Project 1: EasyA (or JustPass)
CS 422 Software Methodologies
Prof. Anthony Hornof - 1-9-2024

0. Deadlines

Initial submission: Due at on Wednesday, January 17, at 8PM. This will be submitted as a single PDF on Canvas. This initial submission will include everything discussed under "Initial Project Plan / SRS / SDS" at https://classes.cs.uoregon.edu/24W/cs422/Project_Evaluation.html except that you will not submit an SRS unless you substantially modify this project description.

Final submission: Due on Wednesday, February 5, at 8PM. It will be submitted as a zip file on Canvas. It will include everything else discussed at:
https://classes.cs.uoregon.edu/24W/cs422/Project_Evaluation.html

Presentations: In class on Monday, February 6. Eight minutes per group.
Please see: https://classes.cs.uoregon.edu/24W/cs422/How_to_Present.html

1. Concept of Operations (ConOps)

The goal of the project is to build a system called *EasyA* that students can use to figure out which professors in which classes are giving the most *A*s, and which professors are giving the fewest *D*s or *F*s. The system could help students to an "easy *A*" or make sure that they "just pass". The system will use grade data from 2013-2016 at the University of Oregon. The data are provided by the *Daily Emerald*.

Current System
The goal is to improve up on the current system at https://emeraldmediagroup.github.io/grade-data/ by providing better side-by-side comparisons of different faculty teaching the same class, and different faculty across different class levels (such as 200-level classes versus 400-level classes). The system you build could be either a standalone system that produces PDF or screen output, or an online system such as the current system.

Justification of a New System
The current system does not adequately show views of the data that would be most useful to students. For example, Figure 1 shows a side-by-side comparisons, within a single graph, of the percent of *A*s assigned by different faculty teaching the same class across

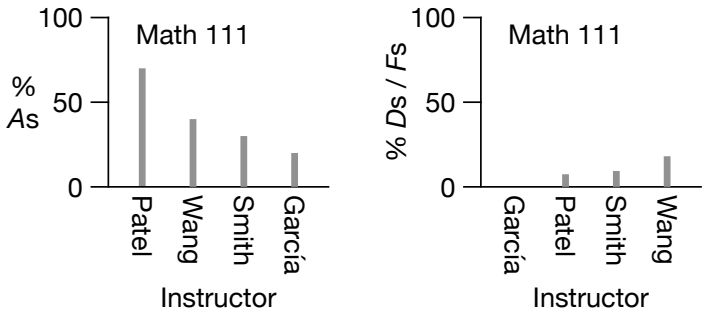

Figure 1. Side-by-side comparison of different instructors teaching the same class. The left panel shows the percent of *A*s assigned by the instructor, and the right panel the percent of *D*s and *F*s.

multiple years, and the percent of *D*s and *F*s assigned by the same instructors for the same classes. This is a view of the data that would be more useful to students.

User Classes

The two main users classes here include:

(1) A student using the system to select a class or instructor.

(2) A system administrator installing the system, or updating it with new data.

Operational Scenarios (Use Cases)

There are two main use-cases, described here very briefly:

(1) A student needs to take Math 111 this term, and there are many different faculty teaching the class. The student wants to know which instructors are most likely to give *A*s, and which are most likely to give *D*s and *F*s. The student uses the system to compare the grading history of the instructors teaching this term, side-by-side.

(2) A system administrator acquires new data for the system, and updates the system with the new data.

2. Functional Requirements

Views of the Data

The user should be able to view data in the following manner:

(a) Within a single graph, broken out by instructor for:

(1) A single class (such as "Math 111").

(2) A single department (such as "Math").

(3) All classes of a particular level within a department (such as all "Math 100-level" classes, from 100-level classes through 600-level classes).

Figure 2 shows examples of such graphs.

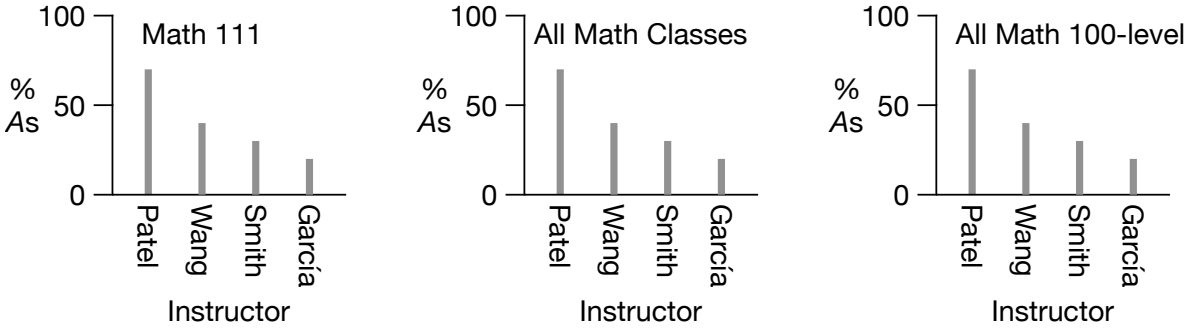

Figure 2. Within a single graph, instructor data for a single class, a single department, or a single level within a department.

(b) All classes of a particular level within a department. This would help students, for example to choose which electives to take. Figure 3 shows an example.

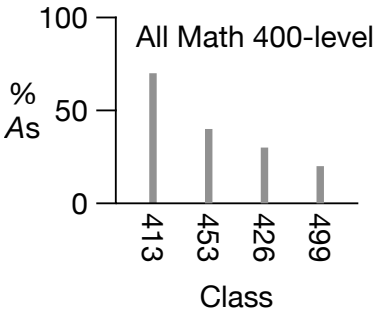

Figure 3. Side-by-side comparison of different classes at the same level.

(c) "All Instructors" versus "Regular Faculty"
For all graphs, there should be two options as follows:

(1) "All instructors" (the default)
(2) "Regular Faculty", which are the faculty listed within departments in the 2014-2015 UO Course Catalog, the listings of which are available here: https://web.archive.org/web/20140901091007/http://catalog.uoregon.edu/arts_sciences/
Open a department name in the list on the right, and click on the "Faculty" Button.

(d) "Easy As" versus "Just Pass"
For all graphs, there should be two options, as follows. The selection should be indicated on the *Y*-axis label:

(1) "Percent As" or "% As" (the default)
(2) "Percent Ds or Fs" or "% Ds / Fs"

(e) An option to show the class count
Ideally, your system will have an option to show the number of classes represented in each bar in the bar graph. Figure 4 shows an example.

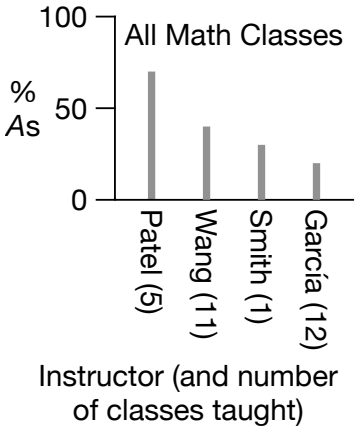

Figure 4. A class-count has been added to the *X*-axis labels.

Side-by-Side Viewing
The system should support some sort of side-by-side viewing of graphs, such as to show Figure 3 (for Math 400-level classes) alongside a separate, similar graph for all Computer Science 400-level classes.

Graph Formatting
Graphs should be easy-to-use and easy-to-read such as those shown in Figures 1 through 4. For example, bars in bar graphs should be narrow, clearly visible, easy to compare, and ordered to support the user's task, which will usually be from highest to lowest, or lowest to highest.

Fully-Populated Initial System
The system that is delivered should not require any administrator steps to convert or prepare data for the system other than (optionally) setting up a database (if your system uses a database). But, for example, there should not be any web scraping or data conversion required for the initial use of the system.

The Administrator Adding New Data
With regards of the use-case of a system administrator updating the system with that new data:

1. You should provide a program that permits an administrator (such as the instructor) to replace all of the data in the system (all grade data, and all instructor data) using the "gradedata.js" file that was provided to you (or a .csv or other single file that you created from the gradedata.js file), and by scraping the instructor names from the "way back machine". Replacing the data should be quick and easy (such as, able to be accomplished by an administrator unfamiliar with the system in less than five minutes).
2. The data file (that the system administrator will use to load the new grade data) can be in a file format of your choice, such as the original "gradedata.js" file that is provided to you for this project, or a CSV file. If it is a format of your choice, you would ideally provide an appropriately-formatted file that contains all of the data in "gradedata.js".
3. A "scraper" for extracting the faculty names from the "way back machine" data at https://web.archive.org/web/20140901091007/http://catalog.uoregon.edu/arts_sciences/ would ideally be provided. You can use the python library BeautifulSoup to help with this.
4. When replacing the system data, ideally, discrepancies between names found in "gradedata.js" and the names found in the scraped instructor data would be easy to resolve using the administrator tools, to ensure that the data in your tables is clean, consistent and accurate. (Optionally, as the two sources of data are brought into alignment, the tools could generate statistics such as lists of names from both data sources that have yet to find a match, so you can see how your data resolving process needs to be further improved.)
5. Any software tools that you create for the administrator to load data can be separate standalone applications, not integrated with the main program used by students. These can be command-line tools (without a graphical user interface). Instructions for their use should of course be provided. (Optional: Some sort of statistics could be provided when loading the new data, such as how many classes or instructors were added, and any naming discrepancies that were resolved.)
6. The new data should overwrite the old data.

3. Non-Functional Requirements

Natural Sciences Only
Your systems only needs to include departments that are in the Natural Sciences, which are listed here: https://casstudent.uoregon.edu/departments-and-programs/natural-sciences/. If you can gather and provide data beyond that, that would be great, but not necessary.

State the Source of the Data
The system should clearly and prominently state the source and nature of the data. This would include, at a minimum:

(1) That the data were copied directly from https://emeraldmediagroup.github.io/grade-data/ (and the date the data were copied).

(2) That the data are limited as stated on that website. The exact text should be included, starting with “If your class doesn't show up here, it means the data was redacted….”. This explanation should be in quotes and should include an appropriate reference to this web page, including the date that the data were copied.
(3) The years that are included in the data.

Grade Data
The Daily Emerald article presenting the grade data is here:
https://www.dailyemerald.com/news/academics/check-out-the-emerald-grade-tracker-to-see-what-grades-uo-professors-give-out/article_6794975c-7fad-5550-b2be-e32a496b3cab.html

A file containing the grade data is here:
https://emeraldmediagroup.github.io/grade-data/gradedata.js

4. Technical Requirements

Standalone or UO-CS-Web-Based
The system can be standalone, with all data on a local hard drive, or it can be a web-based system. If it is a web-based system, must provide a thorough set of instructions, and all files necessary, to create a mysql server on ix.cs.uoregon.edu, and access the system on ix. Installation must be as straightforward as the instructions at: https://systems.cs.uoregon.edu/wiki/index.php?n=Help.ToolsMysql

No Internet Access Required to Install or Run
The system should not require Internet access to be installed, or to run. The one exception is that instructions may be provided to install, run, and access software running on a UO CS server. Do not submit a URL of an installed system for your project. Your system, installation, and your code may not access any pre-installed server.

Data Storage
Data should be stored either local, or on ix.cs.uoregon.edu using either mysql or mongo.

No Login. There should be no login required for a student to use the software.

Build-Related Constraints

Target Platform. The software should run on a laptop or desktop machine. It can optionally be accessed on a web browser on smartphone.

System Document File Formats

All system-related and system-development-related documents that are intended for human reading must be in either plain text or PDF. For example, Microsoft Word, Microsoft Excel, or markdown language documents must be converted into plain text or PDF.

Programming Constraints

- The system may be built using Python 3 along with The Python Standard Library https://docs.python.org/3/library/index.html, but no other imports except for:
  mysql, pymongo, and matplotlib
  This means that the only GUI package that can be used is tkinter.
- Python code must run in Python 3.10 through 3.12.
- The system may be built using Java and Java Standard Edition modules, no other imports.
- The system may be built in C/C++, the C++ standard library, Cocoa, and no other components.
- Java code must run in Java 19 or 20.
- Instructions must be provided for how to compile the code.
- No server connections may be required for either installing or running the software, except for setting up mysql on ix, and you must provide instructions on how to set this up. The instructions should be as the MongoDB and MySQL guidance at https://systems.cs.uoregon.edu/wiki/index.php?n=Help.Tools
  (along with whatever files are need to create your tables and such)
- No virtual environments may be required to run your projects.
- No gaming engines such as Unity may be required to run your projects.

Installation

- There can be at most 20 user actions to compile the code and run the program.
- An experienced computer programmer should not require more than 30 minutes working alone with the submitted materials to compile and run the code.