Team Assignment 4

Computer Science 2212b

Team 6

Allan Yan
Tyler Swartzenberg
Valmir Verbani
Vanessa Seguin
Vinayak Modi

Due: April 2, 2014

1. REVISED USER STORIES

2. DESIGN REPORT

At ATripleV (also known as Team 6), we have designed a custom gradebook application for a teachers' or professors' grade keeping needs. The gradebook is a simple tab-windowed program that uses instructors' data to manage and perform various grade keeping tasks. In order to make our application simple and intuitive to use we attempted to apply as many best coding practices as possible while providing a high level of utility.

When creating our gradebook software, we were able to apply a variety of design principles. Our program maintains a high level of efficiency, as our design implements efficient calculation and data storage algorithms. We also increased efficiency by minimizing redundant code. This high level efficiency includes large amounts of testing to establish our program's reliability. Strong coding procedures were used to ensure successful data persistence in order for our program to be used by teachers and professors in the future. We also made effective use of re-engineering which increased our maintainability. By shifting our bi-weekly duties to a new aspect of the project, it was necessary to be clean and concise with our code, allowing other group members to easily understand it. Furthermore, by mapping out a bi-weekly UML diagram, we were able to get a better idea of how our classes were interacting as our project grew in size and complexity.

In addition, while creating our gradebook software, we were able to apply a variety of object-oriented design principles. Following good design principles, we implemented encapsulation to protect the program from unwanted outside modification to variables, in line with good coding principles. We also favoured composition over inheritance, as our course and student classes managed array lists composed of deliverables, grades, and students. Our gradebook follows the SOLID principle, as every object in our system have single responsibilities, and all the object's

services are focused on carrying out that single responsibility. This is incredibly important when working separately in a team that totals five members. By following the SOLID principle, the chances of a class needing to be changes is minimized which results in a high level of cohesion.

When assigning duties for our bi-weekly reports, we would assign one person to deal with a specific set of classes, which no other team members would need to change, which effectively managed the Open/Closed Principle. In addition, we attempted to use higher-level classes, such as the App class, in order to follow the Dependency Inversion Principle. Our code exemplifies the DRY principle, as each piece of knowledge must have a single, unambiguous, authoritative representation within our system, which can be seen in our code and comments. We aimed to reduce redundancies and make our code as precise as possible.

It is clear that our gradebook took the opportunity to apply a variety of general and object-oriented design principles. However, there were a few features of our program with which we particularly excelled, and others with which we did not.

There were several strong areas in terms of code design for our project. The development process we used followed the Agile design method. The unit tests ensured that all user stories were completed and functioning as desired. Lastly, our program code aligns very closely to the UML diagram, which will make the program more understandable to outside programmers, thus increasing the maintainability of the code. Our software was designed to be user-friendly, and strived to integrate usability well with utility.

In the planning stage of the project, we used the divide and conquer principle to split responsibilities and divide the project up into individual subsystems. This helped us create a timeline and project plan with specific due dates, which in turn helped keep us on schedule. Later on, we

used it to divide the program into a collection of subsystems, then into different classes, and then into different methods. We used a mix of top-down and bottom-up design in order to make sure that the program had a clear structure, and then worked down to the individual methods and lower-level constructs. Then, moving from the bottom up, we took a look at the constructs and determined how to make them more reusable as they were put together in the higher level constructs. For example, we decided to use ArrayLists, as a lot of functionality that we needed were already built into the Java API.

Following good design principles, we implemented encapsulation to protect the program from unwanted outside modification to variables, in line with good coding principles. Keeping with the SOLID principles, we implemented the single responsibility principle, for example in our code the Course, Student, Grade, Deliverable classes all are only responsible for changes that affect those objects. This results in our code having a higher level of cohesion, increasing reusability, robustness and understandability. We also followed the Open/Closed Principle in order to keep our code bug-free, but allow extensions to increase the functionality going forward if necessary. The Dependency Inversion Principle was implemented here in the CSV suite, in integrating the export CSV functionality. The user can input any type of file, txt, csv, etc. and the file will be written out correctly. Implementing this principle decreases rigidity and fragility within the code. We also implemented the Principle of Least Knowledge within the code so each class only has access to the data it needs to perform actions. This decreases coupling and allows us to make the code more readable. Lastly, we implemented the DRY Principle in our code, when a variable is called multiple times within a code segment, we create a temporary object to point at it instead of using multiple calls repeatedly.

Another strong point in our program was the separation of GUI and business logic. This makes the code more modular, and easier to make changes on either side. For example, if we wanted to make changes to the GUI, the function calls that each button triggers would not change, and vice versa. If we wanted to change functionality, the GUI would still remain the same.

Our code implements several types of cohesion. There is functional cohesion such that each method only completes one task. This lessens side effects and makes the system easier to understand, update, and for code reusability. There is layer cohesion where the Course class is at the top of the hierarchy, while Grade and Deliverable at the bottom. Course can access student, grade, and deliverable data, while each student can only access their own grades, and grades and deliverables can only see their own data. Communicational cohesion is evident as well; the only methods that modify data are kept within their object classes. This makes the code easier to commit changes as all the data for a certain object are kept in the same place.

As a group, we have agreed that despite having a functional program for the gradebook, it still had areas of the design that were weak. One of these weak designs in our code was that initially our program had a remove student method that was supposed to, as the title says, remove a student from the array list. At the beginning we had created main methods which had created a student and was able to change each student's first name, last name, and id number. These methods all worked fine alongside with the remove method, which deleted this student from the course. The way our GUI works now: it removes the student from the course without calling this remove method. This deletion occurs this way because the array list of students is displayed in the GUI once you click on the down arrow in the student display. Once you select that certain student, its index in the ArrayList is also selected and then deleted. Time was initially spent writing this method and writing tests

methods to see whether this method works or not which was pointless considering we did not even need it. With more GUI knowledge or experience this would have prevented us from wasting time. The second downside to our code is our separate App class. The reason why this addition is not necessarily the greatest is because it is just another unnecessary class addition. Essentially the App class is the main method that was initially in our GradebookUI class. We would easily have gotten away with keeping this main method in our GradebookUI and still have the same results.

Another downfall to our project is that we created our grade constructor that takes in three variables: the first is the double which is the grade value, the second is a string which is the title of the grade, and the third is another double which is the weight of the grade. This is a bad design for the fact that each grade should not have its own weight. The deliverable constructor should be the only constructor that has each grades actual weight average.

Another real fallback of our code is at beginning we deleted our combo box because our JTable code was not properly implemented. Due to this change, our JTable had been integrated on its own in a way where we had to implement everything else around it rather than the other way around. Even the code that had been originally developed prior to the JTable had to be re-implemented. The JTable code was very robust, it seemed that we could not change anything of JTable but rather change all the other code to properly function with it. This was a very poor design because due to this change, it set our team back a couple of weeks. It caused the majority of the tests to fail and to get the frame of the project fully re-organized to accommodate with this change. All these changes led our group to a substantial amount of lost time. This lost time would have been contributed to having the project done ahead of time; leaving our group with a lot more time to fully test if the code had any bugs and fix any small mistakes that were made in this project.

The GUI rapidly became one of the most important parts of the project. We didn't realize this until well into the project. The implications of certain design choices became more and more clear as the term went on. The initial design of the GUI was created using NetBeans, mainly for layout and component binding. We chose to start with NetBeans as it quickly allowed us to see how the GUI would look and to make changes quickly to develop a design that worked best for everyone.

In designing the initial layout, our group wanted to focus on keeping things simple for the user. Minimizing the number of buttons and options would allow the program to be easy to learn and use. We decided on a top menu bar with a generic 'File' option which would hold basic commands such as 'Save' and 'Exit' to start. The menu bar also held commands for Adding, Editing and Deleting. Within these commands, drop down menus would allow you to choose between Course, Student and Deliverable manipulation. In the dropdown options, Course was always listed first as, especially in the case of Adding an Object, you could not do certain actions without first creating a Course. Dropdown options were also included for importing and exporting lists of Objects. Once each command was chosen, the program initially opened a new window, however it was quickly decided that have a bunch of individual windows opening did not create a good user experience and we opted to create new tab panes for each command.

Aside from the menu bar and the main panel area where the tabs would open, the GUI also included a combo box which populated all courses. Initially this list was only used to open a course's JTable, however this was changed later in the project and the combo box was turned into an "active course" option which, when working with Student and Deliverable objects, the active course selected would be the course affected by these options. This was mostly done based on an issue with JTable implementation. However, by changing this portion and removing the option to choose

a course and its associated students or deliverables in each tabbed window when adding/editing/deleting objects, it ended with creating other issues. These issues can be seen in Fig 1 and Fig 2.

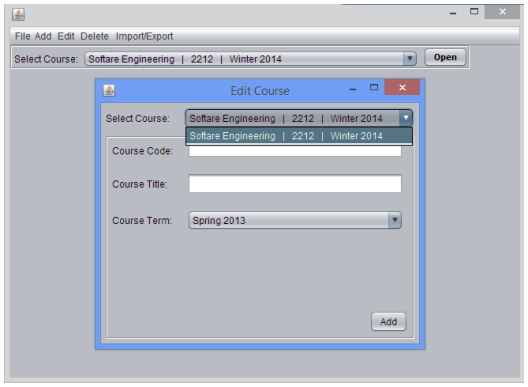


Figure 1 Initial Implementation with combo boxes to select active course in Add/Edit/Delete interfaces

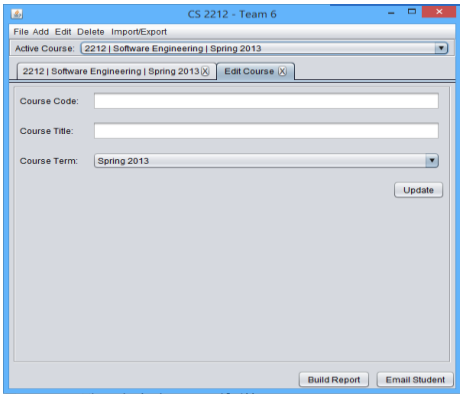


Figure 2 Final implementation without combo box in Add/Edit/Delete interface

Once the basic layout was chosen, NetBeans was used as facility component binding so that when windows were resized by the user, all the elements would stay in place. All action listeners and commands were implemented separately as NetBeans seemed to make the code very messy and hard to read by the members of the group who were not directly involved in the coding of the GUI. It was important to all our group members that each person in the group be able to understand each step of the implementation of the program. Commenting code frequently and implementing well formatted code meant that each member would understand the code. This lead to many minor changes to the GUI as the project progressed which lead to a high deletion count in the Git repository as we attempted to “clean” up the code.

As mentioned before, about half way through the project, as we started to implement more parts to the project, especially the input validity testing, it became more clear that some of our initial decisions were originally correct versus the changes that were made later on. When testing validity and avoiding errors, we realized that certain functionality that we wanted to implement in the

program was just not possible. When looking into the issues, it became clear that the initial layout decisions were correct and were best able to handle all issues that would come up. Unfortunately, this was realized too far into the process to change things back. By realizing these mistakes too late, it forced our team to continue to integrate our code with these changes.

Ultimately, the decision to keep the design simple helped us as we were able to change things easily. We added small alert JLabels which allowed small messages to be displayed when the user attempted to submit information or do something that was contrary to the program functionality. This lead to a less jarring experience for the user than using pop-up messages while still providing them with guidelines for using the program.

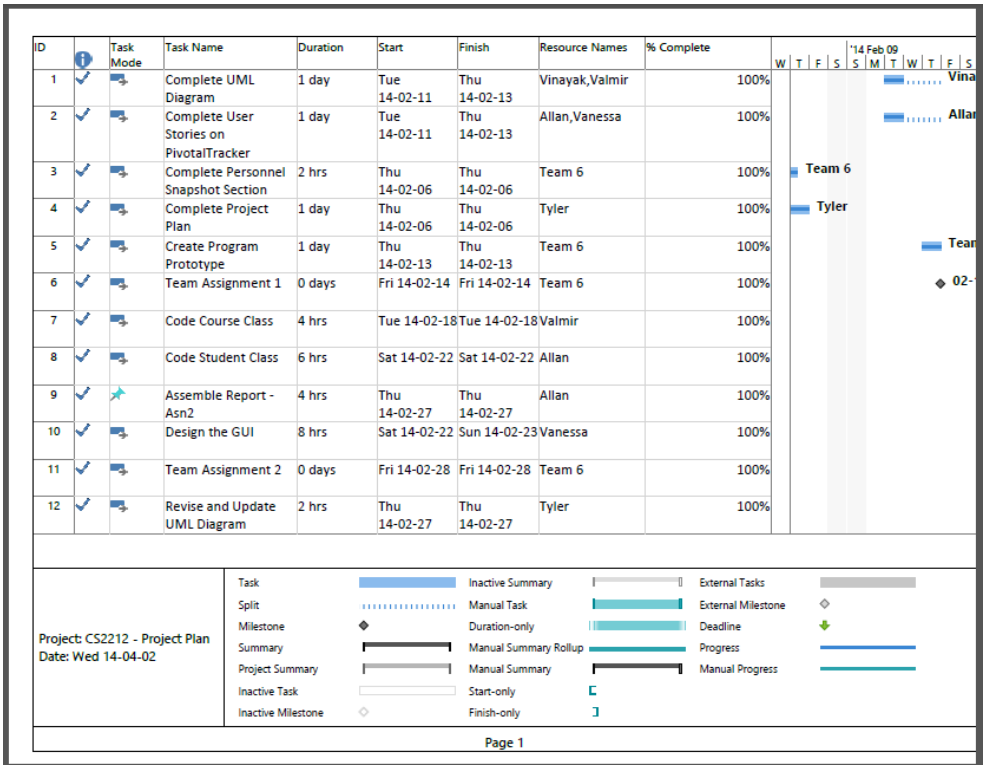
Our project design both reused code, and created reusable code of its own. While reusability was not one of the priorities of this project, our code exemplifies none the less, as it is efficient and simple to use previously written code. The reason for reusability not being a priority in this project was due to its context. Since this a project being completed for a course, it is not possible for the instructor to request changes or changes that might benefit from reusability, such as making the application a web-based application. However, there was a fair bit of reusability involved nonetheless.

We often reused code in the development of this application, whether it was originally written for the application, or otherwise. Major examples of reused code not written for the application originally would be code for classes specific to specialty labs. For example, the classes for adding the JTable functionality reuse code from the JTable lab provided by the instructor. More specifically, the CourseTableModel class uses code from the model class in the lab. The same is true for other specialty functions such as sending out the email and generating the report. In addition to

this, the GUI classes also heavily reuse code. While not exactly reusing, GUI classes use a lot of code generated by NetBeans, which makes the process of created GUI items much easier and more efficient. Once the initial code was generated through NetBeans, it was heavily reused within the project. The best example of this would be creating a new tab for an action such as adding a student. Once the first tab was made, the other tabs were made by reusing code from it and simply modifying it. Almost all smaller components were also created by reusing code such as buttons and associated listeners and actions. This reuse of code allowed us to create components much more quickly and efficiently. Reusing code also made it easier for us to work together because even if one member did not fully understand the code for a component, the member could duplicate it and modify it by modifying small portions of the code.

There is a lot of reusable code in this project, but it may not be applicable to all situations. If the client wants the application to be web-based instead of being desktop based, all functional code can be used. This includes basic classes such as the Grade and Student classes, along with other classes related to specialties, such as the CourseTableModel class. These classes would require extremely minimal changes, if any in such a situation. The GUI classes are less flexible because it may not be possible to implement them over the web, making them largely useless. However, the GUI classes contain many actions and methods used to tie the application together. All of these methods and actions could be reused as they are. Overall, the code is highly reusable in such a situation where a similar application is being created. However, if the client wants a new unrelated application, most of the code in this project would not as easily be recycled. The reusable code in this situation would be the GUI elements, which can be easily adapted and modified to fit another application.

3. PROJECT PLAN



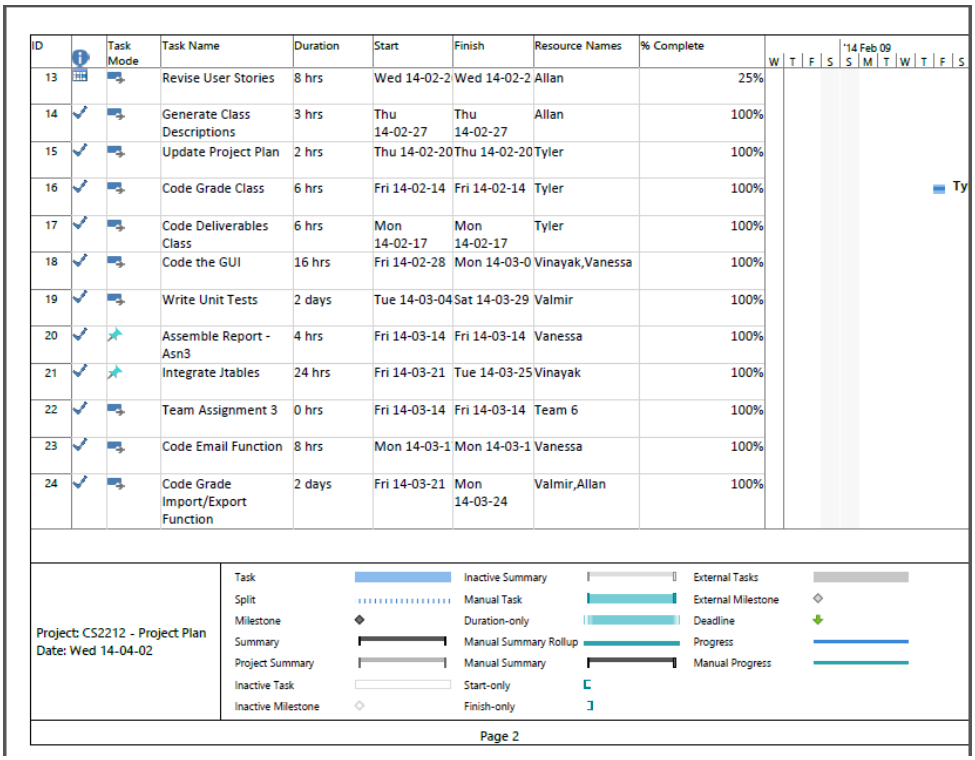
ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names	% Complete
1		Complete UML Diagram	1 day	Tue 14-02-11	Thu 14-02-13	Vinayak,Valmir	100%
2		Complete User Stories on PivotalTracker	1 day	Tue 14-02-11	Thu 14-02-13	Allan,Vanessa	100%
3		Complete Personnel Snapshot Section	2 hrs	Thu 14-02-06	Thu 14-02-06	Team 6	100%
4		Complete Project Plan	1 day	Thu 14-02-06	Thu 14-02-06	Tyler	100%
5		Create Program Prototype	1 day	Thu 14-02-13	Thu 14-02-13	Team 6	100%
6		Team Assignment 1	0 days	Fri 14-02-14	Fri 14-02-14	Team 6	100%
7		Code Course Class	4 hrs	Tue 14-02-18	Tue 14-02-18	Valmir	100%
8		Code Student Class	6 hrs	Sat 14-02-22	Sat 14-02-22	Allan	100%
9		Assemble Report - Asn2	4 hrs	Thu 14-02-27	Thu 14-02-27	Allan	100%
10		Design the GUI	8 hrs	Sat 14-02-22	Sun 14-02-23	Vanessa	100%
11		Team Assignment 2	0 days	Fri 14-02-28	Fri 14-02-28	Team 6	100%
12		Revise and Update UML Diagram	2 hrs	Thu 14-02-27	Thu 14-02-27	Tyler	100%

'14 Feb 09 W T F S S M T W T F S

Project: CS2212 - Project Plan
Date: Wed 14-04-02

Task, Split, Milestone, Summary, Project Summary, Inactive Task, Inactive Milestone, Inactive Summary, Manual Task, Duration-only, Manual Summary Rollup, Manual Summary, Start-only, Finish-only, External Tasks, External Milestone, Deadline, Progress, Manual Progress

Page 1



ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names	% Complete
13		Revise User Stories	8 hrs	Wed 14-02-2	Wed 14-02-2	Allan	25%
14		Generate Class Descriptions	3 hrs	Thu 14-02-27	Thu 14-02-27	Allan	100%
15		Update Project Plan	2 hrs	Thu 14-02-20	Thu 14-02-20	Tyler	100%
16		Code Grade Class	6 hrs	Fri 14-02-14	Fri 14-02-14	Tyler	100%
17		Code Deliverables Class	6 hrs	Mon 14-02-17	Mon 14-02-17	Tyler	100%
18		Code the GUI	16 hrs	Fri 14-02-28	Mon 14-03-0	Vinayak,Vanessa	100%
19		Write Unit Tests	2 days	Tue 14-03-04	Sat 14-03-29	Valmir	100%
20		Assemble Report - Asn3	4 hrs	Fri 14-03-14	Fri 14-03-14	Vanessa	100%
21		Integrate Jtables	24 hrs	Fri 14-03-21	Tue 14-03-25	Vinayak	100%
22		Team Assignment 3	0 hrs	Fri 14-03-14	Fri 14-03-14	Team 6	100%
23		Code Email Function	8 hrs	Mon 14-03-1	Mon 14-03-1	Vanessa	100%
24		Code Grade Import/Export Function	2 days	Fri 14-03-21	Mon 14-03-24	Valmir,Allan	100%

'14 Feb 09 W T F S S M T W T F S

Project: CS2212 - Project Plan
Date: Wed 14-04-02

Task, Split, Milestone, Summary, Project Summary, Inactive Task, Inactive Milestone, Inactive Summary, Manual Task, Duration-only, Manual Summary Rollup, Manual Summary, Start-only, Finish-only, External Tasks, External Milestone, Deadline, Progress, Manual Progress

Page 2

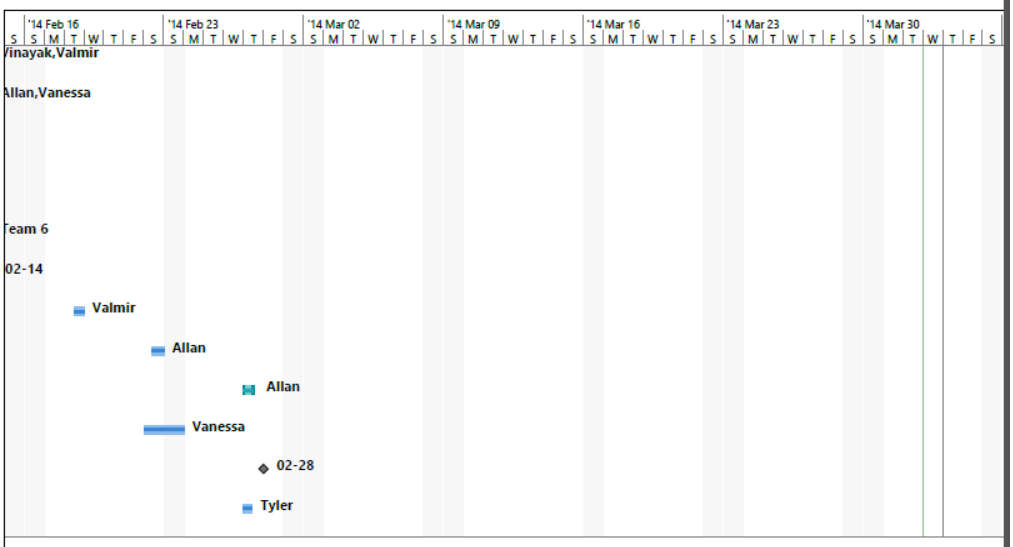
ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names	% Complete
25		Update Data Persistence	2 hrs	Wed 14-03-26	Wed 14-03-26	Allan	100%
26		Code PDF Generation ability	8 hrs	Mon 14-03-24	Mon 14-03-24	Allan,Vanessa	100%
27		Write Design Choices Report	3 days	Thu 14-03-27	Mon 14-03-31	Team 6	100%
28		Final PivotalTracker Submission	5 hrs	Thu 14-03-27	Thu 14-03-27	Allan	100%
29		Ensure Code Coverage in Unit Tests	2 hrs	Mon 14-03-31	Mon 14-03-31	Valmir	100%
30		Final Update of GUI	24 hrs	Mon 14-03-2	Wed 14-03-2	Vanessa	100%
31		Final update of Gantt Chart	2 hrs	Tue 14-04-01	Tue 14-04-01	Vanessa	100%
32		Team Assignment 4	0 days	Wed 14-04-0	Wed 14-04-0	Team 6	100%
33		Final Code Submission	0 days	Wed 14-04-02	Wed 14-04-02	Team 6	100%

'14 Feb 09

W T F S S M T W T F S

Project: CS2212 - Project Plan
Date: Wed 14-04-02

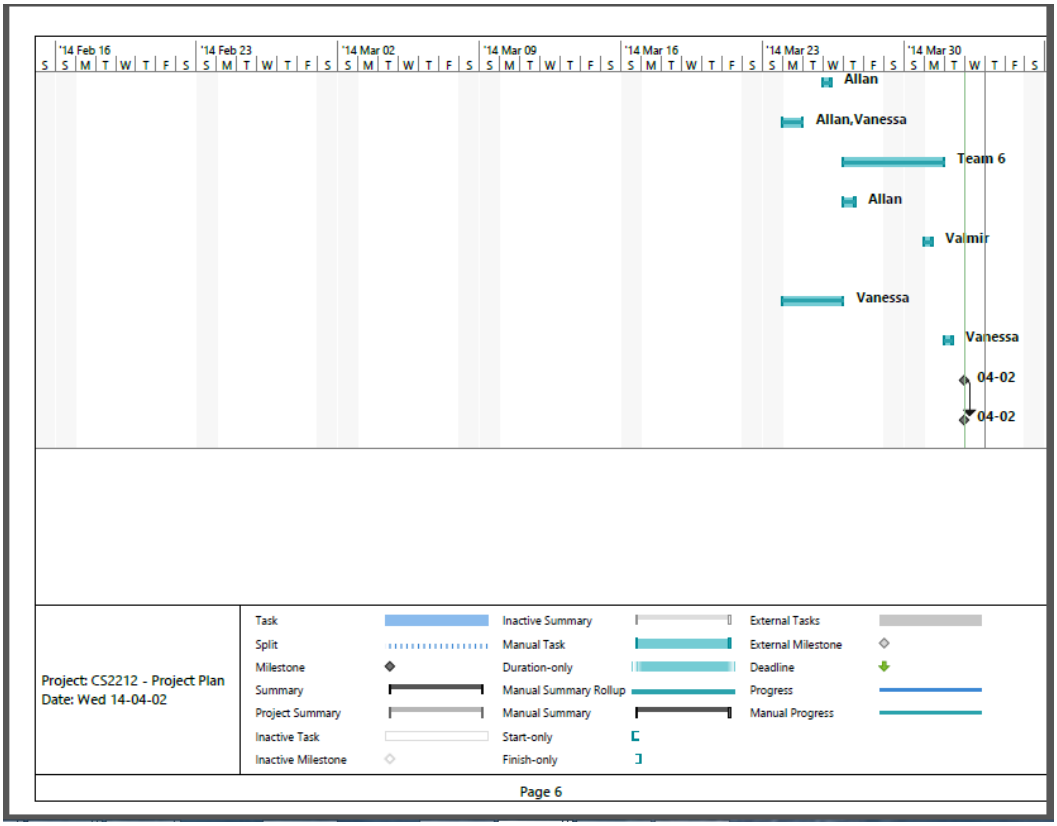
Task	Inactive Summary	External Tasks	
Split	Manual Task	External Milestone	
Milestone	Duration-only	Deadline	
Summary	Manual Summary Rollup	Progress	
Project Summary	Manual Summary	Manual Progress	
Inactive Task	Start-only		
Inactive Milestone	Finish-only		



Project: CS2212 - Project Plan
Date: Wed 14-04-02

Task	Inactive Summary	External Tasks	
Split	Manual Task	External Milestone	
Milestone	Duration-only	Deadline	
Summary	Manual Summary Rollup	Progress	
Project Summary	Manual Summary	Manual Progress	
Inactive Task	Start-only		
Inactive Milestone	Finish-only		


'14 Feb 16
'14 Feb 23
'14 Mar 02
'14 Mar 09
'14 Mar 16
'14 Mar 23
'14 Mar 30
Allan
Allan
Tyler
Tyler
Tyler
Vinayak,Vanessa
Valmir
Vanessa
Vinayak
03-14
Vanessa
Valmir,Allan
Project: CS2212 - Project Plan
Date: Wed 14-04-02
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress
Page 5


'14 Feb 16
'14 Feb 23
'14 Mar 02
'14 Mar 09
'14 Mar 16
'14 Mar 23
'14 Mar 30
Allan
Allan,Vanessa
Team 6
Allan
Valmir
Vanessa
Vanessa
04-02
04-02
Project: CS2212 - Project Plan
Date: Wed 14-04-02
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress
Page 6

4. UNIT TESTS

Non-GUI Classes	**GUI Classes**
Course	AddCourseUI
CourseTableModel	AddDeliverableUI
*CSVParser	AddStudentUI
*CSVReader	App
CSVSuite	DeleteCourseUI
*CSVWriter	DeleteStudentUI
Deliverable	EditCourseUI
EmailHelper	EditDeliverableUI
Grade	EditStudentUI
ReportData	GradeBookUI
*ResultSetHelper	JTableUI
*ResultSetHelperService	SetupUI
Student	

** These classes do not have tests because they were not written by the group. They are an open Java library*