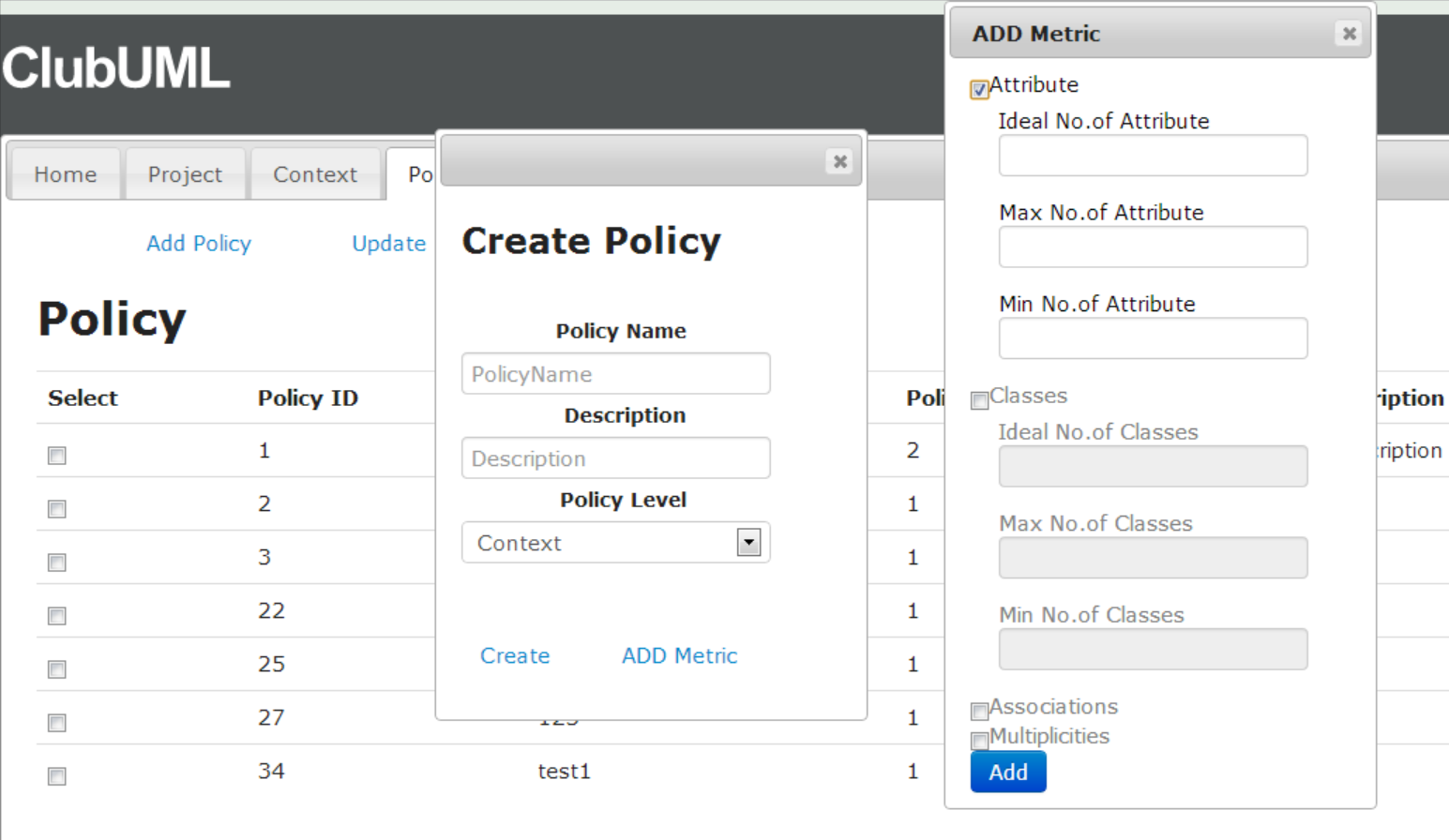
Policy data come from MySQL in this table. There are two buttons: Add Policy and Update Policy. ”Update Policy” button has same function with “Add Policy” button.



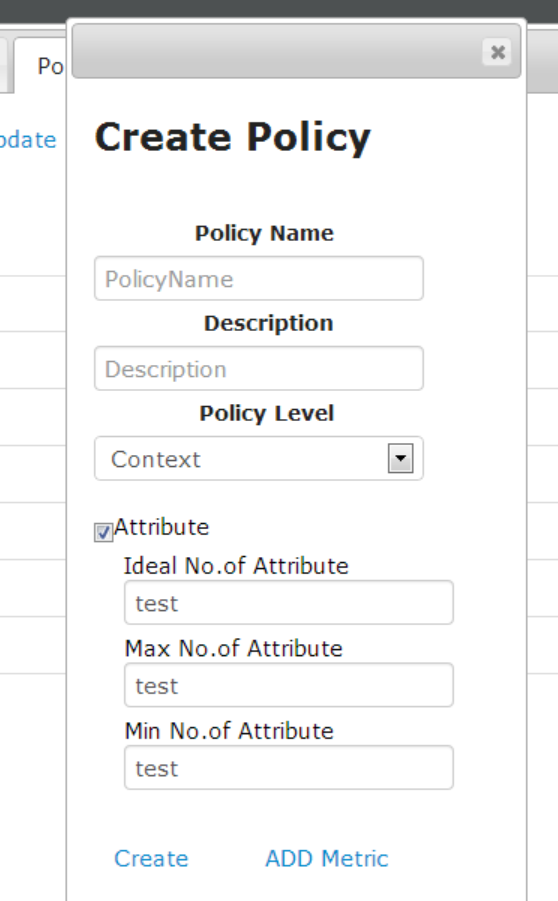
After the user click button, page pop-up a new window. The user needs to add Metric at first.



The user cannot insert any information if user doesn’t check the checkbox in ADD metric window and then background color will remain grey.

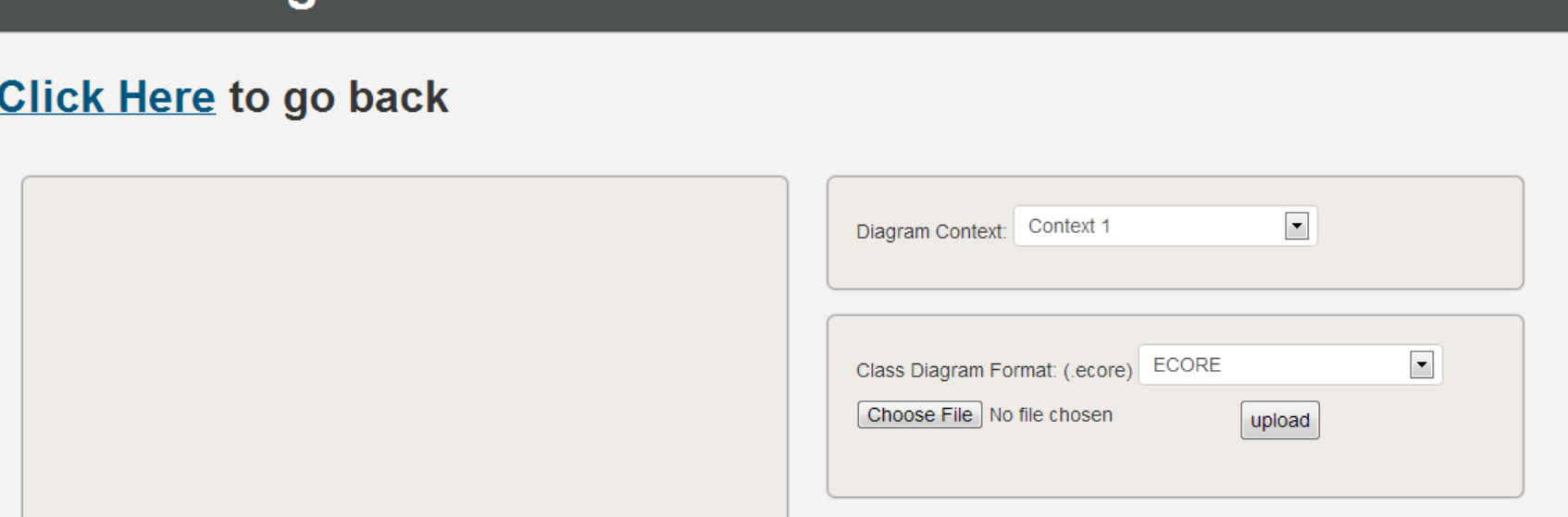


After click “add” button, create Policy window will show metric table.



Display.jsp:(Weiqi Kong)

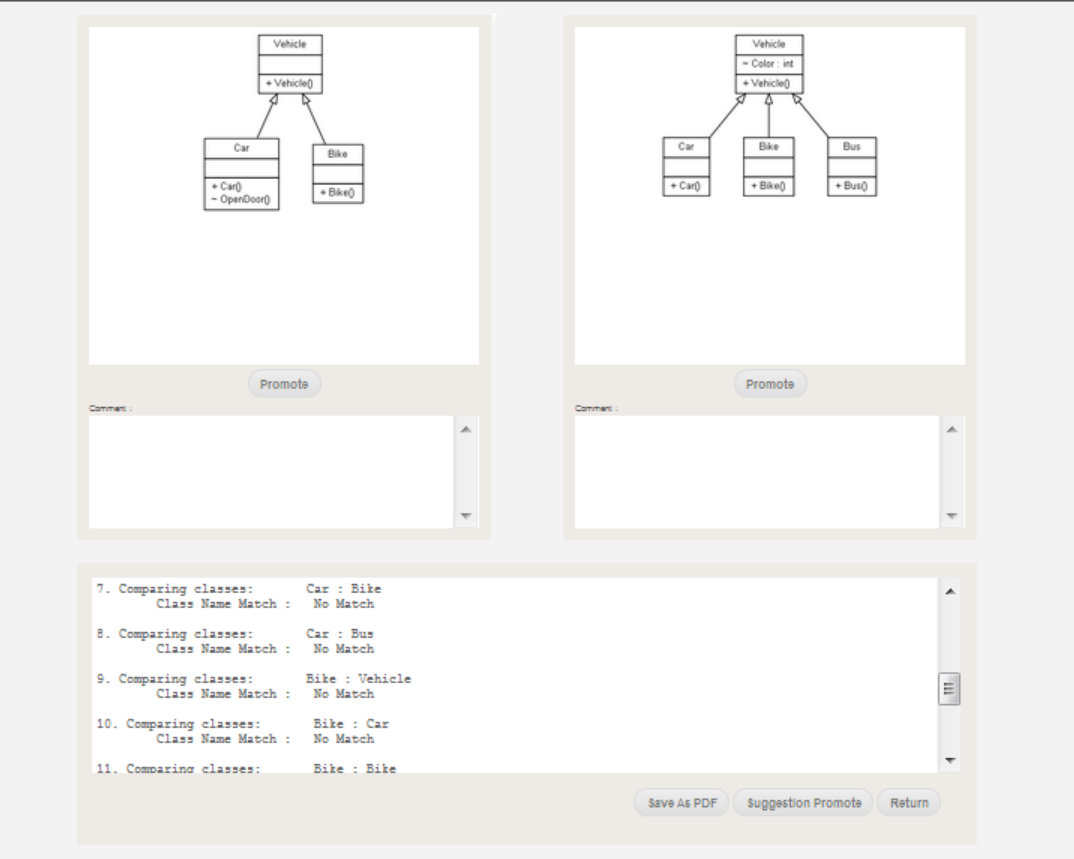
Add Diagram Context in Display page.



Promote.jsp:(Weiqi Kong)

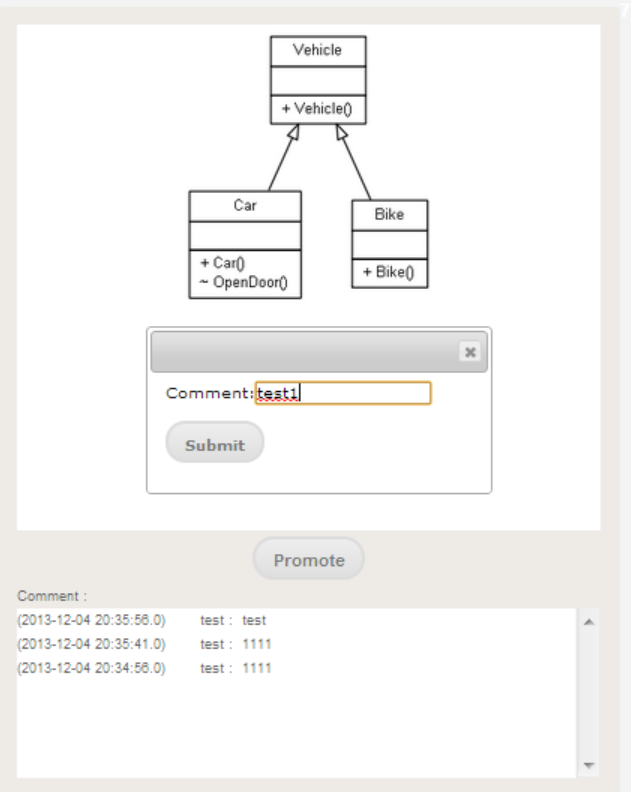
Resize the background color of diagram.

Report will show in the bottom of diagram instead of popping up a new window.



When user click “Promote” button,

Using pop-up window to insert new comment and then the comment will show up in the comment box.



## Server-Side

***Controller changes***

***Compare Framework (Prasanna):***

The existing application only supported the comparison of Ecore class diagrams and generating the reports. New Compare design was developed to support the comparison of two Ecore diagrams or two XMI class diagrams or an Ecore and an XMI diagrams. Existing Compare process was refactored and new functionality were added. Compare framework consists of 2 part DiagramParser and UnifiedAlgorithim(Compare)

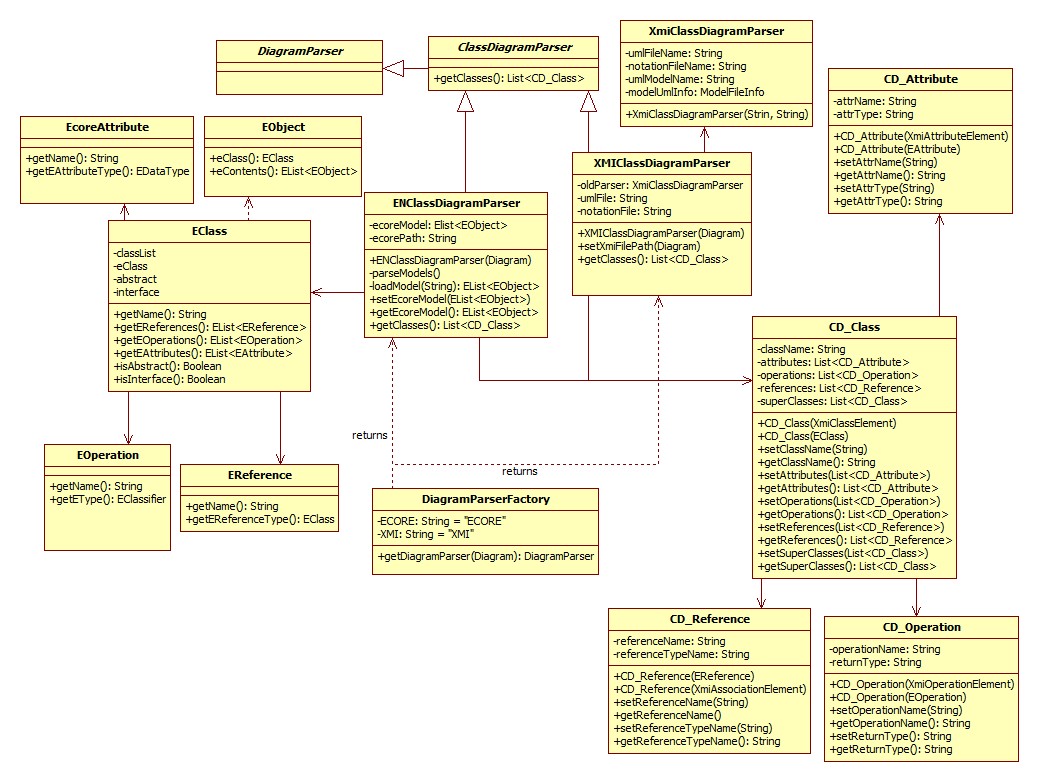
After User clicks the Compare button in the compare.jsp, request is handled by compare.java, which gets the diagramIDs of the selected diagrams and which is used to get the diagram objject from diagramDao class. Diagram object is used to set the diagram path in the DiagramCompare class. In DIagramParser's process method, DiagramParserFactory is intantiated and getDiagramParser method isinvked passing diagram path, which returns diagramparser object of the respective diagrams. DiagramParser object is then used by UnifiedAlgorithim to do the comparision of the diagrams.

Working of DiagramParser and Compare is explained below:

***Diagram Parser:***

A Factory Method Design Pattern is used, which allows to create objects without specifying the exact class of object that will be created, in this case, creating the diagramparser object without knowing diagram type, either Class Diagram Parser or Sequence Diagram Parser(Please refer to parser diagram in **Architectural and Design Overview**), but only Class diagram is supported by current application so, it will create either XMI Class Diagram Parser or Ecore Class Diagram Parser based on the file type of the class diagram.

DiagramParserFactory class instantiates and returns the DiagramParser object, at runtime depending upon the diagram type, object returned will be either of type ENClassDiagramParser or XMIClassDiagramParser.



**Ecore Diagram Parser:**

Ecore Data model of org.eclipse.emf.ecore package is used to model and parse the ecore class diagrams. loadModel method creates instance of ResourceSet api, which takes the diagram path as paramater and returns Elist of type EObject which is the list of Ecore Diagram classes. getClasses method traverse through the EObject List(i.e.list of EClass) and creates the instance of CD\_Class for each EClass, and stores the attribute, operation, references and superClass in the List of type CD\_Attribute, CD\_Operation and CD\_Reference respectively. So once we get the diagramparser object, it used getCLasses method to get the ECLass list, and traverse through it to get respective attributes, operations and references using getEAttributes, getEOperations and getReferences method of CD\_Class.

**XMIDiagramParser:**

XMI Diagrm Parser is implemented using a Java SAX parser. XMIClassDIagramParser gets the file path for XMI .notation and .uml files, along with extension and pass it to old XMI parser class, XmiClassDiagramParser, which calls instances following classes to parse the XMI elements.

ParseXmi : This class implements a SAX parser. Please see the documentation on SAXParser for more details. This class opens a XMI files breaks the file into elements and attributes.

ModelFileInfo : This class contains a list of all the Elements present in the XMI file. The name of the file the elements are associated with.

XmiElement : This class contains a list of all the child elements present in the Element. A list of all the attributes. And parent Element.

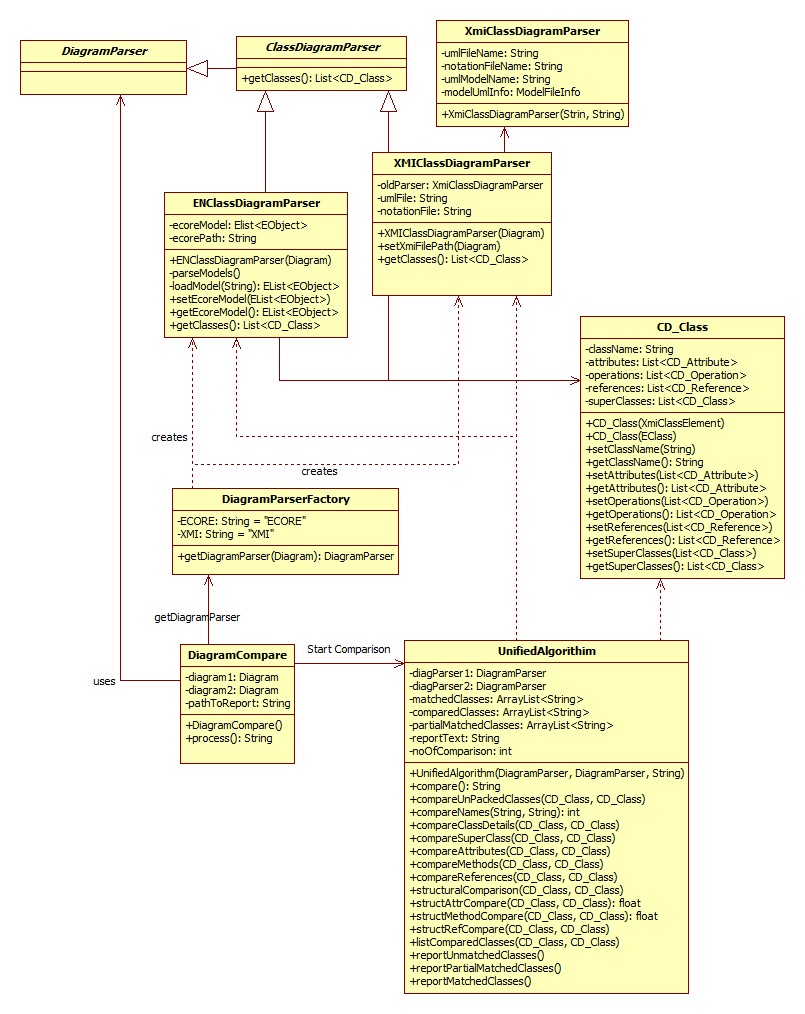
XmiIdToElement : So most Elements are referenced using Id. So we created a Hashmap of all the ID to elements. 5. Attribute : An Attribute has a name and value.

XmiClassDiagramParser return a parser object, which is passed to getClasses method. getClasses method retrieves XMI class elements List(i.e. list of XMI classes in a diagram) and creates the instance of CD\_Class for each XMI Class, and stores the attribute, operation, references and superClass in the List of type CD\_Attribute, CD\_Operation and CD\_Reference respectively (similar to ECORE ).

In DiagramCompare class once we get the diagramparser object, it uses getCLasses method to get thelist of classes, and traverse through it to get respective attributes, operations and references using getEAttributes, getEOperations and getReferences method of CD\_Class.

***Compare Framework (Prasanna):***

The existing Compare package supported the comparison of Ecore class diagrams only and generating the reports of the comparing Ecore elements. New Compare design was developed to support the comparison of two Ecore diagrams or two XMI class diagrams or an Ecore and an XMI diagrams, making use of the DiagramParser, as explained above.

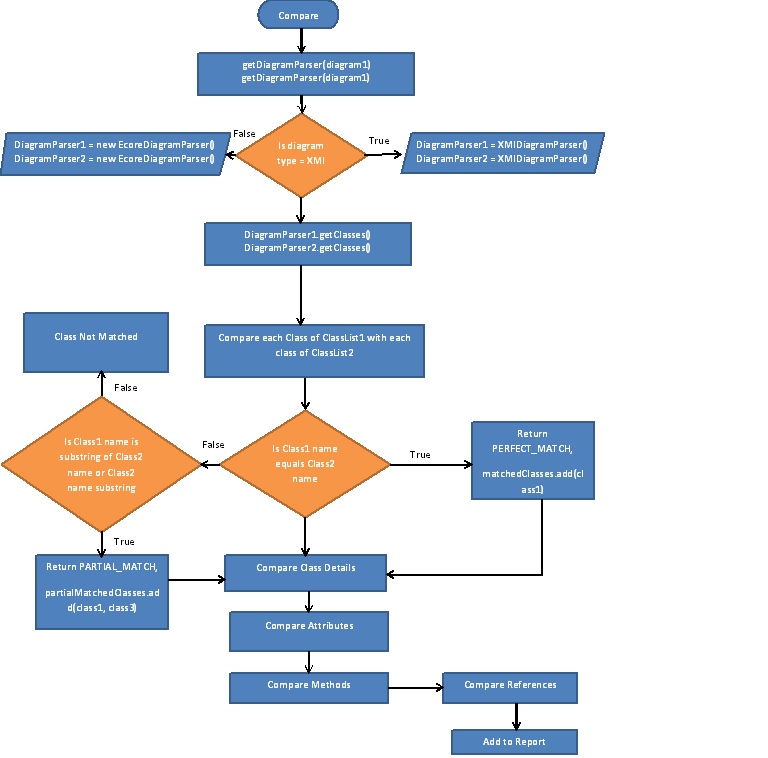


DiagramCompare's process method gets the DiagramParser object, Ecore or XMI based on the diagram type and instantiates UnifiedAlgorithm passing the parser object for both diagrams. UnifiedAlgorithm's compare method compares the diagrams both Ecore or XMI using the diagramparser object, which is used to get the list of classes in the diagrams and list of respective attributes, operations, references in each class, and each comparison step is added to both PDF and text report .

Compare Algorithm:

Depending on the diagram type, Diagramparser object (i.e. ENClassDiagramParser or XMIClassDiagramParser) is used to get the list of classes of two diagrams selected for comparison.

Each class in first diagram is compared with every classes in second diagram. First Name of each classes are compared. If they are exact match(i.e. same lexical name), it returns PERFECT\_MATCH and class name is stored in matchedClasses list. If one class name is a Substring of another e.g Vehicle and Vehicles, PARTIAL\_MATCH is returned and Class names are stored in partialMatchedClasses list. For Perfect and Partial matched classes, structural comparison is made i.e. comparing attributes, operations, references and superclass of matched classes. Attributes and Operations are checked for name and type match, data type and return type respectively. All the remaining classes are added to report as Unmatched classes.

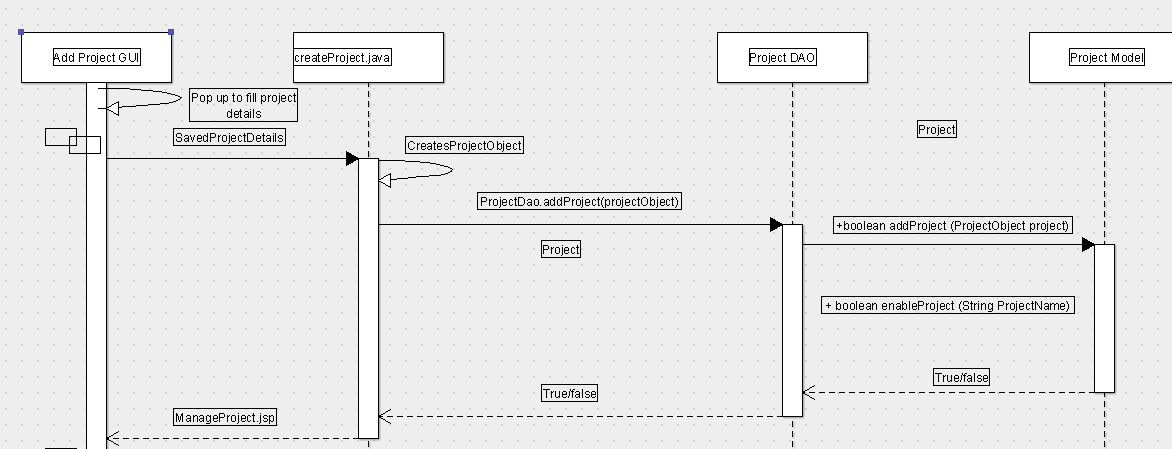


## Client-Server Side

Sequence diagrams (Indrajit)

The sequence diagrams have been made based upon the use cases.

Add Project sequence diagram.



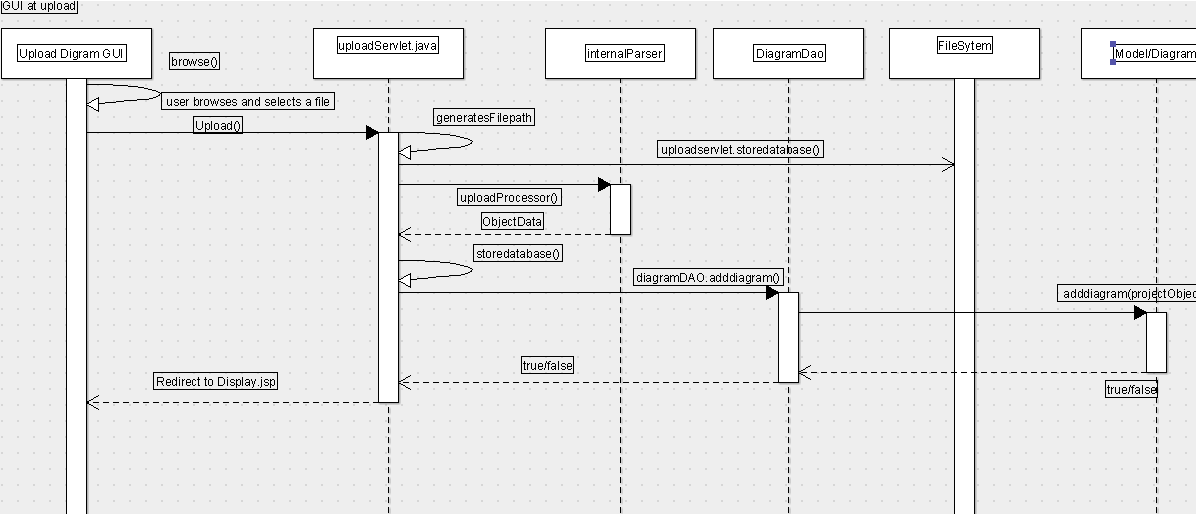
The add project high level sequence is as follows

1) The GUI saves all the project details and then sends it to the createProject.java

2) CreateProject.java creates a Project object.

3) ProjectDAO then saves the project to the database.

2) Upload Diagram sequence diagram



The steps for the upload diagram sequence diagram is as follows.

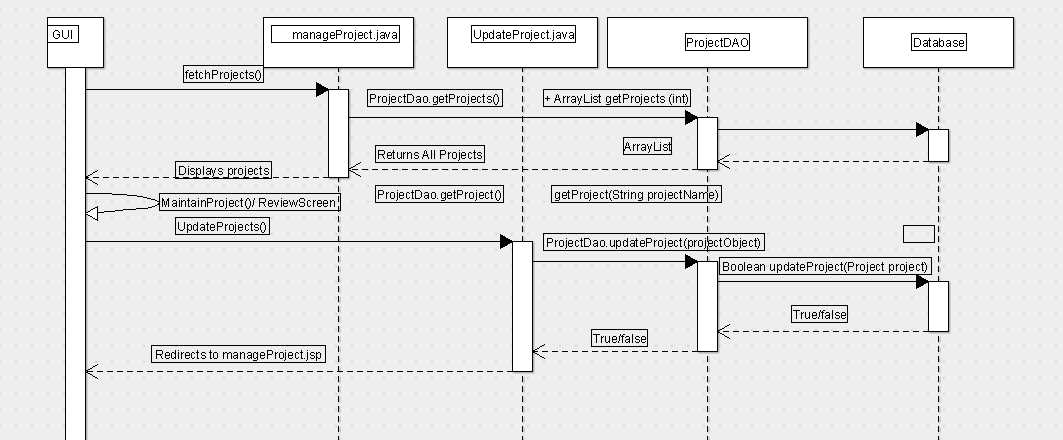
1) The user browses the image to upload

2) The user uploads and the upload servlet generates a file path for the image.

3) The internal parser separates the parts and assigns to variables.

4) The diagramdao adds the image to the database.

Maintain Project



The steps for the sequence diagram for the maintain project .

1) The GUI fetches the list of all projects and displays using manage project servlet.

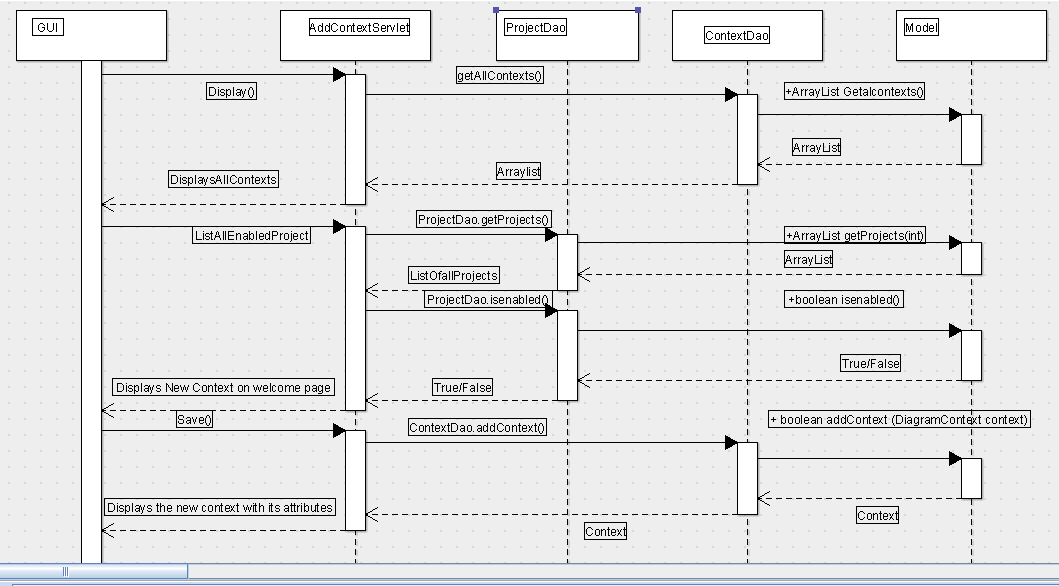
2) The maintain project tab results in a review screen where the details for the project can be added.It has options for enable disable and changing name and description.

3) The update project creates an object and that object is sent to the projectDAO which modifies the database with the new project fields.

4) Returning true or false depending upon the status in the database.

Add Context sequence diagram.

This functionality does not exist but can be used to implement.



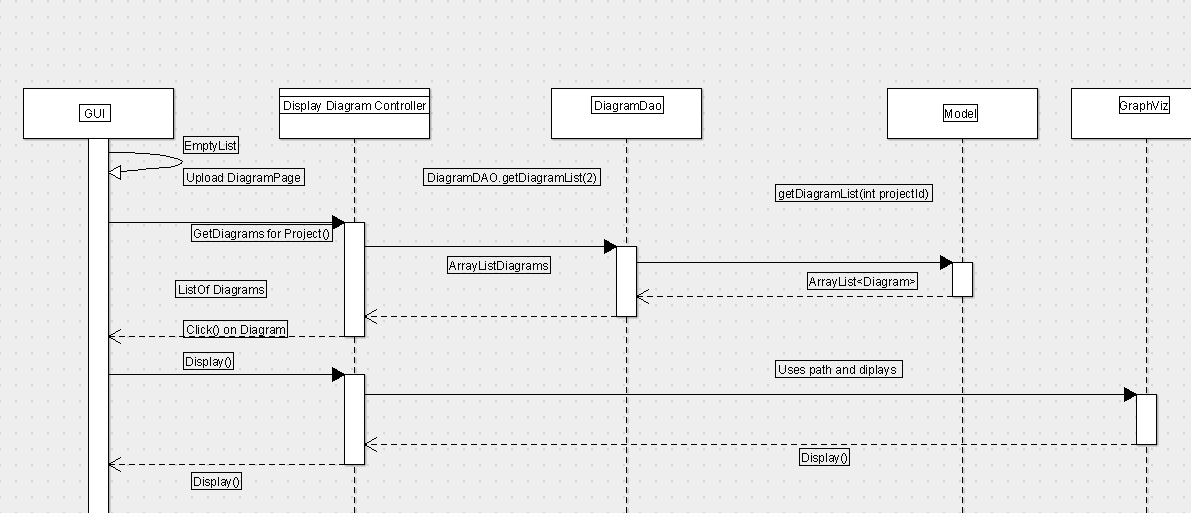
The steps for add context sequence diagram

1) The requirement is to have all contexts and a list of enabled projects enabled by add context servlet and project dao and context dao.

2) Since multiple projects can have one context we need enabled projects for that.

3) Save() results in creating a context object with all the details and the contextdao adds the new context to the database.

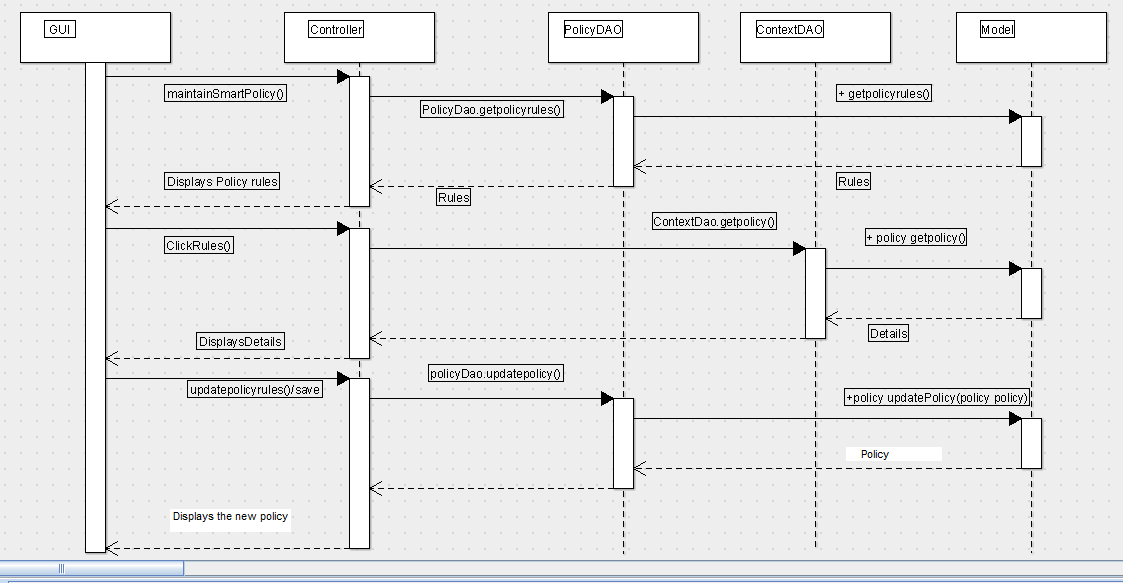
Display Diagram sequence diagram



This functionality exists in the application.

1) The user will be taken to the upload diagram page and the user can then select the appropriate diagram from the file system and then on the display option the display diagram controller will make the call to the graph viz using the path that is generated by the display diagram controller and render the image.

Enable Disable smart policy .



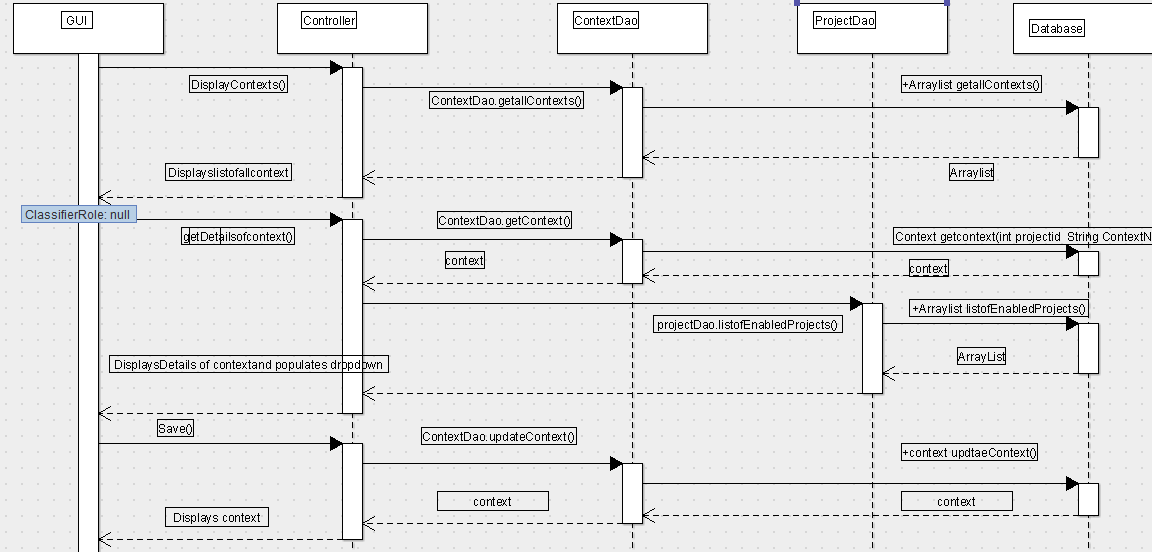
This functionality does not exist in the application, though the sequence diagrams can be used to crate the appropriate DAO and servlets.

The Function of disabling the smart policy will involve the use of a default policy which is in place to evaluate the diagram.

Each of the policies have the flag in the database that help them to be enabled or disabled.

The prospective controller can create a new object for the updatepolicy option and then call the dao to update the policy in the database. It can be as simple as setting a flag in the database.

Maintain Context sequence diagram



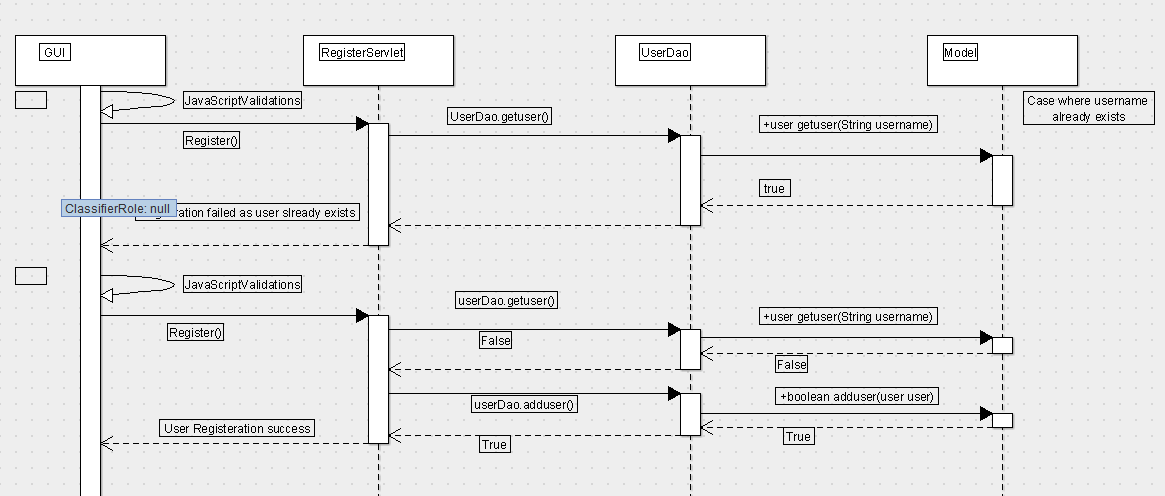
This functionality is implemented using context dao , project dao and a servlet. The servlet to enable the functionality needs to be written .

The GUI will display the list of all the contexts enabled and disabled.

The servlet will have to fetch the context object to populate the update context .This function can be enabled with the Context dao which has methods to get the context object implemented using JDBC.

The changes can be made to the GUI regarding the name ,description , and the status. The prospective update context servlet can create the new context object and store it in the database using the context DAO. The new context will be returned and the manage context JSP can compute and send the display JSP to display the modified context.

Register Servlet

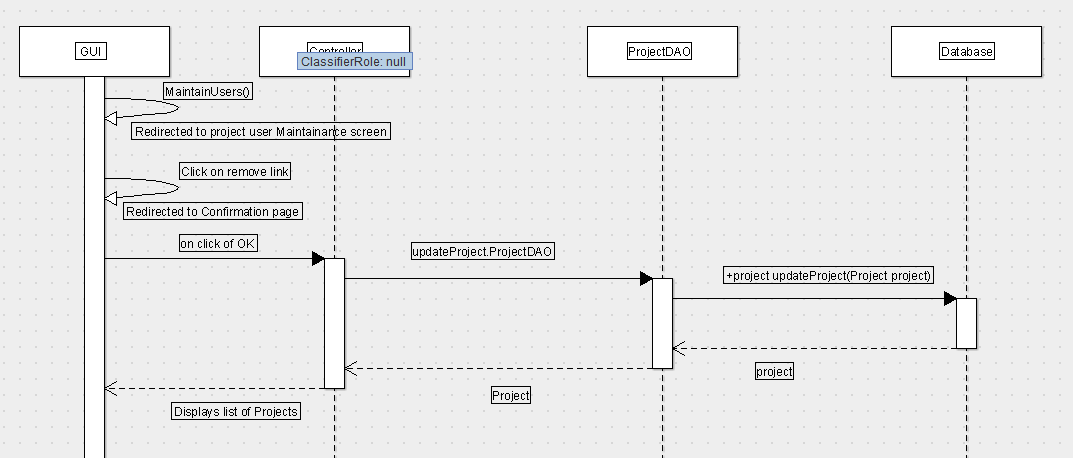


This is the basic registration functionality implemented in the earlier semester but the sequence diagram can be used

The register servlet does a call to the user dao to ensure that the user name does not already exist. There can be some java script validations to ensure that the email address format is correct and the password requirements are met in order to make the login a little more secure.

The servlet would return the result that the user already exists if the user name is found in the database and would be prompted to use a different user name else would successfully register the user.

Maintain Project Users



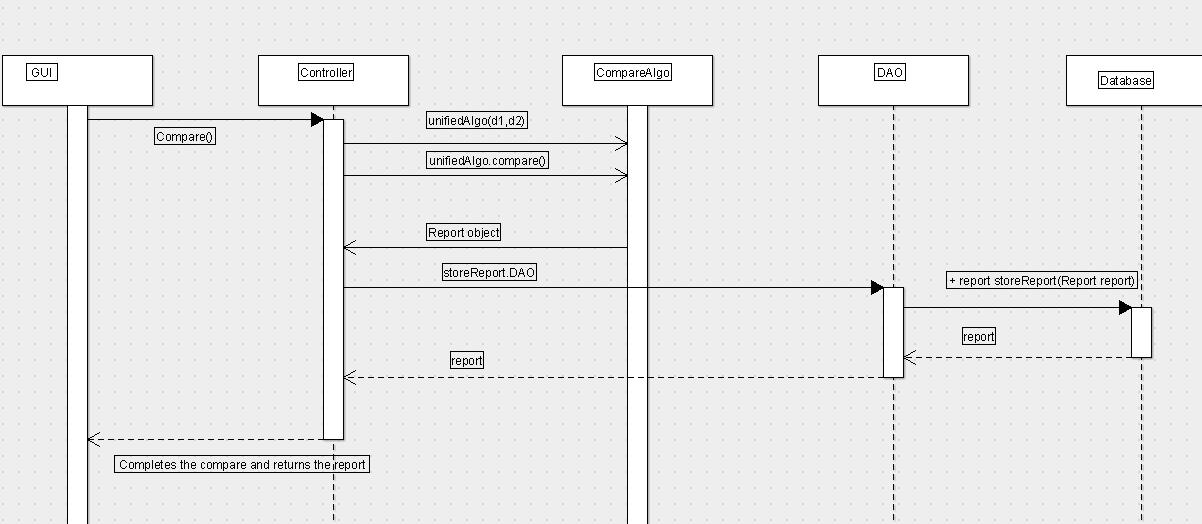
The steps for the abouve sequence diagrams are as follows.

1)The maintain user projects we can add and remove the users from a existing project.

2) On clicking the confirmation page the project DAO will modify the project object in the database.

3)The controller will facilitate the list of new modified projects.

Comparison Diagram



The steps for the above sequence diagram is as follows

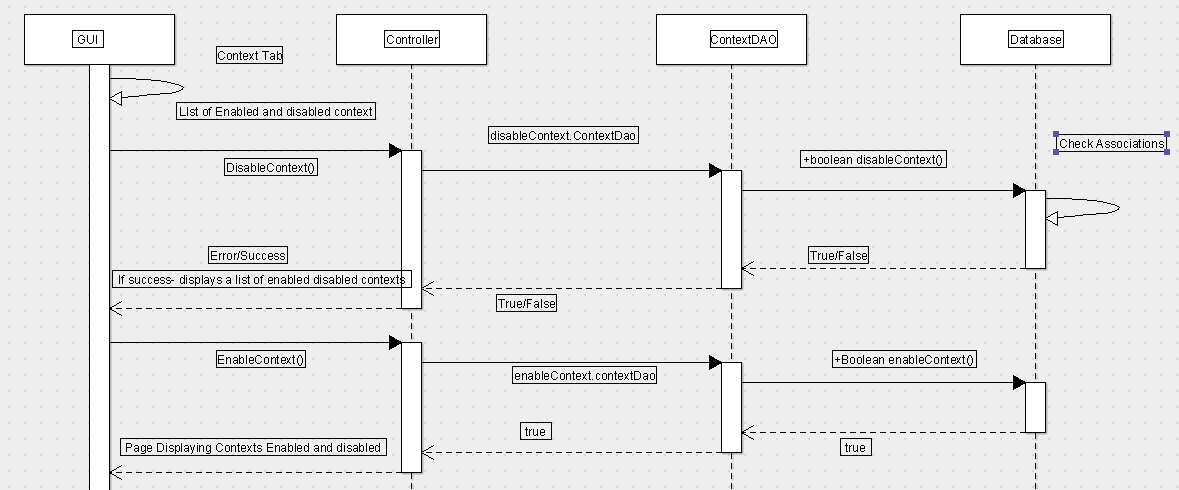
1) The controller will invoke the unified algorithm for the compare

2) The compare algorithm will return the report object .

3) That report object can be stored in the Database by the said DAO.

4) The controller will then send the GUI the report to display.

Enable Disable context



The steps for the enable disable context sequence are as follows

1) The GUI gets the list of all the enabled and disabled contexts.

2) The GUI then we have the option of either enabling the context or disabling it.

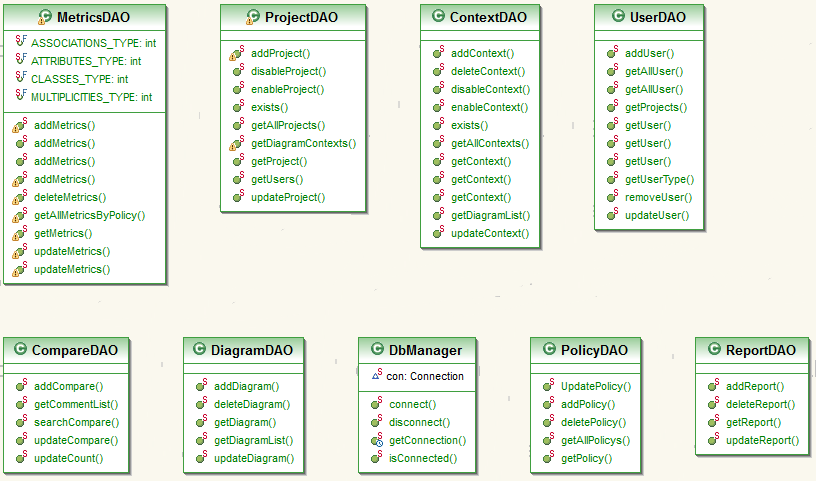
3) The said controller will call the DAO to change the filed in the database to reflect the change on the GUI.

# Class diagrams

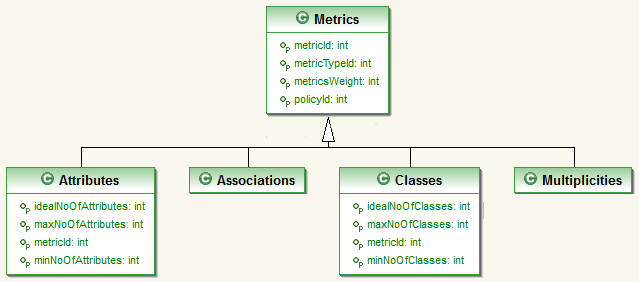
These diagrams give a fair description of the various classes involved in the controller. Mostly consisting of servlet classes, DAO classes and domain objects (Classes which describe the data being inserted and or being retrieved from the database)

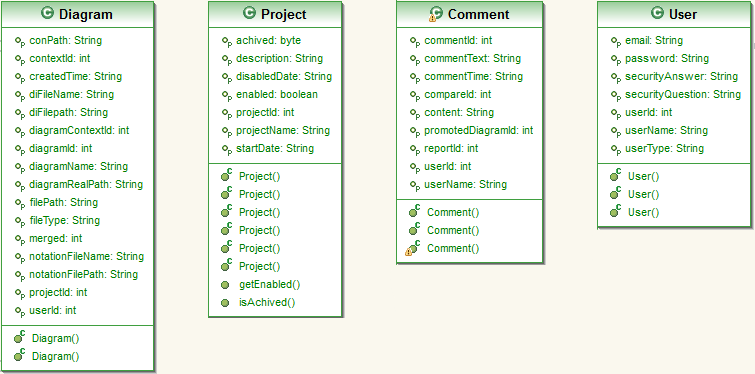
### DB Classes

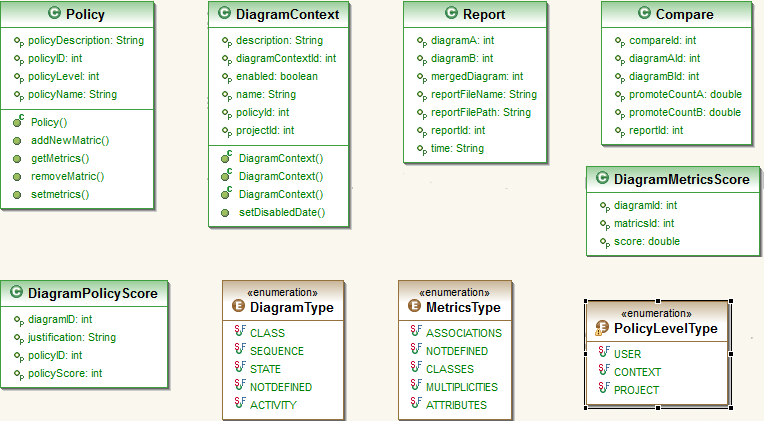
These include DAO classes. These classes create a level of abstraction between database access and application code.



### Domain classes

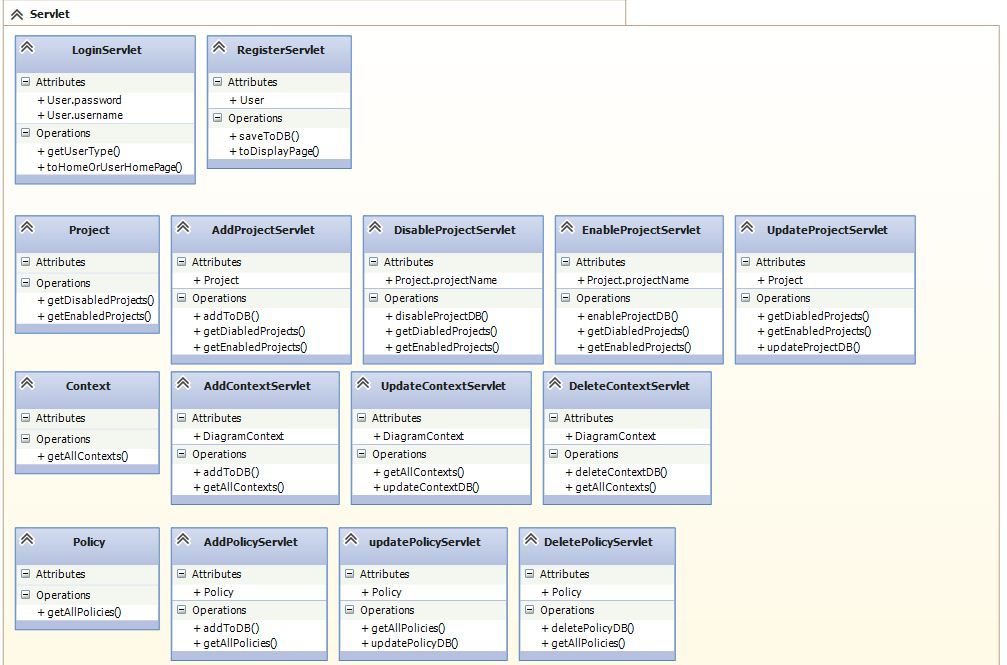
Domain classes describe the data that is being retrieved and or inserted into the database. The data access methods from afore mentioned DAO classes accept as input and return as output data of the type of these classes. 





### Servlet classes

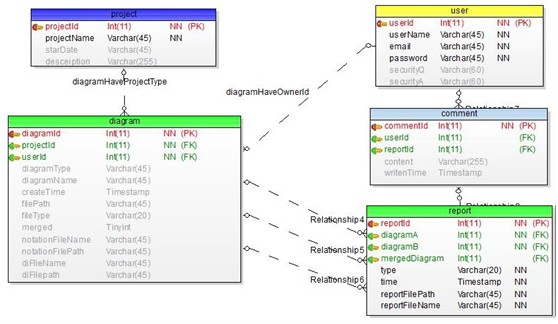
These are the classes which contain the code which connect the GUI to the engine of the application. They act as a gateway between user and the system.



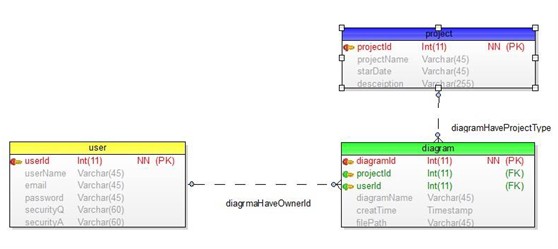
## Database

(Jiayuan)

According to the improvement of the project made to the project, we need to make several changes. This section shows the basic database structure.



Modification for User – Project – Diagram



Changes:

1. ProjectId used to be put in the user table. Now we put the diagram table in the middle of user and project, so the user table and project table do not have a direct relationship any more. Diagram table contains both projectId and userId.

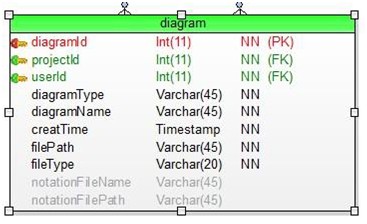
2. Add start date in the project table.

Benefits**:**

1. Improved the old version where one user could only join in one project. Now one user could join as many projects as he/she wants.

2. Diagrams can be grouped by projects. So the user could choose diagrams from the same project to compare or merge.

Modification for Diagram Table



Changes:

3. Add diagram type attribute. (class diagram or sequence diagram)

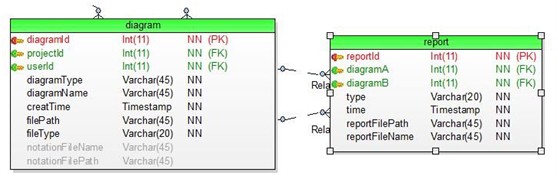
4. Add file type attribute. (Ecore file or UML file)

5. Add notation file name and file path

Benefits:

1. Make the diagram table support new diagram and data types.

Modification for Diagram – Report



Changes**:**

**2.** Add type attribute to report table. (compare or merge)

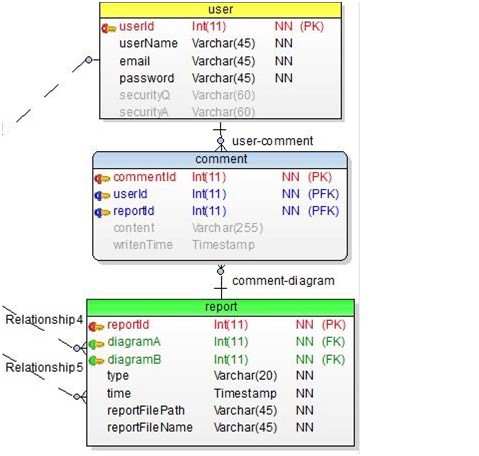
**3.** Add reportFileName attribute to report table.

**4.** Switch comparedTime attribute to time.

Benefits**:**

5. Make the report table support new functions and data types.

Modification for User – Comment



Changes**:**

6. Comment table connects to the report table instead of diagram table.

7. Comment table contains reportId instead of diagramId.

Benefits**:**

1. User could leave his/her comment under the compare or merge result. Old version database could only let user leave the comments under one diagram.

Example SQL Insert Queries

See Appendix B: Database Query for the query needed to set up the MySQL tables used by the Spring 2013 version of ClubUML. This section shows some queries that can be used to insert new values into the MySQL tables and then check the results.

***Insert Project***

Insert into project VALUES (000001, 'projectA', 'Jan2013', 'projectDesceiption')

select \* from project



***Insert User***

insert into user VALUES (000001, 'admin', 'admin@clubuml.com', 'adminpd','','')

select \* from user



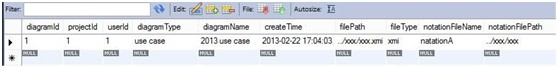
***Insert Diagram***

insert into diagram (diagramId, projectId, userId, diagramType, diagramName, filePath, fileType, notationFileName, notationFilePath)

VALUES (000001, 000001, 000001, 'use case','2013 use case','../xxx/xxx.xmi','xmi',

'natationA' ,'../xxx/xxx')

select \* from diagram



***Insert Report***

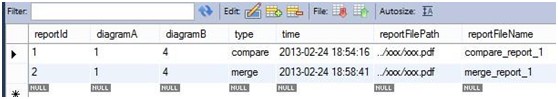
insert into report (reportId, diagramA, diagramB, type, reportFilePath, reportFileName)

VALUES (000001, 000001, 000004, 'compare','../xxx/xxx.pdf','compare\_report\_1')

insert into report (reportId, diagramA, diagramB, type, reportFilePath, reportFileName)

VALUES (000002, 000001, 000004, 'merge','../xxx/xxx.pdf','merge\_report\_1')

select \* from report

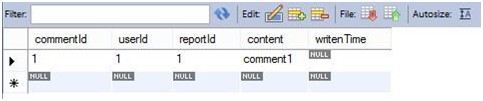


***Insert Comment***

insert into comment(commentId, userId, reportId, content)

VALUES (000001, 000001, 000001, 'comment1')

select \* from comment



# Test

## Test Cases :

(Aindra)

**Register New Account** :

Register With Valid data :-

1. Description : To test the new user registration

2. Pre-Condition: User goes to the ClubUML homepage

3. Post-Condition: User enters valid input to the registration form and navigates to the home page.

4. Data required: The user needs to enter the non-existing username

5. Steps :

a) User should find and click the 'Register' button

b) User should input the username, password and test it with multiple alphanumeric characters

c) The user should enter exactly the same password in the 'Confirm Password' box.

d) The user enters an email and tests it with multiple addresses and then selects a security question from the drop down menu and enters the answer for the question in the next tab.

e) The user clicks on the 'Register' button and gets a message 'Registration Success'.

Register With Invalid data :-

1. Description : To test the new user registration with invalid data

2. Pre-Condition: User goes to the ClubUML homepage

3. Post-Condition: When the user enters an invalid input data in any of the text field in the registration form, he gets a message stating the input condition which is invalid.

4. Data required: No necessary data is required by the user to run this test

5. Steps :

a) User performs all the similar steps in the previous steps and inputs invalid data in the various text fields. The various use-cases for entering the invalid data is described in the subsequent steps.

b) Empty username/ Username longer than 10 characters/ Use of

non-alphanumric characters in the username

c) Empty password/ Password longer than 10 characters

d) The user should enters a different password in the 'Confirm Password' box.

e) The user enters an empty email or an email in a wrong format.

f) The user leaves the security question field and the answer field blank

g) The user clicks on the 'Register' button.

h) Expected result: Registration fails, user is provided with appropriate message and should be given the chance to correct the errors and continue. The Error messages should be disappeared after continuing.

* Register With Existing Username :-

1. Description : Tests the ability of the system to detect that the username that is being entered during registration is not available

2. Pre-Condition: The Registration test case should succeed before this test case is performed

3. Post-Condition: The user should fail to create the account and should stay on the registration page

4. Data required: No necessary data is required by the user to run this test

5. Steps :

a) Register a user with a specific username according to Register New Account use case

b) Try to register another user with the same username.

**Upload Diagram:**

* Upload Valid file:-

1. Description : Test the ability of the system to accept valid files from the user

2. Pre-Condition: The login should be successful.

3. Post-Condition: The website can successfully accept Ecore and papyrus formats for class and sequence diagrams

4. Data required: Valid Ecore and Papyrus files

5. Steps :

a) Login and then the user should land into the page which has an upload feature

b) Select Ecore file format

c) Click choose file and in the dialog box, select the ecore class diagram.

d) Click the 'Upload' button

e) Perform similar steps for Papyrus file format. User should be able to select multiple files if required in step 4.

f) Repeat 1-4 steps for sequence diagrams.

* Upload Invalid file :-

1. Description : Tests the ability of the system to reject invalid files, as denoted in the alternate flow of Upload Use case

2. Pre-Condition: The login should be successful.

3. Post-Condition: Test succeeds, meaning that website behaves in a way that no invalid file can be uploaded.

4. Data required: Valid and invalid Ecore and Papyrus files for class diagram and sequence diagram.

5. Steps :

a) Login and then the user should land into the page which has an upload feature

b) Select Ecore file format

c) Click choose file and in the dialog box, select the Papyrus class diagram.

d) Click the 'Upload' button

e) Perform similar steps for random file formats except Ecore format. Also try sending multiple files that only some of them are invalid or not Ecore format

f) Perform the above steps and select Papyrus in step b.

g) Perform the same steps for sequence diagrams.

**Login :**

* Login using valid username and password :-

1. Description : Tests the ability of the system to accept a valid username and password

2. Pre-Condition: None

3. Post-Condition: If the login succeeds, the user lands into its main home page

4. Data required: Valid username and password of a registered user

5. Steps :

a) Go to the ClubUML page and enter the valid username and password of a registered user in the appropriate text boxes

b) Click Login and the user lands into the main home page

* Login with an invalid username :-

1. Description : Tests the ability of the system to reject an invalid username

2. Pre-Condition: None

3. Post-Condition: If the test succeeds, the system rejects the invalid username with an appropriate message. The server denies the attempt and redirects the user to the Login failed page.

4. Data required: Invalid username

5. Steps :

a) Go to the ClubUML page and enter an invalid username and random password of a in the appropriate text boxes

b) Click Login and the user lands into the Login failed page

* Login with an invalid password :-

1. Description : Tests the ability of the system to reject an invalid password

2. Pre-Condition: None

3. Post-Condition: If the test succeeds, the system rejects the invalid password with an appropriate message. The server denies the attempt and redirects the user to the Login failed page.

4. Data required: Invalid password

5. Steps :

a) Go to the ClubUML page and enter the valid username and an invalid password in the appropriate text boxes

b) Click Login and the user lands into the Login failed page.

**Compare Diagrams :**

* Compare Uploaded Diagrams :-

1. Description : Tests the ability of the system to compare two valid uploaded Ecore or Papyrus diagrams

2. Pre-Condition: The user should have logged-in and the Upload file test case should succeed

3. Post-Condition: The system is able to compare two diagrams and successfully output the report.

4. Data required: Two diagrams in valid formats with a known set of differences to fit the test purposes

5. Steps :

a) Upload two diagrams for the purpose of test.

b) Select the diagrams and click on the 'Compare' button

* Save Comparison Report:-

1. Description : Tests the ability of the system to save the comparison report

2. Pre-Condition: The user should have logged-in and the Upload file test case and Compare uploaded diagram test case should succeed

3. Post-Condition: The system is able to export the report as PDF

4. Data required: Two diagrams in valid formats should be uplaoded for test

5. Steps :

a) Upload two diagrams for the purpose of test.

b) Select the diagrams and click on the 'Compare' button

c) On the report page click on the 'Save' button

**Add New Project:**

**Add project with Active Status:-**

1. Description: To test the add new project feature with Active Status

2. Pre-Condition: User should successfully log into the application as PolicyManager role

3. Post-Condition: New project is added, and PolicyManager is able to see the newly added project listed under the Current Projects in project screen

4. Data required: The user needs to enter unique Project Name, Project Description and project status: Active (by Default Active)

5. Expected Result: PolicyManager should be able to add a new Active Project

6. Steps:

a) PolicyManager should click on the “Add Project” button in the project tab of the home page

b) A new project entry window should pop up with input fields:

 Project Name

 Description

 Project Status (Drop down with option: Active Project and Disabled Project)

c) PolicyManager should enters the Project Name, Description, and selects Project status as Active Project

d) PolicyManager clicks on the “Add Project” button to save the project details

e) Project should be stored in the database in project table with unique projectId assigned to it by system, enabled as true, StartDate as Current System Date, and disabledDate as null.

f) The newly added project should be displayed along with the existing projects in the Current Projects section in Project Screen  
  
**Add Project with Disabled status:-**  
1. Description: To test the add new project feature with disabled status  
2. Pre-Condition: User should successfully log into the application as PolicyManager role  
3. Post-Condition: New project is added, and PolicyManager is able to see the newly added project listed under the Disabled Projects in project screen  
4. Data required: The user needs to enter unique Project Name, Project Description and project status: Disabled (by Default Active)  
5. Expected Result: PolicyManager should be able to add a new Disabled Project  
6. Steps:

a) PolicyManager should click on the “Add Project” button in the project tab of the home page

b) A new project entry window should pop up with input fields:

 Project Name

 Description

 Project Status (Drop down with option: Active Project and Disabled Project)

c) PolicyManager should enters the Project Name, Description, and selects Project status as Disabled Project

d) PolicyManager clicks on the “Add Project” button to save the project details

e) Project should be stored in the database in project table with unique projectId assigned to it by system, enabled as false, StartDate as Current System Date.

f) The newly added project should be displayed along with the existing projects in the Disabled Projects section in Project Screen.

## Test Results

**Test Case Generation [Login page]:**

As per the test cases in the previous section, and few additional ones added, the following are

the test cases generated

 **Precondition**: Login page with blank fields

Test Check Item Test case Steps to Test data / Excepted Results Actual case Objective Execute Input Result

TC001 Login Page Leave all Click Login Username: By leaving all fields PASS

fields as NULL as blank and on click

blank and Password: Login button then

click Log-in NULL mandatory symbol (\*)

button should appear in

front of Username

and Password fields.

TC002 Username Enter Invalid NA Username: By entering invalid PASS

Username xyz [not in Username then an

DB] error message

should appear as "

Please Enter Valid Username.

TC003 Username Enter Valid NA Username: Allow user to PASS

username ClubTester proceed.

TC004 Password Enter Invalid NA Password: By entering invalid PASS

password \*\*\*\*\*\*\*\*\* password, an error

message should

appear as "Please

Enter Valid password".

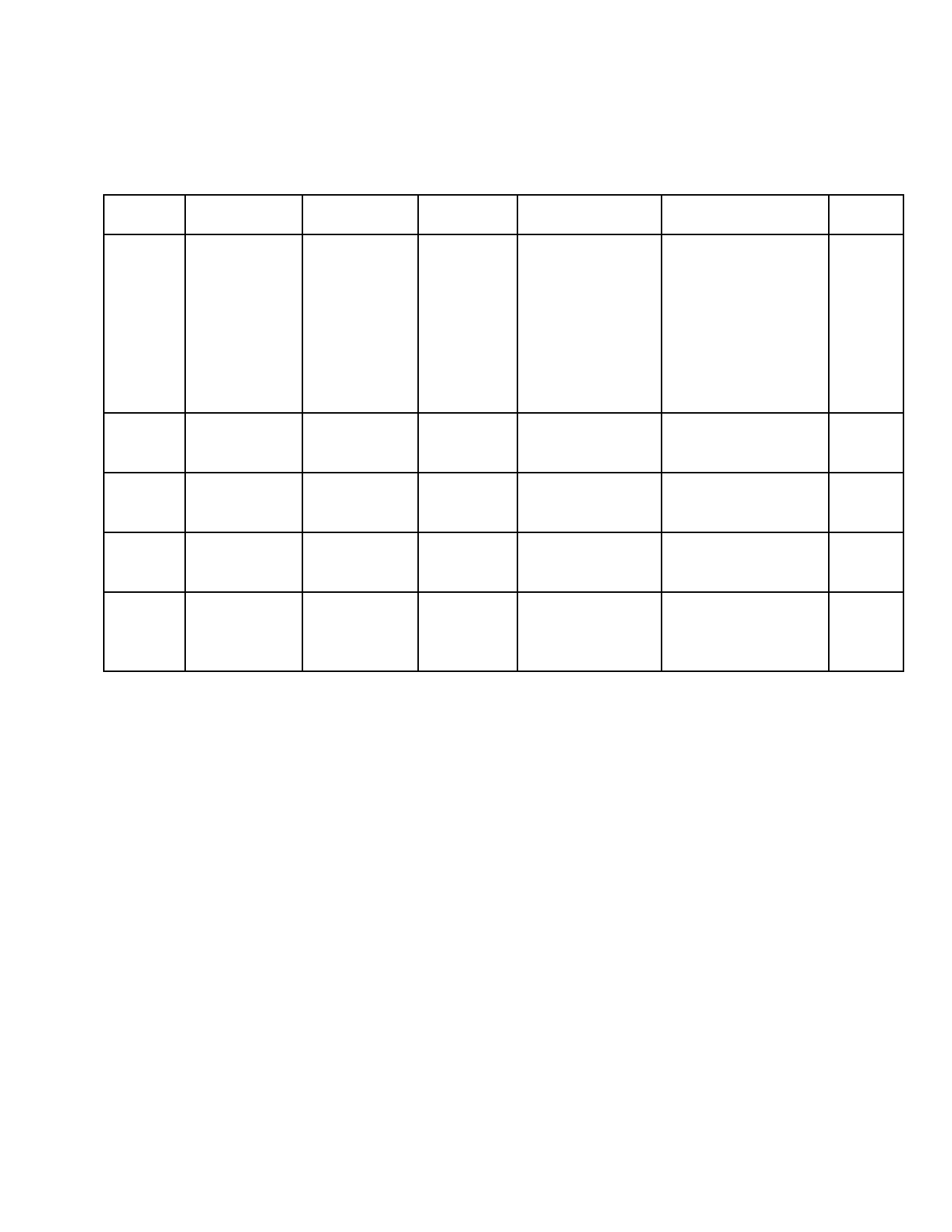
TC005 Password Enter valid NA Password: Allow user to PASS

password \*\*\*\*\*\*\*\*\*\*\*\*\* proceed.

TC006 Log-in button Correct Click Login Take user to PASS

credentials button respective page.

 **Post Condition**: Continue with access to the respective page after Logging.

**Test Case Generation [Registration page]:**

 **Precondition**: Registration page with blank fields

Test Check Item Test case Steps to Test data / Input Excepted Results Actual case Objective Execute Result

TC001 Registration Leave all Click Username: By leaving all fields PASS

Page fields as Create NULL as blank and on

blank and account Password: click Login button

click button NULL then mandatory

Register Email id: symbol (\*) should

button NULL appear in front of

Username,

Password and email id fields.

TC002 Username Enter Valid NA Username: Message PASS

username Sarvesh "Registration

completed" appear.

TC003 Password Enter valid NA Password: Message PASS

password \*\*\*\*\*\*\*\*\*\*\*\*\* "Registration

completed" appear

TC004 Email id Enter valid NA Email: Message PASS

email id Svic@gmail.com "Registration

completed" appear

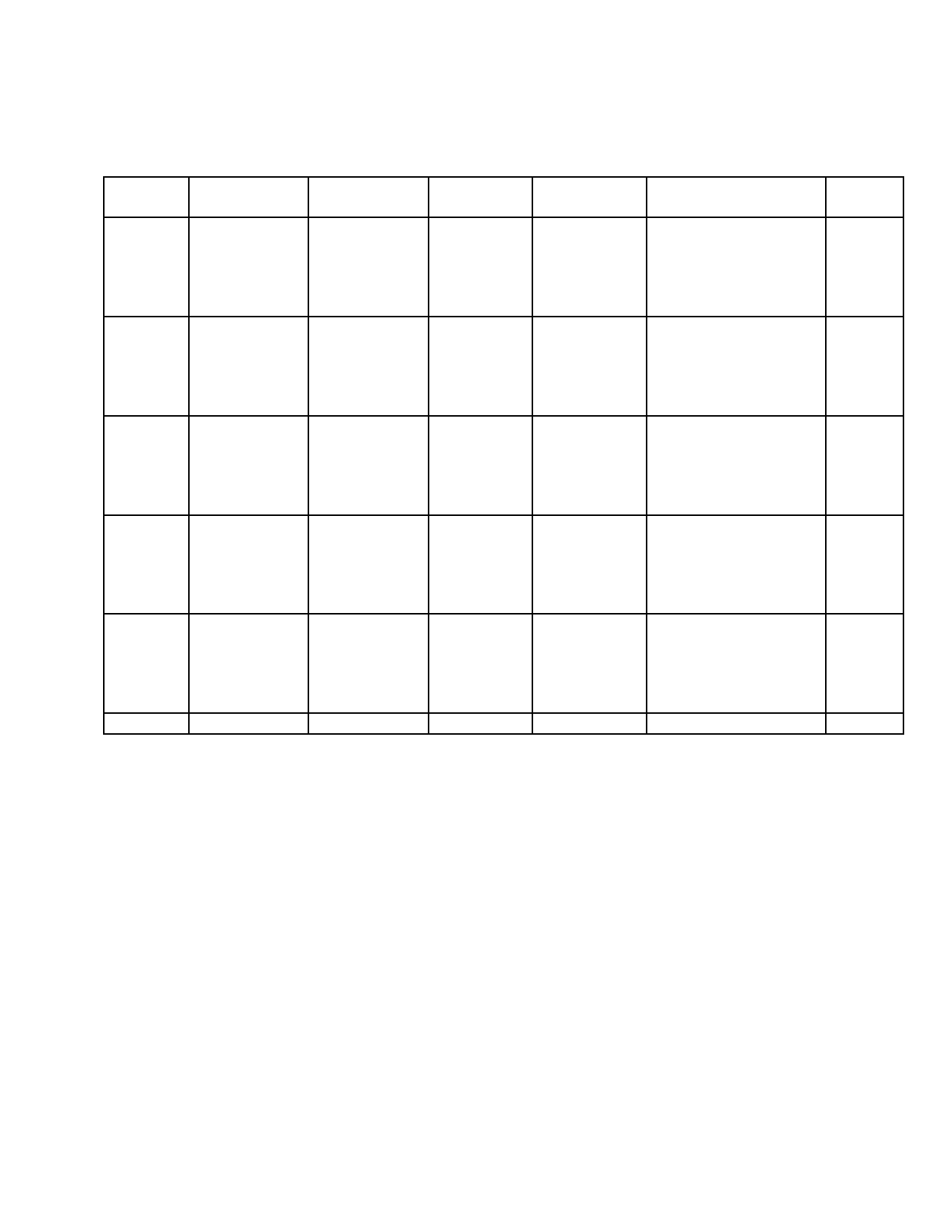
TC006 Create Correct Click Message PASS

account credentials Create "Registration

button account completed" appear

button

 **Post Condition**: Continue with access to the settings page after registration.

**Test Case Generation [Upload page]:**

 **Precondition**: The login should be successful.

Test Check Item Test case Steps to Test data / Excepted Results Actual case Objective Execute Input Result

TC001 Upload file Leave all Click NA Shouldn't proceed to PASS

fields as Upload compare and merge

blank and functionality.

click Upload

button

TC002 Upload file Select Ecore Select NA Shouldn't proceed to PASS

drop down ECORE compare and merge

selection and Select XMI functionality.

try selecting file Click

XMI files Upload

TC003 Upload file Selecting Click NA Allow user to PASS

Ecore and Upload proceed.

further

proper ecore files selection

TC004 Upload file Select XMI in Click NA Shouldn't proceed to PASS

drop down Upload compare and merge

selection and functionality.

try selecting

ecore files

TC005 Upload file Selecting Click NA Allow user to PASS

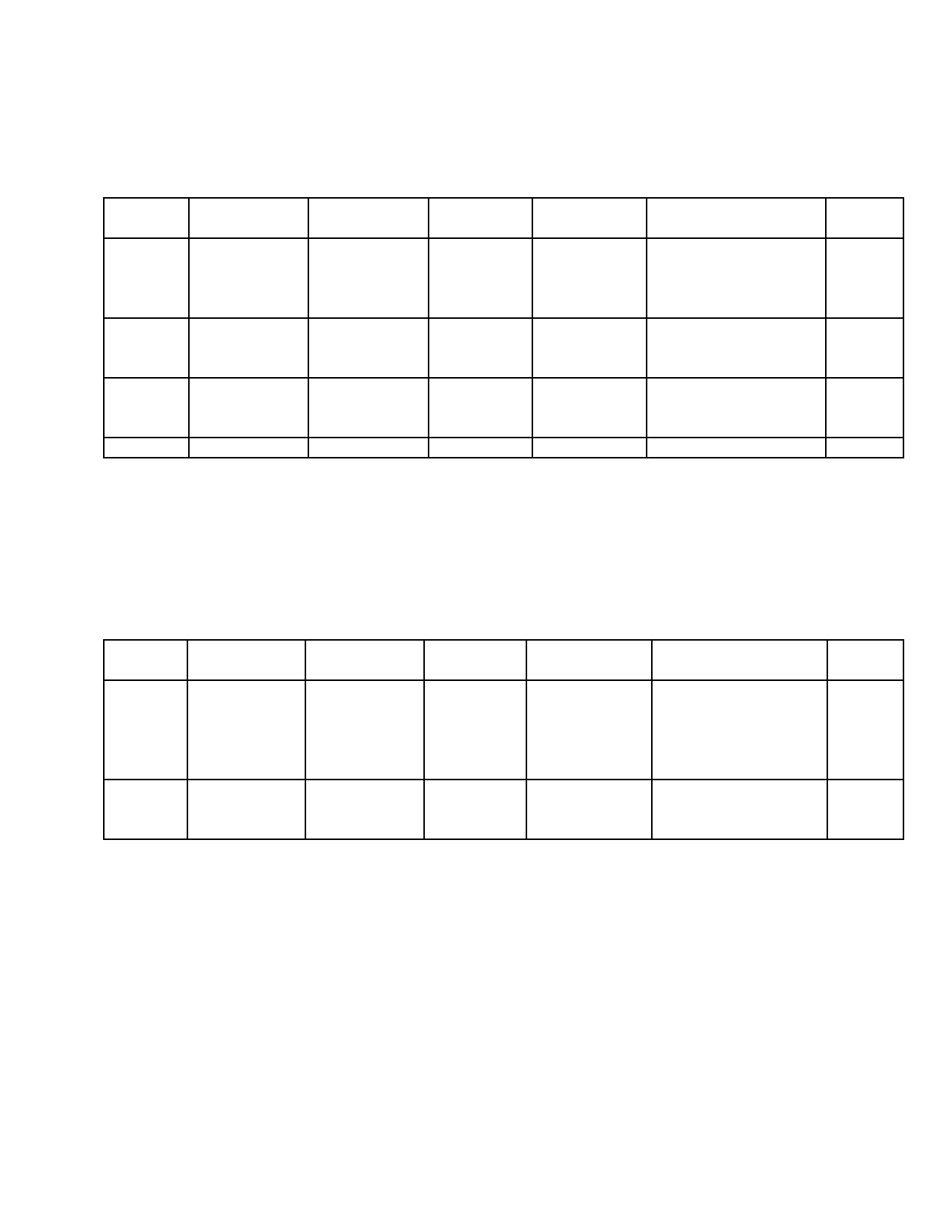
XMI and Upload proceed.

further

proper XMI

files selection

 **Post Condition**: Continue with Compare and Merge functionality.

**Test Case Generation [Compare Upload page]:**

 **Precondition:** The user should have already logged-in and the Upload file test case should

succeed.

Test Check Item Test case Steps to Test data / Excepted Results Actual case Objective Execute Input Result

TC001 Compare Selecting Click NA Shouldn't allow to PASS

diagram only one Compare proceed and

diagram generating the

report.

TC002 Compare Selecting Click NA Shouldn't allow to PASS

diagram more than Compare proceed for

one diagram comparing

TC003 Compare Selecting Click NA Compare and PASS

diagram exactly two Compare generate report

diagrams

 **Post Condition**: The system is able to compare two diagrams and output the report.

**Test Case Generation [Display diagrams]:**

 **Precondition**: User already uploaded the files

Test Check Item Test case Steps to Test data / Excepted Results Actual case Objective Execute Input Result

TC001 Test1.ecore Display the Choose a Ex: Successfully PASS

UML diagram Test1.ecore displays the UML

diagram diagram

Click on

display

TC002 Nothing. To throw null Click on Nothing Successful display PASS

Blind display exception display of exception

click

 **Post Condition**: nothing

# Conclusion

All of us did big contributions to the project though most of us didn’t have much experience. We’ve learned to use new tools (Git, JIRA) and new SDLC (Scrum) which are popular in industries. These will be helpful for our careers. Also it made us have a better understanding of Software Engineering after the hands on experience.

Scrum is helpful for small team development like us. Some of the team members had experience on it, which makes it easy to start. We used JIRA as the tool that helped us to track the tasks assigned to each person and everyone worked on those tasks. The problem we had was that we didn’t have much “daily stand-up” meetings to synchronize our progress. Ideally, team members should meet everyday but we only met once a week. If we meet more frequently, each of us would have a better understand of whole system and finish the works more efficient.

Git also helped us to achieve the goal. It is very popular in the industries and very helpful when doing team development. It was a hard time to use it at the beginning since most of us knew nothing about it at that time. After we used several times and read the instructions for Git bash and EGit (Eclipse Git), Git became a very convenient and useful tool for us. The problem we had was that we always met conflicts when merging code together. That caused by committing .class files or changes by different persons to same file. Git would be more powerful if we overcome this problem.

# Appendix

## A. Smart Policy Comparison Algorithm Overview

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document Type |  |  | Algorithm Summary |  |  |
| Activity |  |  | Analysis |  |  |
| Document Version |  |  | 1.0 |  |  |
| Document Status |  |  | Draft |  |  |
| Date | **Version History** | **Version** |  | **Description** | **Author** |
| 10.28.2013 | 1.0 | Initial Draft |  | Create draft of comparison algorithm overview | Seth Lee |
| 11.12.2013 | 1.1 | Updated Draft |  | Update for team feedback on proposed algorithm | Seth Lee |

**Brief Description**

The purpose of this document is to review at a high level the algorithm that will be used to support the 'smart policy' functionality incorporated into the ClubUML application as it pertains to the comparison of two Diagrams, of the same Context, in the same Project.

'Smart policy' functionality refers to the systematic application of a set of policies, or rules, to the comparison of two UML diagrams (initially for UML Class diagrams) to determine which diagram is preferred of the two diagrams. The comparison algorithm relies on the definition of a set of policies that utilize the metadata that comprises the UML diagrams.  
Each policy is associated with a policy score, which provides the algorithm with a sort of weight for the application of the policy to the diagram's metadata.

Refactoring functionality refers to the systematic application of a set of rules to an individual UML diagram (Class diagram initially) to restructure the diagram at an optimized level of abstraction.

**Preconditions**

In the spirit of comparing diagrams that are at a high level similar, it is assumed that the diagrams being compared belong to the same Context, a construct designed to partition diagrams within a project based on some level of functional or technical similarity.

Configurable point values have been assigned in the database to each smart policy and refactoring rule.  
Rules have configurable point values to allow the application to be flexible and respond to the needs of the client organization.

Smart policy policies may be applied either at the default or context-levels. In the absence of a context-level policy, the default policy will be applied to the comparison. The algorithm described in this document assumes that the appropriate policy level has been determined (e.g. default policy versus context-level policy) and is being provided to the algorithm as an input.

**Basic Flow**

1. Apply smart policy rules

1.1 For each diagram:

1.1.1 Gather metrics

1.1.1.2 Total number of attributes

1.1.1.3 Total number of Classes

1.1.1.4 Number of associations missing multiplicities

1.1.1.5 Number of classes without associations/relationships

1.1.2 Calculate the average number of attributes per Class. For each class:

1.1.2.2 Calculate the total number of attributes

1.1.2.3 Calculate the number of Classes

1.1.2.4 Calculate the average number of attributes / Class

1.1.2.5 Normalize the number of associations missing multiplicities by dividing the number of missing multiplicities (1.1.1.4) by the total number of classes (1.1.2.3).

1.1.2.6 Normalize the number of missing associations by dividing the number of missing associations (1.1.1.5) by the total number of classes (1.1.2.3).

1.3 Calculate smart policy score:

1.3.1 Average number of attributes per Class

1.3.1.2 For each diagram, compare the average number of attributes/Class (1.1.2.4) to the configuration value for the average attributes/Class rule for the ideal number of attributes/Class (idealNoOfAttributes), the maximum number of attributes/Class (maxNoOfAttributes) and the minimum number of attributes/Class (minNoOfAttributes).

* If the average number of attributes/Class = idealNoOfAttributes, averageAttributesScore is 0.
* If the (A) average number of attributes/Class > idealNoOfAttributes < maxNoOfAttributes OR (B) minNoOfAttributes < average number of attributes/Class < idealNoOfAttributes, then averageAttributesScore is Absolute Value (idealNoOfAttributes - average number of attributes/Class) \* avgAttributesBetweenPoints
* If the (A) average number of attributes/Class >= maxNoOfAttributes , averageAttributesScore is (avgAttributesMaxPoints + ((average Number of Attributes - maxNoOfAttributes) \* avgAttributesOverMaxPoints ) OR (B) average number of attributes <= minNoOfAttributes, averageAttributesScore is (avgAttributesMinPoints + ((minNoOfAttributes - average number of attributes) \* avgAttributesUnderMinPoints)

1.3.2 Total number of Classes

1.3.2.1 For each diagram, compare the total number of Classes (1.1.1.3) to the configuration value for the total number of Classes rule for the ideal number of total Classes (idealNoOfClasses), the maximum number of total Classes (maxNoOfClasses) and the minimum number of total Classes (minNoOfClasses).

* If the total number of Classes = idealNoOfClasses, totalClassesScore is 0.
* If the (A) total number of Classes > idealNoOfClasses < maxNoOfClasses OR (B) minNoOfClasses < total number of Classes < idealNoOfClasses, then totalClassesScore is Absolute Value (idealNoOfClasses - total number of Classes ) \* totalClassBetweenPoints
* If the (A) total number of Classes >= maxNoOfClasses, totalClassesScore is (totalClassMaxPoints + ((total number of Classes - maxNoOfClasses) \* totalClassOverMaxPoints) OR (B) total number of classes <= minNoOfClasses, totalClassesScore is ((minNoOfClasses - total number of Classes) \* totalClassUnderPoints)

1.3.3 Associations that are missing multiplicities

1.3.3.1 For each diagram, calculate the missing multiplicities (1.1.1.4) score

* If there are no missing multiplicities, missingMultiplicitiesScore is 0.
* If there are missing multiplicities, missingMultiplicitiesScore = normalized number of missing multiplicities (1.1.2.5) \* missingMultiplicitiesPoints

1.3.4 Classes that are missing associations

1.3.4.1 For each diagram, calculate the missing associations (1.1.1.5) score

* If there are no missing associations, missingAssociationsScore is 0.
* If there are missing associations, missingAssociationsScore = normalized number of missing associations (1.1.2.6) \* missingAssociationsPoints

1.3.5 Total smart policy score

1.3.5.1 For each diagram:

smartPolicyScore = averageAttributesScore + totalClassesScore + missingMultiplicitiesScore + missingAssociationsScore

1. Refactor each diagram to achieve an optimized model:

2.1 Syntactical validation

2.1.2 No two classes have the same name

2.1.3 No two types have the same name

2.1.4 No two attributes within a class have the same name

2.1.5 No attributes of a class have the same name as any attributes of its super classes

2.2 Optimization

2.2.1 For each node, identify any attributes of children that are the same in all children and bring them up to the superclass. Remove these attributes from the children.

2.2.2 For each node, identify any attributes that remain in the children following the previous step that are common between children. Create an 'extract superclass' as a subclass of the node that will be the super class of the children. Bring the common attributes up to the new 'extract superclass' and remove from the children.

2.2.3 Repeat steps for each branch moving down to the leaves.

2.3 Examine nodes for any share attributes with other nodes. If several nodes share common attributes, create a super class for those nodes and bring shared attributes up to the new super class, removing them from the nodes.

2.4 Calculate refactoring score

2.4.1 Calculate the number of occurrences of step 2.2.1 (shared attributes of all children rolled up to node).

2.4.2 Calculate the number of occurrences of step 2.2.2 (shared attributes of a subset of children rolled up to new 'extract superclass').

2.4.3 Calculate the number of occurrences of step 2.2.4 (shared attributes of root nodes rolled up to new superclass).

2.4.4 A refactoring score is calculated for each diagram if it has passed syntactical validation.

2.4.4.1 Refactoring score (refactoringScore) = (2.4.1 \* siblingAttributesAbstractionPoints) + (2.4.2 \* subsetSiblingAttributesAbstractionPoints) + (2.4.3 \* rootAttributesAbstractionPoints)

1. Calculate total score

The scoring of two compared diagrams is based on two principles:

* Achieving a lower smart policy score
* Achieving a lower refactoring score

3.1 Scenario 1 : Both diagrams pass syntactical validation

3.1.1 The diagram with the lower sum of smartPolicyScore + refactoringScore is promoted

3.2 Scenario 2 : Diagram A passes syntactical validation, Diagram B does not

3.2.1 Diagram A is promoted

3.3 Scenario 3 : Neither diagram passes syntactical validation

3.3.1 The diagram with the lower smartPolicyScore is promoted

**References**

Algorithmic logic associated with the refactoring process is based on the following:

Lano, K. & Kolahdouz-Rahimi, S. (2013) *Case Study: Class Diagram Restructuring*

**Additional Reference:**

The ClubUML Spring 2013 project summary report containing much additional information can be found [here](https://blackboard.neu.edu/bbcswebdav/pid-7155118-dt-content-rid-10226505_1/xid-10226505_1).

Additional recommendations regarding project guidelines may be found [here](https://blackboard.neu.edu/webapps/portal/frameset.jsp?url=%2Fwebapps%2Fblackboard%2Fcontent%2FlaunchLink.jsp%3Fcourse_id%3D_1851669_1%26tool_id%3D_137_1%26tool_type%3DTOOL%26mode%3Dview%26mode%3Dreset).

## B. Glossary:

Branch (code) – A version of the Git repository that has been individually allocated to a developer for development and testing.

Bug – In Jira, a bug tracks a software defect identified during development or testing that will be reviewed by the project team as part of the backlog. Team members can track a bug by creating a new Issue of type ‘Bug.’

Compare (ClubUML feature) – A feature of the ClubUML application which allows a user to see two UML diagrams side-by-side, and determine manually or systematically which diagram is preferred (‘better’).

Context – Logical grouping of UML diagrams based on a common trait or feature. Each Project in ClubUML is expected to have at least one Context associated with it.

Diagram context – Context associated with a UML diagram at the time that it is uploaded to the ClubUML application.

Ecore – File format provided by the Eclipse Modeling Tool (or other UML modeling tools supporting the Eclipse Modeling Framework).

Issue – In Jira, an issue is an item that may be of type ‘Epic’, ‘User Story’, ‘Bug’, ‘Task’ or ‘New Feature.’ An issue represents a unique entity in Entity with an identifier.

Merge (ClubUML feature) – A feature of the ClubUML application which allows a user to manually or systematically bring together individual elements of two UML diagrams into one consolidated UML diagram.

Merge (code) – To merge changes to source code or other Git repository items from an individual developer’s branch into the main shared repository.

Metric - A rule associated with a policy, used by the Smart Policy scoring algorithm to assess one UML diagram versus another diagram. (For example, in the current scope of Smart Policy functionality supported by the ClubUML application, there are 4 possible metrics associated with Smart Policy functionality. The metrics are: Average number of attributes; Total number of Classes; Number of missing associations; Number of missing multiplicities.)

Policy - A set of metrics associated with a context, used by the Smart Policy scoring algorithm to compare or merge two UML diagrams.

Policy Manager – End user who is responsible for establishing Context designations in the ClubUML application, for the establishment of Smart Policy rules, and for associating Contexts with a set of rules.

Point – In the context of Smart Policy, points are assigned to the outcome of the application of a rule to the analysis of the element(s) of two UML diagrams. For example, a rule is associated with a number of points, based on the relative weight of that rule to the Smart Policy algorithm.

Refactoring score – Value associated with a UML diagram being compared or merged by the application, which represents the number of refactoring steps applied by to the diagram and the weight of those steps. In general, a lower score indicates a 'better' diagram.

Rule – In the context of Smart Policy, a rule is the discrete algorithmic function which is applied to elements of two UML diagrams to determine which diagram is preferred. An example of a rule might be a function that assigns preference to a diagram with a greater number of levels of hierarchy (e.g. super and sub classes). Another example of a rule might be a function that assigns preference to a diagram which includes operations (methods) over one that does not.

Smart policy – The systematic application of a set of rules when analyzing two UML diagrams, wherein the system emulates the logical process of determining which aspects of a diagram are preferable to those of another diagram.

Smart Policy score - Value associated with a UML diagram being evaluated based on a set of metrics as defined by the Smart Policy. The score is comprised of a tally of the points associated with the rules/metrics being against which the diagram is being evaluated. In general, a lower score indicates a 'better' diagram.

Sub-task – In Jira, this is a work item that may be directly created from a user story. The majority of work items that will be tracked for the project are created as sub-tasks.

Task – In Jira, this is a work item, assigned to a team member, against which the team member will track effort and status that does not align with a user story. Work items that align with user stories will be created as sub-tasks.

User story – In Jira, this is corresponds to an aspect of a system feature to be completed by the team. A story is linked to one or more tasks, which are assigned to the team members responsible for completing the tasks.

## C. Guidelines for future batches:

* The DiagramParser is created in a very flexible way such that you can have it pull put different data from the files. For now the DiagramParser can extract Classes and Attribute information from the file. Note that the parser EncoreDiagramParser and XMIDiagramParser inherits from the ClassDiagramParser which in turn inherits from the DiagramParser. These two are defined in the package – controller.diagramParser and are not to be confused with the legacy xmi and encore parsers elsewhere. These two – ENClassDiagramParser and the XMIClassDiagramParser internally takes the corresponding legacy parsers and uses them to extract data as per the abstract operations defined in the parent class – ClassDiagramParser. Any new operation to be added, should be added as an abstract method in the ClassDiagramParser and then implemented in the ENClassDiagramParser or XMIClassDiagramParser.
* De-couple the Merge Algorithm, which is using the XMI Parser. You can do this by using the new DiagramParser instead of the XMI parser.
* The Scoring algorithm which was implemented for Class Diagram – ClassDiagramScoreGenerator, uses preset values of penalty points. They are stored in a property file – clubUMLProperties.properties
* Each diagram should be uploaded against a particular context. For now, there is a default context created for each project.

## D. Guidelines and suggestions for future batches (Controller side)

* Detach hardcoded value from file into a config file and read them during runtime.
* Uploading diagram to COE server fails because we don’t have permission to upload images on the server yet. Look into this to rectify operation of application on the server side
* DB login values are hardcoded into DBManager.java. This is usually the point of failure when multiple users use different login data for their local mysql files
* DBManager.java has multiple instances of hard coded values for login as per differing OS. Server is based in linux which differs from flow for windows and MacOS. This is a point of failure when deploying to server
* Take care not to commit individual config files
* Servlet for context needs to be created, consider implementing relation between context, user, project and policy as per documentation.

## E. Database Creation Scripts

/\*

Created: 2013/2/15

Modified: 2013/11/25

Model: MySQL 5.0

Database: MySQL 5.0

\*/

-- Create tables section -------------------------------------------------

-- Table project

CREATE TABLE project

(

projectId Int(11) NOT NULL AUTO\_INCREMENT,

projectName Varchar(45) NOT NULL UNIQUE,

startDate Varchar(45),

description Varchar(255),

enabled BOOLEAN NOT NULL DEFAULT true,

disabledDate Timestamp,

PRIMARY KEY (projectId)

);

-- Table user

CREATE TABLE user

(

userId Int(11) NOT NULL AUTO\_INCREMENT,

userName Varchar(45) NOT NULL UNIQUE,

email Varchar(45) NOT NULL,

password Varchar(45) NOT NULL,

securityQ Varchar(60),

securityA Varchar(60),

userType char(2) NOT NULL DEFAULT "U",

PRIMARY KEY (userId)

);

-- Table userproject

CREATE TABLE userproject (

userId int(11) NOT NULL,

projectId int(11) NOT NULL,

PRIMARY KEY (userId, projectId)

);

-- Table diagram

CREATE TABLE diagram

(

diagramId Int(11) NOT NULL AUTO\_INCREMENT,

projectId Int(11) NOT NULL,

userId Int(11) NOT NULL,

contextId int(11) DEFAULT 1,

diagramType Varchar(45),

diagramName Varchar(45),

createTime Timestamp NULL,

filePath Varchar(45),

fileType Varchar(20),

merged Tinyint,

notationFileName Varchar(45),

notationFilePath Varchar(45),

diFlieName Varchar(45),

diFilepath Varchar(45),

PRIMARY KEY (diagramId)

);

-- Table report

CREATE TABLE report

(

reportId Int(11) NOT NULL AUTO\_INCREMENT,

diagramA Int(11) NOT NULL,

diagramB Int(11) NOT NULL,

mergedDiagram Int(11),

type Varchar(20) NULL,

time Timestamp NOT NULL,

reportFilePath Varchar(200) NOT NULL,

reportFileName Varchar(45) NULL,

PRIMARY KEY (reportId)

);

-- Table comment

CREATE TABLE comment

(

commentId int(11) PRIMARY KEY AUTO\_INCREMENT,

compareId int(11) NOT NULL,

userId int(11) NOT NULL,

commentText varchar(255) NOT NULL,

commentTime timestamp NULL,

promotedDiagramId int(11),

userName varchar(45)

);

-- 2013/10/22 Create Table Policy --

CREATE TABLE policy

(

policyId INT(11) PRIMARY KEY AUTO\_INCREMENT,

policyName VARCHAR(45) NOT NULL UNIQUE,

policyDescription VARCHAR(255),

policyLevel INT(11) NOT NULL

);

-- 2013/10/22 Create Table DiagramContext --

CREATE TABLE diagramContext

(

diagramContextId Int(11) PRIMARY KEY AUTO\_INCREMENT,

description VARCHAR(255),

name VARCHAR(45) NOT NULL,

policyId INT(11) NOT NULL,

projectId INT(11) NOT NULL,

enabled BOOLEAN NOT NULL DEFAULT true,

disabledDate timestamp

);

-- 2013/10/22 Create Table DiagramPolicyScore --

CREATE TABLE DiagramPolicyScore

(

diagramId INT(11) NOT NULL,

justification VARCHAR(100),

policyId INT(11) NOT NULL,

score INT(11) DEFAULT 0,

PRIMARY KEY (diagramId, policyId)

);

-- 2013/11/12 Create Table compare --

CREATE TABLE compare

(

compareId int(11) NOT NULL AUTO\_INCREMENT,

diagramAId int(11) NOT NULL,

diagramBId int(11) NOT NULL,

reportId int(11) NOT NULL,

promoteCountA int(11) DEFAULT 0,

promoteCountB int(11) DEFAULT 0,

PRIMARY KEY (compareId)

);

CREATE TABLE metric

(

metricId int(11) NOT NULL AUTO\_INCREMENT,

policyId int(11),

metricTypeId int(2) NOT NULL,

metricsWeight int(11) NOT NULL,

PRIMARY KEY (metricId)

);

CREATE TABLE metricType

(

metricTypeId int(11) NOT NULL,

description varchar(255),

metricTypeName varchar(30) NOT NULL,

PRIMARY KEY (metricTypeId)

);

CREATE TABLE DiagramMetricsScore

(

diagramId int(11) NOT NULL,

metricId int(11) NOT NULL,

score int(11) NOT NULL,

PRIMARY KEY (diagramId,metricId)

);

CREATE TABLE attributes

(

metricId int(11) NOT NULL,

idealNoOfAttributes int(11),

maxNoOfAttributes int(11),

minNoOfAttributes int(11)

);

CREATE TABLE classes

(

metricId int(11) NOT NULL,

idealNoOfClasses int(11),

maxNoOfClasses int(11),

minNoOfClasses int(11)

);

-- Create relationships section -------------------------------------------------

ALTER TABLE diagram ADD CONSTRAINT diagramHaveOwnerId FOREIGN KEY (userId) REFERENCES user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE diagram ADD CONSTRAINT diagramHaveProjectType FOREIGN KEY (projectId) REFERENCES project (projectId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE report ADD CONSTRAINT Relationship4 FOREIGN KEY (diagramA) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE report ADD CONSTRAINT Relationship5 FOREIGN KEY (diagramB) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE report ADD CONSTRAINT Relationship6 FOREIGN KEY (mergedDiagram) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE comment ADD CONSTRAINT Relationship7 FOREIGN KEY (userId) REFERENCES user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE userproject ADD CONSTRAINT userprojectHaveUserId FOREIGN KEY (userId) REFERENCES user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE userproject ADD CONSTRAINT userprojectHaveProjectId FOREIGN KEY (projectId) REFERENCES project (projectId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE comment ADD CONSTRAINT commentHaveCompareId FOREIGN KEY (compareId) REFERENCES compare (compareId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE comment ADD CONSTRAINT commentHaveUserId FOREIGN KEY (userId) REFERENCES user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE compare ADD CONSTRAINT compareHaveReportId FOREIGN KEY (reportId) REFERENCES report (reportId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE compare ADD CONSTRAINT compareHaveDiagramAId FOREIGN KEY (diagramAId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE compare ADD CONSTRAINT compareHaveDiagramBId FOREIGN KEY (diagramBId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE DiagramPolicyScore ADD CONSTRAINT DiagramPolicyScoreHavePolicyId FOREIGN KEY (policyId) REFERENCES policy (policyId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE DiagramPolicyScore ADD CONSTRAINT DiagramPolicyScoreHaveDiagramId FOREIGN KEY (diagramId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE diagramContext ADD CONSTRAINT DiagramContextHavePolicyId FOREIGN KEY (policyId) REFERENCES policy (policyId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE diagramContext ADD CONSTRAINT diagramContextHaveProjectId FOREIGN KEY (projectId) REFERENCES project (projectId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE diagram ADD CONSTRAINT diagramHaveDiagramContextId FOREIGN KEY (contextId) REFERENCES diagramContext (diagramcontextId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE metric ADD CONSTRAINT metricHavePolicyId FOREIGN KEY (policyId) REFERENCES policy (policyId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE metric ADD CONSTRAINT metricHaveMetricTypeId FOREIGN KEY (metricTypeId) REFERENCES metricType (metricTypeId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE DiagramMetricsScore ADD CONSTRAINT DiagramMetricsScoreHaveDiagramId FOREIGN KEY (diagramId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE DiagramMetricsScore ADD CONSTRAINT DiagramMetricsScoreHaveMetricId FOREIGN KEY (metricId) REFERENCES metric (metricId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE attributes ADD CONSTRAINT attributesHaveMetricId FOREIGN KEY (metricId) REFERENCES metric (metricId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

ALTER TABLE classes ADD CONSTRAINT classesHaveMetricId FOREIGN KEY (metricId) REFERENCES metric (metricId) ON DELETE NO ACTION ON UPDATE NO ACTION

;

-- insert sample data--------------------------------------

insert into policy (policyName,policyDescription,policyLevel) values ("policy 1","policy 1 description",2);

INSERT INTO policy (policyName, policyDescription, policyLevel) VALUES ('Policy2', 'Policy Description', 2);

INSERT INTO metricType (metricTypeId, description, metricTypeName) VALUES (1, 'ASSOCIATIONS', 'ASSOCIATIONS');

INSERT INTO metricType (metricTypeId, description, metricTypeName) VALUES (2, 'MULTIPLICITIES', 'MULTIPLICITIES');

INSERT INTO metricType (metricTypeId, description, metricTypeName) VALUES (3, 'ATTRIBUTES', 'ATTRIBUTES');

INSERT INTO metricType (metricTypeId, description, metricTypeName) VALUES (4, 'CLASSES', 'CLASSES');

INSERT INTO metric (metricId, policyId, metricTypeID, metricsWeight) VALUES (1, 2, 4, 10);

INSERT INTO metric (metricId, policyId, metricTypeID, metricsWeight) VALUES (2, 2, 3, 10);

INSERT INTO classes (metricId, idealNoOfClasses, maxNoOfClasses, minNOOfClasses) VALUES (1, 4, 5, 2);

INSERT INTO attributes (metricId, idealNoOfAttributes, maxNoOfAttributes, minNOOfAttributes) VALUES (2, 4, 5, 1);