InsightVis: Staff Dashboard for a Third-Year Software Engineering Course

Syed Ishtiaque Ahmad, Lucas Zamprogno

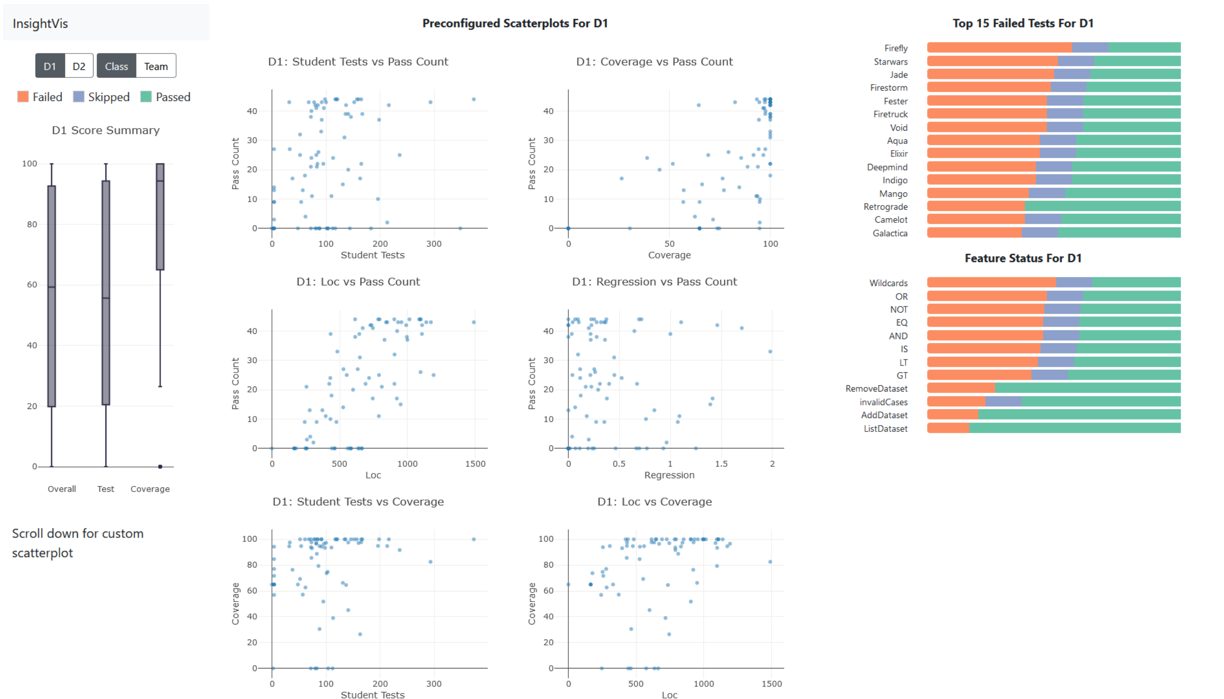

Fig. 1. The class tab of the InsightVis dashboard, the main landing page of the viz. The left column is a navigation bar with page controls, a legend, and box plot showing a high level grades summary. The center section has a small multiples scatterplot view showing relationships between various course attributes. Users can scroll down to access a larger scatterplot with configuration options. The right column highlights the tests and project features that are the least implemented by the class.

Abstract— Large classrooms place a high demand on course staff who are often strained for time. Course instructors need to be able to make informed decisions about course changes, and teaching assistants need quick access to information that helps them support students. To this end, we developed InsightVis, a dashboard to improve access to information for course staff of UBC's Introduction to Software Engineering. It features a class view that provides an overview of the whole class and supports broad analytics, and a team view to show a team's current and historical progress on developing their course project. InsightVis will be integrated into the currently used Classy course management system used by the course.

Index Terms—Design studies, time series data, tabular data, visualization in education

1 Introduction

Computer Science 310 is UBC's Introduction to Software Engineering course, and is a requirement for all Computer Science majors. As such the class have an enrollment of up to approximately 380 students during one term. The course has a heavy project focus, with teams of two working over three months to implement functionality according to a series of specifications. More details about the course will be discussed in Section 2. We make an attempt to tailor the course in ways that encourage a good environment for students to develop their project, pushing industry standard best practices like maintaining a strong test suite and requiring the use of a linter to adhere to code style standards.

However, with such a large course it can be hard to get an insight about how all teams are doing, or how various factors are related to their success in the project. To accommodate such a large class size, the course typically employs just under twenty teaching assistants (TAs) to assist teams in their activities. The class forum Piazza[1] can see up to 2,000 posts over the course of a term through which they can access TA help, as well through dozens of lab and office hours. Given the amount of demand on TAs, having resources available to facilitate TAs helping students effectively is important to help them keep up this demand. There is an existing dashboard as part of the Classy[2] course management system that the course uses which has most of the data required, however, it is almost exclusively in a tabular format and information needed is often split across many tabs. Our aim in this work was to improve the abilities of course staff to monitor and tailor the course by providing a more accessible overview of this information in the form of a visual dashboard.

- *Syed Ishtiaque Ahmad is with the University of British Columbia. E-mail: siahmad@cs.ubc.ca.*
- *Lucas Zamprogno is with the University of British Columbia. E-mail: lucasaz@cs.ubc.ca.*

[1]https://piazza.com
[2]https://github.com/ubccpsc/classy

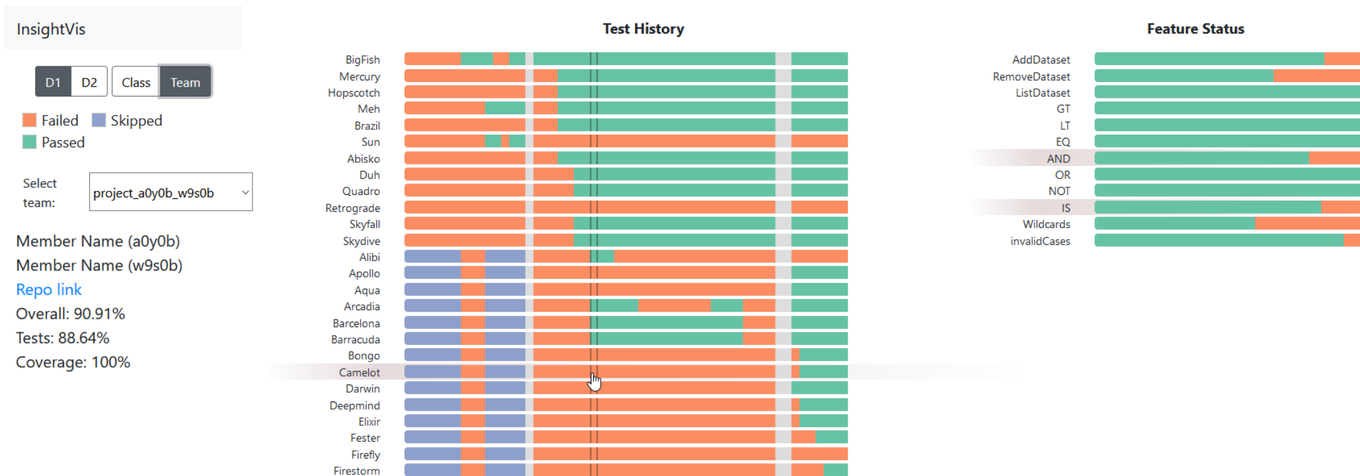

Fig. 2. The full team view tab of the dashboard. The left hand navigation panel contains a team select dropdown, and general information about the selected team. The right hand side contains a combination of test history and feature readout of the most recent commit. Linked highlighting shows the relationships between features and tests. Some of the test lists have been truncated to increase readability of this figure.

There are two broad use cases we are targeting with this work. The first is to assist in monitoring the state of the class, and discovering information that may be useful for modifying the course in the future. The second use case is for providing information about individual teams' performance. This case mainly arises either when a TA is assisting a team and wants to get some context, or for identifying and monitoring struggling teams.

The contribution of this paper is the design of this new system, and the discussion of the features of interest for a class settings. In Section 2 we discuss the more of the system context and our data and task abstractions. Section 3 describes the final configuration of our system. The details of this implementation are discussed in Section 4. We discuss the results of the project in Section 5. In Section 6 we elaborate on the specific strengths and future work needed as well as lessons learned. Lastly, in Section 7 we discuss related work in the field.

2 Data and Task Abstractions

In this section, we discuss the context and details of the data used in our visualization, as well as the contexts in which this data would be viewed by a user.

2.1 Data

We will start with a description of the source of our data and the context surrounding it, then move on to a more detailed description of the data used by our system.

2.1.1 Domain

The most important existing system involved in data generation in the course is called Autotest. Autotest is the system responsible for running and grading student projects and delivering feedback. Students are graded every time they push changes to their repository, and a record is kept of each grading run, leading to a grading history of each project along with the actual software artifacts in the repository. The project students must implement is a simple data storage and query language system. Project work is divided up between multiple specifications called *deliverables*, which typically encompasses three weeks of the term each and its own set of automated tests the students need to pass, though the specific details of these tests are hidden from the students. For this project, we narrowed our focus to only two deliverables, `D1` and `D2`. We chose to exclude `D3` due to the fact that it is currently undergoing changes and is already different at the time of writing from when our data was generated, and will likely be changed even more once this system is integrated into the course.

The main component of the grade is how many tests are passed out of the total test suite, this makes up the test score, worth 80% of their final grade. However many of the tests require more than one "feature" to work in order to pass. In the context of the course project, a feature is some subset of the project specification, for instance, a certain query operator like a logical `AND` or the ability to process wildcard characters. The feedback delivered to students is how well they are performing across the various features required in the deliverable, for instance, they might be passing six of ten queries requiring `AND` functionality. As well as being shown to the students this information is also available to course staff.

The second component of the grade comes from the students' coverage of their source code with their unit tests, and is worth the other 20% of their grade. This self-testing grading coupled with not directly showing which of our tests they are failing encourages thorough self-testing and the use of test-driven development. In addition to the Autotest data, the student projects are maintained in locally hosted GitHub repositories. Given this, we have access to the states of each project at the time that Autotest graded the project. From this, we can get details like lines of code and number of tests by cloning and inspecting their repositories.

2.1.2 Data specification

The source of most of our data is an approximately 100MB JSON file which is the database entries of all grading runs in one term. The remainder of the data is the source code of each student project. This is acquired by cloning each team's repository from our locally hosted GitHub instance. After processing our dataset is essentially tabular despite never being directly stored as such. The data we ended up using is likely best discussed categorized by its source, either determined by the course test suites and grading process, or metrics derived from student source code.

The data from the course include grades and test data. Grades are all on percentage scales based on the number of tests passed, and coverage scores obtained. Number of tests passed has more nuance. This attribute is capped at the number of tests in a deliverable, which is typically in the 40-60 range though this varies by term and deliverable. Additionally, we may choose to display *subsets* of the tests passed, such as when displaying how many tests *that require a specific feature* a student is passing. As a result, the maximum of this may be lower depending on the context.

Regression rate is one metric which is derived and defined by us for this project. It is intended to represent how much a team is having their passing tests revert to a failing state, which we were originally going to represent as a simple count. However, we realized teams that push changes more frequently would have more chances for breaking changes to be seen by our grader, so we decided to normalize it as an

Table 1. Details of attributes used in our dashboard. Those above the divider are visualized, those below were also used but only for internal computation and labelling purposes.

Attribute Name	Attribute Type	Description
Test score	Quantitative	Percentage of test passed against student code, 0-100%
Coverage score	Quantitative	Percentage of code executed by student tests, 0-100%
Overall score	Quantitative	Computed as (0.8 * Test score) + (0.2 * Coverage score), 0-100%.
Test pass count	Quantitative	Number of tests passed, historically maximum is below 60, unlikely to go notably higher
Lines of code	Quantitative	Number of source lines of code in the student project. Typically 2000-3000, hypothetically unbounded
Student tests	Quantitative	Number of student tests run to generate coverage. Typically lower than 300, hypothetically unbounded
Regression rate	Quantitative	How often a test goes from passing back to failing, normalized by number of commits. Typically less than 3.
Deliverable	Categorical	Either "D1" or "D2"
Test state	Categorical	Either "passed" , "failed" , or "skipped"
Team id	Categorical	Identifier for a team, composed of members' Computer Science IDs
Commit SHA	Categorical	Unique identifier for a commit
Timestamp	Sequential	Denotes the time that the commit was made

average rate of regressions per graded commit.

Lines of code and number of tests come from students and so are much less controlled. Individual differences in approach and thoroughness when completing the project can lead to widely varying values here, both of which are hypothetically unbounded. In practice, the number of tests is constrained by a time limit on how long their code is allowed to run in the grading service, and lines of code would be constrained by students being unlikely to include dead code or useless files in their projects. From the student data, we also used their team id and commit SHAs as unique identifiers, as well as the timestamps from the commit events as sequential markers for building out test histories. These attributes are never directly visualized outside of the team id being used as a label.

2.2 Tasks and Use Cases

There are two broad categories of use we are trying to simplify with our solution, tasks focused on observing the class as a whole, and those focused on a particular team.

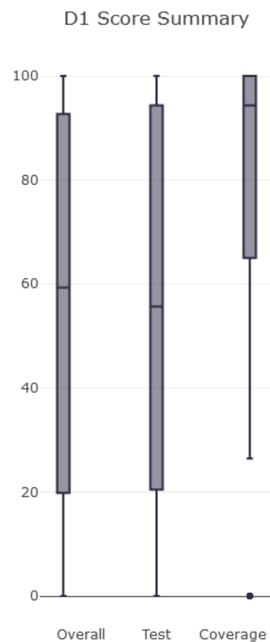

Fig. 3. The global class score display from the class view navigation bar. Shows the general state of progress on the project across the class.

2.2.1 Class focused

One common question that arises is very simply "How is the class doing" or "how far have teams progressed" ? A common use of this information is that a general summary of the whole class like this can inform how much load TAs should expect at labs and office hours in the coming weeks. The class doing poorer than expected at a given point in time (current determined by the grade median compared to prior terms) is a fairly strong sign that TAs are likely going to see an increased number of students showing up to labs and office hours. This summary is also important in gauging if new test batches or project changes are too challenging or too easy. This would be seen as either abnormally low grades on the deliverable, a certain feature, or a test depending on the scale of the change. Previously the only methods of investigating this are by looking at whole deliverable medians, or visually scanning a table of recent commit runs[3].

2.2.2 Team focused

The majority of day-to-day TA work is focused on helping teams work through the challenges of building their projects, as well as assessing and checking in with groups at the end of a deliverable. When teams ask for help it is often useful to get a quick sense of a team's general progress up to this point. This would involve questions like "how are they doing on X feature" , "what functionality is holding them back" , and "had they ever had this functioning correctly in the past" . With this information it is easier to make judgements about how the team may need to approach their project in order to improve.

3 Solution

Here we discuss the context and evolution of our visualizations, as well as the idioms that were used in the final build of our system.

3.1 Early concepts and design philosophy

Our initial concept for the InsightVis dashboard followed the split between class overview and individual teams that has been mentioned in prior sections, but was only planned to include two specific visualizations. The first was going to be a configurable scatterplot for the class view, where the user can change the axes, and click on points to see the team view for that team. This largely made it into the final implementation unchanged as seen in Figure 5, and will be discussed more in Section 3. The team view however, went through much more change. The initial concept was a Sankey-like diagram where all student tests start out as failing, and you can see them flow into a passing stream as the project is built as well see when the tests flow back to a failing state if something breaks. The concept behind this was that it would provide quick information about overall progress in the form of the size of the passing stream, as well as how smooth the progress has been by seeing how often flows from passing to failing occur and

[3]Or in some more unfortunate cases having students ask if a test is impossible, and finding out the answer is "yes" .

how large they are. One downside we identified shortly before starting was that it would be hard or impossible to pick out individual tests or specific numbers using this encoding. Individual tests were of special concern since students often ask about one specific issue or subset of functionality they are working on when requesting TA assistance. After producing a prototype as seen in Figure 8, it was clear that this idea was not going to produce very usable results.

From there we pivoted to a strong functionality focused philosophy, our designs do not need to be interesting but they should be accurate and informative. To the informative end, we gave our selves some easier to use frameworks to simply allow us more time to build in more figures. For accuracy, we tried to use visualizations based on aligned scales wherever possible. Fortunately, our scatterplot concept already fit this framework and did not need to be fundamentally redesigned. In the end, we believe this core lead to a vastly more useful design, if only at the expense of some “wow factor” .

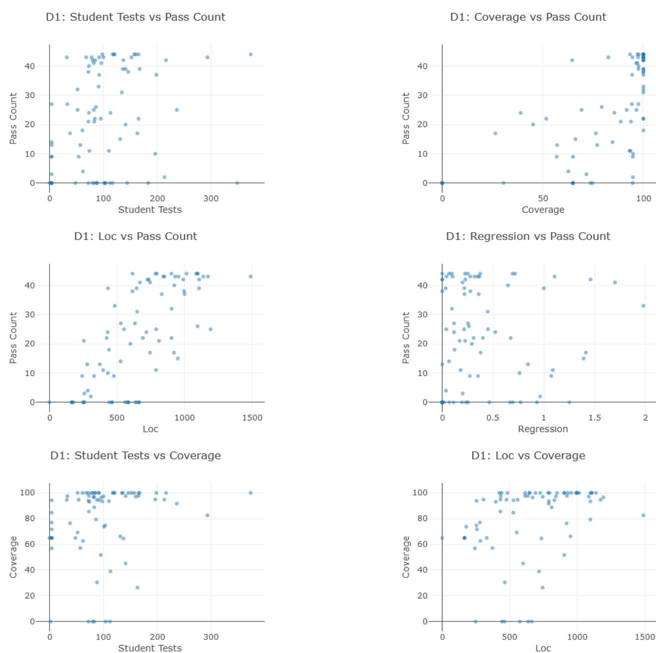

Fig. 4. Small multiples scatterplots from the class view. The six scatterplot configurations were chosen by the authors as the most likely to be meaningful for course design questions.

3.2 Resulting solution

Our solution is split across two tabs, one for the class view and one for the team view, as well as a side navigation bar. This navigation bar remains largely the same between views showing controls and a legend, but does get some context specific information depending on the tab.

3.2.1 Class view

Table 2. What why how analysis of the class view.

What	Tabular grade records with tests passed and grades
What	Mined repository data, derived regression score
Why	Discover trends and outliers in team performance
Why	Summarize test data from many teams
How	Scatterplots, small multiples, box plot, stacked bar charts
How	Scatterplot: Separate, change, select, juxtapose
How	Tests: Order, align, aggregate

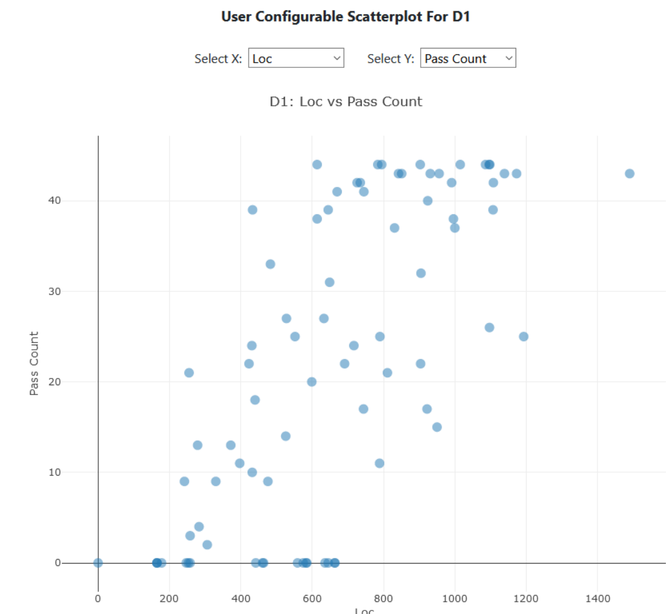

Fig. 5. The configurable scatterplot from the class view, it is larger and placed below the small-multiple scatterplots. The axes attributes can be selected from dropdown menus above.

Our course view features three main components, a broad grade summary, scatterplots of project attributes, and the status of tests across the whole class. A view of the whole view can be seen in Figure 1. The grade summary uses a box plot to display the median, quarter intervals, and maximum and minimums of student overall, test, and coverage scores. This was chosen for being compact, and easily understood at a glance. Users can also get detailed numbers by hovering over a box to display the exact statistics. The summary plot can be seen in Figure 3.

We used two variations on scatterplots to display overall course trends and relationships. Immediately displayed on screen is a small multiples view showing six scatterplots selected as the most potentially interesting or meaningful relationships. The top four involve various attributes interactions with test pass rates, and the bottom two show interactions with coverage. The small multiples scatterplots can be seen in Figure 4. If the user has a specific question that is not answerable using the small multiples scatterplots, they can scroll down to a larger scatterplot which features configurable axes. Attributes are selectable via dropdown menus and the scatterplot will update to reflect the new configuration. The configurable scatterplot can be seen in Figure 5. On all the scatterplots points start out partially transparent, and get more opaque as they overlap to reduces issues with occlusion when the axes have a small number of discrete values. Our scatterplots are also interactive. Selecting a point (representing a team) on a plot will switch to the team view for that team. To assist selecting overlapping or tightly grouped points, or to see what teams share an area, users can use a selection tool to pick a subset of points. This triggers a popup showing all the teams selected, from which users can select a specific team from a list to go to their team view. Also available on all scatterplots are tools for zooming and download figures as images.

Lastly, the test statuses are displayed on the right hand side, both in the form of individual tests and as overall feature sets, which can be seen in 6. The individual tests are labelled with internal test code names, and the feature statuses are labeled with the feature name. These are essentially horizontal stacked bar graphs with sub-bars representing the number of teams failing, skipping, or passing that test. The bars are sorted with by proportion of failed tests, with the most failed tests at the top. Skipped tests, despite also not awarding marks, are not included in the most failed calculation because a skip is typically a symptom of failing a different prerequisite test or feature. As a result a test being skipped does not indicate people are struggling on that test. Axes for

specific numbering is omitted as relative challenge is primary question served here as opposed to exact pass or fail rates.

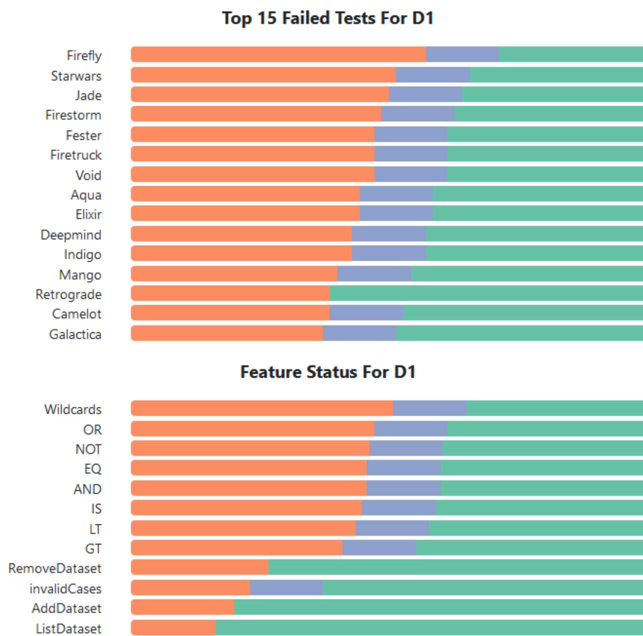

Fig. 6. The tests status display in the class view, represented as horizontal stacked bar charts. The sub-bars indicate test state, where orange is failed, blue is skipped, and green is passed. Tests are sorted by most failing first to bring attention to most challenging or potentially buggy tests. The labels in the top plot are code names for single tests, whereas the labels in the bottom are names of features comprised of multiple tests

3.2.2 Team view

Table 3. What why how analysis of the team view.

What	Tabular grade records with tests passed and grades
Why	Discover patterns or anomalies in test histories
How	Express, order
How	Shared encoding, linked highlighting, side-by-side views

The team view can be seen in Figure 2 which contains a modified version of the navigation bar, and a view of the teams tests currently and over time. The navigation bar in our team view does not contain a visualization. Instead, it holds a dropdown menu for team selection and a plain text readout of team information. This answers very simple questions of "Who is this team" and "How are they doing" that do not require any extra visualization.

The other component of this view contains two plots, a history of the status of each test, and the current status of each feature. The feature view here is nearly identical to the stacked bar chart approach used in the class tab. The one difference here is that skipped tests are collapsed into the failed category to match the output available to students [4]. The test history portion is visibly similar to the stacked bar charts due to encoding similar, however, the horizontal axis in this case represents a commit/grade instance and states will not be clustered together. This view is in essence "test by commit" matrix, with the rows visually split by vertical space. This is to help users focus on the states of individual tests over time. A closeup of part of this history can be seen in Figure 7.

[4]This simplification is due primarily to display limitations of other mediums displaying this same data.

This tab does have interaction with the user's cursor. If the user wants to focus in on a certain commit, they can mouse over it and get some lines to highlight that specific commit. Clicking will take the user to that specific commit on the students repository so they can inspect the code changes made presented there. Users can also use hover interaction to get linked highlighting to better see the relations between tests and the features they require. Hovering over a test will highlight it, along with any of the features that comprise it. Conversely, hovering over a feature will highlight it, along with any test that depends on it.

4 Implementation

In this section we discuss the technical detail of our implementation as well as what components each author did work on.

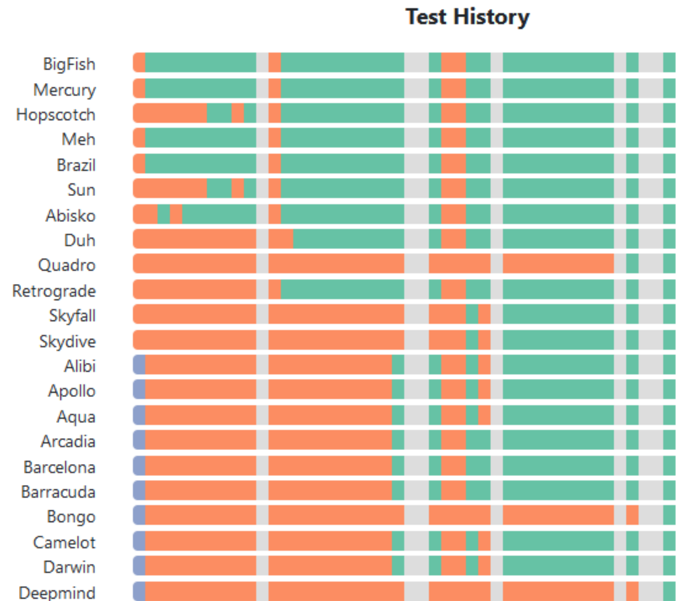

Fig. 7. The test history from the team view. Each row is a single test labeled with its code name, and each vertical slice represents a graded commit. In addition to the fail, skip, and pass colors, grey is used here to represent a failed build and the absence of a grade.

4.1 Technical Details

The first challenge of the project was data preprocessing. The processing was done in two phases. First, we filtered out a portion of commit events to include only entries for D1 and D2 made before the grading deadline. Second, we got out lines of code metric with a bash script that checks out the proper commit for each deliverable and gets summary statistics from the `src` directory. The high level coordination of this was done by using a JavaScript script that clones each team's repository and used `git` commands. This could typically be done easier through the GitHub API but that was unavailable on UBC's self-hosted GitHub instance. After gathering all the required data for our visualization we performed further data cleaning, filtering, aggregation and derivation to create a JSON structure indexed by team's name that facilitated creating our visualization.

The back-end server of our solution is a bare-bones restify server in Node.js. This is essentially a cut down version of the CPSC 310 starter code originally developed by Reid Holmes but has also been worked on by Lucas in the past.

We did the rest of the project in JavaScript along with a handful of libraries. For generating our grid layout, buttons, navigation bar, and popup dialog box in our dashboard we used Bootstrap[5]. We also used plain HTML and CSS to improve the front-end design. To make our

[5]https://getbootstrap.com/

design interactive and dynamically generate HTML fragments we used Javascript, jQuery[6] and Handlebars[7].

The scatterplots and box plot from the class view were done using Plotly.js[8]. The box plot is largely stock aside from color and layout configuration. We modified the scatterplots more, applying transparency to data points, adding on click interactions, and implementing the select behaviours.

The horizontal stacked bar plots and the test history we generate through a combination of Handlebars and Bootstrap. Handlebars is used to dynamically generate appropriate DOM elements based on the data, where they are formatted as components of a Bootstrap progress bar for styling. This functionality could be replicated with divs and background fills, the use of bootstrap here was largely a small quality of life choice as it was already in use for layout.

4.2 Work Breakdown

Due to Lucas having a difficult course with work front-loaded in the term, and Syed having a difficult course with a final and project, work ended up split primarily based on time. Syed wrote the project proposal and did the initial data pre-processing and GitHub repository mining to get our final data models. He then did the initial implementation of the class view featuring the configurable scatterplot, box plot, and the "Top Failed Test" stack bar chart. Lucas then did the initial implementation of the team view with the team feature status, test history, and the linked highlighting between the two. Then there was a period of time where they both added to and improved each others sections. Syed improved the styling of the highlighting in the test view, and Lucas added the class feature status and new data options in the parser to feed to the configurable scatterplot as suggested by a CPSC 310 instructor. At this point, Lucas largely took over and implemented most of the changes suggested by Tamara, as well as any preparation for the presentation and the first draft of this report.

5 Results

In this section we discuss some sample usage scenarios, and discuss how effective we believe our solution will be when integrated with a live course.

5.1 Usage Scenarios

Here we will walk through three possible scenarios of use for our system. All these scenarios are rooted in past events that the authors have encountered while working in the course.

5.1.1 Scenario 1: Planning a course change

In this scenario, an instructor is considering a change to a component of the project. They notice that many students write a very large number of tests in an attempt to increase their coverage score, however they often fail to think much about what they are testing, and end up writing many redundant tests. This takes up both their time and server time to run their tests, so the instructor is considering a cap on how many tests are allowed. However, setting too low of a cap could make getting a good coverage score prohibitively challenging. To remedy these concerns they can view the small multiples scatterplots on the class tab. Here they can see that the number of tests is only loosely related to the final coverage score, many teams are able to efficiently get high scores using a relatively small number of tests, despite the portion of the class writing hundreds of tests and struggling to get high coverage. However, they can also see the scatterplot showing that teams with notably low coverage often have lower test scores, so the impact of reducing students ability to cover their code could have additional indirect effects on their grade. The instructor now has more information to make an informed decision that was not available before.

5.1.2 Scenario 2: New tests introduced

In an attempt to detect an edge case that students often fail to account for, a new test has been introduced. It is just over a week into the deliverable, and course staff want to ensure that the new test is not unfairly hard on students. Looking at the class tab, the change is too small to see on the overall grades or even the feature the test falls under, but in the top failing tests view they can see that the new test is not being passed by anybody! They quickly realise that a resource required for the test was not also added to the deployed grading container, and can fix the error before students notice the test was impossible or waste time trying to make it pass.

5.1.3 Scenario 3: Team requesting TA help

A team has posted to piazza requesting help with a certain cluster and not providing much more information. A TA can go to the team view, select that team from the drop-down menu, and view their feature status and test history. Mousing over the feature they requested help with shows that a few tests needing that feature had previously passed, but some started failing recently. Hovering over the place where one test starts failing, they see another handful did as well. However, more tests became passed overall so the students may not have been too concerned. The TA can then click on that commit to go to the students repository, investigate what change they made, and provide guidance about the misunderstanding or technical error that lead to the issue.

5.2 Effectiveness of Solution

We were unable to do any proper evaluation of our system due to both the fact that the project for the term had ended, and it would require integration with existing infrastructure which will be discussed more in Section 6.2. We did get a comment from a CPSC 310 instructor that it "looks great" while giving feedback, so that is at least a good sign.

We can give our own opinions of how effective we believe the current solution to be based on our experiences as teaching assistants in the course. We believe this tool will be quite effective in enabling course staff to do their tasks in two main ways. The first is by bringing to light data that was previously completely inaccessible. Prior to this tool, there was no way to see data about number of tests or lines of code in a project other than manual inspection. Relations between tests and features has previously not been visible to most TAs as well. Secondly some of our visualizations take previously accessible data and present it in a more usable manner. Viewing team test histories is possible with current tools, but may not fit on a screen, would not show summary data about the team, and could not be easily compared to the current state of their features.

6 Discussion and Future Work

In this section we discuss the strengths of our solution, lessons learned during its creation, and limitations that may guide future work on the project.

6.1 Strengths and Lessons Learned

The most notable takeaway from the development of this project for us came from our decision to change from our initial plan of using D3.js and a Sankey-like flow diagram to bar plots and a library. Looking back it seemed we were inadvertently prioritising frameworks that might give more control to us, and idioms that would be more *interesting* to look at. This ended up colliding with the reality that we need to make a useful and usable system. Moving past these initial plans and adopting higher level libraries gave us more time to add extra content, and switching to more simple visualizations using aligned scales made each component more effective. This leads to what we think are the main strengths of our system which are the fact that it contains a wide variety of information centralized in one place, and that this information is able to be read accurately in a very short amount of time. Existing tools for the course either do not present some of this data at all, or has it scattered through different tabs or presented in tables. Our system should be both faster and more informative for users.

[6]https://jquery.com/
[7]https://handlebarsjs.com/
[8]https://plot.ly/

Fig. 8. An early prototype of the idea to use a Sankey diagram to show tests as flowing between passing and failing states. Even though this version was buggy (pass and fail bars are offset by one increment and flows from fail to pass are not displayed) it was clear that this would not be a practical idiom to use in the scenario no matter how much we may be able to polish it.

6.2 Limitations and Future Work

One limitation of our system is that it was developed fairly strictly for a standard 1080p screen. Any reduction in resolution risks having different visualization elements overlap with each other. Increasing resolution has less dramatic interference but does add extra whitespace between elements, which is both visually unappealing and makes it harder to keep two figures in view simultaneously. Ideally what we would like to see is that after a certain threshold the page reorders itself to a strictly vertical layout to be easily scrolled through, and fit well on mobile devices. Many TAs do value being able to quickly get team information on mobile devices, and this is currently completely unusable in the system's current form.

Also related to scaling, the test history in the team view should support scaling both vertically and horizontally but currently does not. Existing test suites do fit on a 1080p screen, however extended suites or smaller viewports would prohibit seeing all tests at once and introduce scrolling. Similarly absent is horizontal scaling which is needed to compensate for the fact that different teams can have very different numbers of grade records. A low number of records results in a lot of extra white space between the test history and the feature status. A high number starts overlapping with the cluster status display on the right and detail become occluded.

Our scatterplots could also be extended to show best fit lines for the data in each plot. This is supported by the Python version of Plotly however is unfortunately absent in the JavaScript version. It should be possible to manually calculate these lines and plot them manually, but we would also like to support non-linear relationships which may be more challenging to implement.

The largest challenge that needs to be dealt with is integrating the visualization with Classy and hooking it into live data instead of our preprocessed archival data. One design decision that will have to be made is whether to take data directly from the existing course database and transform it appropriately client-side, or to create a new database view to that can be queried by our front end. Thankfully one improvement that can be made here is to begin logging information such as lines of code and number of tests during the grading process so all the data used is already centralized to begin with. Additionally the implementation may need to be partially rewritten into TypeScript to match the rest of the Classy architecture, as well as adapted to provide data on the deliverable D3 which was skipped due to its ongoing changes.

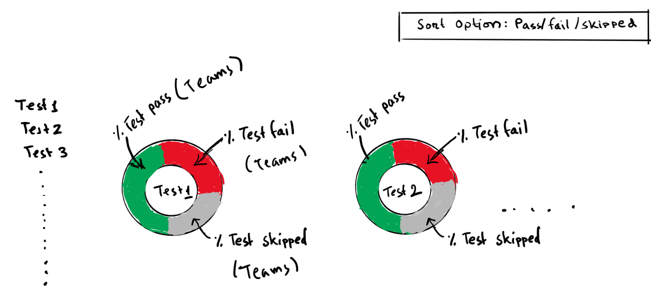

Fig. 9. An early plan for a test view, decided against in favor of the view shown in Figure 6. This view was discarded due to not being compact enough to fit many tests on the screen at one time.

7 Related Work

In this section, we look at prior work using similar visual idioms used in our system as well as previous applications of visualization in educational settings. 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 class view, they produced multiple scatterplots that plotted individual students in relation to grades and interaction with course materials. They also plotted out a course structure, 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 which aligns with the goals we had when planning this project.

Strandberg, Afzal, and Sundmark used test result data and represented test status in an overview form using circular progress graphics [4]. Different colors in the circular progress graphic depict the progress of different test results, for instance, green shows passing tests, red shows failure, while orange indicates that some tests remained

invalid, unmappable, or unloadable. This is similar to our test summary and team views represented with a color coded stacked bar charts to show proportions or instances in time that tests have a particular state. In fact, this almost exactly matches one of our early designs that we decided against in favour of the stacked bar charts. This earlier prototype can be seen in Figure 9. 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 team history graph in the team view.

Prior work has taken similar approaches to implementing scatterplots as we did in our system. Metejka, Anderson, and Fitzmaurice took a similar approach to dealing with over plotting and occlusion in scatterplots [2]. Their focus was on finding a model for scaling opacity depending on the characteristics of a plot using user survey data. This sort of model would be useful in our scenario to replace the process of manually tweaking opacity based on trial and error. Sadana and Stasko looked at designing visualizations for tablet interfaces [3]. They adopted a similar approach to using configurable scatterplots as in interface on tablets where screen real estate is limited

Vigentini, Clayphan, and Chitsaz discuss the benefits of data provided by the FutureLearn platform for MOOCs [5]. With their system, they promote data availability to help instructors. They provide customizable dashboard components through an open sourced repository. Areas of investigation they look to support include learner engagement, evaluation, education design, and research. Components provided to create a dashboard include layout options, a variety of chart types and features, and other interactions. This work has very similar intentions to ours in enriching what course staff is able to accomplish in their course.

8 Conclusion

In this paper, we described our course dashboard InsightVis designed for UBC's Introduction to Software Engineering course. It features two views, a class overview and a team view. The class view includes a box plot to show the current grade distribution of the class on the project. It also contains a small-multiples collection of scatterplots to view relationships between different course attributes. Relationships not covered by the small-multiples can be viewed using a configurable scatterplot. Test views show how the class as a whole is performing across various tests and project features. A team view shows an individual team's test history and current performance across project features. Linked highlighting allows TAs to view the links between tests and features. Interacting with the scatterplots in the class view can be used to locate teams that can be immediately inspected in the team view. These visualizations help course staff assess the state of the course, and allow teaching assistants to quickly view information about teams they are assisting.

Acknowledgments

We would like to thank Tamara Munzner, Reid Holmes, Patrick Huber, and Julia Zhu for their feedback on earlier plans and iterations of our system.

References

[1] M. Ginda, M. C. Richey, M. Cousino, and K. Börner. Visualizing learner engagement, performance, and trajectories to evaluate and optimize online course design. *PLOS ONE*, 14(5):1–19, May 2019. doi: 10.1371/journal.pone.0215964

[2] J. Matejka, F. Anderson, and G. Fitzmaurice. Dynamic opacity optimization for scatter plots. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 2707–2710. ACM, New York, NY, USA, 2015. doi: 10.1145/2702123.2702585

[3] R. Sadana and J. Stasko. Designing and implementing an interactive scatterplot visualization for a tablet computer. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, AVI '14, pp. 265–272. ACM, New York, NY, USA, 2014. doi: 10.1145/2598153.2598163

[4] P. E. Strandberg, W. Afzal, and D. Sundmark. Decision making and visualizations based on test results. In *Proc. 12th ACM/IEEE International Symp. Empirical Software Engineering and Measurement*, pp. 1–10, 2018.

[5] L. Vigentini, A. Clayphan, and M. Chitsaz. Dynamic dashboard for educators and students in FutureLearn MOOCs: Experiences and insights. p. 16.