Curriculum Competency Tree Software

A Capstone Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Daniel Prohaska

with

May 5, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Daniel Prohaska

Capstone advisor: Floryan M. Advisor, Department of Computer Science

Curriculum Competency Tree Software

How can students and educators track understanding of material in a course? Students rely on grades for a variety of reasons, including to qualify for scholarships, to fulfill course prerequisites, and for opportunities in graduate school and career employment. In conventional teaching methods, teachers assign values to assignments and calculate a weighted average for a final grade. Averaging performance produces an amalgamation that can be hard to understand and unbeneficial. Development of a tool to track student progress through a course and use of standards based grading will improve student performance. The tool provides feedback based on student performance and assists with meeting personal goals.

Previous Work

Several students have worked on this competency tree software. My task was to take their product and ensure it was ready for deployment in a course. Systems like displaying the tree and many of the professor operations (such as manually uploading rosters and grades) were complete. I familiarized myself with much of the codebase and went to work on improving it. There was an enormous amount of code with various quirks and issues scattered throughout. I started by creating a Kanban board and adding many of the issues I saw along the way.

I was taking CS 4710: Artificial Intelligence with Professor Floryan who was implementing the standards based grading system the software was designed to model. This allowed me to apply my student perspective, as well as those of my peers, to get a pulse on how the system was received. This opportunity revealed valuable information and helped guide my efforts.

Bug Squashing and Maintenance

The primary purpose of the software is to make it easy for a student to visualize their progress and understanding in a course. The idea is to be able to see all the nodes in the tree and know whether you had reached satisfactory or mastered criteria. Nodes with their prerequisites not satisfied should appear as locked. Feedback I received when talking to a student was that understanding that they would not receive credit for completing an assignment without completing the prerequisites was difficult. Had this software been used, it would have been more obvious that the student should have focused on the prerequisite material beforehand. In looking into this crucial feature, I found and corrected issues where students could not see the entirety of the tree, multiple prerequisites were improperly locking a node, and grading issues were miscalculating a student's mastery of a node. To better align with standards based grading, all references to percentages or a normal 0-100 point scale were eliminated.

In my attempt to model how the Artificial Intelligence course would work within this system, I found that weighting of assignments and the option for a professor to choose how an assignment was deemed competency/mastered needed to be more fine grain. Assignments in the course differed in the number of standards to meet and did not always conform to a one size fits all scale.

I also found a significant amount of code bloat with various files having unused variables and large segments of unused code. Even within the database models there were unused fields that were confusing. The naming schemes that come with multiple developers over several cycles with little communication between them were similarly chaotic. This slowed my progress and will undoubtedly slow the progress of future students as the technical debt of messy code

continues to increase. While refactoring large segments did not fall within the scope of my work, I marked many of these issues with TODO tags so they could be addressed in the future.

Gradescope Integration

The idea of the system allows students to submit assignments at any point throughout the semester. Rather than put the burden on the professor to continually update grades, some sort of outside integration was needed. The UVA Computer Science department uses Collab as well as Gradescope in many of it's courses to house grades, but neither of which have good APIs. I was able to find a public HTML scraper API for Gradescope that allows for easy access to rosters and grades. With this I devised a system where a "bot" account would be created on behalf of the software and be added to a course on Gradescope as a teaching assistant or instructor (from my understanding of Gradescope, professor accounts have more permissions, like adjusting the roster, which are not currently required for the bot to function). This allows us direct access to any course that adds our bot. With this great power comes a great responsibility. It is vital that we protect the credentials that access the bot. Obviously they can't be stored in our public repository so I turned to Docker environment variables (figure 1). They can be passed into a container using an environment file that is not source controlled. With the utilization of Docker environment variables, I also simplified some complex deployment vs. development logic into a few easy to find variables.

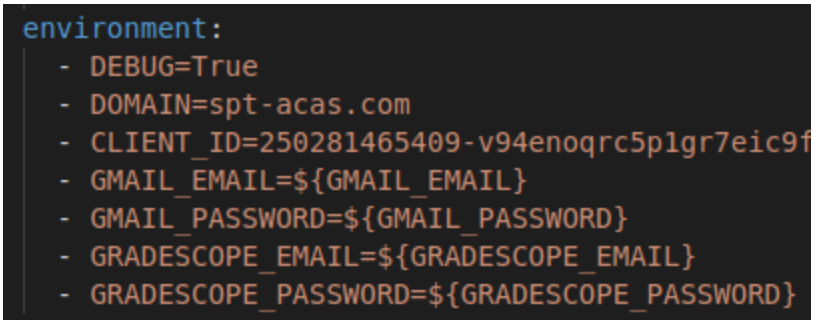

Figure 1: Docker environment variables

Goal Setting

With this new system of grading comes a complex grading scale of various nodes being satisfied and mastered. Previously a student would need to find this information in something like a syllabus, but I created a visualization that shows a student the course policy and their current grade (figure 2). This view shows a matrix of all possible grades and what a student needs to accomplish to get the next highest grade.

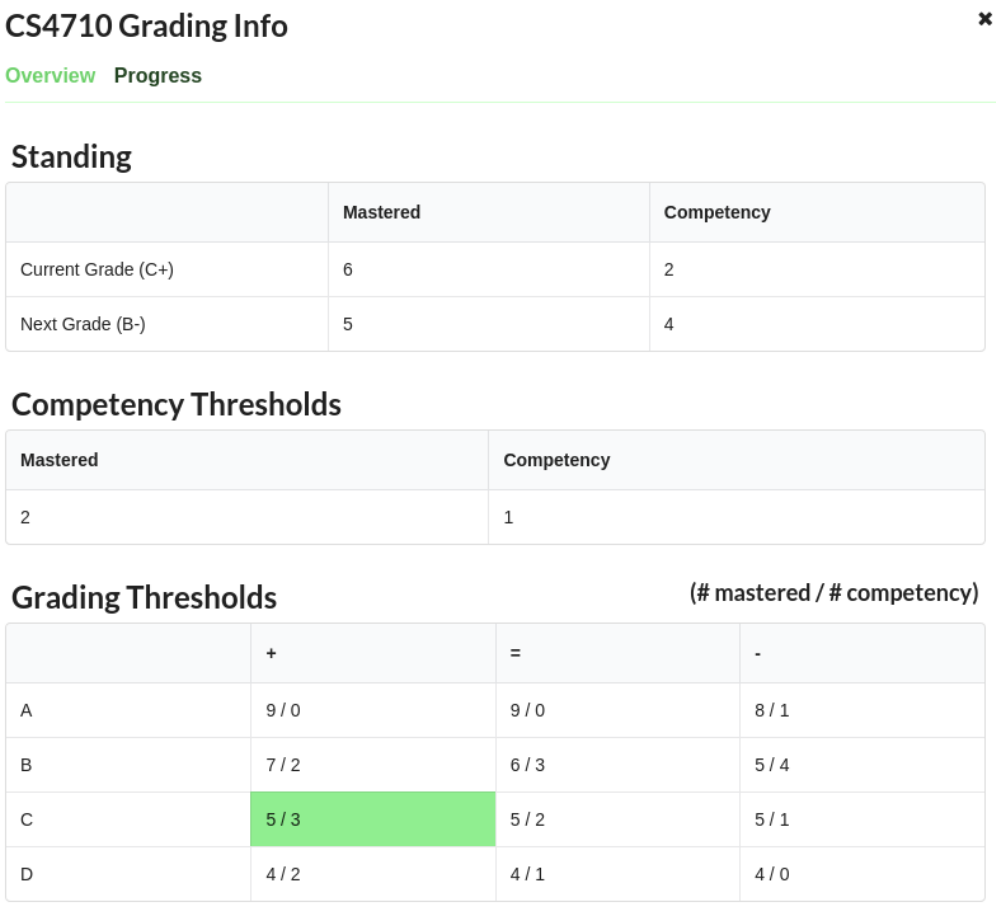

Figure 2: Course grading overview

This in conjunction with a new goal setting view (figure 3) allows a student to map out how they approach a course. This page displays the number of satisfactory and mastered nodes they need to reach a target grade. This gives students a sense of how quickly they should be accomplishing nodes. Many students commented in the Artificial Intelligence course that they

wanted a course calendar for when to accomplish assignments. This is slightly contrary to our goal of a student taking the course at their own pace. Hopefully this view will bridge that gap by giving students the tools they need to plan out their semester.

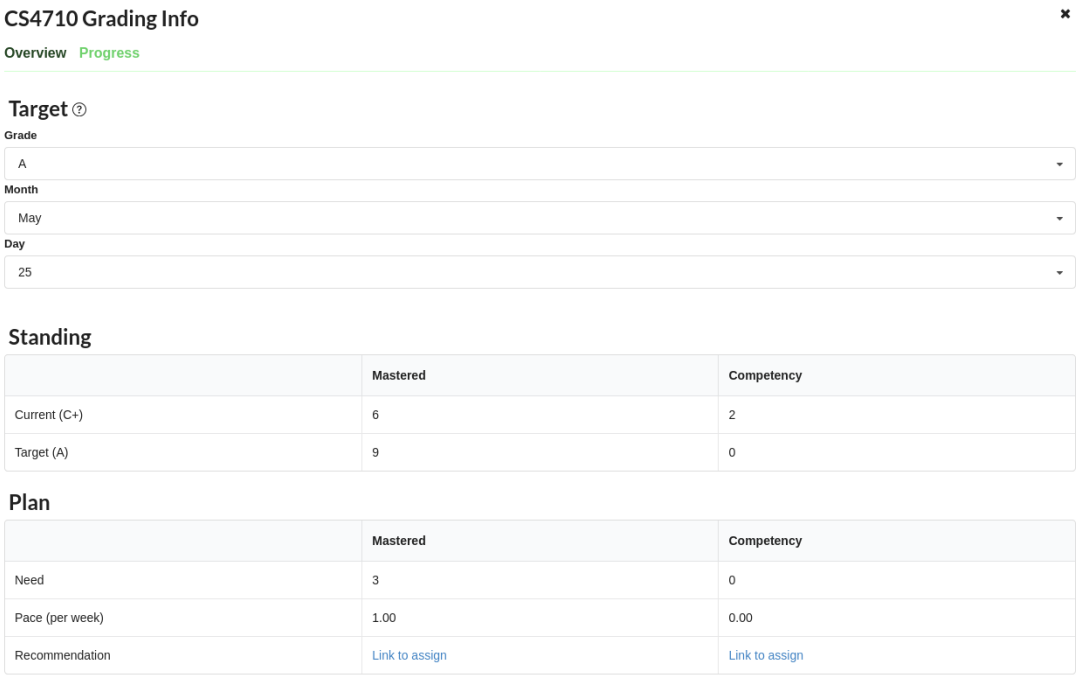

Figure 3: Goal setting tool

Dynamic Group Finding

In many computer science courses there is an enormous demand for help from teaching assistants and a consistent struggle to meet it. To address this issue, other professors have tried manually pairing students together who are willing to help, but this is often time consuming and difficult to arrange. Our goal was to create a system that is opt-in for students and pairs them according to skill and need for help. If a student is far behind, they should be prioritized over one that is far ahead of the class. Likewise, the system should draw on students far ahead in the course to help more frequently. Once students are paired, emails are sent to both students to help arrange a time for them to meet (but this is done exclusively outside of the system). We wanted to introduce a gamification element into helping others to incentivize participation. “Helpers”

would be rewarded a number of points based on the current demand and supply of help. These points could be used for a leaderboard for students to show off to peers and professors (who might see this as a good indicator of how a student would perform as a teaching assistant). Other applications might include badges or some kind of point shop for fun elements outside of the course. Students also tend to form study groups or review sessions which this tool could be adapted to facilitate. The code allowing students to sign up and be emailed upon pairing is mostly set up, however the actual pairing algorithm is incomplete.

Conclusion

In my opinion the grade display and presentation of a course is in a deployable state, but it still has a long way to go before it is adoptable by others. The goal setting code is working, but the group finding is not. Adding new features is appealing, but much of the work with the current features requires a lot of professor oversight to ensure they work properly. Going into the admin panel and inputting values, assignments, etc. is very user unfriendly. Kytos is a custom course/grading tool used at UVA, maintained mostly by Professor Tychonievich. Anecdotally, all students I've encountered love iterating with Kytos, however, it is very unfriendly for professors to use and as such has not been widely adopted.

Future Research

I see there being at least three major avenues of work that are needed to increase usability and value: professor improvements, developer improvements, and feature development. For professor improvements, Gradecope has many issues. The current API relies on Gradescope not changing the layout of the page. While this may not be a huge issue, it does mean that the entire project could break at the flip of a switch. Not all courses even use Gradescope so it might be

beneficial to pursue integration with Collab. For either platform, importing assignments created on the other and turning them into a graph would cut a professor's overhead in half. Some sort of upload for a graph, rather than creating it manually, would be useful. Many course values can only be set from visiting the admin page so they either need to be exposed to professors or have a way of partitioning data on the back end. As is if multiple courses were using the system, all professors could see all grades for all classes which is a significant concern.

As for development improvements, the current development environment is very loose and would benefit from CI/CD tools. There's a lot of code that has been written by a lot of people with little to no quality control. The testing suite is limited and cumbersome. Programs like SonarQube statically analyze code quality and interpret test output. Build managers like Travis CI and GitHub Actions ensure that tests aren't failing. Since we are dealing with sensitive data, it's very important the system is secure. Tools like OWASP ZAP and Fortify look at the security of code, and GitHub Secrets or Hashicorp Vault handle sensitive credentials and tokens. All this data can be fed into a single program like MITRE Heimdall to get a sweeping understanding of the health of the code. All of these tools are used in industry agile environments and future developers would benefit greatly from having them.

As for feature development, the group finder needs work as well as the reward systems surrounding it. Metric data tracking global student progression through the course, ensuring the quiz feature works, prerequisite views that connect with other courses (showing how dependencies interact across courses), generating data on how early A students complete assignments to provide to future students, running grading environments on the server (allowing Collab to be a more viable tool for CS courses) with OpenWhisk managing Docker containers, and stealth assessment are just a few of the possible ways to expand on this project.