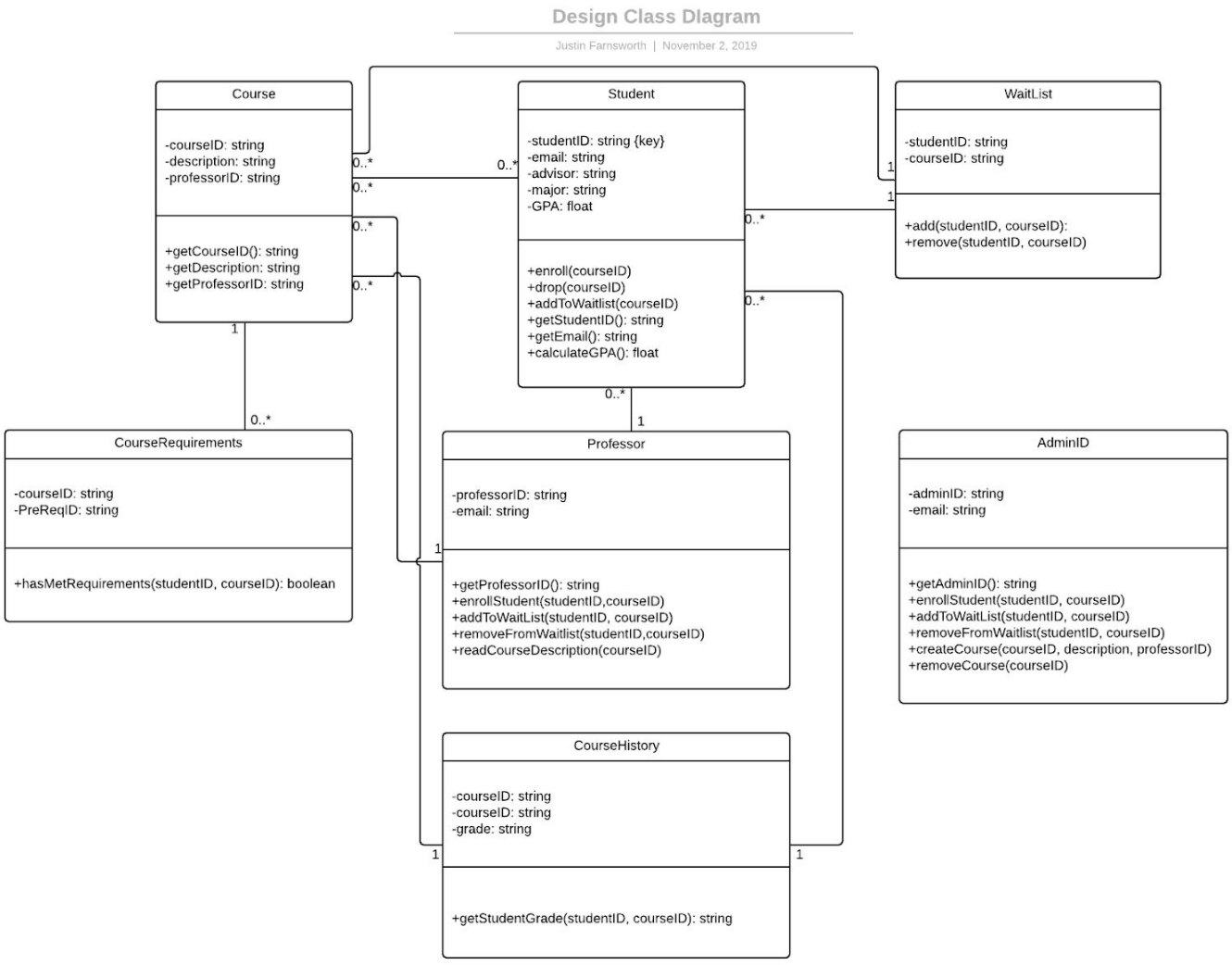
Justin Farnsworth
Software Engineering
November 4, 2019

Assignment 4: Open-Source Software - Analysis and Design

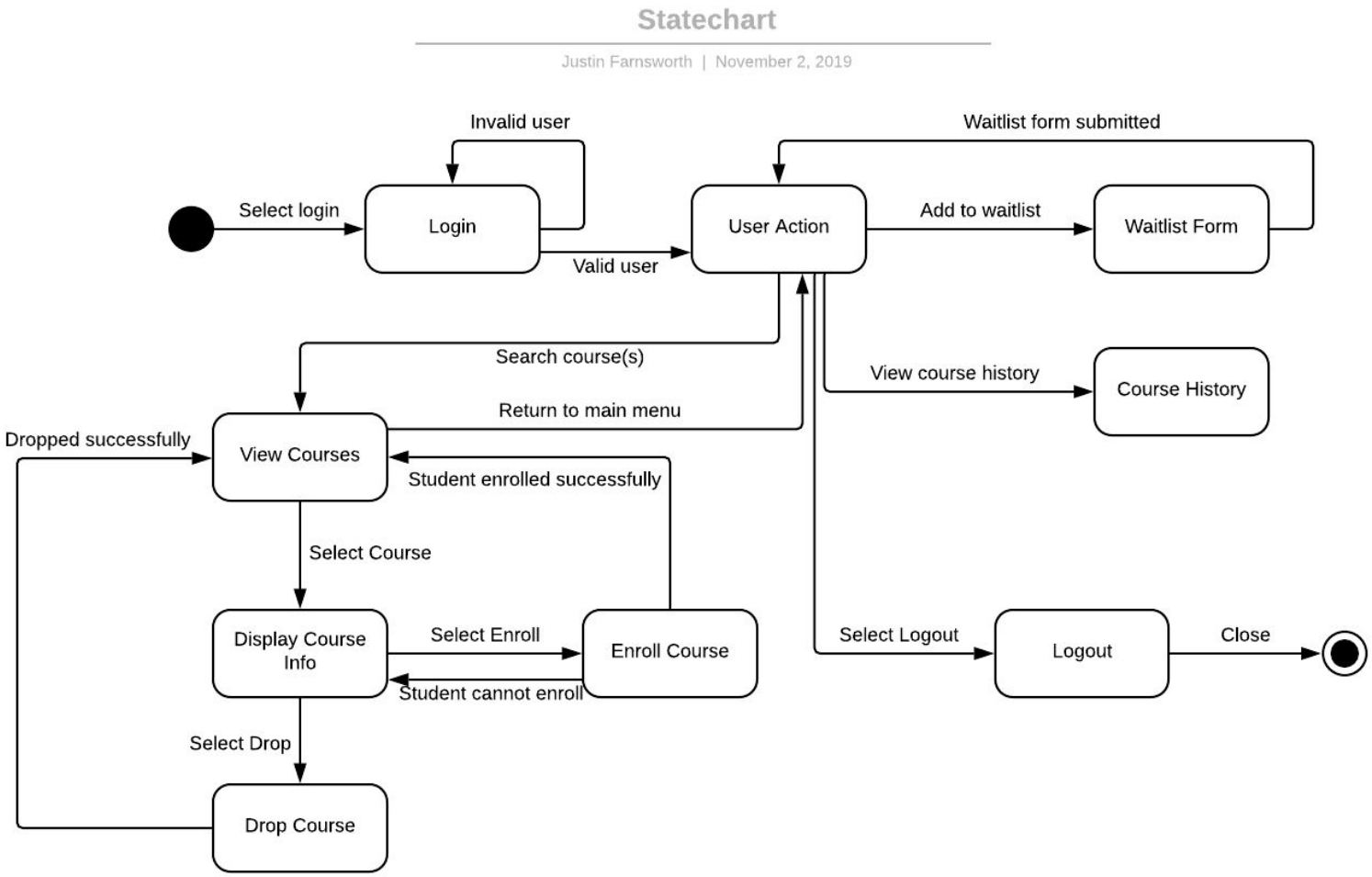
Use Case Description

Use Case Name	Student enrolls into a course
Actor	Student
Preconditions	Student must be logged into their respective account.
Trigger	The student selects a course and clicks the Enroll button.
Scenario	1. The student logs into their account and gets redirected to the menu.. 2. Then the student clicks the View Courses button. 3. The student selects a course from the list of courses. 4. The information of the course is displayed along with a button that says Enroll. 5. The student selects the Enroll button. 6. The program will verify that the student has satisfied the prerequisites. 7. The program will add the student to the course's roster. 8. The student will be redirected back to the View Courses page, but a message will appear saying "Student enrolled into [course name] successfully!"
Exceptions	Student is not logged in or the student didn't satisfy the prerequisites.
Priority	High priority. Other functions such as Drop depend on the Enroll functionality.
Frequency	Frequently
When available	First increment
Secondary actors	Administrators and professors
Channels to secondary actors	PC-based system.

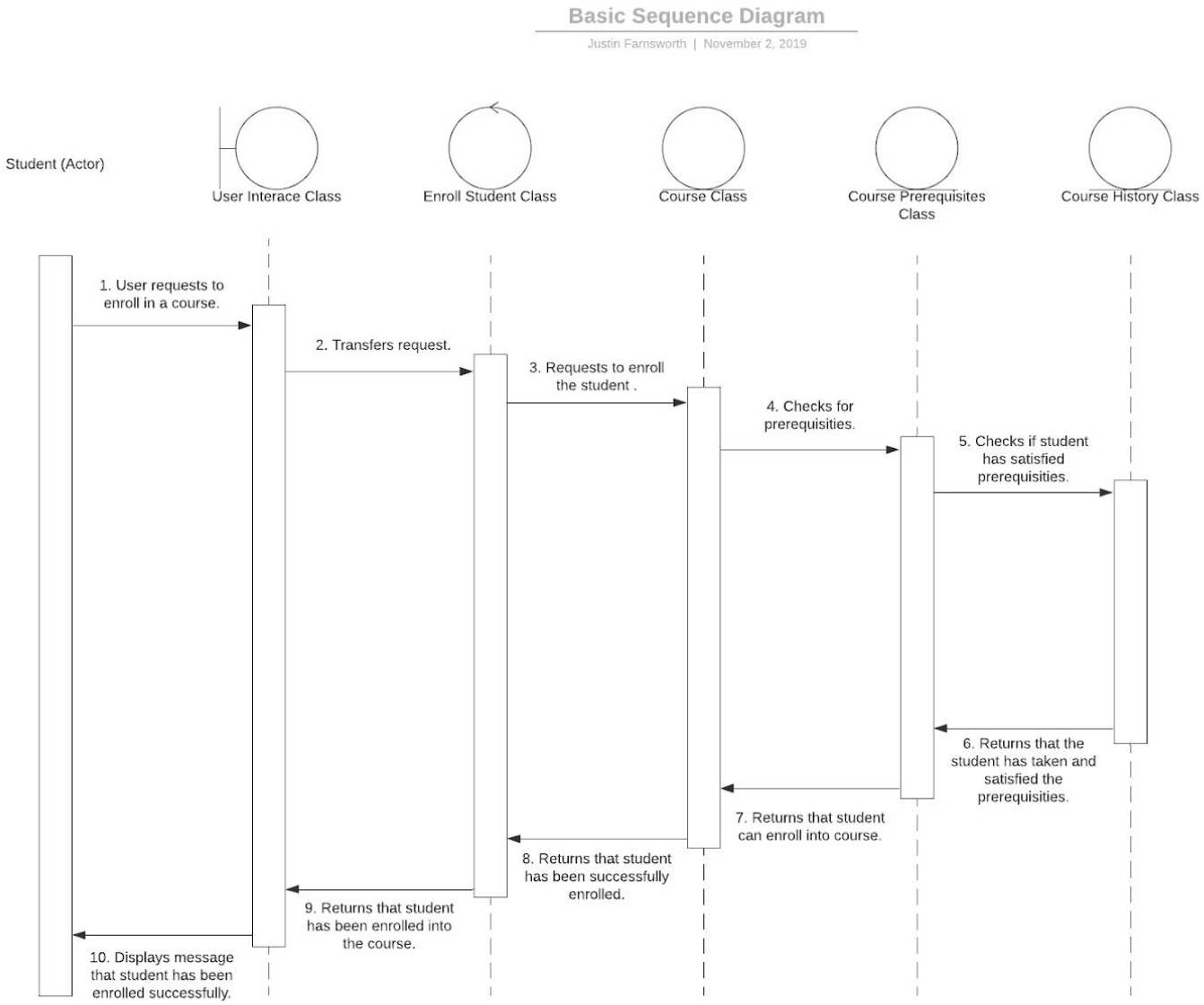
Design Class Diagram



Statechart


Statechart
Justin Farnsworth | November 2, 2019
Invalid user
Waitlist form submitted
Select login
Login
User Action
Add to waitlist
Waitlist Form
Valid user
Search course(s)
View course history
Course History
Return to main menu
Dropped successfully
View Courses
Student enrolled successfully
Select Course
Display Course Info
Select Enroll
Enroll Course
Select Logout
Logout
Close
Student cannot enroll
Select Drop
Drop Course

System Sequence Diagram



Mockup

The mockup will be uploaded separately along with this file due to the large amount of content from it alone.

Eight Golden Rules

1. **Strive for consistency:** The interface will have the same color scheme and the buttons/text will also be the same throughout.
2. **Seek universal usability:** Anyone who is enrolled or works at a college/university will have access to use the program.
3. **Offer informative feedback:** Anytime a user wants to enroll into or drop a course, the user will be notified about the changes and will be asked to proceed anyway. On the next page, the user will be informed about the actions they have chosen, such as the completion of certain tasks.
4. **Design dialogs to yield closure:** If a student enrolls a course, the page will inform the student that they've been enrolled. Similarly, it will tell a student if they've dropped a course as well. The same goes for adding/removing a class from the waitlist.
5. **Prevent errors:** If an error occurs, the program will not continue until the error has been addressed. For example, if a user inserts the wrong login information, the program will not allow the user to login until the correct information is plugged in.
6. **Permit easy reversal of actions:** The user can enroll and drop courses as long as they're available. Also, the previous page will be accessible through a button. For example, if the user accidentally clicks the waitlist button, they will have a button that will redirect them to the main menu page.
7. **Keep users in control:** The program will be simple enough so that the user can intuitively navigate the program. The functionalities will do exactly as they're intended to do. For example, enrolling a student will enroll a student into a course as expected.
8. **Reduce short-term memory load:** Information about the course will be displayed when viewing courses. Similarly, the user's profile will contain all the information about the student.

Test Case

Unit Testing: Every time a module is added to the program, the program will be tested for errors. All the functionalities will be tested numerous times with different inputs. Some inputs will test the program's exception handling to ensure that it is working as intended.

Integration Testing: I plan on taking the top-down integration method since it will make debugging much easier. If problems occur as a result of the unit that I integrated, then I can easily identify and address the source of the problem.

System Testing: Usability testing seems to be the most reasonable option as it will test the functionality as well as the aesthetics of the program.

For debugging, I will use Eclipse for debugging since I can use it on a Windows computer. I'm also aware that Eclipse works on Mac as well.

Functionality Tested	Inputs	Expected Output	Actual Output
Enrolling a student	StudentID: 123456 CourseID: CSC345	Student enrolled successfully!	
Enrolling a student	StudentID: 123456 CourseID: CSC999	CSC999 does not exist!	
Add to waitlist	StudentID: 123456 CourseID: CSC498	Student added CSC498 to the waitlist!	
Add to waitlist	StudentID: 123456 CourseID: CSC100	CSC100 does not exist!	
Remove from waitlist	StudentID: 123456 CourseID: CSC345	Student 123456 removed CSC415 from the waitlist!	
Remove from waitlist	StudentID: 123456 CourseID: CSC345 (again)	Student 123456 is not on the waitlist for CSC345!	

Maintenance and Communication

As the project develops, the incentive for organizing ideas becomes an imperative. Also, contributors of the project should also understand what is needed/requested for the project. To make communication and maintenance easier, I wish to provide personal information, specifically an email address, so contributors can ask me about the project, such as ideas and suggestions. I will also keep the project on GitHub so people can share their ideas and contributions, where they may be reviewed for approval. I will also create a code of conduct that lists out the expectations for the project, including future goals for the project. For example, perhaps another feature could be added or the interface needs to be modified. In general, a sense of community with a common goal is what I would like to promote.

Open Source License

The three types of OSS licenses I considered were the following:

License Type	Strengths	Weaknesses
Apache License	Allows users to use, modify, and distribute and product under this license. It is also compatible with other licenses, such as GPL.	Not compatible with MIT License. Also, unmodified parts of a product must have the Apache License.
GNU General Public License	Commonly used for open-source projects.	Any portion of code that has a GPL includes the entire project into a GPL. The license is impractical for corporate organizations.
MIT License	This license is one of the simplest and has very few restrictions. It is also one of the most commonly used licenses.	Ambiguity with the word 'software' in various jurisdictions can create legal implications. Also, the license doesn't explicitly cover patents and trademark restrictions.

After analyzing the 3 licenses, I believe the Apache License works best for my project. Not only does it provide the most compatibility amongst other licenses, but it also ensures the original code is explicitly labeled as licensed. Also, the GPL can get in the way of users who want to utilize the code for closed-source applications and the MIT License isn't compatible with the Apache License.