Teamline: Visualizing small team code contributions

Nick Bradley
nbrad11@cs.ubc.ca

Felix Grund
ataraxie@cs.ubc.ca

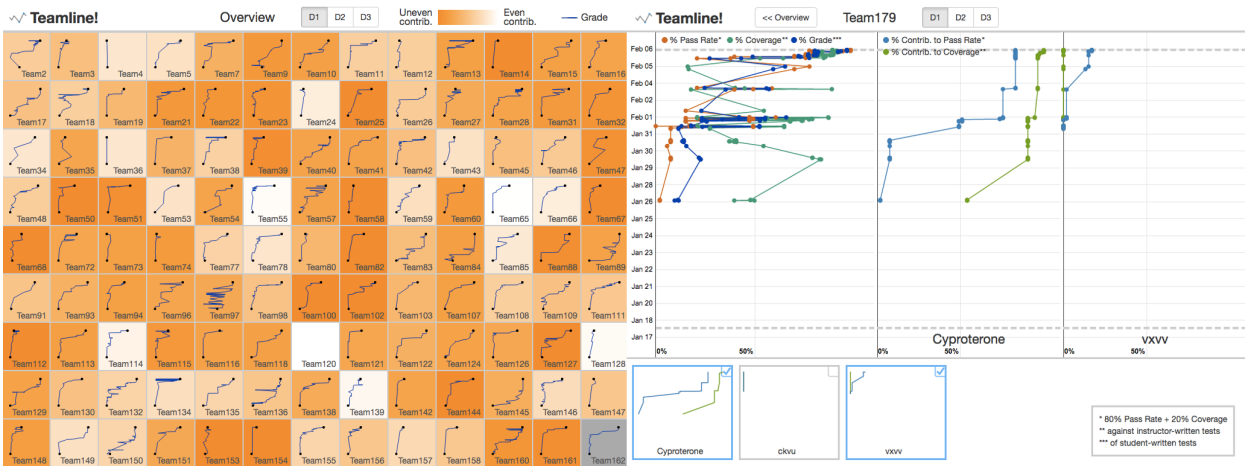


Fig. 1. The Teamline visualization. The overview is shown in the left half of the figure. In this view the user can quickly identify teams with non-uniform contributions by noting cells with higher saturation. The sparkline in the cell indicates the grade for the team over time on the y-axis. Clicking a cell takes the user to the detail view, shown on the right. Here, the grade and contributions of team members are shown in three juxtaposed charts. Comparisons among team members can be done using the gallery view at the bottom. The user can filter by deliverable in either view by selecting the corresponding button at the top of the visualization.

Abstract— Determining and understanding contributions to a shared code base can be helpful for monitoring developer efficiency, allocating work items and understanding how and why changes were made. To support this task, we introduce a method for deriving contribution using test execution results and code coverage reports. Our visualization tool, Teamline, then presents these metrics in two different views. An overview shows the uniformity of contributions across hundreds of teams. The detail view shows the test pass rate progression and offers a way to compare up to ten contributors at-a-glance with a small-multiples view. Pairs of contributors can selected to get a detailed comparison in a large side-by-side view.

1 Introduction

Writing code is a social activity that requires input from all members of a team. Scrum is a popular agile framework that emphasizes this while encouraging awareness of all the work being done by the team. To achieve this, teams run daily Scrum meetings where each developer provides a brief description of what tasks they have done thereby making explicit the contributions of each member.

Scrum works well for co-located teams where it is easy to conduct daily meetings but can be hard to implement when developers are distributed. This is common in open-source projects. Developers typically only interact with each other online so it can be hard for project managers to track individual contributions. GitHub, a popular git hosting service for open-source projects, attempts to address this issue by providing a contributors graph (Fig. 5) but it has very limited functionality.

Our tool, called Teamline[1], aspires to provide a visual means of identifying contributions of each team member. To limit the scope of the project we designed Teamline to meet the needs of a single software engineering course at UBC. The course was designed to support a limited version of Scrum but shares many of the challenges of co-located teams and thus provides a reasonably-sized use-case for us to support.

A learning outcome of this course is to develop project management and Scrum skills in the context of a programming project. In particular, students need to develop the ability to determine how equitably they are contributing to the project. To help them learn this, the teaching assistants (TAs) act as a Scrum leader in short meetings where they assess the contributions of each team member. Between the meetings, the students work as a co-located team, using online project management tools to communicate work items with each other.

It is the responsibility of the TAs to scale-back grades of students who did not make a significant contribution before a deliverable deadline. Unfortunately, this can be a hard task for the TA due to both the amount and uniqueness of code written for the project. While Teamline has been designed explicitly for the task of assisting TAs understand contributions made to the project, the framework we developed should generalize to other team-based coding projects.

This paper makes two major contributions.

- First, we define several derived attributes that are designed to indicate to what level each member of a team contributed to the overall success of a project. We also give a formula that combines individual contributions to provide a single number indicating the uniformity of contributions.

- Second, we provide a prototype visualization that uses the aforementioned contribution attributes to assist users in determining to what extent each member of a team contributed. It uses a heatmap that shows the uniformity of contribution of the users across all cells. This acts as an overview. A detail view shows the contribution of individual commits in side-by-side views, facilitating comparison over time between team members.

[1]Demo: https://nickbradley.github.io/teamline

The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 defines the domain problem and defines the data attributes and tasks. We present our visualization solution in section 4 and describe how it supports the tasks. Implementation details for Teamline are presented in section 5. Section 6 presents a scenario that shows how Teamline is intended to be used to support the intended use-case. A discussion of how well Teamline fits the scenario is given in section 7 where we also discuss future work. Finally, we conclude in section 8.

2 Related Work

Our visualization is designed to show each person's contribution to a team-based project. Previous work by Kelly et al. [7] examined whether visualizing contributions in a team-based, collaborative game leads to fairer contributions by team members. Their approach was based on using an attribute, *meters*, which is derived from existing artifacts that enable awareness of contributions in the game. We use a similarly-purposed derived attribute, *contribution*, that indicates how much each team member contributed to the overall grade. They found that only using this single attribute could undermine the efforts of collaborators if it doesn't adequately "reflect important aspects of individual work in the context of team activity" [7]. Also these tools should be combined with other methods to more robustly evaluate contribution in real-time or retrospectively. This finding supports our use-case. Teamline allows team members to see who is contributing to the overall grade of the project as it progresses and helps the TA better understand how the work was divided within the team during the retrospective meeting.

The most critical aspect of our visualization is the ability to easily and accurately compare indicators of contribution over time. Much work has been done exploring effective ways to visualize comparisons between objects. Gleicher et al. [5] gives a taxonomy of visual designs used for comparison tasks, noting that all designs are assembled using juxtaposition, superposition and explicit encodings (computing the relationships between objects and providing a visual encoding of the relationships). The authors distinguish these categories by the principal mechanism used to make connections between objects: juxtaposition uses the viewers memory; superposition uses the visual system; and explicit encodings use computation to determine the relationships. These categories can be combined to form hybrid categories. Munzner talks about these approaches in detail in [8]. Teamline takes a hybrid approach by both superimposing each contributors' metrics in the same view and by visually encoding the computed contribution score.

Our vis was inspired in part by ShiViz[2] [1] which shows messages being passed among a collection of processes to verify that the happens-before relation is not violated. We also looked at commit graphs, like the one built into BitBucket [3], which visualize commits in time, and gallery and film strip views found in most photo viewing applications.

Some visualization tools can be used to show contribution to team-based projects that use the git source control system. The most relevant to our specific use-case would be GitHub's *Contributors graph* [4]. It uses filled line graphs to show either the number of commits, number of lines added, or the number of lines deleted over time. There is an aggregate view that shows these metrics for all contributors and there are also individual contribution charts. Users can select a time range in the aggregate view to see the individual contributions only during that time range. This visualization is not sufficient for our task because it only shows the number of commits made by each team member which may not be indicative of their actual contribution to the final grade. Other tools exist that also visualize metrics exposed by git, for instance GitHub Visualizer [9], but share similar shortcomings as GitHub's Contributors graph.

3 Data and Task Abstractions

3.1 Domain Background

Our tool uses data collected by AutoTest[3], an automatic grading service used to grade code submissions for students in CPSC310. The course is structured around a term-long coding project that is divided into 5 deliverables (or sprints) completed by teams consisting of 2-3 students. The first 3 of these deliverables are graded by a combination of AutoTest and TAs. Teams manage their shared code on GitHub[4] using a basic git workflow: students pull the latest code changes from GitHub, commit their modified code locally and then push those commits to GitHub for other members to see. Every time a student pushes their changes, AutoTest is automatically invoked and runs a suite of instructor-written tests against the modified code. In addition, it computes the amount of code covered by student-written tests. Results are stored in a NoSQL database with each record corresponding to a single submission (push event).

3.2 Data Description

The grade dataset is treated as a static table which contains over 44,000 submission records for 285 students in 139 teams. From this dataset we use the attributes shown in table 1. From those attributes we created the following derived attributes:

Pass rate contribution. A student's commit is considered as contributing to the team's pass rate if one or more instructor-written tests are passed for the first time in this commit. The contribution is calculated as the number of tests passing for the first time divided by the total number of tests passed by the team. We do not visualize this value directly. Instead, we visualize the accumulated value over all commits to ensure that the plotted points are monotonic. This is a quantitative attribute ranging from 0-100%.

Coverage contribution. We consider a contribution to the coverage grade to occur when a commit increases the coverage grade beyond anything seen so far. The amount of the contribution is the amount by which which it exceeds the running maximum. Again, we accumulate this value to ensure a monotonic line. Note that the coverage is increased when a student writes (more) tests that execute a higher proportion of their code or they change the total lines of code. This is a quantitative attribute ranging from 0-100%.

Overall contribution. The overall contribution is computed as the sum of 80% pass rate contribution and 20% coverage contribution to match the weights used in computing the deliverable grade. This attribute is quantitative and ranges between 0 and 100%.

Within-team contribution uniformity (CU). This measures how evenly the team members contributed. It is a quantitative attribute that ranges from 0 to 1 where 0 indicates that one team member did all the work while 1 indicates a completely uniform workload split where each member contributed equally. This attribute is computed by taking the sorted pairwise difference of the overall contribution, computed as 80% test grade contribution and 20% coverage grade contribution, of each team member. More specifically, if m is the number of team members and u_i is the overall contribution for the ith team member, sorted by overall contribution, then we have

$$CU = 1 - \sum_{i=1}^{m-1} |u_i - u_{i+1}|. \quad (1)$$

3.3 Task Description

After a submission deadline, TAs meet with their assigned teams to conduct a retrospective to discuss any challenges that arose during the sprint and to ensure that the work was equitably distributed among the team members. This typically consists of a TA asking some questions designed to gauge a student's comprehension of the task and code. They may go so far as to explicitly and privately ask each student how evenly they felt the workload was split. Based on the retrospective, the TA assigns a scaling factor to the deliverable grade. For example, if the team got 90% on the deliverable but one member did most of the work, the final grades might be 90%*1.0 = 90% and 90%*0.6 = 54%. Unfortunately, it can be hard to determine how much work was done by each student from these conversations since the team member

[2] https://bestchai.bitbucket.io/shiviz/?

[3] http://github.com/nickbradley/autotest

[4] http://github.com

Table 1. Dataset Attributes.

Attribute Name	Attribute Type	Description
pass rate	Quantitative	Percentage of instructor-written tests that passed against student code.
coverage	Quantitative	Percentage of student code executed by student-written tests.
grade	Quantitative	Computed as 80% pass rate + 20% coverage.
timestamp	Sequential	Unix time of record creation.
commitID	Categorical	The (truncated) SHA-1 hash of the submitted commit.
githubID	Categorical[a]	The GitHub ID of the student making the submission.
team	Categorical[b]	The team number, stored as *teamXXX*, where *XXX* is a number between 2 and 199.
deliverable	Sequential	The submission deliverable, which can have values *d1*, *d2*, or *d3*.

[a] 285 values currently; max < 1000. [b] 139 values currently; max < 1000.

Table 2. What-Why-How analysis of Teamline.

What: Data	Table of graded commits with attributes described in table 1.
What: Derived	Measure of contribution to pass rate and coverage; contribution uniformity. These are discussed in section 3.2.
Why: Tasks	Present the uniformity of contributions and summarize the team's commit history. Identify teams with highly non-uniform contributions.
How: Encode	We used a heatmap with sparklines in the overview and line charts in the detail view. Marks are positioned on a common time scale with color indicating the attribute that is being encoded.
How: Facet	Overview + detail views. Detail view is partitioned into side-by-side views.
How: Embed	Superimpose sparklines on the overview's heatmap cells.
How: Reduce	Filtering is done by selecting the team and the deliverable.

who contributed very little will attempt to spoof the TA while the hard-working one may not want to rat out their partner. One possible solution is to look at the commit history on GitHub to determine how many commits each student made. This can be a decent proxy but can be misleading since different people have different commit habits (i.e. some will commit every line, others only large changes) and they may not reflect the actual contribution to the grade (i.e. commits that don't directly increase the grade).

From the description, we can identify two distinct tasks that a TA is expected to perform. The first is to understand the uniformity of the contributions. To support this task the visualization needs to present the contributions to the user in a way that makes it apparent if contributions of one person where much larger or much smaller than the others. The second task is to summarize the commit history of the team. We discuss how Teamline enables these tasks in the next section.

4 Solution

In this section we describe our visualization solution and analyze it using the What-Why-How framework provided by [8] for which table 2 provides a summary.

Teamline is faceted into two views: an **overview** providing a summary for all teams and a **detail view** for each team. The content shown in both the overview and the detail view is filtered by using the controls above the charts, labelled *D1*, *D2*, and *D3*, to indicate the corresponding deliverable. To filter by team, the user can select the corresponding heatmap cell in the overview or by updating the team number shown in the detail view.

4.1 Overview

The user is first presented the **overview** shown in figure 2[5] which uses a heatmap to encode the within-team contribution uniformity where a higher saturation indicates a more uneven contribution distribution. We initially used red for the hue but later decided on orange because we considered red to be too negative. We chose a heatmap because it is able to accommodate the approximately 200 teams in a single view without needing to scroll. Showing the contribution of every team on a single page is important so that the user is able to quickly identify teams with one member contributing less than the others. To further increase the information density and utility of the overview, we embedded sparklines in the heatmap cells. Sparklines are an ideal choice in this instance because they show trends without the extra clutter of axes found in true line charts. However, they closely resemble line charts which helps maintain consistency with the detail view. We opted to use the same color as in the detail view both for consistency and because it contrasts well with all saturation levels of the heatmap cell. The sparklines help the user see how teams are progressing based on grade and, combined with the saturation, let the user quickly identify dysfunctional teams that warrant further investigation in the detail view.

4.2 Detail View

Clicking on a heatmap cell switches to the **detail view** shown in figure 3. The detail view is based on the team that was clicked on in the overview and can be changed on the fly by changing the team name in the menu bar which will result in redrawing operations of all charts. This view includes a stage area at the top that is partitioned into three side-by-side line charts and a gallery area with a variable number of small line charts at the bottom. The stage area contains the **grade chart** (left) and two **contribution charts** (middle and right). All three charts share a common y-axis which encodes time. We chose this configuration, instead of using the x-axis to encode time, because version control systems commonly design their commit graphs that way and it allows us to take advantage of the wider screen aspect ratio of most modern consumer monitors. The x-axis is used to encode attribute values which differ based on the line chart. For the grade chart, the x-axis encodes the percentage pass rate, percentage coverage, and the overall percentage grade. The contribution charts encode the percentage contribution to pass rate and percentage contribution to coverage with the x-axis. Because all of the attributes are quantitative, we encode them using a line chart. We could have used a scatter plot but decided it was important to connect the point marks to make it easier for the user to trace changes. We also made the line chart interactive by allowing users to see a tooltip containing details about individual commits by hovering over the point marks. To help us choose reasonable colors for the chart lines representing the five attributes, we used a 5-class qualitative color scale from Color Brewer[6] [6].

5 Implementation

The implementation of Teamline consists of (1) a **backend** that retrieves our data from the AutoTest database, applies the required transformations on the data and serves it, and (2) a **frontend** that is responsible for creating the visualizations and handling user interaction in the browser. The following sections describe these two parts in detail[7].

[5] The highlighted cell is used in section 6 and can be ignored for now.

[6] http://colorbrewer2.org/

[7] See appendix A for a breakdown of work by authors.

Fig. 2. Sample overview

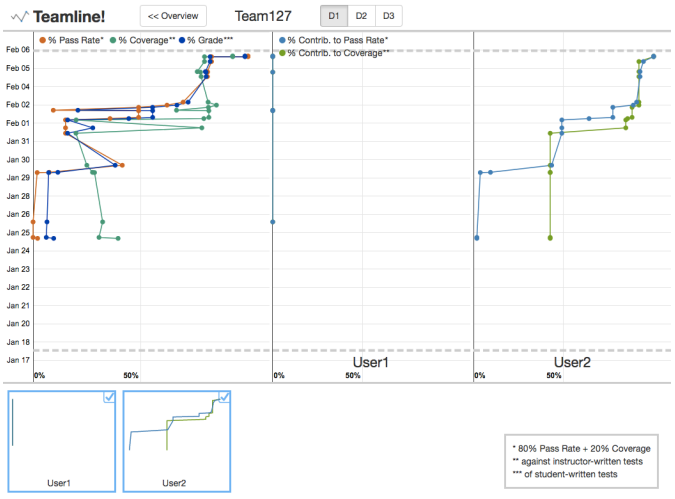


Fig. 3. Sample team view for Team127

5.1 Data Transformation (backend)

Teamline uses the CPSC310 course project data captured in a CouchDB noSQL database during term 2 of the 2016/17 year. We made a copy of this database after the project was complete and deployed it on a personal server. The database is over 14 GB and contains approximately 44,000 commit records which contain details about the committed code including the pass rate against the instructor-written tests and the percentage of code the team covered with their own tests. These fields are extracted from the database using a map-reduce view whose code is available in `contribs-by-team.js`. The view emits a list of commit records which can be optionally filtered by team, deliverable and timestamp. On top of this view, we implemented a list function, whose code is available in `contrib.js`, that takes as input the filtered list of commits and outputs a list of team objects: aggregates of all the commits made by the members of the team. It also computes the contribution metrics discussed in section 3.2.

One of the challenges encountered when preparing the dataset was learning to work with the list function framework used by CouchDB. It operates by automatically iterating over each record emitted by the underlying view which means that the programmer is responsible for keeping track of the records as they are made available. To make this part easier, we ensured that the underlying view emitted records in order first by team, then by deliverable and finally by timestamp. In this way, we were able to determine when records for a new deliverable or team by checking for differences between these fields for the current and the previous record. This check also gave us an opportunity to run aggregates over the entire deliverable or the entire team. Even with these checks implemented, we still need to keep track of which user the record was for since they are interleaved by timestamp.

The Teamline data is extracted from CouchDB by making an HTTP request to the `contribs-by-team` view without specifying any filters. The response contains a JSON file listing metrics for all the teams in the course. A complete description of the emitted fields can be found in `data-guide.md`. Currently, we save this JSON data to the file `teamline-data-min.json` for use by the fronted. Future versions of the frontend could directly request the specific data it needs from the HTTP endpoint without the need for an intermediate JSON file. The decision to use a file was made to simplify the deployment of the demo but the endpoint makes production deployment straight forward.

5.2 Visualization (frontend)

The Teamline visualization was implemented as a Web application frontend with a simple **architecture**: (1) `index.html` with HTML markup and templates, (2) `teamline.js` with program logic of the application (JavaScript) and (3) `teamline.css` with stylesheets for layout and design (CSS).

Teamline uses a number of **libraries and frameworks** aiming to maximize maintainability and reusability according to the *DRY*[8] pattern in software engineering. The Teamline stylesheet (`teamline.css`) is generated with the **SASS**[9] CSS precompiler. We used the selector nesting feature to avoid duplicate CSS selector code and variables for repetitive CSS property values. After the initial HTML markup in `index.html` is rendered on page load, numerous HTML fragments are rendered dynamically with JavaScript based on user interaction and the data received from the backend. To make this process as clean and robust as possible, we included the **Handlebars**[10] templating engine. Templates are contained as `script` tags in `index.html` with a `type="text/x-handlebars-template"` attribute, initially compiled in JavaScript on page load and then parametrized and rendered during runtime of the application. We included **jQuery**[11] for DOM manipulation and general extended JavaScript library functions since it has become a standard in Web programming and the authors are acquainted with it. The line and sparkline charts in Teamline are drawn with **NVD3**[12] which is an extension to **D3**[13] [2]. We decided on NVD3 because it supports our charting requirement with the least amount of repetitive drawing logic and exposes the underlying D3 API for scenarios when its capabilities are not enough for our use-case. We included **Moment**[14] for date formatting because we considered it more intuitive than the respective functions exposed by D3. We included **Twitter Bootstrap**[15] for simple general application styling.

The **key characteristic** of the Teamline application is the concept of *one data object and one state object* that are global to the module. The data object is populated after the data for Teamline was fetched with one initial request to the backend on page load. Teamline does not require further HTTP requests to the backend. We decided on a one-request pattern because this gives us maximum performance and the small amount of data required (~2MB) makes this approach feasible. The state object determines what data and view is visible on the screen. Whenever the state object changes, the view is updated accordingly. This approach is implemented with the *Observer pattern*[16]: events are dispatched upon state change (observable) and the respective listener (observer) updates the view. This state-based implementation enables us to redraw the current view with a simple state-update action and we

[8] http://wiki.c2.com/?DontRepeatYourself
[9] http://sass-lang.com/
[10] http://handlebarsjs.com/
[11] https://jquery.com/
[12] http://nvd3.org/
[13] https://d3js.org/
[14] https://momentjs.com/
[15] http://getbootstrap.com/
[16] http://www.oodesign.com/observer-pattern.html

can be generally open as to how the state is changed. The state object contains fields for the current `deliverableName`, `teamName`, `view` ('overview' vs. 'team') and `users` (visible in the contribution views).

The **overview** view is created as a simple container with a variable amount of sub-containers ('team cells') based on the number of teams in the current deliverable. Cells are sized to fill the available space in their parent container. The background of the cells (encoding the team's distribution of contribution) is created dynamically using the lightness value of the HSL color scale, where 100 is most equal contribution (white) and 50 is least equal contribution (dark orange). The value between 50 and 100 is calculated using the contribution numbers served by the backend for each team (see section 3.2). Sparkline charts (encoding the team's grade) are then drawn to each cell. We initially drew NVD3 line charts for each cell which confronted us with considerate performance issues and we switched to NVD3 sparkline charts in consequence which improved performance significantly (from ~8sec to ~2sec). Since the drawing operations for the sparklines are still visible to the user we only perform the drawing once for each deliverable (i.e. switching back and forth between deliverables will not result in redrawing operations). The sparkline's x-axis width is scaled back dynamically based on the team's final grade because NVD3 scales sparklines to fill the available space by default. Clicking on a team cell switches from the overview to the team view for the target team (by updating the `view` and `teamName` fields on the global state object).

The **detail view** consists of three line charts (the 'stage') and an array of thumbnail line charts (the 'gallery'). All charts are drawn as NVD3 line charts with different configuration settings. In our implementation this is visible in a call chain with different abstraction level and configuration at each step:

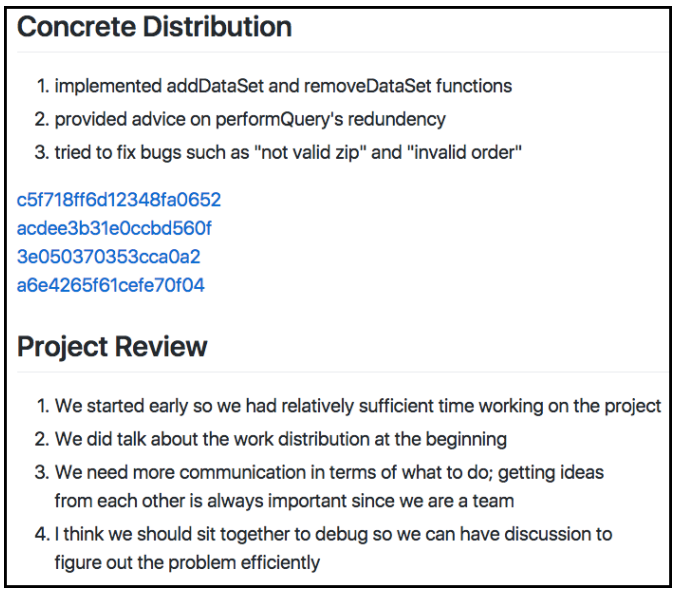
```
drawTeamCharts
  |__ drawGradeChart
    |__ drawLineChart
  |__ drawGalleryCharts
    |__ drawIndividualChart
      |__ drawLineChart
  |__ drawUserCharts
    |__ drawIndividualChart
      |__ drawLineChart
```

Changes in layout and design to the charts were mostly implemented with CSS because NVD3's configuration capabilities often turned out to be too limited for our use-case. Along the same line, we decided to implement the chart legends and the username captions on our own with a dynamic template and HTML/CSS because NVD3 did not provide us appropriate means. Tooltips are rendered dynamically upon hover, parameterizing a template with data for the target commit depending on the chart type (grade chart vs. contribution chart). Whereas the grade chart on the left remains static for the current team view, the contents of the two contribution charts on the right change dynamically depending on what users are selected in the gallery (by updating the `users` field on the global stage object).

6 Results

Teamline was designed specifically for the use-case of supporting TAs in grading students for the course *CPSC 310 Introduction to Software Engineering* at the University of British Columbia. The course is characterized by multiple deliverables and a retrospective session after each deliverable with the TA and each team member individually. Currently, students are required to hand in an individual contribution file for each deliverable that points out how they contributed to their implementation, outlining their major commits and evaluating what went well or bad during the 'sprint'. Figure 4 shows an example contribution file.

Using this contribution file as a basis, the TA will then talk to the team members and try to assess if they contributed evenly to the deliverable. The TA may use information provided by Github, like the deliverable's commit history and contribution graphs (see Figure 5) to obtain a better picture for the assessment. If an uneven contribution is visible, the TA is then to downscale the team member who contributed



Concrete Distribution

1. implemented addDataSet and removeDataSet functions
2. provided advice on performQuery's redundency
3. tried to fix bugs such as "not valid zip" and "invalid order"

c5f718ff6d12348fa0652
acdee3b31e0ccbd560f
3e050370353cca0a2
a6e4265f61cefe70f04

Project Review

1. We started early so we had relatively sufficient time working on the project
2. We did talk about the work distribution at the beginning
3. We need more communication in terms of what to do; getting ideas from each other is always important since we are a team
4. I think we should sit together to debug so we can have discussion to figure out the problem efficiently

Fig. 4. Sample contribution file

less. As described earlier, this assessment can be a very challenging task with only the information provided.

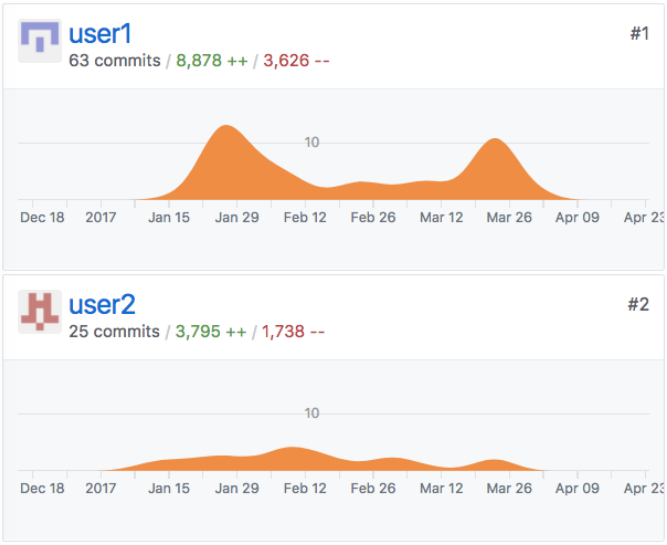


Fig. 5. Sample contribution graph showing number of commits and code churn by contributor

Teamline makes this assessment significantly easier. In this sample scenario, the TA is to assess the contribution of each member in *Team127*, from which the previous sample contribution file and graph (figures 4 and 5) were taken. In the overview, the TA is given a first picture of the team's final grade and distribution of contribution (see figure 2 with Team127 highlighted). Although the contribution file from *User1* (figure 4) would indicate that contribution to the deliverable was distributed fairly and both team members tell the TA exactly that, the TA will see immediately that the team cell has a dark orange color (highly saturated color indicating uneven distribution) and become suspicious upon the students' statements.

The TA then clicks on the table cell and the team view appears (see figure 3). The left chart shows the team's general performance with a very good final grade (blue line) of 99%. Both pass rate (orange line) and coverage (green line) switched significantly back and forth during progress on the deliverable which often indicates regression bugs in the

team's code and uncommenting parts of their code. More significant than the overall performance is the unequal distribution of contribution visible in the two contribution charts on the right. Whereas the chart for User2 on the right shows almost linear improvements in pass rate (blue line) and coverage (green line), both lines remain at 0% on the y-axis for User1 on the left side. This indicates that User1 was not responsible for any improvements on pass rate and coverage in this deliverable and contribution was highly uneven. The charts also tell that User1 is the author of only 4 commits while User2 is responsible for 16 commits, which is another indicator for uneven contribution.

In this scenario, the TA therefore has a much better picture of how each team member contributed and will very likely downscale User1 and/or give more credits to User2.

7 Discussion and Future Work

7.1 Limitations of Contribution Metrics

In section 3.2 we defined two different ways to measure contribution to a team-based coding project and here we discuss some of their limitations. The pass rate contribution is an all-or-nothing measure meaning that it only counts code that increases instructor-written tests, neglecting all supporting code. For example, it could be the case that one member writes many lines of code, commits them without passing any new tests, and then their partner makes a small change which causes several tests to pass. In this example, the partner would be given all the credit. However, this can be largely mitigated by ensuring that instructor-written tests are very focused so that small changes to the code would be captured.

Determining contribution to the coverage score can also be challenging due to the fact that small changes to either the student-written test suite or the actual code can cause large changes to the coverage score on which the coverage contribution is based. We partly address this issue by only counting changes that cause the coverage score to increase beyond the running maximum.

In both cases, the contribution measures lack robustness and are too dependent on individual metrics. For instance, neither takes into account the developer's skill or the time/effort it took for them to write the code. Also, the strong dependence of the contribution measures on the underlying pass rate and coverage make it hard to generalize to datasets that use different measures of code change and improvement. We discuss some approaches to these problems in the future work section.

7.2 Limitations of the Visualization

Time constraints prevented us from a full evaluation of Teamline but we were able to elicit feedback from members of the software practices lab; many of whom are or have been TAs for CPSC310. The most common comment was that the "axes are swapped" which lead to minor confusion when interpreting the visualization. While we noted that other commit visualization tools like `Gitk`[17] show time vertically and that comparing two line charts by stacking them can lead to visual distortion, it might be necessary to reevaluate the decision to show time on the y-axis. Another comment suggested that the class average should be shown in the grade chart in the detail view so that the TA has a benchmark for the selected team.

The grade chart in the detail view can become cluttered, obfuscating certain commits. This issue is partly due to the high variability of grades between commits which causes the lines to cross a large portion of the chart in a zig-zag fashion. Some commits are special in that the student made a request to see their grade, typically because they believe they have passed more instructor-written tests. We may wish to highlight these commits since it is likely that they would form a (nearly) monotonically increasing line. It would also be possible to use the size of the point marks to encode another quantitative attribute such as code churn or the number of lines being committed.

Finally, it could be the case that a TA would like to see how a team has progressed over deliverables. Currently, the TA would need to use their working memory and switch between each deliverable but this is not ideal. Instead, we should include an *ALL* option in the deliverable selector that would remove the deliverable filter.

[17] https://git-scm.com/docs/gitk

7.3 Future Work

As mentioned earlier, it is important that we enhance the robustness of the contribution measures. To do this, we would need to collect data for more metrics including code churn and coverage reports of instructor-written tests. With these, we would be better positioned to understand the flow of the code: where and when lines of code were changed and who made the changes. They would provide a much richer basis for deriving the code contribution attributes.

We need to evaluate Teamline against the TAs who are responsible for grading student projects. We would like to see if the tool makes it more obvious which students contribute less (and by how much) and also if it makes the task of grading qualitatively easier. A quantitative evaluation where the time and accuracy of grading is measured with and without Teamline would also be appropriate. This feedback will be critical in determining if Teamline actually solves the intended problem and would be required if we were to refine Teamline in future iterations.

Currently, the Teamline frontend uses a static JSON file containing metrics for all teams to obtain the data it needs for the visualization. However, this will not work once the tool is deployed since the data will change frequently as students make changes to their code. The solution requires that the frontend query the grade database directly to get the data. While the backend is set up to respond to such queries, the frontend will require minor changes to read data from the backend's HTTP endpoint instead of a JSON file.

Ultimately, we would like to extend Teamline to industrial software development teams so that it could assist with team awareness, task allocation and change summaries. This would likely require defining contribution based on metrics made available by version control and testing systems.

8 Conclusion

In this paper we presented Teamline, a tool that visualizes contributions to a team-based project with an emphasis on contribution uniformity. It has been designed to assist TAs in retroactively scaling back grades of students that contributed less than their partners. The Teamline visualization consists of a heatmap overview that allows users to identify teams with unequal contributions and to compare the overall grades for all teams. The detail view, which can be accessed by clicking one of the heatmap cells, shows the team's overall grade as well as the contribution scores of each team member. A gallery view shows users thumbnail versions of the contribution charts to help identify interesting contribution patterns, two of which can be selected to show in a large juxtaposed detail view.

To create the Teamline visualization, we had to define derived metrics that would show team member's contributions to the project. We based these contribution metrics on the pass rate and coverage scores since these are what the final grade is based on. While the contribution metrics are a reasonable start, we believe that more robust measures of contribution could be derived by capturing additional code metrics in the underlying dataset.

The next step is to evaluate Teamline and collect user feedback before starting the next design iteration. We hope that with reasonable effort, Teamline could be deployed to TAs in future offerings of CPSC310.

References

[1] J. Abrahamson, I. Beschastnikh, Y. Brun, and M. D. Ernst. Shedding light on distributed system executions. In *Companion Proc. 36th International Conf. Software Engineering*, ICSE Companion, pages 598–599, New York, NY, USA, 2014. ACM.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[3] C. Choate. Commit graph: The missing graph for stash, 2016. [Online; accessed Mar. 1, 2017]. Available: https://marketplace.atlassian.com/plugins/com.plugin.commitgraph.commitgraph.

[4] GitHub Inc. Viewing contribution activity in a repository, 2017. [Online; accessed Mar. 5, 2017]. Available: https://help.github.com/articles/viewing-contribution-activity-in-a-repository/.
[5] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, 2011.
[6] M. Harrower and C. A. Brewer. Colorbrewer.org: An online tool for selecting color schemes for maps. *The Cartographic*, p. 27–37, 2003.
[7] R. Kelly, L. Watts, and S. J. Payne. Can visualization of contributions support fairness in collaboration?: Findings from meters in an online game. In *Proc. 19th ACM Conf. Computer-Supported Cooperative Work & Social Computing*, CSCW, pages 664–678, New York, NY, USA, 2016. ACM.
[8] T. Munzner. *Visualization Analysis and Design*. A K Peters Visualization Series. CRC Press, 2014.
[9] A. Zubkov. Visualization github repositories history, 2016. [Online; accessed Mar. 5, 2017]. Available: http://ghv.artzub.com/.

A Task Breakdown

The work for this project was fairly evenly divided with Felix focusing on the frontend implementation and Nick preparing the data set and handling the interim components like the proposal. Table 3 shows the detailed work breakdown.

Table 3. Task Breakdown

Task	Felix	Nick
Pitch (x2)	50%	50%
Proposal	20%	80%
Project Reviews	100%	0%
Related work	20%	80%
Design	50%	50%
Implementation	80%	20%
- Database views	10%	90%
- Overview	90%	10%
- Detail	80%	20%
Presentation	20%	80%
Final paper	50%	50%
Overall	**50%**	**50%**