CSYE 7945 – Fall 2013
ClubUML Project Guidelines

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.

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. Known issues were identified with the existing version of the ClubUML application running on Tomcat 7.0.42 and newer, and a fix will be implemented shortly to address this issue.

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 here. 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). Papyrus diagrams can be exported from the Papyrus UML tool (Available for download here). 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.

Version control of project artifacts will be managed by the Git version control system. The primary project repository storing the artifacts can be found here). A A tutorial is also available that describes how to use the Eclipse Git plug-in, or the Git console..

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.

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 Git repository branch.

Testing

TBD

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.

Appendix:

Additional Reference

The ClubUML Spring 2013 project summary report containing much additional information can be found here.

CSYE 7945 – Fall 2013
ClubUML Project Guidelines

Additional recommendations regarding project guidelines may be found here.

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.

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.

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.

CSYE 7945 – Fall 2013
ClubUML Project Guidelines

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 stpes.

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.

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.

Database creation script:

delimiter $$ CREATE DATABASE `clubuml` /*!40100 DEFAULT CHARACTER SET utf8 */$$ delimiter $$ CREATE TABLE `project` (`project_Id` int(11) NOT NULL AUTO_INCREMENT, `projectName` varchar(45) DEFAULT NULL, `description` varchar(255) DEFAULT NULL, `achived` tinyint(1) DEFAULT '0', PRIMARY KEY (`project_Id`)) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8$$ delimiter $$ CREATE TABLE `user` (`User_Id` int(11) NOT NULL AUTO_INCREMENT, `userName` varchar(45) DEFAULT NULL, `password` varchar(45) DEFAULT NULL, `email` varchar(45) DEFAULT NULL, `project_Id` int(11) DEFAULT NULL, `securityQuestion` varchar(45) DEFAULT NULL, `securityAnswer` varchar(45) DEFAULT NULL, PRIMARY KEY (`User_Id`), KEY `fk_Project_User_idx` (`project_Id`), CONSTRAINT `fk_Project_User` FOREIGN KEY (`project_Id`) REFERENCES `project` (`project_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8$$ delimiter $$ CREATE TABLE `diagram` (`diagram_Id` int(11)

NOT NULL AUTO_INCREMENT, `diagramName` varchar(45) DEFAULT NULL, `createdTime` datetime DEFAULT NULL, `inEdition` tinyint(1) DEFAULT NULL, `owner_Id` int(11) NOT NULL, `filePath` varchar(45) DEFAULT NULL, PRIMARY KEY (`diagram_Id`), KEY `fk_Diagram_User_idx` (`owner_Id`), CONSTRAINT `fk_Diagram_User` FOREIGN KEY (`owner_Id`) REFERENCES `user` (`User_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB DEFAULT CHARSET=utf8$$ delimiter $$ CREATE TABLE `comment` (`comment_Id` int(11) NOT NULL AUTO_INCREMENT, `user_Id` int(11) NOT NULL, `content` varchar(255) DEFAULT NULL, `writtenTime` datetime DEFAULT NULL, `diagram_Id` int(11) NOT NULL, PRIMARY KEY (`comment_Id`), KEY `fk_Comment_Diagram_idx` (`diagram_Id`), KEY `fk_Comment_User_idx` (`user_Id`), CONSTRAINT `fk_Comment_User` FOREIGN KEY (`user_Id`) REFERENCES `user` (`User_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION, CONSTRAINT `fk_Comment_Diagram` FOREIGN KEY (`diagram_Id`) REFERENCES `diagram` (`diagram_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB DEFAULT CHARSET=utf8$$ delimiter $$ CREATE TABLE `report` (`report_Id` int(11) NOT NULL, `diagram_A` int(11) NOT NULL, `diagram_B` int(11) NOT NULL, `comparedTime` timestamp NULL DEFAULT NULL, `reportFilePath` varchar(45) DEFAULT NULL, PRIMARY KEY (`report_Id`), KEY `fk_Report_Diagram_idx` (`diagram_A`), KEY `fk_Report__idx` (`diagram_B`), CONSTRAINT `fk_Report_Diagram_A` FOREIGN KEY (`diagram_A`) REFERENCES `diagram` (`diagram_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION, CONSTRAINT `fk_Report_Diagram_B` FOREIGN KEY (`diagram_B`) REFERENCES `diagram` (`diagram_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB DEFAULT CHARSET=utf8$$ delimiter $$ CREATE TABLE `editinghistory` (`diagram_Id` int(11) NOT NULL, `EditingTime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, `editingHistory_Id` int(11) NOT NULL, `user_Id` int(11) NOT NULL, PRIMARY KEY (`editingHistory_Id`), KEY `fk_EditingHistory_Diagram_idx` (`diagram_Id`), KEY `fk_EditingHistory_User_idx` (`user_Id`), CONSTRAINT `fk_EditingHistory_User` FOREIGN KEY (`user_Id`) REFERENCES `user` (`User_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION, CONSTRAINT `fk_EditingHistory_Diagram` FOREIGN KEY (`diagram_Id`) REFERENCES `diagram` (`diagram_Id`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB DEFAULT CHARSET=utf8$$