Assignment 2:
Component 3

Lindan Thillanayagam (213742176)

Saad Saeed (213968284)

Date: November 19, 2017

Course: EECS 3461

Professor: Melanie Baljko

Conceptual Model

Interface metaphor

Suppose a student manages his/her courses through a guidance counsellor. A guidance counsellor is someone who is assigned the task of managing courses and advising students throughout their time in secondary school. The guidance counsellor will fetch the student's information from a localized database stored on the server or from a predefined checklist. Information includes courses the student has taken, the list of possible courses to take based on the student's academic situation, number of credits completed, and the student's grades for each completed course. Based on all this information, the guidance counsellor recommends the most logical path of action a student should take if they want to succeed academically during their time in secondary school. Furthermore, they are also responsible for completing the student's request to drop or exchange courses. The proposed application will automate the tasks performed by a guidance counsellor.

Interaction Type

The proposed application will perform many tasks; some of these being the ability to drop, add, and exchange courses.These sort of tasks will require the user to interact with the domain by viewing and manipulating their current data. One task specific to the proposed domain is the ability to view academic progress to keep the student up to date, and to guide in picking courses relevant to one's own academic progress. Such task will enable interaction by instructing the user about the choices that can be made.

Interface type

The main interface type that will be supported by the proposed application is the Graphical User Interface (GUI). The application will unfortunately be designed using Java.Swing and resultantly, will make use of the MVC paradigm. This sort of interface will support the user by making them aware of the various options they can select and warn the users when they are about to perform an irreversible action. It is a well known fact that people learn through exploring rather than following a set of instructions. Thus a GUI is perfect for when such usability criteria are the main focus; a GUI can facilitate familiarization with performing the tasks of course management while supporting the exploration paradigm that people enjoy. Furthermore, a GUI can enforce user safety via dialog-boxes and prompt the user to verify their actions before committing to them.

A secondary-but still a very important-interface type supported by the application is the web. In an ideal world, an application of such importance would be linked to a website where

students would be able to perform course management activities by entering their university credentials. This would ensure that the application would be accessible to anyone with an internet connection and login credentials. Since the program is being developed as a standalone application for demonstration purposes, this interface type can be disregarded for now. However, it does not hurt to keep in mind the interface requirements if this application were to go commercial.

Supported Functions and Activities

The application will be able to support the task of obtaining a query from the user and making a decision on it depending on multiple criteria. Furthermore, there is a burden on the user to provide information to the application before it can produce the correct action. For example, the user is responsible with the task of providing a valid course code. If this is satisfied, then the program can either allow the user's action to proceed or prevent the action from continuing due to a specified reason (missing prerequisite or unavailable seating). Additionally, the program is burdened with providing information about the user's academic progress based on the courses taken, the number of credits completed, etcetera. Here, the user has no say in the way the information is generated but it can enhance the user experience by allowing them to make appropriate actions.

Relationship among functions

The proposed application supports four different functions related to course management: adding, removing, exchanging courses, and providing a checklist of all the courses the person has already taken as well as any courses they need to take to graduate. All of the said functions can only be performed temporally; for example, one cannot add and remove courses at the same time. Most of these functions share the same type of user interaction; i.e. inputting text into a textbox or pressing a button, the program fetching data from a database, and displaying information to the user after successfully (or unsuccessfully) completing the requested task. All the course management functions can only be used if the student is enrolled in at least one course; one cannot exchange or drop courses if one is not enrolled in any in the first place.

Information Requirements

The following is a list of all the required information that will ensure the program acts as intended:

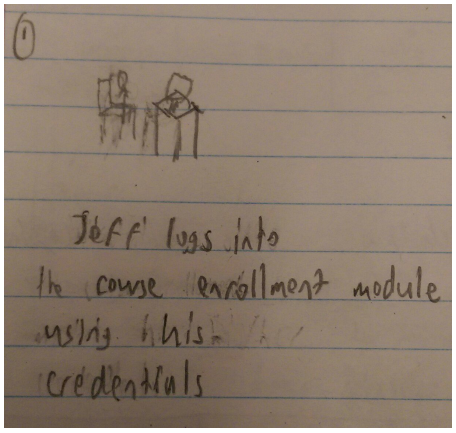
1. Retrieve student information based on login credentials by accessing a database.
2. Use information of student's completed courses as well as the data entered by the user in a textbox to allow or deny enrollment into a course.
3. Use information to update interface to reflect student's academic progress.

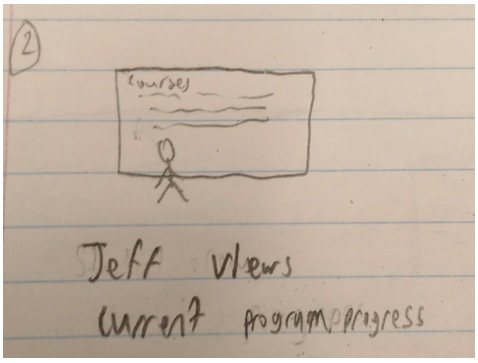
4. Fetch course details from the database associated with the provided course code.
5. Manipulate the database based on whether a course was added, dropped, or exchanged.

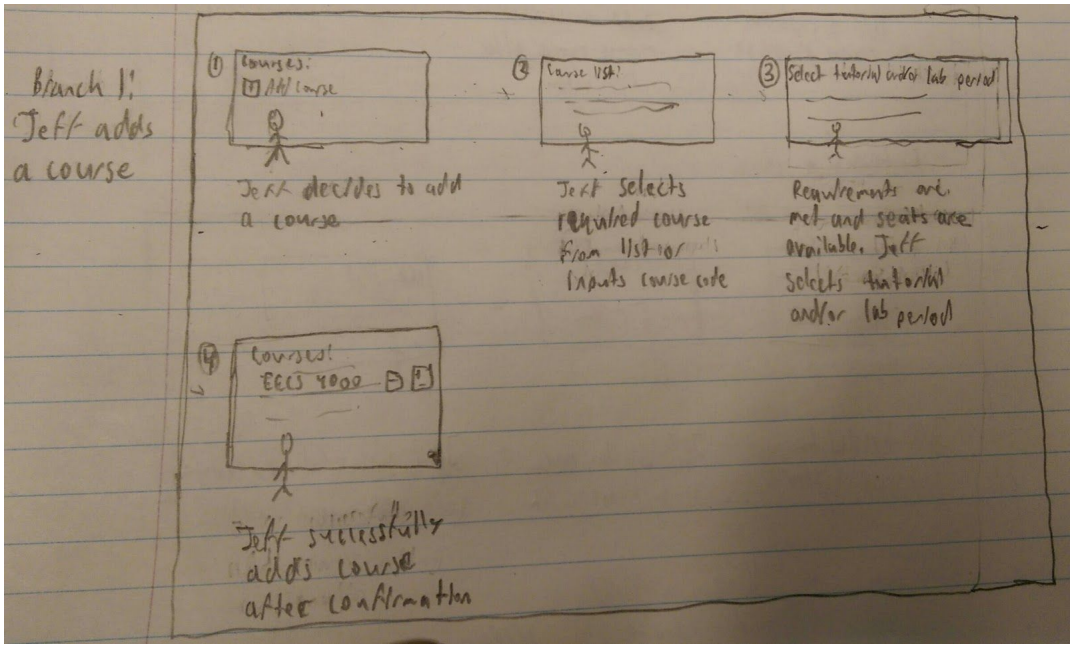
Choice for the Model

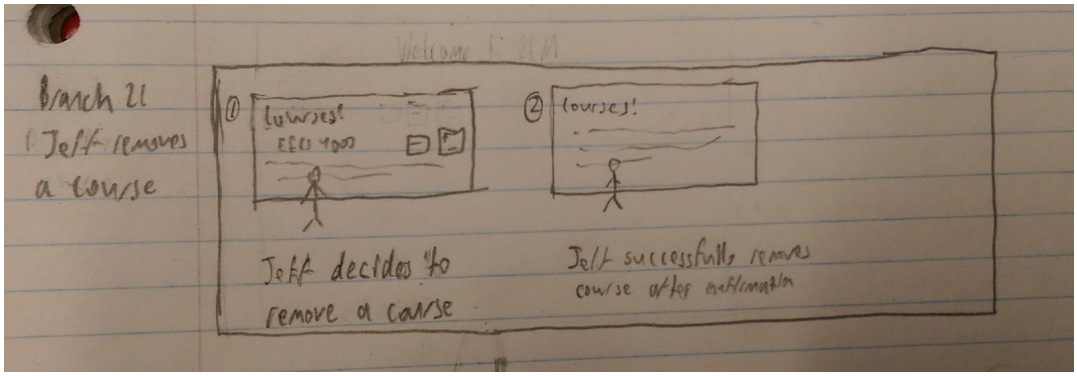
The conceptual model stated above seems the most appropriate for the proposed product as most of the propositions stated are closely linked with the requirements of the product and the end target of the product. Essentially, the program is trying to eliminate the need for an academic advisor by providing all the useful information to the user, enabling them to make an informed decision on managing courses. As a result, employing a GUI would be a reasonable interface choice considering the amount of functionality needed to be incorporated, along with the need to make the application as user-friendly as much as possible. Furthermore, the way the functions are defined makes the most logical sense. For example, the features need to be temporal and constrained (cannot exchange or drop a course when the person is not even enrolled in a course to begin with) in order for them to make the most sense and perform the jobs that they are supposed to perform. In other conceptual models, there was not a way to link all of these things together and this in turn, would have resulted in inconsistencies in the final product.

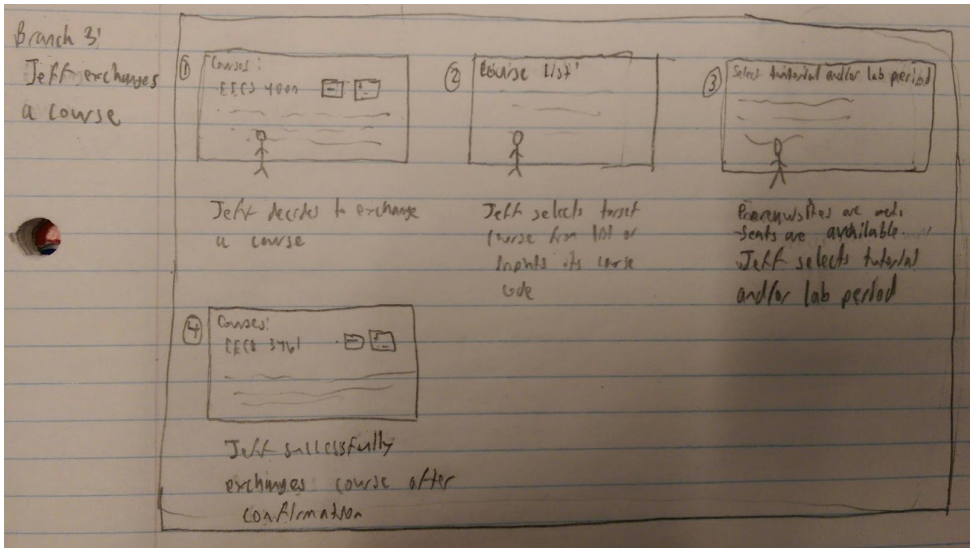
Storyboard






Branch 1: Jeff adds a course
Courses:
Add course
Jeff decides to add a course
Course list:
Jeff selects required course from list or inputs course code
Select tutorial and/or lab period
Requirements are met and seats are available. Jeff selects tutorial and/or lab period
Courses:
EECS 4000
Jeff successfully adds course after confirmation

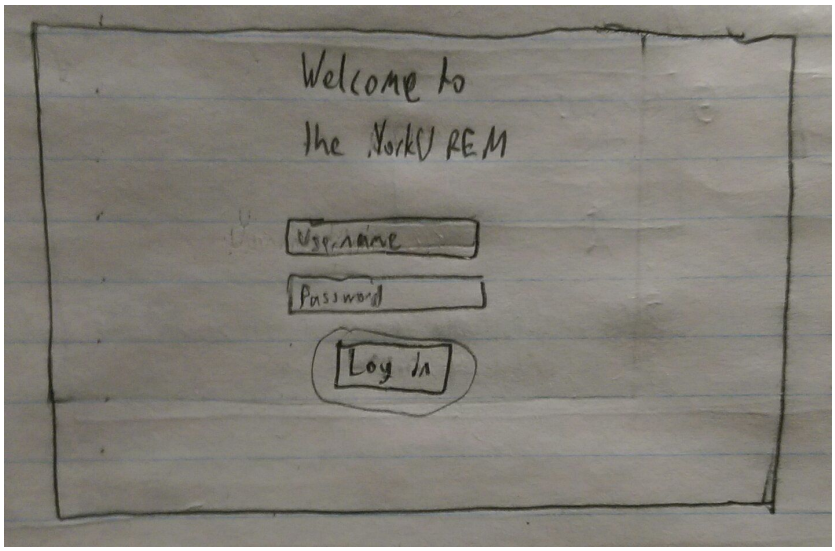

Branch 2: Jeff removes a course
Courses:
EECS 4000
Courses:
Jeff decides to remove a course
Jeff successfully removes course after confirmation

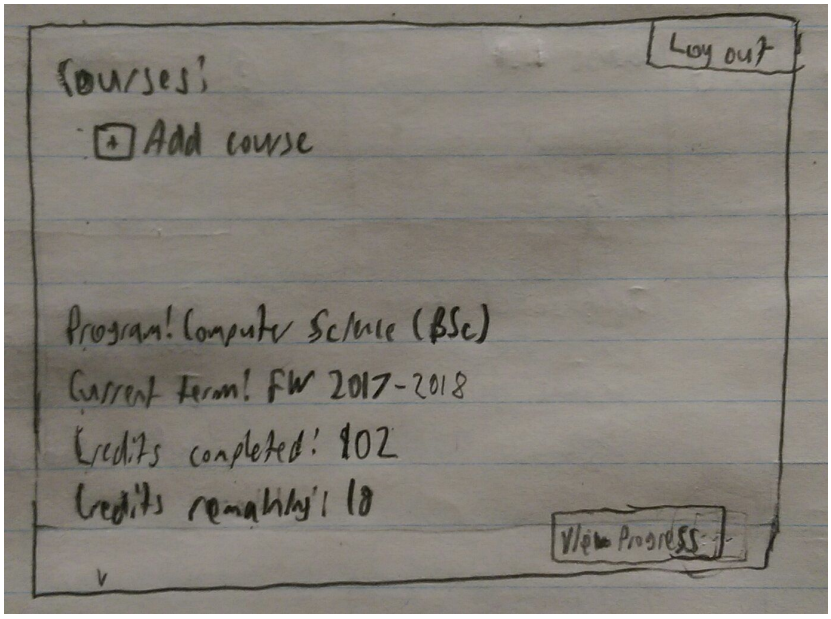

Branch 3: Jeff exchanges a course
Courses:
EECS 4000
Course list:
Select tutorial and/or lab period
Jeff decides to exchange a course
Jeff selects target course from list or inputs its course code
Prerequisites are met, seats are available. Jeff selects tutorial and/or lab period
Courses:
EECS 3761
Jeff successfully exchanges course after confirmation

I. User feedback

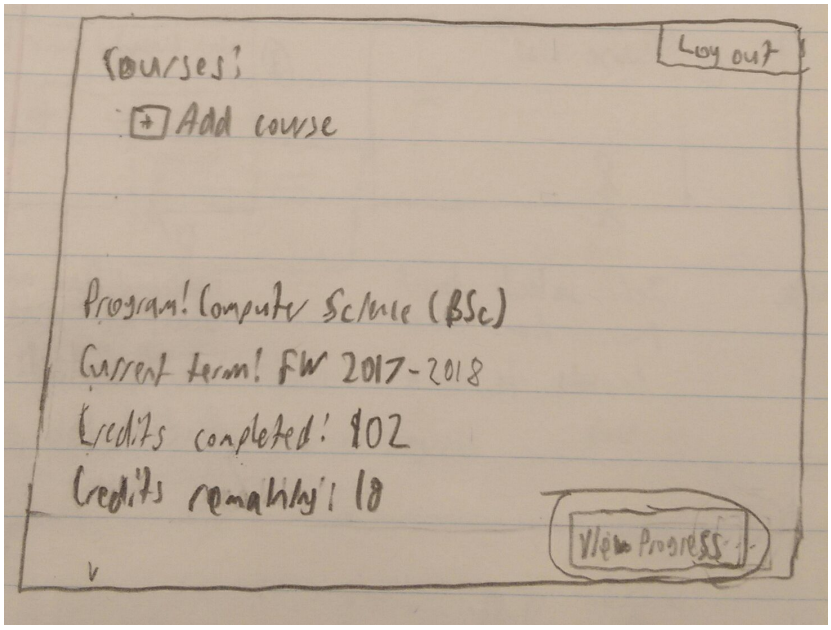
Two potential users were consulted about the storyboard and both stated that it was fine. They were able to follow through the storyboard without any difficulties. Neither of them gave negative criticism about the storyboard.

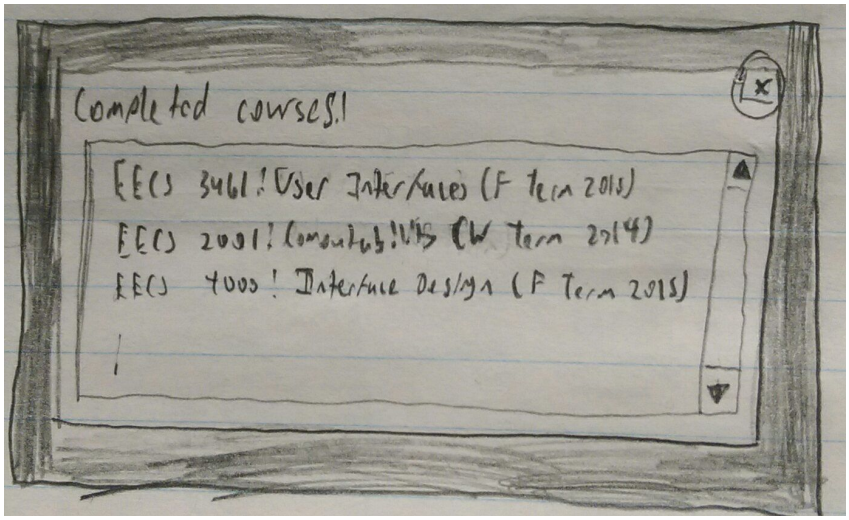
Card-based prototype

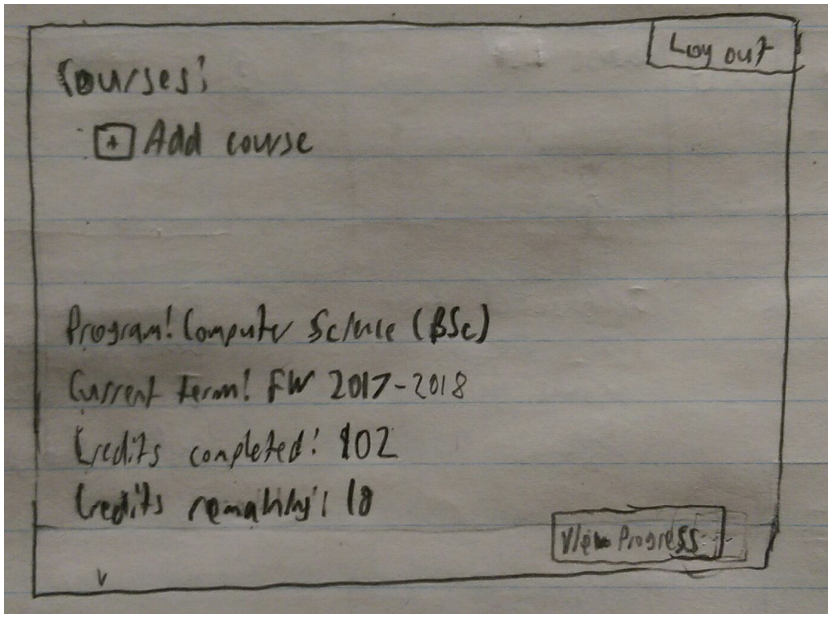




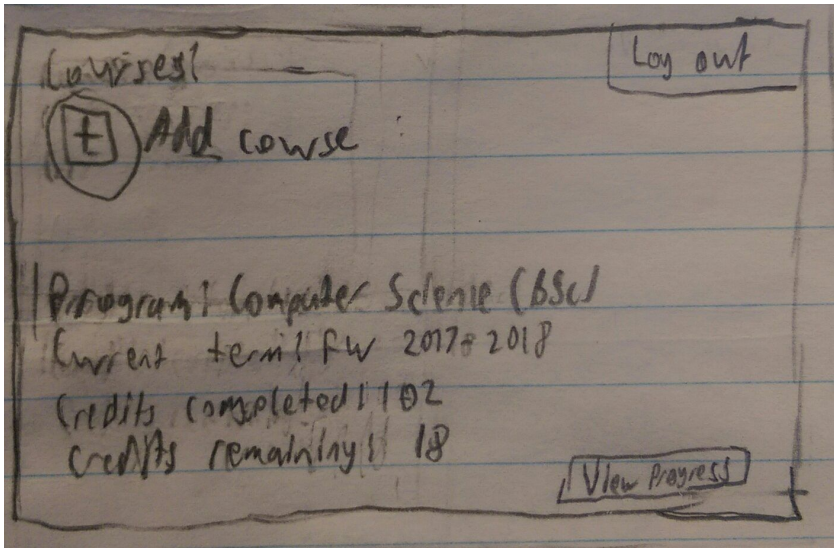
View progress

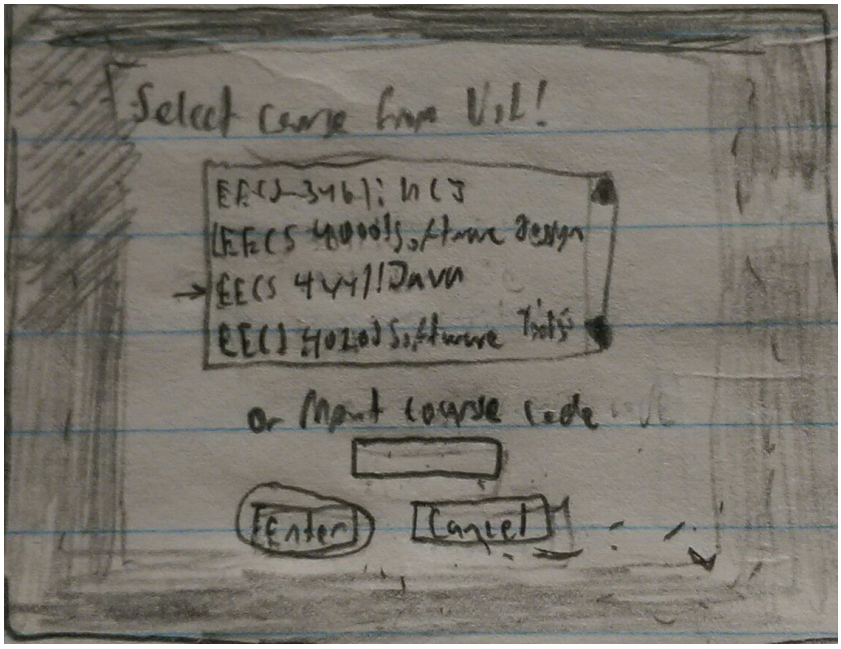


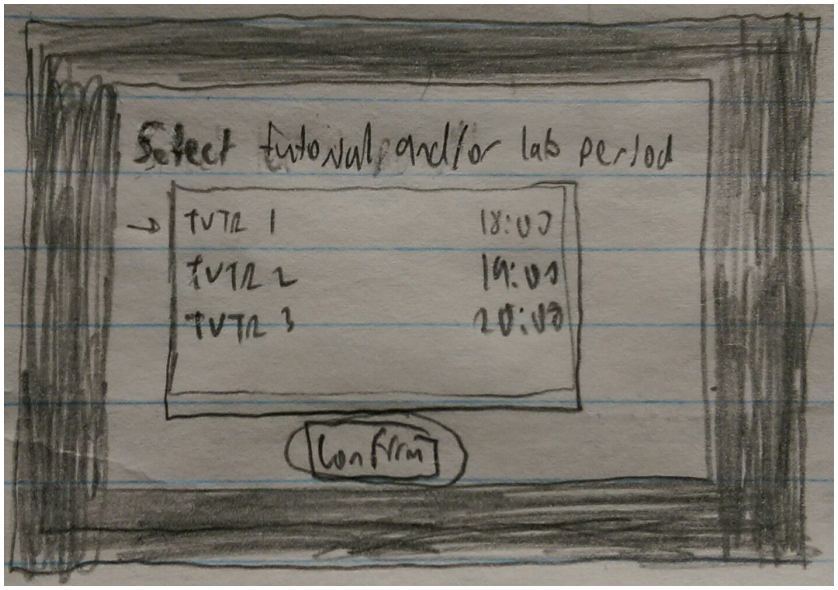


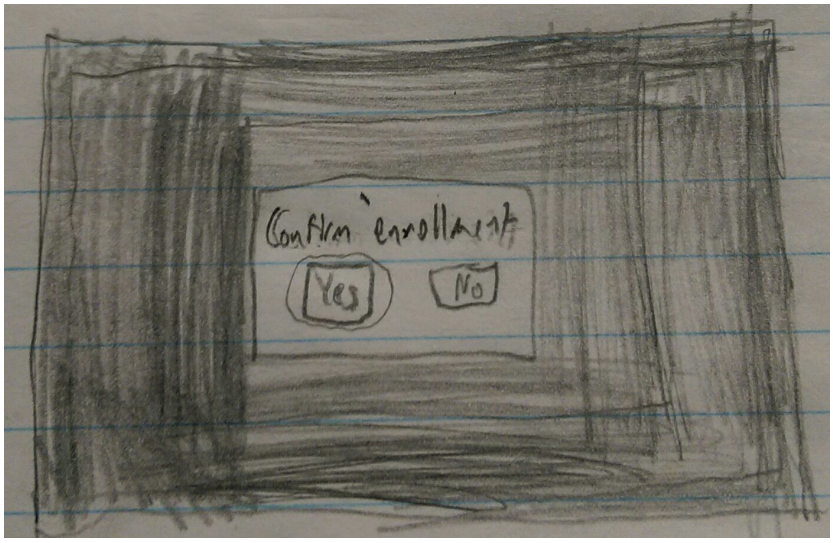

Courses:
Log out
Add course
Program: Computer Science (BSc)
Current term: FW 2017-2018
Credits completed: 102
Credits remaining: 18
View Progress

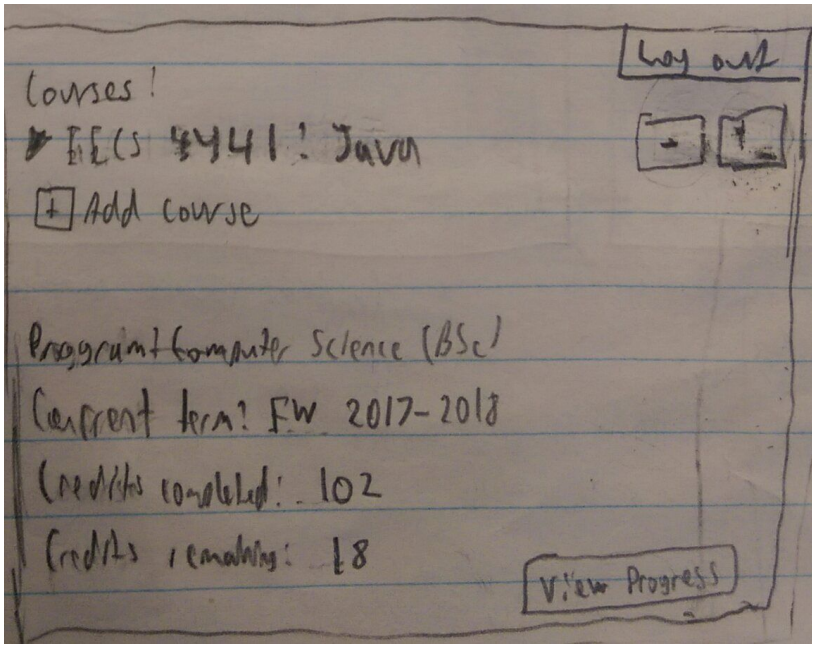
Add course


Courses:
Log out
Add course
Program: Computer Science (BSc)
Current term: FW 2017-2018
Credits completed: 102
Credits remaining: 18
View Progress

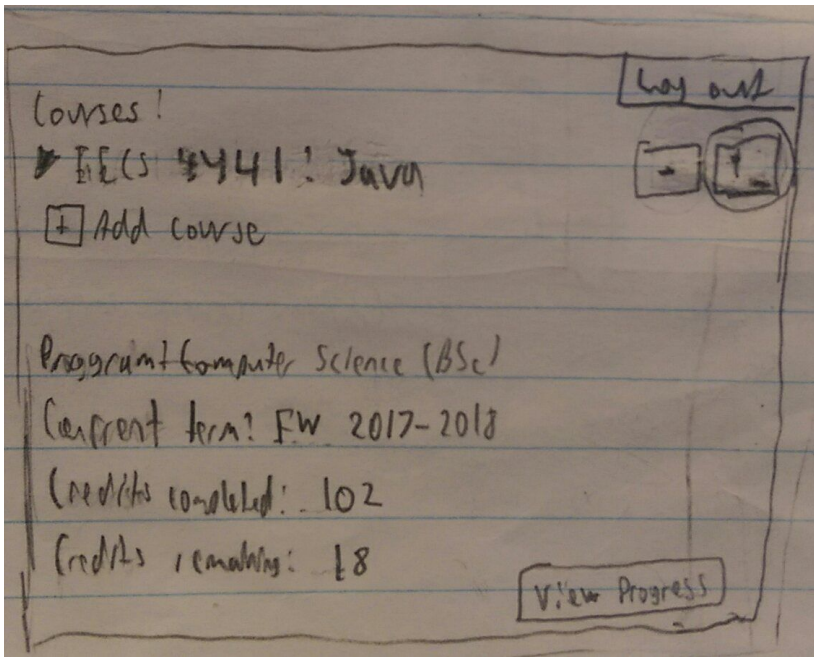

Select course from UofL!
EECS 3461: HCI
EECS 4080: Software design
EECS 4441: Java
EECS 4020: Software Tests
or input course code
Enter
Cancel

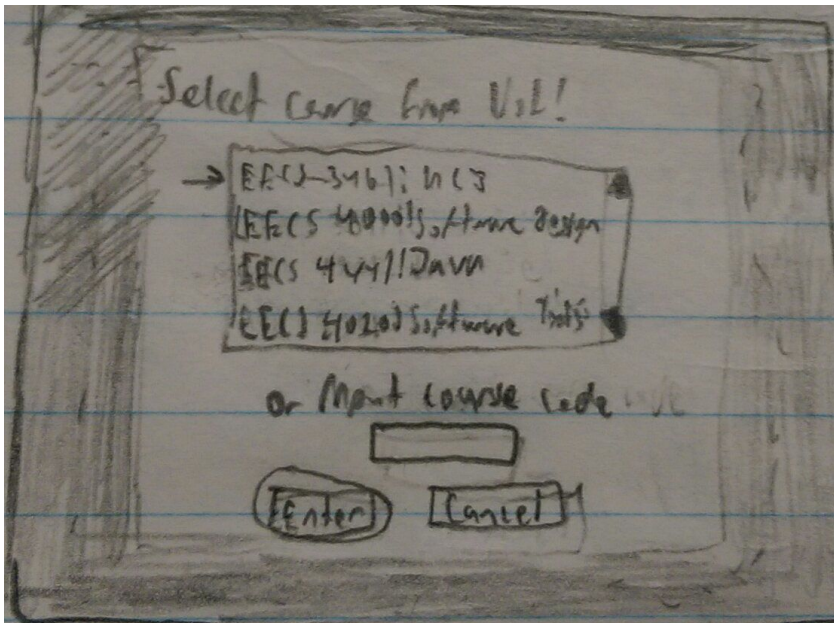

Select tutorial and/or lab period
TUTR 1
TUTR 2
TUTR 3
Confirm

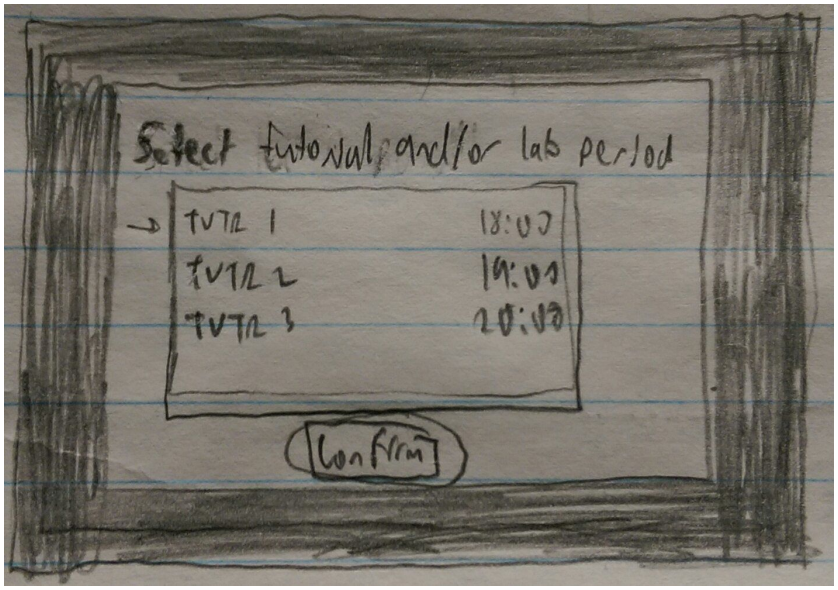

Confirm enrollment
Yes
No

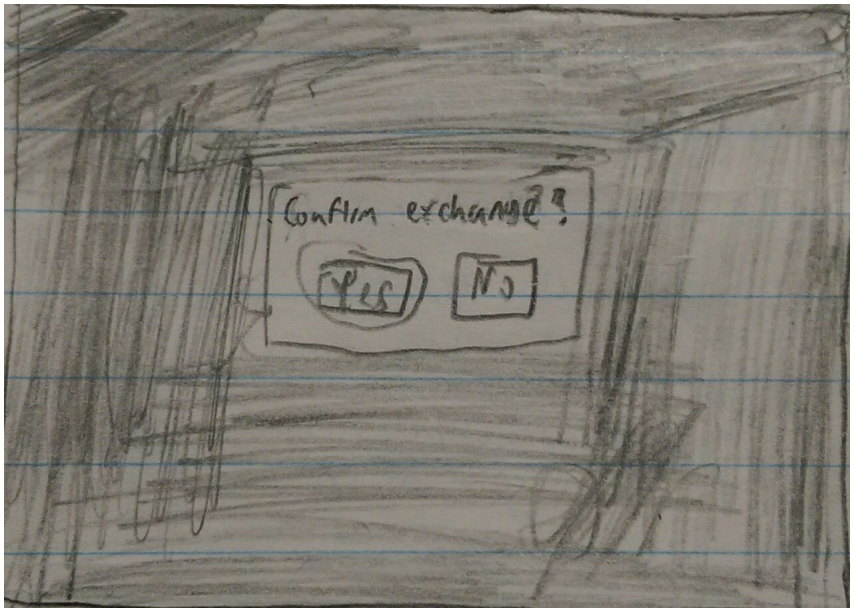

Log out
Courses!
EECS 4441: Java
Add course
Program: Computer Science (BSc)
Current term: FW 2017-2018
Credits completed: 102
Credits remaining: 18
View Progress

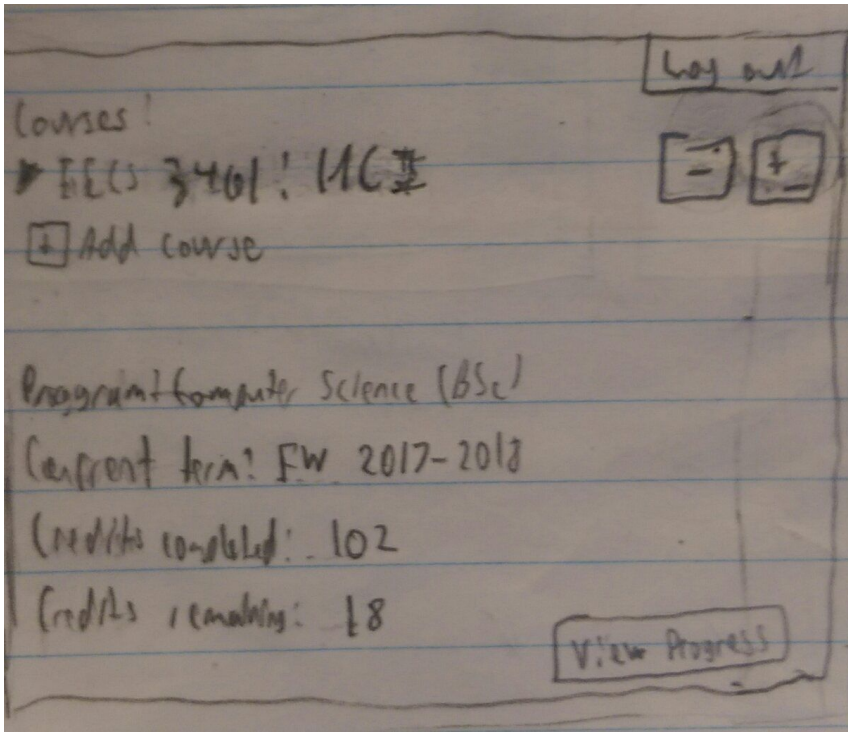
Exchange course



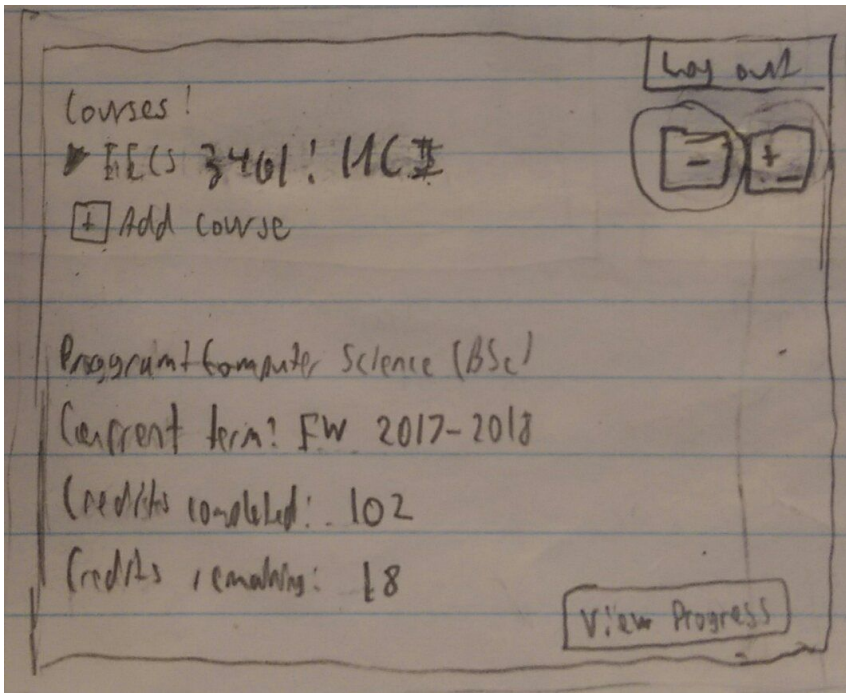


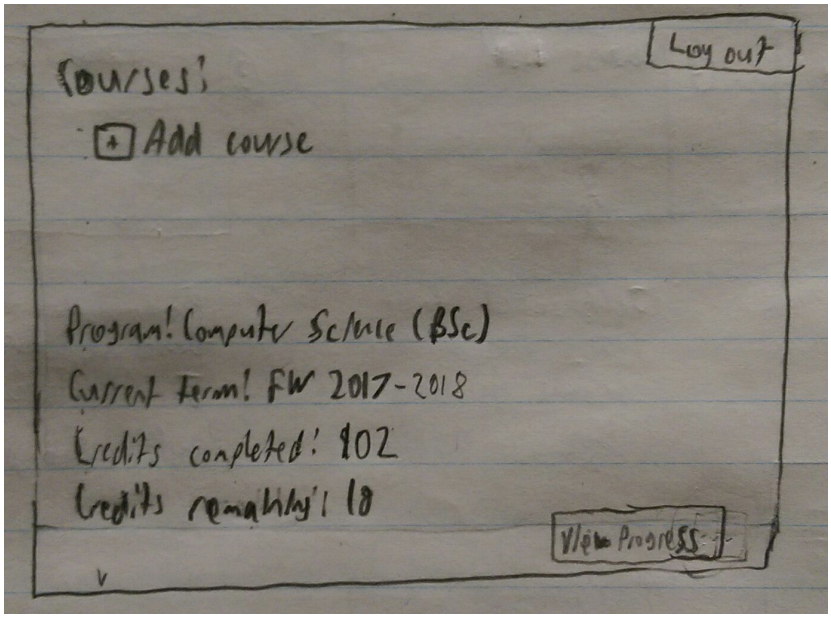

Select tutorial and/or lab period
TUTR 1
TUTR 2
TUTR 3
Confirm


Confirm exchange?
Yes
No


Log out
Courses:
EECS 3401: MC
Add course
Program: Computer Science (BSc)
Current term: FW 2017-2018
Credits completed: 102
Credits remaining: 18
View Progress

Remove course

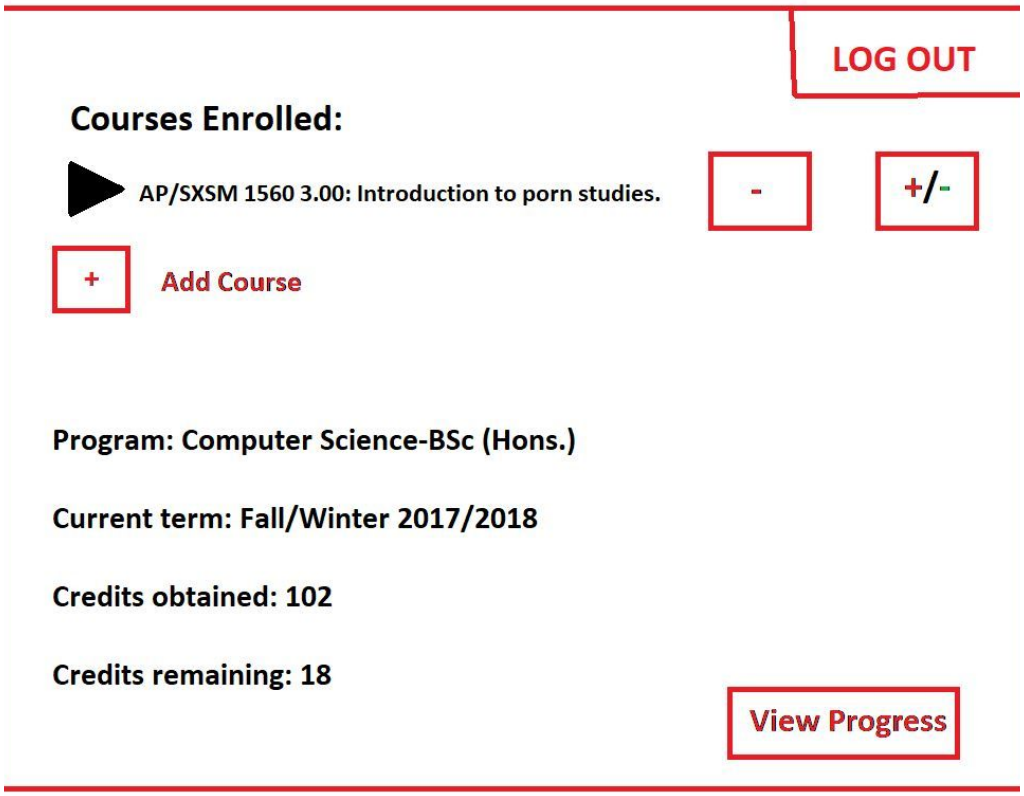




I. User feedback

Two different potential users were consulted about the card-based prototype. Both of them stated that they found the prototype to be amazing. Nevertheless, one of the the potential users inquired about the colour scheme to be used for the application.It was conveyed to the person that based on the inquiry, necessary actions would be taken to ensure that the application looks visually pleasing.

Initial View



The only platform for the product is a .jar file, which is to be used on a computer or laptop. The initial view for the student can vary depending on whether he/she will start managing courses for an upcoming term or if courses were already added, removed, or exchanged. The view above shows the case where a course was added or exchanged as most of the functionality will be accessible at this point. In comparison to the existing REM, many components of this product are simplified by the creation of small buttons which can be intuitively understood. Also, the existing REM lacked minimalism, thus for such an application it was implemented. As a result, due to the spacing, it is easier to locate buttons or information. The text was made much bigger based on a request from a potential user. These improvements lead to an intuitive, enjoyable, and simplified user experience.

User Experience Map

The following user experience map uses the timeline design to depict the three main ways a user can potentially interact with the application as follows:

Experience Map: Jeff uses the new REM to enroll in a course

Jeff logs into the REM using his university credentials.	He's greeted with a new window that offer him 5 clickable buttons He chooses the degree progress report button off to the bottom-right corner to see his progress and what courses he still needs to take.	Jeff presses the add course button, where a new window pops up with a list of courses he is eligible for (based on prerequisites). A textbox will also be provided for entering course code for courses outside of current major.	He adds the course from the list after picking the section and lab that best suits his schedule.	He tries adding a course for which he did not have prerequisites by adding the course code into the textbox. He gets denied.
The REM will use a database to validate his credentials.	Need to access database that will already contain information (courses, credits, faculty).	Need to filter out the courses based on the information provided by the degree progress report.	Should the user be prompted using a popup that states the action was successful?	The user should be provided with a popup notifying about the missing prerequisites.
Need to have a simple login page for the user.	Probably should design the feature so that when a Degree Progress Report button is pressed, the information is brought up in a new window.	Need to make the individual courses in the list interactive so that the user can see the course details before adding it (lab/tutorial).	The database that holds this information should be updated to show a decrease in seats for that particular course.	
	Should the welcome page be plain and minimal?			

Experience Map: Jeff uses the new REM to drop a course

Jeff presses the drop course right next to the course name button to drop a course.	The course gets removed from his schedule and this change is reflected in the degree progress report
The user must be enrolled in at least one course for this action to be valid.	Must update the database to show that a spot has opened for someone else by incrementing the seat counter.
Should there be a prompt that confirms user action? (Safety purposes).	Should there be a prompt that shows the action was successful?

Experience Map: Jeff uses the new REM to exchange courses

Jeff tries to exchange a course by pressing the exchange course button next to the course to be exchanged.	He enters the course code of the course he would like to exchange it with.	After a prerequisite check, Jeff picks the course section that fits his schedule.	The course is swapped and reflected in the schedule.
A popup should appear instructing the user how to exchange courses.	Validation should occur, whether the course is valid or if seats are available. Popup should appear if validation fails.		Must update the database in such a way that the old course has its seat counter incremented and the new course's seat counter decremented

	Should there be a prompt that shows the action was successful?

LEGEND

Green - interaction points
Red - implementation/design ideas
Yellow - implementation questions worth considering

Product variance to Maker Movement

The Maker Movement was an era which involved the rise of many DIY products. This was possible as resources and tools became affordable and accessible to a wide array of people ranging from the poor to the rich. The proposed product differs from applications that might arise from the Maker Movement as it is an improved version of something which already exists. Essentially, nothing is being built from scratch as an existing version of the application is available on the York University website. The Maker Movement therefore, usually supported products that were built from ground up. In addition, the proposed product is something which can be made without use of any special physical resources (such as a laser cutter) and can be made anywhere: from the comfort of one's home to a cafe. Furthermore, most of the products which came from the Maker Movement were combined with physical components whereas this product only manifests itself on a computer device. This is possible due to the existence of software development kits, which provide developers with the virtual tools to unleash their creativity. Software development kits (SDKs) do indeed have a large role in the making of the proposed product as they provide the necessary tools and functionality so that the end product is achievable. For instance, displaying dialog boxes and information on transitioning pages is only possible with the use of the correct SDK. For this to happen, access to a "library" and an IDE (such as Eclipse) is required.