Course Friction Explorer

Visualizing and validating indicators of student struggle

Shizuko Akamoto
shizuko@cs.ubc.ca

Noa Heyl
falkirks@cs.ubc.ca

Marie Salomon
mariesal@cs.ubc.ca

ToTo Tokaeo
tokaeo@cs.ubc.ca

Abstract

This section is left blank in the proposal.

1. Introduction

CPSC 310 is an undergraduate software engineering course offered at UBC Vancouver. The course involves a term-long project consisting of four checkpoints each spanning a period of 2-3 weeks. The project involves teamwork in which the student collaborates with one other (occasionally two) student(s) through GitHub using git version control. AutoTest is an evaluation tool used in this course, in which a suite of private tests are invoked against the team's solution, and feedback in the form of test failure overviews are reported back. Throughout the course of the project, students access resources including labs, office hours, and Piazza, where they seek help from the course staff. CPSC 310 is a large course, with typically more than 300 registered students, and this makes it extremely difficult for the course staff to determine when and why a student is struggling in the course. We define course "friction" loosely as any identifier of student struggles: some examples being ineffective resource allocation, non-standard use of version control, and prolonged period of inactivity. Recognizing these friction is crucial for the course staff as it makes possible early intervention. Course friction arise from many causes, and previous works have extracted and joined data from the various tools the course employs. For example, data set taken from 2020 Winter Term 2 offering contains 25,488 AutoTest results, 5,801 Piazza contributions, and 2,366 office hour visits, joined with each student. With existing data, one way to solve this problem would be to manually inspect each table row and try to identify suspicious patterns. But tabular representations are oftentimes inefficient for searching patterns over large data sets, and thus, CPSC 310 will benefit from a more effective visualization of their course data.

1.1 Personal Expertise

We chose the domain of course friction in part because all of us are currently TAs for some undergraduate software engineering courses at UBC (CPSC 310, 319, 410). We are often faced with the difficulty of identifying when a student is struggling, which delays timely intervention and has repercussions to overall effectiveness of learning. Most undergraduate level software engineering courses employ some sort of source code management, autograding, Q&A platform, from which we can derive identifiers of friction. In particular, CPSC 310's existing datasets are especially familiar to some of us as we have been working with them previously. This makes it an ideal starting point for a more generalized visualization tool on course friction. Finally, from a software engineering perspective, understanding the pitfalls in collaborative software engineering tasks aligns with our interest as well.

2. Related Work

There is a prevalent belief in CS education research that grades in our courses are bimodal, with some population of "strugglers" and others who do well. In past work this has been explained as a "geek gene", causing some students to be predisposed to better outcomes in Computer Science courses [1]. Robins introduced the concept of "learning edge momentum" which posits that the reason CS1 grades appear the way they do is due to the tight linking of knowledge and how it builds on itself [11]. If a student falls behind the impact will compound itself quickly, which they suggest is unique among fields of study. Patitsas et al find the problem space is more complex [10]. They evaluated many course distributions at UBC CPSC and found that very few are actually bimodal. This indicates that there are not necessarily two separable populations of students but instead a single population that undergo struggles in diverse ways.

Even if we can not split students into two populations based on outcomes, there is value in understanding how and why students struggle. Various authors have used different features to create models for identifying these students. In this paper we call these models "red flags". In "Exploring the Value of Different Data Sources for Predicting Student Performance in Multiple CS Courses" the authors use grade information to predict a final course outcome [9]. They find that prerequisite grade or clicker grade strongly predicts final grade. And in "In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments" they use measures of stagnation in grade to indicate struggle on an assignment [2]. Furthermore, Estey et al develop a model using changes in programming behaviour to identify students in need of support [4]. They find that they can identify students who require support in the first few weeks or term and target outreach to them. Neural networks have also been explored in finding students in need of assistance. One paper uses student grades and the number of submissions as features in their model [3]. Together these authors have conceptualized some feature (or "red flag") and demonstrated its relationship to student outcomes. In this paper we contribute a tool that allows quick discovery and validation of features like these in a general purpose way.

We build a dashboard style visualisation for representing student "red flags" and understanding their relationship to outcomes. Dashboards are a common tool for understanding learner behaviour. For example, Kia and their collaborators created a dashboard for visualising learner attributes in a MOOC class on edX

[8]. This dashboard displays attributes such as attendance, gender, and age as bar charts. Ginda and their collaborators also investigate MOOCs, creating conceptual content hierarchies and "learner path" visualisations that show the steps a learner takes through a class [6]. These tools demonstrate the utility of visualisations for educators to understand the experience of their students throughout a course. In this paper, instead of visualising learner attributes directly we visualise learners in terms of cohorts which are defined by models created by the visualisation user.

In "Quality based guidance for exploratory dimensionality reduction" the authors create a general tool and process for reducing a high dimensionality dataset into a single attribute one, allowing a user to pull out interesting elements for further inspection [5]. A user does this by selecting interesting variables and then inspecting their correlations. The problem of identifying struggling students also maps onto dimensionality reduction. However, in our work we are concerned about the correlation between each potential red flag and the true struggling students. This simplifies the problem considerably because we don't care about correlation between every pair of attributes. We also distinguish our work in that we identify correlations between membership in the set of struggling students instead of between quantitative attributes themselves. LineUp introduces a method for visualising multidimensional data in a tabular format. They facilitate the task of ranking based on a user-specified model [7]. LineUp also allows for the comparison of multiple models by displaying them side by side. This idiom is essential to identifying struggling students because users of our tool need to compare their candidate models to decide which is most useful. However, LineUp stops somewhat short of the idiom we need as they don't incorporate temporal data. We need a user to be able to understand how their model for identifying struggling students varies in accuracy and sensitivity over time. LineUp does not treat attributes as time-varying.

3. Data Abstraction

Our data abstraction consists of a series of tables representing various aspects of our dataset. We also expect to produce more derived attributes from these base tables.

3.1 Table autotest_results

Throughout the project, students make incremental submissions of their code by committing and pushing to branches on their git repositories. Each team's repository consists of a single master branch and a number of development branches which are oftentimes per member or per feature. Each project checkpoint has a suite of associated tests that AutoTest runs on each push to these git branches. Students can see the result of the AutoTest run on a specific commit by explicitly requesting AutoBot [1]. AutoTest result requests are rate-limited across branches, for example, one request on any branch per six hours per student.

Autotest results are identified by their feedback_id which is a unique identifier for each result entry. They also have several attributes.

3.1.1 Categorical attributes

- deliv - Checkpoint number
  - 4 categories (example: `"c0"`)
- ref - Git branch AutoTest was called on
  - 1665 categories (example: `"ref/tags/c2-rc5"`)
- is_master - True if ref is the master branch
  - 2 categories (example: `1`)
- feedback_requester - Deidentified user hash of the requester
  - 409 categories (example: `"j+TyZUe34/c1mZOH9tppky9C..."`[2])
- committer - Deidentified user hash of the committer
  - 409 categories (example: `"F+CFt8v9oVAaHZppCNKYTSN..."`)

3.1.2 Ordered attributes

- score - Test score of this AutoTest run
  - Ranges from 0.00 to 100.00 (example: `100.00`)
- visible_score - Test score currently available to the student
  - Ranges from 0.00 to 100.00 (example: `90.00`)
- request_time - Timestamp of when a student requests the result
  - Ranges from 1610400675825 to 1622674908773 (example: `1610401163054`)
- feedback_time - Timestamp of when result is returned
  - Ranges from 1610400675825 to 1619628202579 (example: `1610521081669`)

3.2 Table contributions

Piazza is the most active resource where students seek and receive assistance from not only the course staff, but also other fellow CPSC 310 students. Students can create posts which can be either a note or an answer-wanted question, categorizing them using tags. A Piazza contribution includes every action from creating a post, replying to a post, creating a new followup to an existing post, etc, all of which are recorded with timestamps in this table.

Piazza contributions are identified by their cid which is a unique identifier for each contribution entry. They also have several attributes.

3.2.1 Categorical attributes

- is_anonymous - True if post is created by anonymous contributor
  - 2 categories (example: `0`)
- kind - Contribution kind
  - 12 categories (example: `"followup"`)
- is_project - True if contribution is tagged as project
  - 2 categories (example: `1`)
- anon_id - De-identified id of the contributor
  - 409 categories (example: `"07e7yyUGH4zoF+i5UF3PH9d..."`)
- post_id - Unique identifier of the post the contribution was on
  - 1436 categories (example: `"Ks43Y68znhtzXws8zNnDG..."`)

3.2.2 Ordered attributes

- created_at - Timestamp of contribution
  - Ranges from 1610149993000 to 1620093785000 (example: `1619632354000`)

[1] Autobot is an autograding system built at UBC, students request Autobot by commenting on a Github commit, but we run tests regardless of whether they request it.

[2] Identifier hashes have been truncated to save line space in this paper.

3.3 Table queue_visits

Aside from Piazza, TA-held office hours are also one resource students use for issues that benefit from more synchronous, one-to-one interaction. Access to TA assistance in office hours are regulated by Queue@UBC, an online queue service simulating "lining-up" for help. Each student seeking assistance would enqueue and wait for a notification for their turn. A TA can view all the students currently on the queue, and would pick one to "start answering" thereby dequeuing them. Upon addressing the student's question, the TA would "finish answering", recording answer_finish. Note that contrary to a conventional queue, the TA need not follow FIFO order strictly; this is to prioritize help for the students requiring more immediate assistance.

Queue visits are identified by their qid which is a unique identifier for each queue entry. They also have several attributes.

3.3.1 Categorical attributes

- anon_id - Deidentified identifier of student asking question
  - 409 categories (example: `"DjN2/LSrZHkxfxAk/ka8gIigB6..."`)
- answerer_id - Deidentified identifier of TA answering question
  - 27 categories (example: `"Nxw7gaFw+d2v0moktJ1dGKzd4Ix2..."`)

3.3.2 Ordered attributes

- enqueue - Timestamp of enqueue
  - Ranges from 1600386235000 to 1618015298000 (example: `1618005226000`)
- dequeue - Timestamp of dequeue
  - Ranges from 1600386253000 to 1619307862000 (example: `1619307843000`)
- answer_start - Timestamp of when the TA starts answering
  - Ranges from 1600386239000 to 1618013723000 (example: `1618008486000`)
- answer_finish - Timestamp of when the TA finishing answering
  - Ranges from 1600386239000 to 1618016558000 (example: `1618004632000`)

3.4 Table users

A deidentified user hash, anon_id, corresponds to each student and TA.

3.4.1 Categorical attributes

- withdrawn - True if the user withdrew from the course
  - 2 categories (example: `1`)

3.4.2 Ordered attributes

- first_lab_time - Timestamp of the user's first lab. Null if user is a TA
  - Ranges from 1610384400000 to 1610751600000 (example: `1610751600000`)

4. Task Abstraction

We want to build a visualization dashboard that will help instructors and TAs detect struggling students early on in a course (CPSC 310). To do so, they can use certain indicators correlated with some outcome, for example low final grades, as red flags to identify the students. These indicators may be some patterns dependent on the following:

- Office hour visits
- Piazza contributions
- Auto-grading results

If a student happens to fulfill a red flag, then some intervention may be helpful in keeping the student on track. Although these red flags can be any arbitrary condition, we can also do some prior analysis using previous years data to discover meaningful red flags. To simulate making a prediction, there should also be a way to restrict available data to a certain time-frame. We will also consider calculating some statistics that may describe how well the indicator predicts the outcome.

In order to accomplish this goal we introduce the following high-level tasks

- **T1**: Discover indicators that identify students who are struggling.
- **T2**: Evaluate and compare candidate indicators based on their sensitivity, accuracy, stability, and speed (at identifying struggling students).

5. Solution

Our solution allows instructors and TAs to perform rapid verification on their hypothesized indicators of student struggles. It should provide an intuitive means to visually confirm a certain pattern in student data as a potential red flag leading to some unfavourable outcome, thereby enabling course staff to step in at an earlier stage. The instructors and TAs can use pre-proposed indicators of student friction from previous studies to verify against their own course dataset, but the visualization should also guide them to explore their dataset and discover more red flags.

We propose a solution consisting of two main idioms

5.1 Circular Packing

An example of this view is shown in Figure 1.

- Outcome of interest is represented as a central circle
- Each indicator being verified is positioned as circles around the outcome
- Students identified by each indicator (ie. belonging to the cluster described by the indicator) are represented as sub-circles inside.
- The size of an indicator circle is proportional to the size of the contained student cluster.
- Each indicator circle is colour-coded with saturation encoding in the red hue, encoding its similarity to the outcome circle. Similarity is defined by member containment: the indicator cluster contains a student also present in the Outcome cluster. This value should be binned into several saturation ranges to avoid the problem of indistinguishable colour encoding.
- Outcome and identifiers to be verified can be configured in the collapsible side panel.
- Interactions include zooming into a single cluster, dragging (useful when many indicators are simultaneously visualized), and potentially hovering to display more detailed information about the cluster. The user can also interact with the time slider to step through the evolution of each cluster in a specified time interval.

5.2 Table

An example of this view is shown in Figure 2.

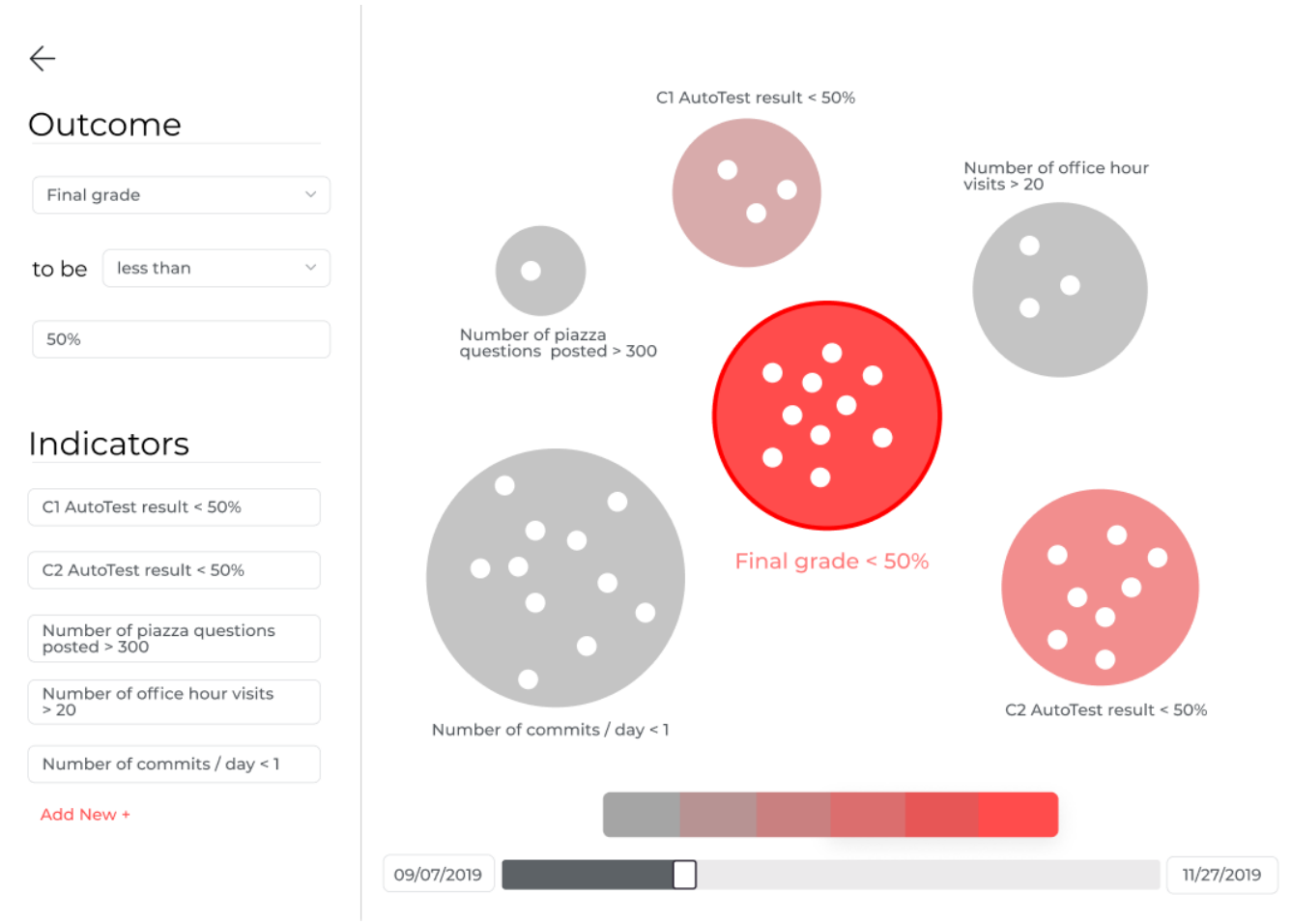

Figure 1. Circular packing view of 5 possible indicators of a failing final grade.

- The user navigates to a per indicator table visualization by selecting the indicator circle in the circle packing of visualization 1.
- The categorical attributes are: students in the selected indicator cluster on the vertical axis and the different indicators on the horizontal axis. (NOTE: these indicators need not be part of what was selected in visualization 1).
- The quantitative attribute of each cell is the derived value of student data based on the indicator. Rows with students also contained in the outcome cluster should be colour-coded with red colour. Interactions include sorting on the indicator columns and filtering on the rows.
- Further details of this visualization is currently under discussion, but the goal is providing opportunities for explorations to identify identifiers other than the set from visualization 1.

5.3 Scenario

Imagine yourself as an instructor for CPSC 310. It is currently midterm and your students have just passed their deadline for submitting code for checkpoint 1 of the project, and AutoTest executed test suites against each of their codebase to give an interim checkpoint grade. It is a good time to evaluate how your students are progressing in the course, and if any of them are showing signs of struggle. You would want to identify struggling students and reach out to them earlier so that they can make improvements toward the end of the term. From the studies you have read previously, you know that several factors are often proposed as causes of friction:

- Starting course assignment late
- Performing suboptimally on one of the earlier assignment
- Seeking assistance repeatedly in a short span of time

Now, if these indicators were verified to also be applicable to CPSC 310, then a bulk of your work is complete. The rest of the work involves identifying students with these particular characteristics, which is comparably straightforward. You turn to Course Friction Explorer for performing the work of friction verification against datasets from the most recent course offering: 2019 Winter Term. You're most interested in knowing whether these proposed friction indicators can actually identify students that have failed the course in that term. So, in the circular packing visualization, you set the "Outcome" circle to be "Students who received final grades less than 50%". Then, you add each one of the aforementioned characteristics as an indicator circle. These circles are, "Students who started checkpoint 1 after Oct 1. 2019", "Students who received AutoTest results of less than 50% on checkpoint 1", and "Students who create more than 5 piazza posts daily on average". After all the interested indicators are added, you notice the visualization displays a central red circle corresponding to the Outcome cluster, and three surrounding Indicator circles of different sizes.

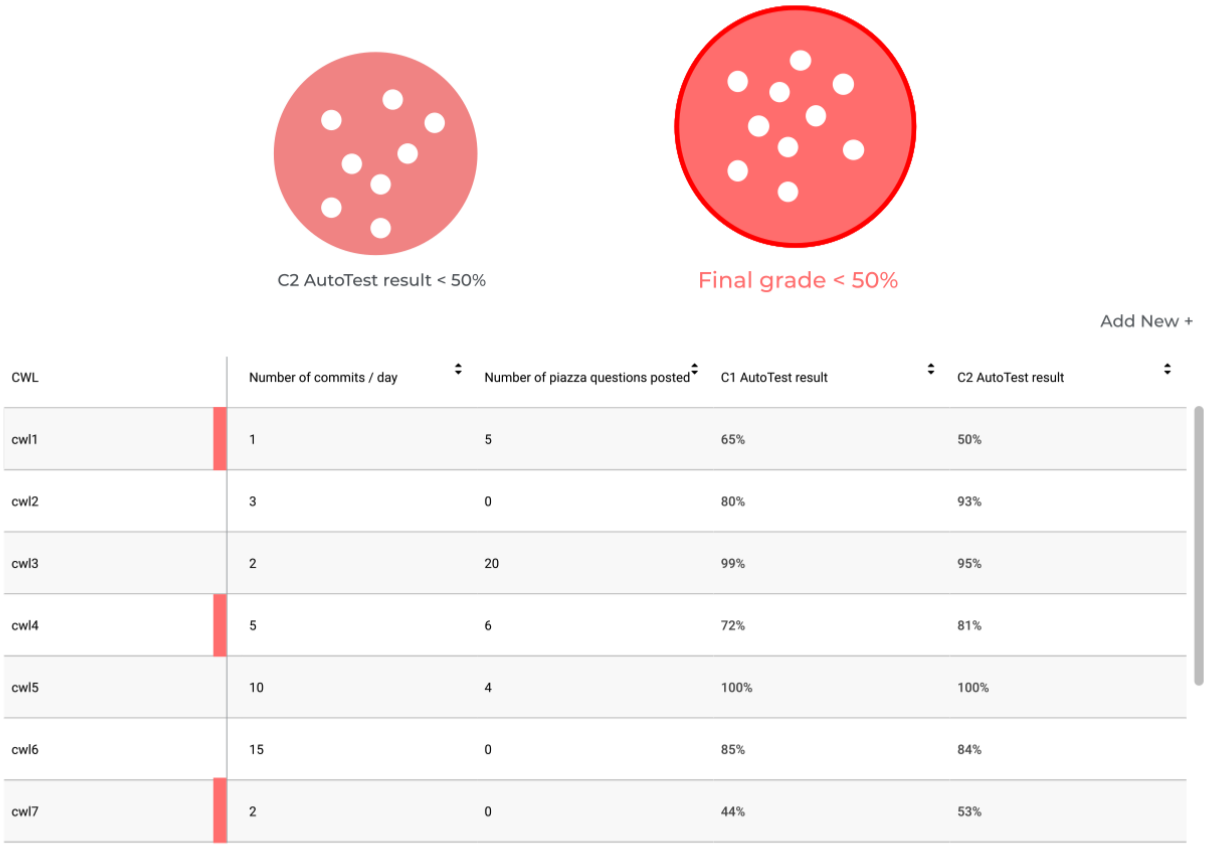

Figure 2. Tabular view of the results of a specific indicator.

In particular, the circle corresponding to the indicator "Students who received AutoTest results of less than 50% on checkpoint 1" is coloured with saturated red, very much similar to the colour of the Outcome circle. You click into the particular indicator circle to inspect the member students, and realize most of them are also members of the Outcome circle. You perform the same verification on datasets from multiple different past offerings, all leading to this convergent result. At this point, you're more confident of the applicability of this indicator on CPSC 310.

Learning from this verification, you can make a preemptive move of reaching out to your students from your current term whose checkpoint 1 AutoTest result reported a failing grade.

5.4 Implementation Approach

We plan to implement our visualizations as a web application, from multiple factors such as our familiarity with web technologies and languages, it being platform agnostic, and easier potential integration with other services like user authentication. Tentatively, we have selected python [3] for our data layer, and Flask. [4] as our backend framework. Python offers an abundance of data analysis libraries which will be a critical part of our solution, and Flask is light-weight and above all, integrates with python well. For the frontend we will be using React [5] and d3.js [6] for laying out our visualizations, user interaction and querying our backend. D3.js has a well-established community with strong documentations and examples, and especially good customizability. We may also include other frontend component libraries like Bootstrap [7] to expedite our development process.

6. Milestones and Schedule

Table 1 displays the proposed development schedule of the Course Friction Explorer from the initial planning to writing and submitting the final paper. We are estimating a total of 81 person-hours on task, with the main hours spent at implementing the Course Friction Explorer followed by writing the paper. However, estimating and planning person hours on task tends to be very difficult and error-prone. Therefore, we will be iteratively revising the allocation of hours, to confirm their accuracy or adapt where needed.

7. Discussion

This section is left blank in the proposal.

[3] https://www.python.org/

[4] https://flask.palletsprojects.com/en/2.0.x/

[5] https://reactjs.org/

[6] https://d3js.org/

[7] https://getbootstrap.com/

	Task	Due date	(Total Hr. / Per person)	Description
1	Pitch	Sep. 29	8/2	Create content and rehearse pitch
2	Proposal	Oct. 21	28/9	Discuss the project, create illustrations and write the project proposal
3	Learning and understanding the tools	Oct. 28	40/10	Read the documentations and examples. Learn how to use the tools and how they can interact.
	- d3.js	Oct. 24	16/4	
	- Flask	Oct. 26	12/3	
	- Python	Oct. 28	12/3	
4	Project Update I	Nov. 16	12/3	Prepare slides for the Project Peer Reviews
5	Project Update II	Nov. 24	16/4	Prepare slide for the Post-Update Meeting and demonstrate the prototype
6	Implementation	Dec. 6	160/40	Implement and complete the Course Friction Explorer
	- Backend: Setup, Data, Configs, Querying, DSL	Nov. 11	60/15	Setup the environment, clean and work with the data in the backend, do the configs and create queries.
	- Frontend: Circular packing, Table	Nov. 26	60/15	Implement the frontend by i.a. creating circular packing and table models.
	- Analysis	Dec, 6	40/10	Generating additional properties people might use
7	Draft of the final paper	Dec. 8	20/5	Write draft of the final paper
8	Presentation	Dec. 15	16/4	Prepare slides for the final presentation and talking points
9	Final paper	Dec. 17	24/6	Finish the paper and include final changes and conclusion

Table 1. Overview of project milestones and person hours allocated to each

References

[1] Alireza Ahadi and Raymond Lister. 2013. Geek Genes, Prior Knowledge, Stumbling Points and Learning Edge Momentum: Parts of the One Elephant?. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (San Diego, San California, USA) *(ICER '13)*. Association for Computing Machinery, New York, NY, USA, 123–128. https://doi.org/10.1145/2493394.2493416

[2] Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D. Hundhausen, and Abigayle Peterson. 2021. In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 467–473. https://doi.org/10.1145/3408877.3432357

[3] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 111–116. https://doi.org/10.1145/3017680.3017792

[4] Anthony Estey, Hieke Keuning, and Yvonne Coady. 2017. Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 189–194. https://doi.org/10.1145/3017680.3017790

[5] Sara Johansson Fernstad, Jane Shaw, and Jimmy Johansson. 2013. Quality-based guidance for exploratory dimensionality reduction. *Information Visualization* 12, 1 (2013), 44–64. https://doi.org/10.1177/1473871612460526 arXiv:https://doi.org/10.1177/1473871612460526

[6] Michael Ginda, Michael C. Richey, Mark Cousino, and Katy Börner. 2019. Visualizing learner engagement, performance, and trajectories to evaluate and optimize online course design. *PLOS ONE* 14, 5 (05 2019), 1–19. https://doi.org/10.1371/journal.pone.0215964

[7] Samuel Gratzl, Alexander Lex, Nils Gehlenborg, Hanspeter Pfister, and Marc Streit. 2013. LineUp: Visual Analysis of Multi-Attribute Rankings. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2277–2286. https://doi.org/10.1109/TVCG.2013.173

[8] Fatemeh Salehian Kia and Simon Fraser. 2016. Learning Dashboard: Bringing Student Background and Performance Online.

[9] Soohyun Nam Liao, Daniel Zingaro, Christine Alvarado, William G. Griswold, and Leo Porter. 2019. Exploring the Value of Different Data Sources for Predicting Student Performance in Multiple CS Courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 112–118.

https://doi.org/10.1145/3287324.3287407

[10] Elizabeth Patitsas, Jesse Berlin, Michelle Craig, and Steve Easterbrook. 2016. Evidence That Computer Science Grades Are Not Bimodal. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 113–121. https://doi.org/10.1145/2960310.2960312

[11] Anthony Robins. 2010. Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education* 20, 1 (2010), 37–71. https://doi.org/10.1080/08993401003612167 arXiv:https://doi.org/10.1080/08993401003612167