Computer Science 2212b
Team 7
Assignment #4

April 2nd, 2014

Nick Delben
Ben Dumais
Alex Gyori
Holly Voysey

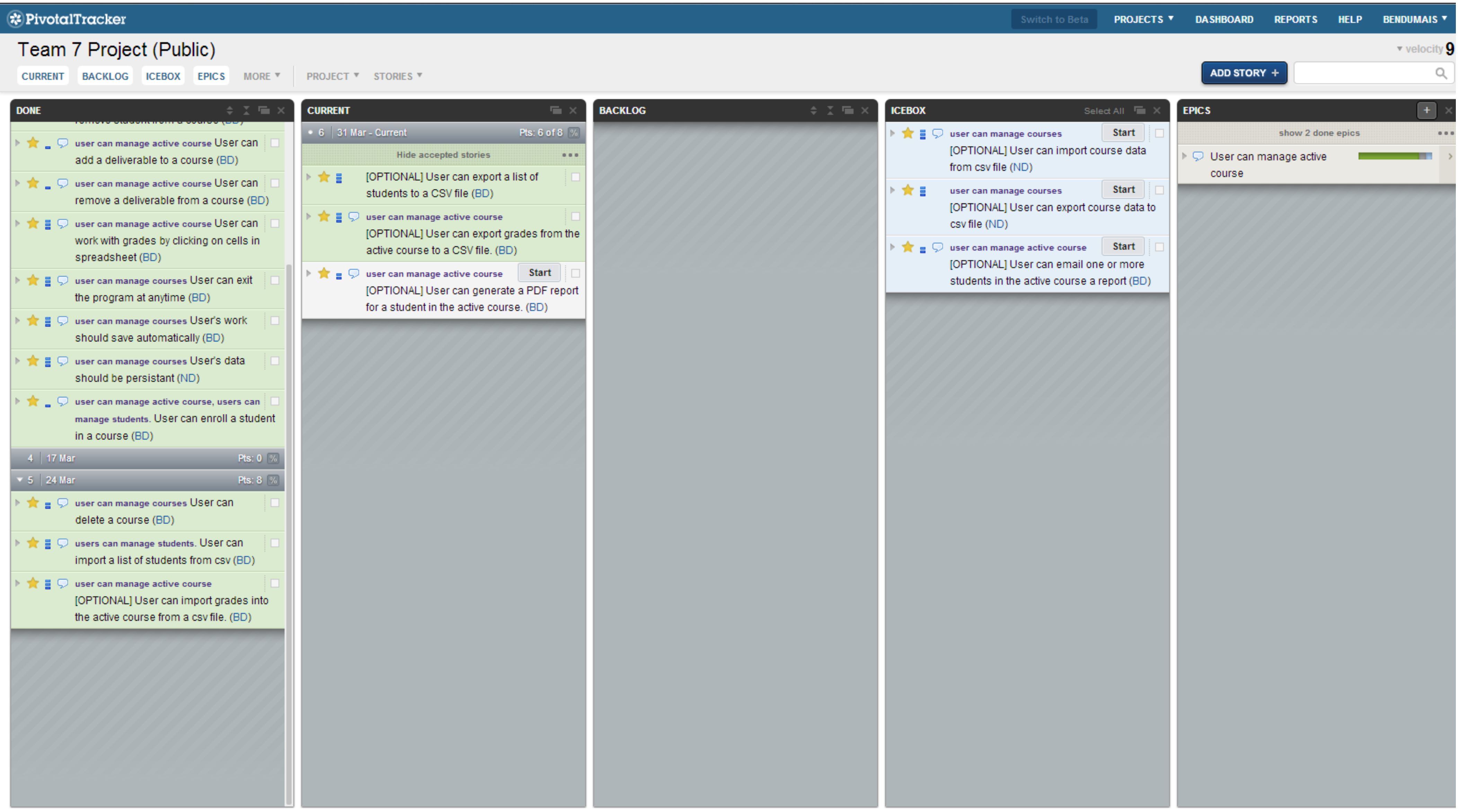
Revised User Stories

Completed since last report:

- User can remove a course (Can remove active course via remove course button, or remove multiple courses through the new Database Manager, done by opening the database manager, navigating to the course tab and either searching for or removing the course code of the course you wish to remove and pressing the appropriate remove button)
- User can Import a list of students from CSV file (done through Import Students button, entering the filepath of the file you wish to import, and clicking the import button. Program will automatically reject faulty input, enroll existing students and create new ones)
-User can Import grades from CSV (done through import grades button, entering filepath, and clicking import button. Program will automatically reject faulty input and assign grades that valid)
- User can export students to CSV (done by selecting students to export from active course and clicking Export Students button. Select export destination (ie which folder to export to) and click okay)
- User can remove students (modified since last report: user is now able to remove or un-enroll multiple students at once through Unenroll button, as well as through database manager. Students who are chosen to be removed from database are also removed from all other courses they are enrolled in)

- User can add a student (Add from database has been implemented since last report, this can be done by selecting “add existing” option after selecting enroll student button)

Link to PivotalTracker: https://www.pivotaltracker.com/s/projects/1004174



Team 7 Project (Public)

DONE

- user can manage active course User can add a deliverable to a course (BD)
- user can manage active course User can remove a deliverable from a course (BD)
- user can manage active course User can work with grades by clicking on cells in spreadsheet (BD)
- user can manage courses User can exit the program at anytime (BD)
- user can manage courses User's work should save automatically (BD)
- user can manage courses User's data should be persistant (ND)
- user can manage active course, users can manage students. User can enroll a student in a course (BD)

4 | 17 Mar — Pts: 0

5 | 24 Mar — Pts: 8

- user can manage courses User can delete a course (BD)
- users can manage students. User can import a list of students from csv (BD)
- user can manage active course [OPTIONAL] User can import grades into the active course from a csv file. (BD)

CURRENT

6 | 31 Mar - Current — Pts: 6 of 8

Hide accepted stories

- [OPTIONAL] User can export a list of students to a CSV file (BD)
- user can manage active course [OPTIONAL] User can export grades from the active course to a CSV file. (BD)
- user can manage active course [OPTIONAL] User can generate a PDF report for a student in the active course. (BD) — Start

BACKLOG

ICEBOX

Select All

- user can manage courses [OPTIONAL] User can import course data from csv file (ND) — Start
- user can manage courses [OPTIONAL] User can export course data to csv file (ND) — Start
- user can manage active course [OPTIONAL] User can email one or more students in the active course a report (BD) — Start

EPICS

show 2 done epics

- User can manage active course

Design Report

From the initial team meeting, through to the testing of the final product, designing the system has played a fundamental role; allowing the team to produce a product that not only meets all of the requirements, but achieves it in a way, that is efficient and one that reduces time taken to implement. This means that any future developments can also be easier implemented and also takes the user of the program into consideration; they are the ones using the system and therefore it should be intuitive and easy to use. Throughout the development of the gradebook application the team have adhered to a number of different design principles. To ensure the application was developed effectively, the divide and conquer principle was used. This allowed the initial requirement of a gradebook application to be broken down into smaller parts; these were the user stories and each story is a specific task relating to the end requirements of the system. This allowed iterations of development to be done and also helped in spreading the tasks out into manageable parts, which also helped in the distribution between different group members. Different design techniques were followed in both the designing of the code and Graphical User Interface (GUI).

Throughout the implementation of the code, we tried to reduce coupling as much as possible, thus also reducing the interdependencies between certain methods and classes. Throughout the development of the application, coupling was reduced as much as possible.

Content Coupling was minimized throughout the system, for example in the implementation of the Student, Grade and Deliverable classes, this was done through encapsulation; making a boundary by hiding certain parts of the classes and only certain methods see particular variables

[1]. This was implemented by making the instance variables of the class private and using getter and setter methods to allow different levels of access.

To ensure there was no Common Coupling throughout the application, no global variables were used, but instead local variables were used and when required were passed to different modules as a parameters of methods. Meaning that only modules where the variable is essential, can modify the variable, allowing debugging and maintenance to become easier as the variable can easily be tracked [2]. Another improvement that reducing common coupling brings to the application, is that it increases the security; as global variables are more likely to be accessed by unauthorised people, these variables could potentially hold sensitive data; which may have bad consequences in the wrong hands.

Reducing coupling has improved the application by decreasing the interdependency between parts of the system, such as methods and classes, thus reducing the likelihood of errors as changes do not having a rippling effect like they would if there was interdependency in the system. It also makes the code easier to understand and reuse, as it is clearer which parts of the system is responsible for a particular functionality.

Although the group tried to follow as many design principles as possible there where some we did not follow, for example the group did not adhere to parts of the SOLID principle as well as we could of done. The Open/Closed principle, suggests that classes should only be extended and not modified, meaning that it will be done by adding new modules and not by modifying code that already works. Not following this principle meant that we were sometimes changing well tested code that was fully functional, with code that had not been tested. This meant there was potential for the system to get additional bugs, which would be hard to fix as it’s not obvious what caused the problem. To follow the principle more closely, when new features adding to the functionality of the system wanted to be added it would have been written in separate methods and added to the current code, allowing easy debugging if something went wrong. It also allows us to easily go back to previous versions where the system was working by just deleting the function and the function call.

Another principle that wasn't followed as well as possible was the Dependency Inversion Principle, to adhere to this principle, high level modules must not depend upon lower level modules, as there are certain methods in the system that rely upon other methods to function. To improve our system to adhere to this principle, instead of having modules depend on others it would be better to use Dependency Inversion, meaning that the higher class would be independent of the lower class. This would make the higher module more reusable as it could be reused without also having to reuse the lower class as it is no longer dependent upon another module.

One of the biggest areas of the code's design we felt was strong, was the use of composition over inheritance, in the system composition was used instead of inheritance. For example, there is a 'HAS A' relationship between the grade and deliverable classes, meaning there is composition. This means that a grade object does not exist outside the deliverable object, and that there are grades encapsulated in the deliverable object, therefore grades can only be given to students (for that particular deliverable) when the deliverable has been made. To implement this in the system, when a new deliverable was created, the constructor also created a new LinkedList of grades, which are then connected to only that particular deliverable. This is also the case in the deletion of a deliverable, that all the grades associated with the certain deliverable will be deleted with the deletion of a deliverable. The deliverable class can still use and access the methods of the Grade class, for example the removeGrade method in the deliverable class uses the getStudentID method from the Grade Class. The main benefit of using composition over inheritance is that the system has become more flexible as extensions can be easily added [3], not only be there is a reduction of code duplication.

Although there are a few aspects of the system that could have been improved on, one of the biggest weakness of the application was the way the course, deliverable and student data was held. This data is held in linked lists and this was originally implemented in the classes as a temporary solution; a quick solution which was working and allowed the group to work on implementing other parts of the application. Although the linked list way of holding the data works and is quick with small quantities of data, with larger amounts of data this method could potentially become slow and inefficient.

This could be improved by using a different data structure instead of a linked list, such as tree which would create a much more efficient way of dealing with data, thus speeding up the application's processes involving data, such as deletion and insertion. Although changing the data structure would not have been a big problem at the beginning of the application's implementation, going back and changing at the end is more difficult. This is due to the fact that lots of the functionality is dependent upon the data, and any changes could cause parts of the application to no longer work and testing to be redone.

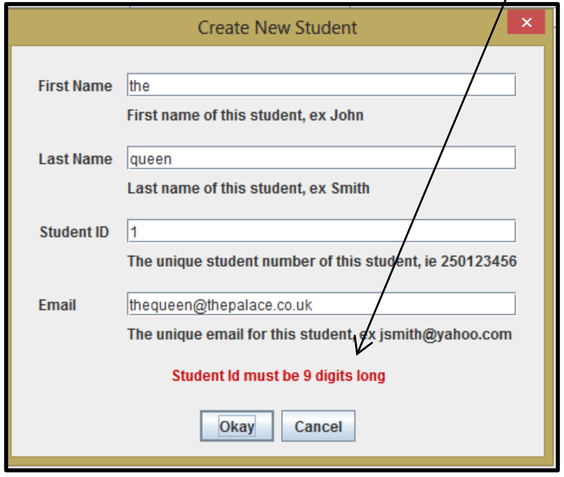
As it is discussed by Stone et al.[4:25] the most important factor we thought about when designing the GUI was the user, this is due to the fact that the end user is the person/people who will be using the system on a regular basis and it is imperative they know how to use the system effectively. As the user is a teacher/ professor, it was assumed that there would be an average knowledge of computer systems, thus the system was designed to be straightforward for all users.

To enhance the systems ‘learnability’ we grouped certain things together, making sure that it was predictable; such that someone who had just opened the system for the first time could easily navigate and press buttons knowing how the system would respond. Buttons which have similar functionality are grouped together with a group box, and given a title generalizing the functionality, meaning that it is even easier to locate the desired functionality of a system. The system was also designed to be predictable for the users, meaning that the user would be confident using the system; as it is intuitive to use. Operation visibility helps the application become predictable, in the way that if a user clicked a button there would be a response from the system. Error messages, on screen instructions all gave feedback to the user, helping them to know what they are doing, on-screen instructions were also included to further help the user to know exactly what to do; such as being told what format data needed to be in. Not only does predictability increases the user’s performance using the application, but it also improves their satisfaction with the application and thus their happiness with using the overall application [5].

The familiarity of using the system, allows users to be more comfortable using our system than other applications that go for a ‘new look’, the system was designed so that it looks similar to other gradebook applications, making the transition from other software even easier for the end user. The use of natural language and the lack of jargon allowed the novice user to be at ease, knowing they will always be able to understand the processes of the system.

Giving the users high quality feedback from the system, was also an significant issue; giving users adequate feedback is essential to tell the user that there actions have been processed [6], such as if there was an error; then there will be an on-screen message alerting the user. Instead of the program just randomly crashing or an error message appearing with technical errors and jargon, we made an effort to make the error messages as user friendly as possible, such that a non-technical user will be able to understand what went wrong and how to fix it. We also made the decision to not over use pop up boxes, only having them for important things in the system; this is mostly used when the system is confirming important actions. If the user's action is an irreversible action such as deleting a course, with a potentially bad repercussion, we need to make sure the user is happy to proceed and is not doing it accidentally.

Fig.1 Screenshot of an on screen error message, from the gradebook



Create New Student

First Name: the
First name of this student, ex John

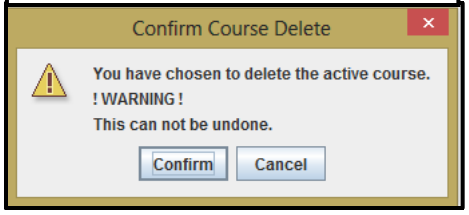
Last Name: queen
Last name of this student, ex Smith

Student ID: 1
The unique student number of this student, ie 250123456

Email: thequeen@thepalace.co.uk
The unique email for this student, ex jsmith@yahoo.com

Student Id must be 9 digits long

Fig.2 Screenshot of a pop up message to the user.



Confirm Course Delete

You have chosen to delete the active course.
! WARNING !
This can not be undone.

Perhaps the biggest HCI design principle the group followed while designing the GUI was consistency; making all of the different aspects of the system look the same, as discussed by Cox [7], consistency allows the user to know that it is all part of the same application and creates a more professional look. We kept as many parts of the system looking the same, from the font style and size, to the background colour and the positioning of buttons. It was also important to make sure each different GUI screen was also kept consistent. This helps speed up how quickly the user can complete tasks, such as data entry, although there are different GUI's for adding a new course, student and deliverable they are all so similar that it is it easy to learn the process.

The main benefit from consistency is that the user can easily learn the same and similar tasks become very efficient to do because after a while the actions become built in to the user.

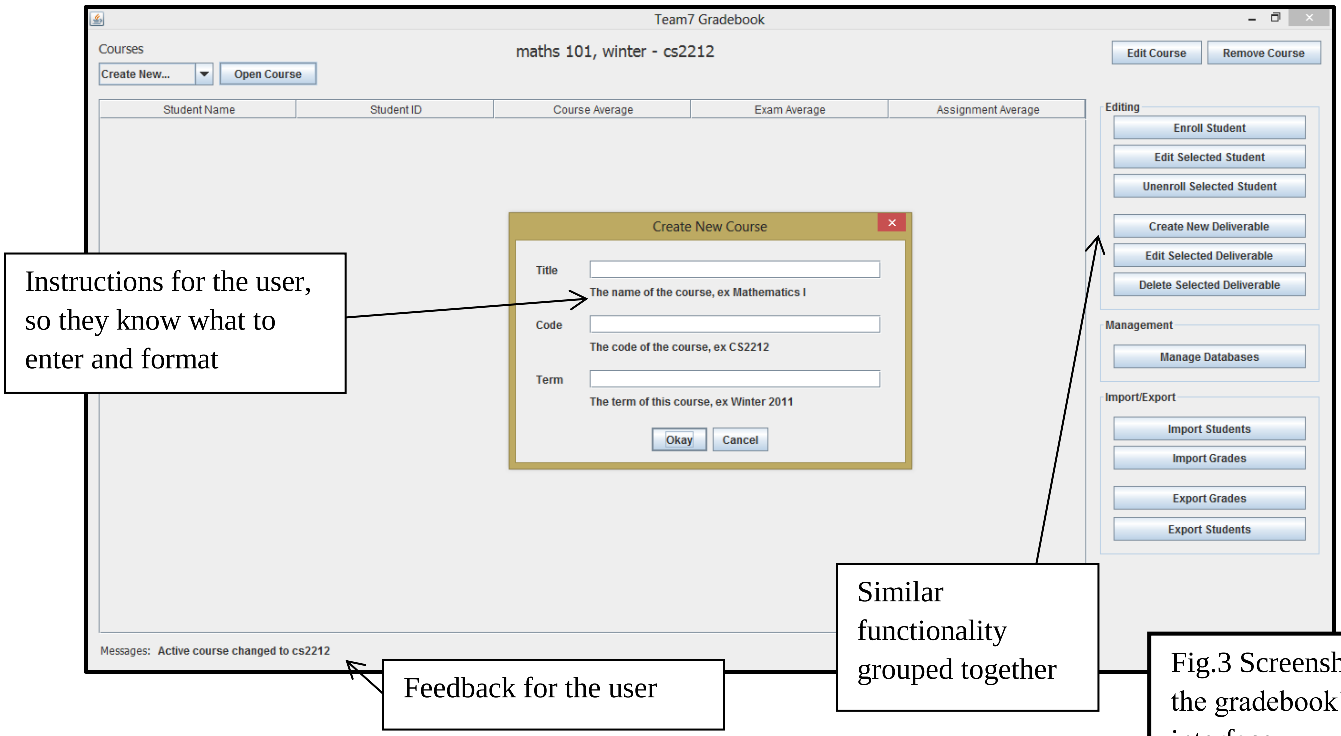

Team7 Gradebook
Courses
Create New...
Open Course
maths 101, winter - cs2212
Edit Course
Remove Course
Student Name
Student ID
Course Average
Exam Average
Assignment Average
Editing
Enroll Student
Edit Selected Student
Unenroll Selected Student
Create New Deliverable
Edit Selected Deliverable
Delete Selected Deliverable
Management
Manage Databases
Import/Export
Import Students
Import Grades
Export Grades
Export Students
Create New Course
Title
The name of the course, ex Mathematics I
Code
The code of the course, ex CS2212
Term
The term of this course, ex Winter 2011
Okay
Cancel
Messages: Active course changed to cs2212
Instructions for the user, so they know what to enter and format
Similar functionality grouped together
Feedback for the user

Fig.3 Screenshot of the gradebook's user interface

When designing the back end of the gradebook system, the reusability of code was important to think about. Making the code more reusable would decrease the amount of work that would be needed if the client wanted to migrate the application into a different format, such as from a desktop application into a web or mobile app.

A variety of different techniques were used, in trying to make the code reusable, such as when designing the classes we tried to make sure that each class had a particular purpose[8]. This allows it to be obvious what is happing in each class and separates different parts of the code, for example keeping business logic out of the GUI classes as much as possible. This means that if a particular functionality was needed to be reused then that particular class or method responsible could be copied to the new application.

Designing the application with reusable code, has allowed us to be more prepared for the future demands of the client, if we were going to change this desktop application into a web app then modification to the code would be required. Although some of the code would stay the same, some parts would be required to be rewritten and added to. When the original application was designed it was structured into separate layers; such as the data access, business logic layer and the front end (GUI). This means that most of the parts of the application not involved in the interface can be kept and parts of that code can be reused in the new application, leaving only the front end of the system being the only major part to be rewritten. This is because it would be a lot of work trying to get the user interface to look the same on the web app as it did in the desktop application. There are also different things to think about with the web application which would not have been thought of before, such as browser compatibility and how users interact differently with a web application as opposed to a desktop application, such as the users only interacting with the different web pages. The back end of the application although it uses the same logic and ideas will also need some rewriting. Whilst some larger components of the code can be reused; saving time and money it may be easier to rewrite smaller parts of the code, which will allow the application to be fully functional as a web application [9]. The application would also need to host the database backend of the application; this could be done with a Database Management System (DBMS) such as MySQL, which can integrate with the front end of the website application.

Sources:

[1] S, Lambert. (2012, Novemeber 2012) Quick Tip: The OOP Principle of Coupling. GameDevelopment. [On-line] Avaliable at:http://gamedevelopment.tutsplus.com/tutorials/quick-tip-the-oop-principle-of-coupling--gamedev-1935 [Accessed: 27th April 2014]

[2] Boryoswich. (2007, May 3) . *Design Principles: Coupling (Data and otherwise)*.Toolbox [On-line]. Avaliable at: http://it.toolbox.com/blogs/enterprise-solutions/design-principles-coupling-data-and-otherwise-16061 [Accessed:April 1st 2014]

[3] B, Venners. (1998, Novemeber 1). *Inheritance versus composition: Which one should you choose?*.JavaWorld. [On-line].Avaliable at: http://www.javaworld.com/article/2076814/core-java/inheritance-versus-composition--which-one-should-you-choose-.html [Accessed: March 18th 2014]

[4] Stone, et al. (2005, April 25). *User Interface Design and Evaluation*. (1st Edition). [On-line]. Available: http://booksite.elsevier.com/9780120884360/casestudies/Chapter_01.pdf [Accessed: March 27th 2014].

[5] Gajos et al. *Predictability and Accuracy in Adaptive User Interface.* CS Washington University, 2008.[On-line] Available at:https://www.cs.washington.edu/ai/puirg/papers/kgajos-chi08-predictability.pdf [Accessed: March 27th 2014].

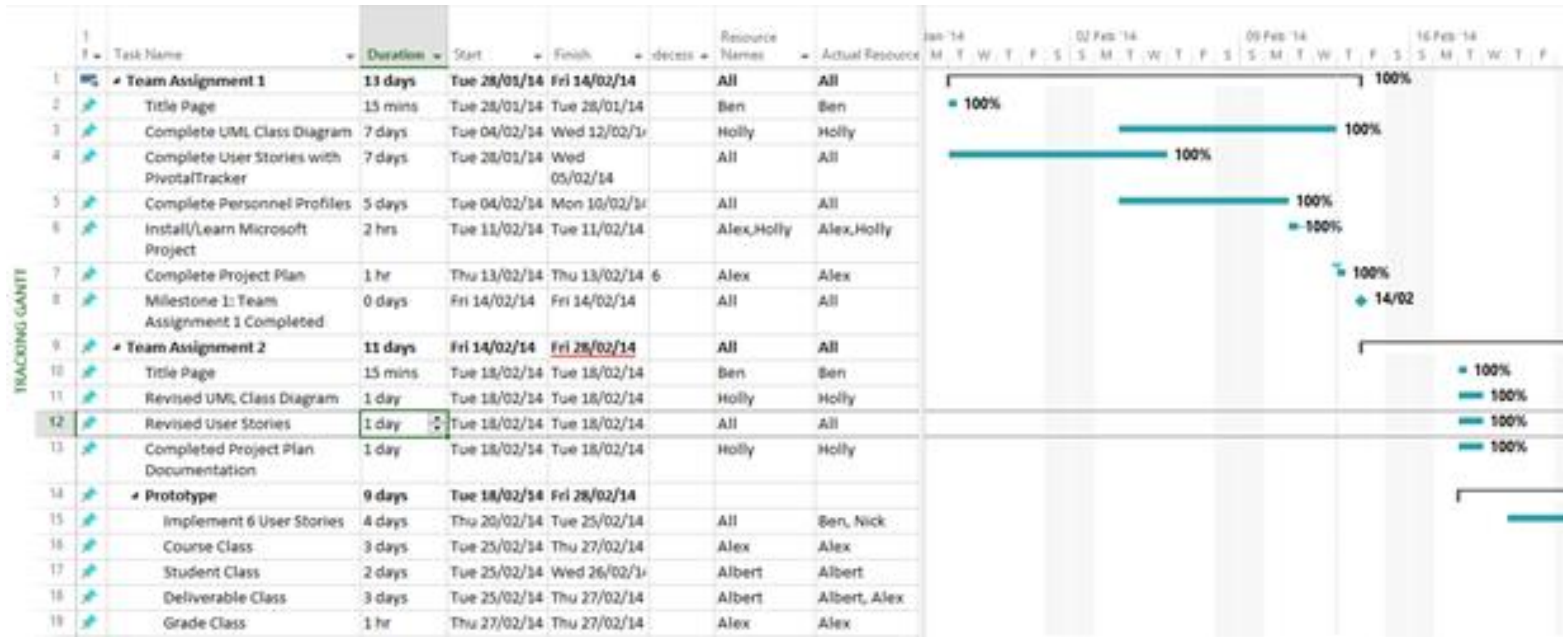
[6] M. Pérez-Quiñones, J.Sibert (1995, December 12).*A Collaborative Model of Feedback in Human-Computer Interaction.* [On-line] Available at: http://www.sigchi.org/chi96/proceedings/papers/Perez/map1txt.htm [Accessed: March 28th 2014]

[7] P.Cox (2012, August 3). *Maintaining Consistency in Your UI Design.* Design Shack. [On-line] Available at: http://designshack.net/articles/graphics/maintaining-consistency-in-your-ui-design/ [Accessed: March 28th 2014].

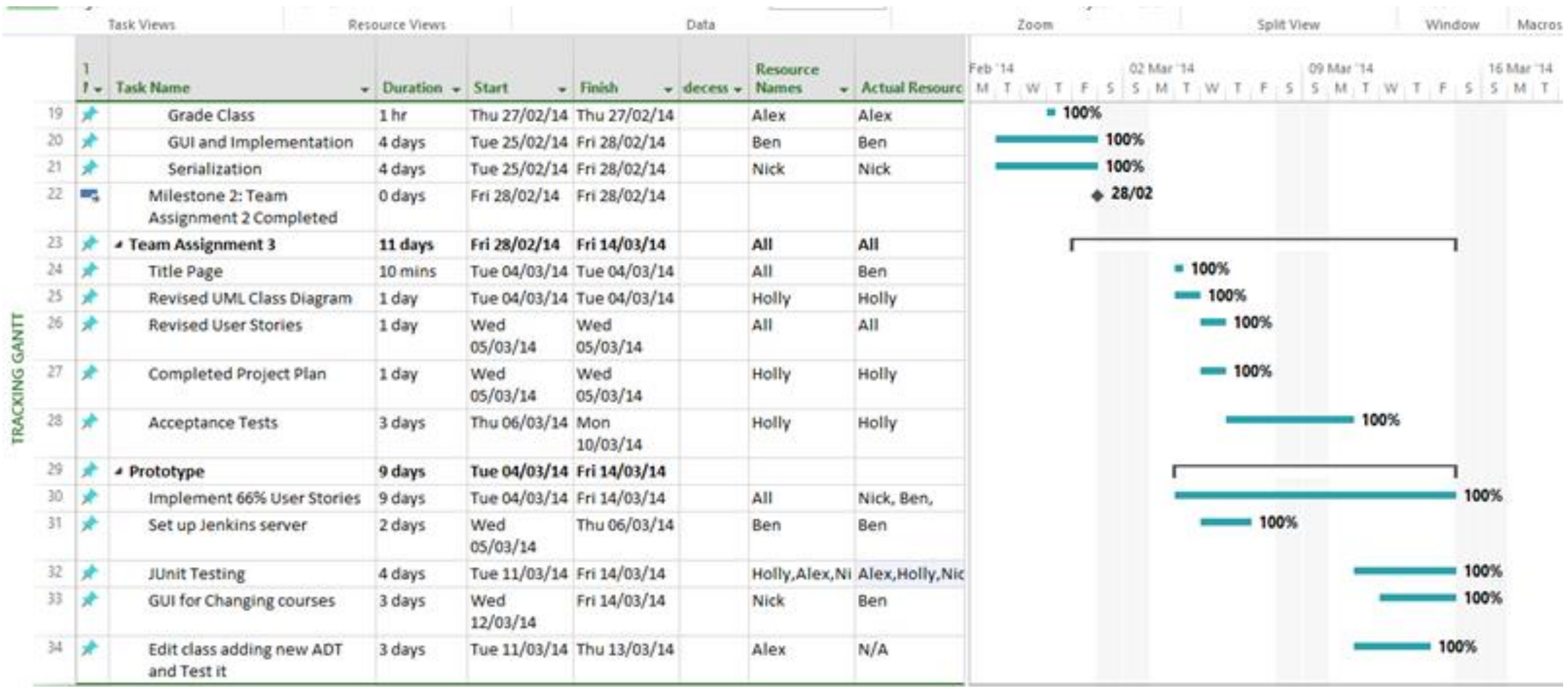
[8] Hoskinator (2006, June 19). *10 tips on writing reusable code*. Blogspot. [On-line] Available at: http://hoskinator.blogspot.co.uk/2006/06/10-tips-on-writing-reusable-code.html [Accessed: March 31st 2014]

[9] A. Puder. (2004). *Extending Desktop Applications to the Web* .San Francisco State University. [On-line] Available at: http://www.puder.org/publications/dorea04.pdf [Accessed: March 30th 2014]

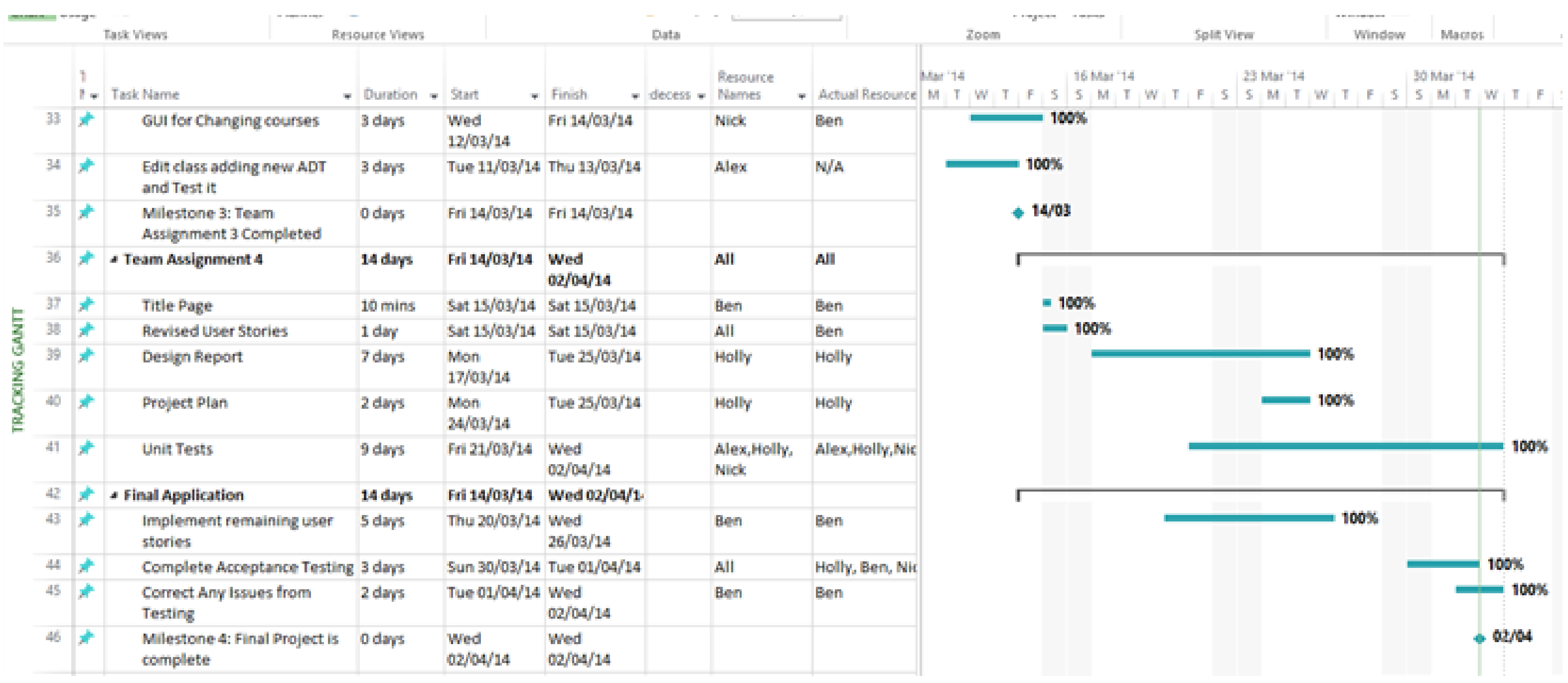
Project Plan



ID	Task Name	Duration	Start	Finish	decess	Resource Names	Actual Resource
1	**Team Assignment 1**	**13 days**	**Tue 28/01/14**	**Fri 14/02/14**		**All**	**All**
2	Title Page	15 mins	Tue 28/01/14	Tue 28/01/14		Ben	Ben
3	Complete UML Class Diagram	7 days	Tue 04/02/14	Wed 12/02/1		Holly	Holly
4	Complete User Stories with PivotalTracker	7 days	Tue 28/01/14	Wed 05/02/14		All	All
5	Complete Personnel Profiles	5 days	Tue 04/02/14	Mon 10/02/1		All	All
6	Install/Learn Microsoft Project	2 hrs	Tue 11/02/14	Tue 11/02/14		Alex,Holly	Alex,Holly
7	Complete Project Plan	1 hr	Thu 13/02/14	Thu 13/02/14	6	Alex	Alex
8	Milestone 1: Team Assignment 1 Completed	0 days	Fri 14/02/14	Fri 14/02/14		All	All
9	**Team Assignment 2**	**11 days**	**Fri 14/02/14**	**Fri 28/02/14**		**All**	**All**
10	Title Page	15 mins	Tue 18/02/14	Tue 18/02/14		Ben	Ben
11	Revised UML Class Diagram	1 day	Tue 18/02/14	Tue 18/02/14		Holly	Holly
12	Revised User Stories	1 day	Tue 18/02/14	Tue 18/02/14		All	All
13	Completed Project Plan Documentation	1 day	Tue 18/02/14	Tue 18/02/14		Holly	Holly
14	**Prototype**	**9 days**	**Tue 18/02/14**	**Fri 28/02/14**			
15	Implement 6 User Stories	4 days	Thu 20/02/14	Tue 25/02/14		All	Ben, Nick
16	Course Class	3 days	Tue 25/02/14	Thu 27/02/14		Alex	Alex
17	Student Class	2 days	Tue 25/02/14	Wed 26/02/1		Albert	Albert
18	Deliverable Class	3 days	Tue 25/02/14	Thu 27/02/14		Albert	Albert, Alex
19	Grade Class	1 hr	Thu 27/02/14	Thu 27/02/14		Alex	Alex



ID	Task Name	Duration	Start	Finish	decess	Resource Names	Actual Resourc
19	Grade Class	1 hr	Thu 27/02/14	Thu 27/02/14		Alex	Alex
20	GUI and Implementation	4 days	Tue 25/02/14	Fri 28/02/14		Ben	Ben
21	Serialization	4 days	Tue 25/02/14	Fri 28/02/14		Nick	Nick
22	Milestone 2: Team Assignment 2 Completed	0 days	Fri 28/02/14	Fri 28/02/14			
23	**Team Assignment 3**	**11 days**	**Fri 28/02/14**	**Fri 14/03/14**		**All**	**All**
24	Title Page	10 mins	Tue 04/03/14	Tue 04/03/14		All	Ben
25	Revised UML Class Diagram	1 day	Tue 04/03/14	Tue 04/03/14		Holly	Holly
26	Revised User Stories	1 day	Wed 05/03/14	Wed 05/03/14		All	All
27	Completed Project Plan	1 day	Wed 05/03/14	Wed 05/03/14		Holly	Holly
28	Acceptance Tests	3 days	Thu 06/03/14	Mon 10/03/14		Holly	Holly
29	**Prototype**	**9 days**	**Tue 04/03/14**	**Fri 14/03/14**			
30	Implement 66% User Stories	9 days	Tue 04/03/14	Fri 14/03/14		All	Nick, Ben,
31	Set up Jenkins server	2 days	Wed 05/03/14	Thu 06/03/14		Ben	Ben
32	JUnit Testing	4 days	Tue 11/03/14	Fri 14/03/14		Holly,Alex,Ni	Alex,Holly,Nic
33	GUI for Changing courses	3 days	Wed 12/03/14	Fri 14/03/14		Nick	Ben
34	Edit class adding new ADT and Test it	3 days	Tue 11/03/14	Thu 13/03/14		Alex	N/A



	Task Name	Duration	Start	Finish	decess	Resource Names	Actual Resource
33	GUI for Changing courses	3 days	Wed 12/03/14	Fri 14/03/14		Nick	Ben
34	Edit class adding new ADT and Test it	3 days	Tue 11/03/14	Thu 13/03/14		Alex	N/A
35	Milestone 3: Team Assignment 3 Completed	0 days	Fri 14/03/14	Fri 14/03/14			
36	**Team Assignment 4**	**14 days**	**Fri 14/03/14**	**Wed 02/04/14**		**All**	**All**
37	Title Page	10 mins	Sat 15/03/14	Sat 15/03/14		Ben	Ben
38	Revised User Stories	1 day	Sat 15/03/14	Sat 15/03/14		All	Ben
39	Design Report	7 days	Mon 17/03/14	Tue 25/03/14		Holly	Holly
40	Project Plan	2 days	Mon 24/03/14	Tue 25/03/14		Holly	Holly
41	Unit Tests	9 days	Fri 21/03/14	Wed 02/04/14		Alex,Holly, Nick	Alex,Holly,Nic
42	**Final Application**	**14 days**	**Fri 14/03/14**	**Wed 02/04/1**			
43	Implement remaining user stories	5 days	Thu 20/03/14	Wed 26/03/14		Ben	Ben
44	Complete Acceptance Testing	3 days	Sun 30/03/14	Tue 01/04/14		All	Holly, Ben, Nic
45	Correct Any Issues from Testing	2 days	Tue 01/04/14	Wed 02/04/14		Ben	Ben
46	Milestone 4: Final Project is complete	0 days	Wed 02/04/14	Wed 02/04/14			

Unit Tests

<u>Non GUI Classes</u>**:**
App
Course
CourseException
CSVManger
Deliverable
DeliverableException
Grade
GradebookException
GradeImportResult
GradeTableModel
GUIHelper
ImportedGrade
ImportResult
InvalidCsvException
PersistanceManager
Student
TableRow

<u>GUI Classes:</u>

CourseGUI
DatabaseManagerGUI
DeliverableGUI
ImportGUI
MainWindow
StudentGUI
StudentSearchGUI