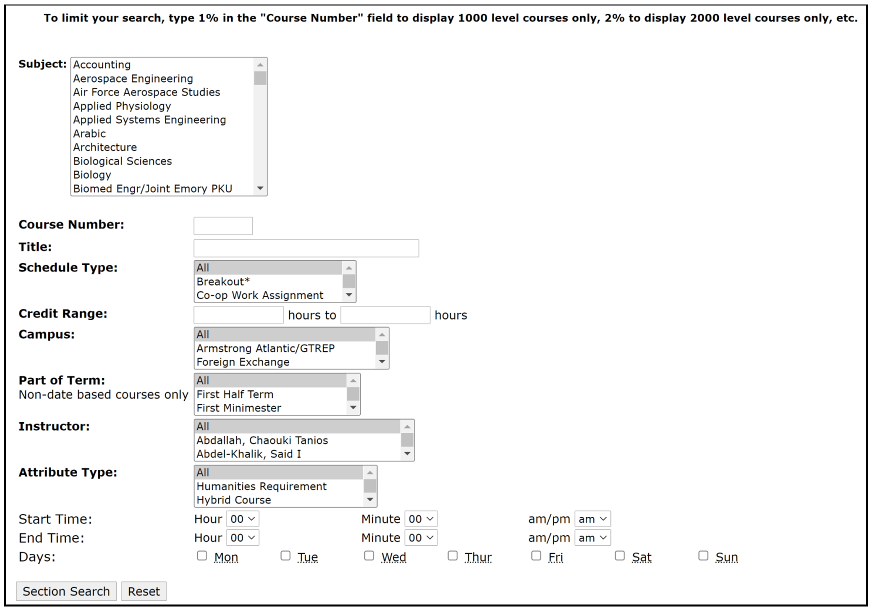
Assignment P1, Summer 2021

Ryan Hodge
rhodge32@gatech.edu

1 QUESTION ONE

In which I examine the *course search interface* within the "Look-Up Classes" portion within Georgia Tech's OSCAR registration system (Figure 1).



To limit your search, type 1% in the "Course Number" field to display 1000 level courses only, 2% to display 2000 level courses only, etc.

Subject: Accounting, Aerospace Engineering, Air Force Aerospace Studies, Applied Physiology, Applied Systems Engineering, Arabic, Architecture, Biological Sciences, Biology, Biomed Engr/Joint Emory PKU

Course Number:

Title:

Schedule Type: All, Breakout*, Co-op Work Assignment

Credit Range: hours to hours

Campus: All, Armstrong Atlantic/GTREP, Foreign Exchange

Part of Term: Non-date based courses only: All, First Half Term, First Minimester

Instructor: All, Abdallah, Chaouki Tanios, Abdel-Khalik, Said I

Attribute Type: All, Humanities Requirement, Hybrid Course

Start Time: Hour 00 Minute 00 am/pm am

End Time: Hour 00 Minute 00 am/pm am

Days: ☐ Mon ☐ Tue ☐ Wed ☐ Thur ☐ Fri ☐ Sat ☐ Sun

Section Search | Reset

Figure 1- *"Advanced Search" within OSCAR Look-Up Classes*

In general, the user's goal when interacting with the "Look-Up Classes" search is to learn what courses and course sections are available in a given period of time, and which ones are viable for the user, based upon user-specific constraints (e.g. conflicting commitments, academic interests, degree program requirements, etc.).

1.1 Processor model

If we consider this interface with a *processor model* of the user, the user's role is primarily to provide inputs such that a full list of courses can be filtered to relevant results. In order to maximize efficiency of the interface, we could spend time examining a metric such as *average time elapsed before submitting a search*. The action of resetting the search could be indicative that the previous search was not sufficiently informative, so we might also try to reduce *search resets per user*.

Using a small number of screens, the interface enables the user to provide input for any/all available filters on the dataset, which meets the processor model's definition of usability.

1.2 Predictor model

The *predictor model* would encourage us to survey students about their reactions to the search interface. The primary question is whether the experience of using the interface aligns with how the user thinks about searching for classes. For instance, we could seek to understand whether users are able to translate their own time and academic constraints into the terminology and format used by our filters. We could ask why individual users choose the basic or advanced search flows, and how their goals differ in those situations. After a search, do the results serve as a satisfactory answer to the constraints they intended to enter? And if not, why?

For our interface to be successful from a predictor standpoint, we want to be confident that our layout properly corresponds to the way students think about finding courses.

1.3 Model comparison

The processor model naturally recommends efficiency improvements. One efficiency problem with the current interface is that the default, basic search flow will only show section-level results for a single class at a time. This means the user will have to search a number of times in order to view results for multiple classes.

To speed up search submissions, the processor model might suggest eliminating the basic search flow altogether, and move all user input (term, subject and filters) to a single screen. The most-used filters, based on usage logs, could be prioritized toward the top of the screen. The most used filter inputs could be suggested earlier in the input list. An extra submit button could be included at the top of the search screen to reduce scrolling.

On the other hand, the predictor model would focus on user understanding. It might be more straightforward to remove "online" as a *campus* selection and instead use a radio button or checkbox. Some users will not be aware or have physical difficulty pressing control as they click to choose multiple options for a filter. Because of this, designing a multi-select which uses only the mouse would be preferred.

The search results do not display the active filters, which might cause users to doubt they included the correct criteria. Displaying the current search input somewhere on the results page might give users more confidence they've achieved their goal.

2 QUESTION TWO

In which I discuss a **music streaming app** as an application used in several distinct contexts.

2.1 Relevant contexts

2.1.1 *Home listening*

In a home setting, there are typically no constraints to listening. Volume doesn't need to be monitored (ignoring that we may disturb the neighbors). It's common in this environment to play music through an external system.

Alterations—In this setting, the application could attempt to integrate seamlessly with other systems. For instance, upon arriving at home, the app might automatically play music over an external sound system, or at least actively prompt the user with this option.

At the same time, a smart TV in the home (if not already in use) could provide a visual of what song is playing and what songs are up next. However, if the app (or connected system) detects voices talking while the music is comparatively loud, it might temporarily turn down the music to not drown them out.

2.1.2 *Public transit*

It's generally considered rude to listen to music out loud on public transit, and rather listening be done with headphones. Depending on how busy the bus or train is, one might have limited physical room to retrieve a device from a bag or pocket in order to control the music player.

Alterations—A context-aware design could play similar music when a playlist or album ends, to limit the need for the user to directly control the device in a crowded environment. Additional prompting before playing if headphones aren't detected could help the user to avoid an awkward social situation.

2.1.3 ***While driving***

While driving, some functions of a music player are unsafe or infeasible to use. It's obviously dangerous to take attention from the road or hands from the wheel to type an artist or album name. Music playing loudly has the potential to cover up other sounds around the vehicle, such as emergency sirens or honking.

Alterations—Similar to our "home" context, integrating with external systems, like the car's onboard media system, would be helpful here. That way, the music could be managed via physical controls- for instance, on the steering wheel- that demand less attention. If no media system is available, the music app interface could use a more simplistic UI allowing a swipe anywhere on the screen to change songs, so a driver could ideally perform the action without looking.

In this mode the application could also detect important sounds like sirens, and turn down the music if above a certain threshold to ensure the driver is aware.

2.1.4 ***Background during other tasks***

Listening to music is not always (and arguably usually not) the listener's primary task. If the user is trying to focus on a task requiring significant attention, certain types of music or frequent changes in volume level might be distracting.

Alterations—When the user is in predetermined locations- for instance a workplace or classroom- the app might adjust to only play types of music the user has (a) designated for this purpose manually or (b) listened to in this setting before. Normalizing volume and crossfading between tracks would help prevent jarring transitions. The interface could display UI elements to allow the user to give feedback on whether particular songs were a good or bad choice.

3 QUESTION THREE

3.1 Gulf of execution in Canvas

3.1.1 *Identify intentions*

Within the context of the system, the goal of "submitting an assignment" means the student must *upload* the document through the Canvas web application, to associate it with the assignment in question. Assuming that the student has experience working with files on websites, this is fairly easy to identify.

3.1.2 *Identify actions*

The interface actions that need to be identified are

- Navigating to the assignments area of Canvas
- Choosing the relevant assignment
- Choose "Start Assignment"
- Click "Choose File" and submit the file via the user's file explorer
- Check the End-User License Agreement (EULA) checkbox
- Click "Submit Assignment"

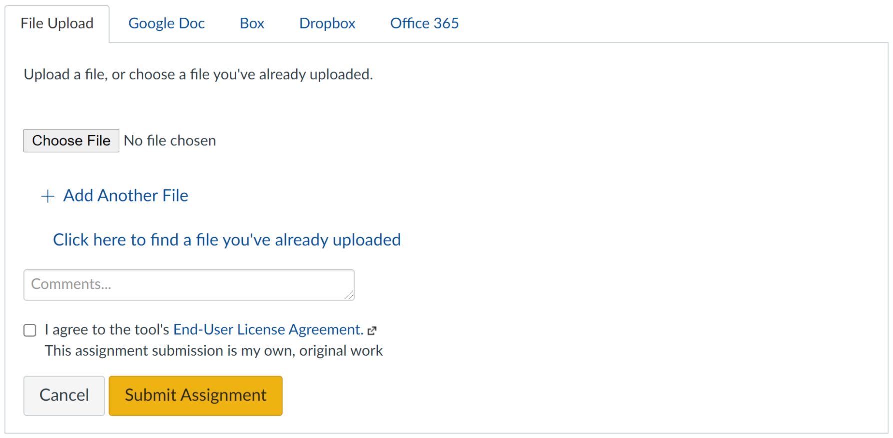
The correct page within Canvas is labelled *Assignments*, which is straightforward to identify. The assignments listed on this page are titled by the instructor, so the names should be familiar to the user. If the user clicks *Assignments*, they are shown a list of assignments they can pick from.

However, "Start Assignment" as a label does not necessarily match the user's expectations. "Start" implies that no work has been done on the assignment yet, but in our case the student is trying to submit a finished assignment. The lack of other options on this screen may be enough to encourage the student to click, though.

3.1.3 *Execute in interface*

Beginning from the course homepage, the student can find navigation options in the sidebar menu of Canvas. On a larger screen, this is immediately visible, but could be collapsed on a smaller screen, making it harder to find. As mentioned earlier, *Assignments* is the option they should choose and appears toward the top of the list. However, *Files* and *Grades* are other menu options that might sound correct to a student.

Clicking *Files* is a dead end, because in this context, *Files* means files that have been provided by the instructor of the course, not files the user uploaded. The student would have to notice this and try a different page. However, the *Grades* page includes links to each assignment, which will take the user to the right place.



Upload a file, or choose a file you've already uploaded.

I agree to the tool's End-User License Agreement.
This assignment submission is my own, original work

Figure 2- *The File Upload area from the a Canvas assignment page*

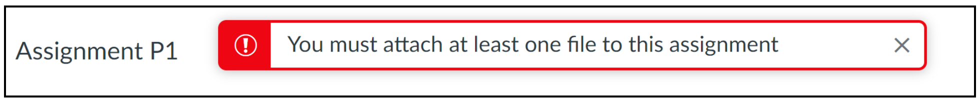
Once they've chosen the right assignment, the assignment screen prominently displays a file upload area (shown in figure 2). The text given instructs the user to upload a file by choosing it. The submit button is large and contrasts from other controls on the page.

3.2 Gulf of evaluation in Canvas

3.2.1 *Interface output*

If the user doesn't check the EULA agreement checkbox, a dialog appears reminding the user to agree.

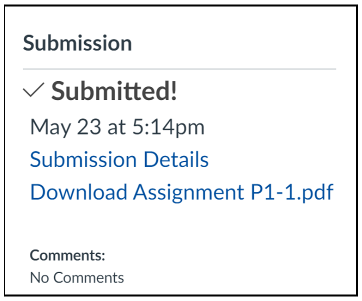
If the user doesn't attach at least one file, an error message is shown (figure 3).



You must attach at least one file to this assignment

Figure 3- *error message if no file is attached*

Otherwise, on success, a message with the text "Submitted!" appears on the right hand side of the screen (figure 4).


Submission

✓ Submitted!

May 23 at 5:14pm

Submission Details

Download Assignment P1-1.pdf

Comments:

No Comments

Figure 4- *Success state after assignment upload*

3.2.2 *Interpretation*

The forgotten file state and the submitted state are easy to interpret because the text they display to the user is clear about what happened. The user sees the name of the file they uploaded under the success message, as well.

Unfortunately, the forgotten EULA state uses the terminology "submission pledge" which does not match the label on the checkbox itself. That could be potentially confusing feedback.

3.2.3 *Evaluation*

Evaluation is simple for this interface.

Because the non-success states show error messages, it's easy to evaluate that an assignment was not uploaded in these cases.

The checkmark and "Submitted!" label, along with the name of the uploaded file, give the user confidence that they were successful.

4 QUESTION FOUR

The activities I'll discuss here are using my oven and using my dishwasher.

4.1 Wide gulf of evaluation: preheating my oven

In the house I'm renting, the oven is quite old- likely from the late 1970s or 1980s. A goal I often have is to preheat the oven to a specific temperature, before putting food into the oven. To do this, I turn a knob to specify the temperature and then flick a switch to "bake," which turns the oven on.

The gulf of evaluation for whether I've successfully done this task is quite wide, because there is no visual indication (like a temperature reading) that the oven has finished preheating. An orange light does turn on and off after setting the oven to bake, but because it goes back and forth, the most I can understand is that this is tied to the heat cycling on and off. The only way I can evaluate that the oven reached the right heat is to use my own thermometer, or see if the food I'm baking takes the expected time to cook, or longer.

4.2 Small gulf of evaluation: checking my dishwasher's progress

The dishwasher in my kitchen is much newer, probably replaced in the last couple of years. It's a whirlpool model with a small, LED screen at the top of the door. After loading the dishwasher and beginning a wash, the gulf of evaluating the current status of the dishes is very narrow. Bending down and looking at the LED screen will report a simple status, such as "WASH," "DRYING," or when finished, "DONE." Even if I can't hear the dishwasher over other noise, that status message is very clear.

4.3 Activity comparison

While I don't have much hope for updating my 1980s oven, a clear difference I see is that there is no **dedicated status indicator** on the oven, like there is on the dishwasher. The orange light is very ambiguous, and leaves a lot for the user to guess.

An LED status screen like my dishwasher would obviously be better, but a more era-appropriate solution could be having two lights. To shrink the gulf of evaluation for preheating, one could show whether the oven has reached the set temperature yet, and the other could serve the same purpose as the existing light-display whether the heating cycle is on or off. Alternatively, a built-in thermometer could be a nice addition to keep track of preheating progress.