Plan

For this iteration we are going to implement some more user stories, but focus on getting the product ready for a final release by doing additional refactoring and testing.

The SQLDatabase class was properly implemented at the start of this iteration. The intention was for it to have been finished earlier, but a small bug in the onUpgrade method and a simple lack of understanding of the version property led to a number of run-time exceptions. Fortunately, with a working database development has sped up considerably as it can be relied upon to store data persistently and in a manner that is relatively simple to access.

# User Stories

Specifically, Ian, Anthony, and Graham are implementing the Grade Calculator, Calendar, and Reminders user stories. David is handling the final testing along with our customer. We are all attempting to contribute to refactoring this application. Ideally, anyone who notices a design smell will either write a TODO in the code, an issue in Github, or simply fix it right away.

The user stories we are working on this iteration are as follows:

1. Grade Calculator: As a student I’d like to be able to view my grades for each course.
   1. Priority: high
   2. Cost: 1 week
2. Reminders: As a student I’d like to be able to view my grades for each course:
   1. Priority: high
   2. Cost: 1 week
3. Schedule: As a student, I’d like to be able to view my current schedule.
   1. Priority: medium
   2. Cost: 1 week

# Developer Tasks

The Grade Calculator user story will be implemented so that students using the application can keep track of their performance in a particular course, and potentially their degree program. This will be done by extending the functionality of the Database (Stub and SQL) to find tasks by the associated user and course attributes, modify the individual tasks, and by extending the Task class to store the grade and weight of an assignment or test. The Grader class was also created to handle the calculation of the grade and remaining weight for each Course, it has been tested with the GraderTest class.

The Schedule user story involved modifying the views to accept date and time input from a more specialized source, instead of text input. This input is then serialized and saved in the database. Tasks can have the date and time updated or modified, while course records are still not editable. Ideally, we’ll be able to create a view to display this information in a way resembling the “Week at a Glance” page that Aurora has.

The Reminders developer tasks are mostly limited to research, as none of us have experience working with device API’s. We should be able to go through the device API for the supplied Nexus tablet and genymotion to learn what functions need to be called. This will allow us to set an alarm to go off when a task due date has been reached.

The three refactoring items we are focussing on for this iteration are:

1. Strategy Pattern for ArrayLists
2. Strategy Pattern for Grade Calculation
3. Abstract Database Layer

These items all have design smells of one form or another. The first two deal with violations of the Single Responsibility Principle, while the last deals with the Dependency Inversion Principle.

The first refactoring item was fairly straightforward. The developer tasks are: Create static class called ArrayConverter, write functions to convert ArrayLists into String arrays with relevant data, update abstract ListItem class, and write unit tests.

The Grade Calculation function was partially implemented shortly after the iteration 2 due date, as we had hoped to have it ready then. Unfortunately, it violated the Single Responsibility Principle by forcing a presentation class to calculate the grade in a course from the list of tasks. The developer tasks for the refactor are: create static class called Grader, write functions to calculate current course grade and the amount of weight left for tasks for the course, and write unit test to ensure viability.

Abstract Database Layer refers to the need to make use of the Database interface that was partially written for iteration one. This is being done because the presentation layer of the application should not be forced to depend on the details of the persistence layer, in case there are bugs. The developer tasks are: update Database interface with methods from Stub and SQLDatabase classes, rewrite staticDB to use an object of type Database, rewrite Presentation classes to accept Database item from staticDB after its initialization.

# Additional User Stories

If time permits we will also attempt to implement other user stories from our original vision statement. These user stories include:

* Passing Grade: As a student, I’d like to be able to view my grades for each course and calculate my overall GPA for the term and my degree
  + Priority: low
  + Cost: 1 week
* Organize Courses into Terms: As a student, I’d like to be able to organize my courses into terms so that I can easily find course information
  + Priority: low
  + Cost: 1 week
* Edit and View Prof Info: As a student, I’d like to be able to enter and view my prof and TA information so that I can easily find my professor’s office room, hours, etc.
  + Priority: medium
  + Cost: 1 week

The costs for these items have been revised as they would all require significant changes to be made to the application, and would require additional tests to be written. Other user stories we will most likely not be able to implement include: class attendance, change settings, degree requirements, update academic history.

# Testing

One of, if not the most important things that we have to do with this iteration is test the application. We need to thoroughly test the important units, as well as test how well they work together. Many of these tests are already written; however, they need to be done in more detail. Specifically, we are going to try and focus on the edge cases where bugs can be left unnoticed for long periods of time. Furthermore, scaffolding needs to be added to the UIs themselves, so that user input will be less likely to result in a crash. For example, the weight of the tasks needs to be between 0 and 1, and the UI should not accept any other input for that category.

Additionally, we will do as much acceptance testing as we can. We will make use of the Robotium plugin for Android studio, but since we can only use it for free a small number of times we will need to be careful and not waste it. This means that we will need to push the acceptance testing back as much as possible, or test it ourselves without Robotium.