InsightVis: A Visualization Tool to Assist CPSC 310 Course Management and Design

Lucas Zamprogno
lucasaz@cs.ubc.ca

Syed Ishtiaque Ahmad
siahmad@cs.ubc.ca

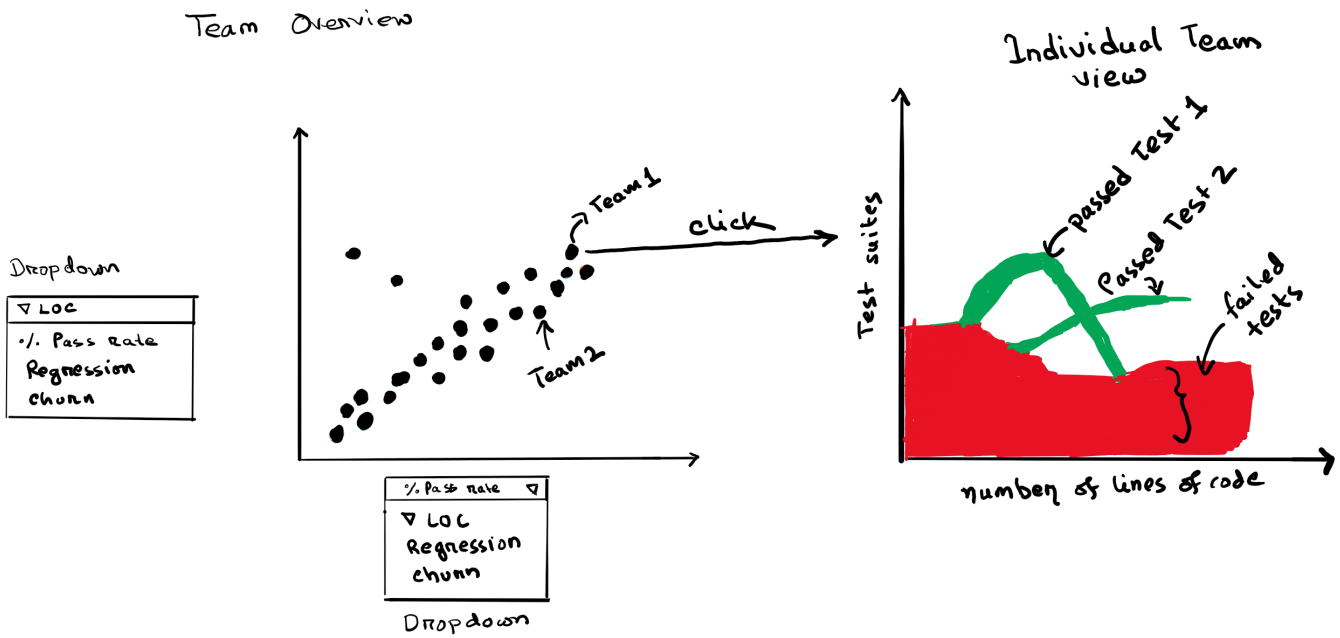

Figure 1: Sketches of planned views for whole-course and individual-team views. On the left is the team overview visualization that shows all the teams and their relation between percentage pass rate versus change in line of code. On the right is a flow diagram that shows individual test performance verses number of lines of code.

1 INTRODUCTION

CPSC 310 is a Software Engineering course at UBC, and is required for all Computer Science majors. As such the class can have very high enrollment (approximately 380 students in busy terms) and employs a large (approximately 18) TA team. The course has a heavy project focus, with teams of two working over a couple months to implement functionality according to a specification. This specification is split into multiple chunks of functionality, called deliverables, which have their own set of automated tests used for marking. We make an attempt to tailor the course in certain ways to encourage a good environment for students to develop their project, pushing good practices like good unit testing and requiring the use of a linter. However with such a large course it can be hard to gain an overview of how all teams are doing, or how various factors related to their success. We would like to improve this by providing a more accessible overview of this information.

2 PERSONAL EXPERTISE

The motivation for this project arises from our first-hand experience of working as teaching assistants for CPSC 310. We have seen teams struggle with passing test cases, performance falling across deliverables and dealt with teams vague piazza questions trying to get a sense of their struggles.

Furthermore, we maintain a service called Classy[1] that displays all the teams performance metrics like test scores, test coverage, test passed and overall score across deliverables. So we already know what data are available and what can be derived.

Additionally, we have designed the test suites and methods to automatically grade student projects using Autotest[2]. As a result, we have a clear understanding of the test suites and can use our knowledge to interpret results from Classy to figure out which test suites the teams are failing or passing.

Lastly, this project relates to our area of expertise that is software engineering and it would be interesting to figure out the problems to design an effective course.

Table 1: Some Typical Attributes. We may extract or derive more data as the project progresses if needed.

Attribute Name	Attribute Type	Description
testPercentage	Quantitative	Percentage of test passed against student code
coverageGrade	Quantitative	Percentage of code executed by student tests
finalGrade	Quantitative	Computed as 0.8*testGrade + 0.2*coverageGrade.
timeStamp	Sequential	Time when teams pushed the code
commitSha	Categorical	Sha-1 hash of submitted commit
teamId	Categorical	Contains the team number
deliverable	Categorical	The submitted delieverable, which is either d1, d2 or d3
testSuite	Categorical	All the test cases the team has to pass during a particular deliverable
state	Categorical	Indicates whether test passed or failed

3 DATA AND TASK

Our solutions use a combination of existing data and derived data. We have archival data of all a prior term's Autobot runs and test results over the duration of the course. Additionally we can still access the repositories of these teams, so it should be possible to mine additional data related to their code base and project history. The derived attribute *ChangeInLOC* describes the difference between current lines of code and previous existing line of code across two commits, found on our local GitHub instance. The *changeInTest* attributes would be computed from test results overtime. We visualize this along with other code metrics (Table 1) to gain a complete understanding of difficulty of tests, struggling teams, the hidden relationship between test cases.

Here we use the what-why-how framework [2] to abstract our solution to the vis domain.

What. Table of graded submission items with the attributes described in Table 1. This number can grow over time, but for the dataset, we are working with, there may be roughly 180 teams, with approximately 200 commits per team, for a total of 36,000 commits. Each continuously growing team repository (over a series of commits) ends with roughly 3,000 lines of code, with some reaching around 7,000 maximum, totalling to 540,000 lines for all teams. The *testPercentage, coverageGrade* and *finalGrade* attributes have values between 0-100. Each deliverable (d1, d2 and d3) contain roughly 50 tests each. A test has a *state* attribute with 3 levels (i.e. passed, failed or skipped). The *timeStamp* attributes contains date-time in the range between start and end of the course for a specific semester.

Why. Get an insight into how the class is doing as a while, as well as how individual teams are performing will help course staff target who requires help, and how. Additionally, the difficulty of tests and test dependencies could help the instructor design the course more effectively for future terms.

How. Our visualization will include:

- Scatterplots with points as a mark representing teams and configurable axes that can display the relationship between two attributes at a single instance. For example, the correlation between *changeInLOC* verses the *testPercentage* shows how the class is progressing, and outliers may indicate teams that are adding a lot of code without making progress, or who have made very efficient progress.
- Flow diagram encoded with color which differentiates between passed and failed tests. The width of the flow represents the number of tests passed or failed. This could track teams progress over time.
- Interactions in the Vis:
  - Hovering over the points in scatterplots will display *teamId*
  - Dropdowns for both axes of the scatterplot to select attributes of interest to visualize the correlation between them
  - Animation to show changes in scatterplots when attributes are selected
  - Hovering over the flow will display the name of the test and highlight it paths to bring it into focus
  - Filter results of interest by selecting dropdown options

[1]https://github.com/ubccpsc310/classy
[2]https://github.com/nickbradley/autotest

4 PROPOSED SOLUTION

We will have encoding for the main overview that that shows various metrics across teams, and an individual team visual encoding that shows the number of test cases passed as a team progresses through their project (i.e. as code changes/commits increase). This is represented in figure 1.

For the team overview, we are thinking of using scatterplots with both the x-axis and the y-axis being configurable by selecting values from a list of options like change in lines of code, percentage of tests passed, code churn, and regression frequency.

The individual team view will contain a flow diagram that has all the test cases represented in y-axis and change in number of lines of code in x-axis that changes over time. Initially, the encoding will start with all the test cases failing from the initial commit with no work done. The width of the flow presents the number of test cases failing. As time progress some test will pass and will emerge as a green color flow which will cause the width of failing flow to decrease. At some point later, the passed test case may fail and merge back with the main flow of failed tests, increasing the width of the red flow again.

As a stretch goal, we are thinking of including a test view which will display the performance of tests across all teams according to Figure 2. The test will be represented as doughnut shape pie charts displaying the percentage of teams passing, failing, and skipping that test using different colors. We would be able to sort the tests shown based on these percentages. This will help us identify the tests that teams are struggling to complete and give us overall

Table 2: Estimated work timeline

Task	Est Hours	Deadline	Description
Pitch (x2)	8	Oct. 8	Create slides, rehearse pitch
Proposal	15	Nov. 4	Discuss project ideas, create mockups and write the proposal
Learning d3.js	10	Nov. 10	Read d3.js documentation and learn to program using d3
Project Review 1	3	Nov. 19	Prepare slides
Project Review 2	3	Dec. 4	Prepare slides, have some version of demo ready
Implementation		Dec. 10	Completed Vis tool
- Prepare Data	15	Nov. 9	Clean, filter and restructure existing data. Fetch data from GitHub.
- Application structure	4	Nov. 11	Setup frontend, include all external libraries
- User interface (buttons, navbar)	6	Nov. 14	Setup the UI layout, add button and navigation control
- Team overview vis	26	Nov. 29	Implement overall teams view including search, filter and fetching data
- Individual team vis	18	Dec. 8	Implement individual team view for test cases
Presentation	12	Dec. 10	Prepare slides and rehearse
Final Paper	20	Dec. 13	Finalize paper. Draft to be written between Dec 3 - 10

sense of test difficulty across all teams. This can be important when introducing new tests to the grader (which is relatively common) and ensuring that it isn't disproportionately easy or challenging. If it seems like adding this feature is in scope, we may will elicit more feature ideas from course staff in advance of implementation.

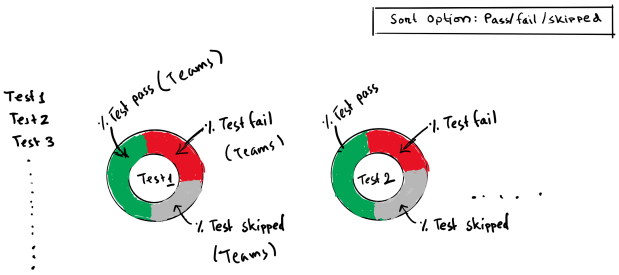

Figure 2: The test view shows the test list. Each test is represented by doughnut shape pie charts that show the percentage of the test passed, failed or skipped. The results can be sorted according to passed, failed or skipped tests.

5 USAGE SCENARIO

Imagine you are a TA and you want to know the correlation between teams writing a lot of code and percentage of tests passed to see if this gives you any insight about how teams are progressing. You are also interested in finding outliers, both for highlighting teams writing quality code (low number of lines with a high pass rate) or messy/non-functional code (high number of lines but a low pass rate). The TA would visit our application landing page. They will see a scatterplot which has a drop-down in both the x-axis and the y-axis that has a list of options. By default, it will already have selected options which are change in the line of code (y-axis) versus the percentage of the test passed (x-axis). For this scenario, they do not need to change the selected options in the drop-down. The result will show teams represented as points. A team that has a lot of code change over time and is still failing tests may show as an outlier, suggesting this team may have had to refactor many times and is struggling to make progress.

When the TA clicks on any team of interest it will open up another view with a different encoding that will show the flow diagram. If there are tests rapidly switching back and forth between the passing and failing flows, this would indicate that this team is struggling to keep this test passing. There is a good chance they are using some patchwork code to get it working, and do not have a good understanding of the underlying functionality they need to implement.

The TA could then return to the class overview scatterplot, and change one of the axes if they want to see a different correlation. An animation will transition between the two scenarios showing how individual teams move and the overall class dynamic changes under the new conditions.

6 IMPLEMENTATION APPROACH

We will implement our visualization as web application since web applications are platform independent, generally very popular and also because of our familiarity with using JavaScript[3]. Furthermore, building it as a web application will allow us to integrate with Classy and Autotest easily. Note that integrating with these services are not intended within the scope of this project, but we would like it to happen at a later date. For our core visualization, we will be using d3.js[4] because it seems flexible, and simply because we want to learn d3.js. Additionally, we will use a front-end component library like Bootstrap[5] to build user interface components and for setting the layout of the application. We will also use jQuery[6] for any DOM manipulation we need.

7 MILESTONES AND SCHEDULE

We are prepared to spend about 140 hours together towards this project. Table 2 provides the tasks breakdown for the project. Work

[3]https://developer.mozilla.org/en-US/docs/Web/JavaScript
[4]https://d3js.org/
[5]https://getbootstrap.com/
[6]https://jquery.com/

may be distributed unevenly between members *over time* as Lucas has work front-loaded with courses largely wrapping up mid-November whereas Syed will have exams into December.

8 PREVIOUS WORK

Ginda et al. designed metrics and visualizations intended to help track student engagement and performance in an online course setting [1]. Visualizations were used to show how students progress through course material, and how different metrics correlate with each other and overall performance. Similar to our intended approach, they produced multiple scatterplots that plotted individual students in relation to grades and interaction with course materials. They also plotted out a course structured, and used color and position coded lines to show how students moved forwards and backwards through course materials to see patterns in how students sequence their learning and review content. They suggest that this information can be used to optimize current course offerings and plan future courses.

Strandberg et al. used test result data and represented test status in an overview form using circular progress graphics [3]. Different colors in the circular progress graphic depicts the progress of different test results, for instance green shows passing of tests, red shows failure, while orange indicates that some tests remained invalid, unmappable, or unloadable. This is similar to our test view summary represented with a colour encoded doughnut pie charts to show percentage of instances that it is passed, failed, or skipped. Furthermore, in the paper test failure pattern across time are represented as heatmaps (to include a diversity of test machines) which performs a similar task to our individual team view flow diagram that shows the test failing patterns over time.

REFERENCES

[1] 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 (May 2019), 1–19. https://doi.org/10.1371/journal.pone.0215964

[2] Tamara Munzner. 2014. *Visualization Analysis and Design.* CRC Press.

[3] Per Erik Strandberg, Wasif Afzal, and Daniel Sundmark. 2018. Decision Making and Visualizations Based on Test Results. In *Proc. 12th ACM/IEEE International Symp. Empirical Software Engineering and Measurement.* Article 34, 10 pages.