Teamline: Visualizing small team code contributions

CPSC 547 Project Proposal

Nick Bradley
nbrad11@cs.ubc.ca

Felix Grund
ataraxie@cs.ubc.ca

1 Introduction

Our tool visualizes data collected from AutoTest[1], 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/sprints, only the first 3 of which are graded by a combination of AutoTest and TA, completed by teams consisting of 2-3 students. The teams manage their shared code on GitHub[2] 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 private suite of tests against the modified code. Results are stored in a NoSQL database with each record corresponding to a single submission (push event). The relevant attributes are briefly described in Table 1. We have collected data for over 24,000 submissions for the first two deliverables; complete data for the third deliverable will be available on March 13. There are 285 students in 139 teams.

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. We may go so far as to explicitly and privately ask each student how evenly they felt the workload was split. The TAs assign 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 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 (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).

2 Proposed Solution

Our solution is to create a derived quantitative attribute *commitContribution* that describes the impact of a submission on the overall grade. In particular, the attribute is the difference between the current submission and the previously graded one. We visualize this along with other code metrics (Fig. 1) to gain a more complete understanding of each student's contribution: those who made more grade-improving submissions should receive a higher retrospective grade. Note that this visualization is designed to assist the TAs in making a judgment when assigning a grade and cannot replace them since students may have chosen a different way to divide the work among team members.

Here we use the what-why-how framework [1] to abstract our solution to the vis domain.

What. Table of graded submission records (items) with the attributes described in Table 1. The dataset is static once it has been loaded on the page but is dynamic in that the dataset grows with each new submission.

Why.Compare the contributions of team members and derive a retrospective grade for each of them.

How. Our vis includes many encodings and idioms:

- Separate individual submissions and align them on two (or three) parallel axes by timestamp.
- Submissions are represented by area marks. Marks that overlap are collapsed into a single point mark whose size encodes the number of submissions that were collapsed.
- Marks are coloured with luminance encoding the overall grade for the submission (max luminance for a grade of 100Note that the overall this is relatively unimportant compared to the ability to accurately compare the number and effect of submissions. This information is also available by hovering over the point mark. For these reasons, we are okay using an encoding that is of low effectiveness and may not be visible on all marks.
- Key data is always visible while interaction allows details about a submission to be seen:
  - Hovering over a single mark displays a popup with detailed information about the submission.
  - Clicking a single points opens a new browser tab that displays the corresponding commit on GitHub.
  - Hovering over a grouped mark expands it to show all submissions it includes. Once expanded, the above interactions are allowed.

3 Scenario

Imagine you are a TA tasked with scaling the final grade of each team member by the amount they contributed. Upon meeting the team, you open their Teamline to get a sense of the team dynamics: did they start early? did they work consistently? what was their final grade? This information is immediately available to you because Teamline defaults to showing the team-view of the most recently due deliverable. Within the view, the you find that the team made a few early submissions that increased their grade and then made a large number of submissions very close to the deadline. From the navigation pane, you notice that one of the team members contributed significantly to final grade unlike the other.

You then proceed to discuss with each team member individually about their contributions to the project. The one who made the larger contribution according to Teamline, Bob, had a clear understanding of the code and was able to discuss a couple of challenges he encountered. At the end of the retrospective, Bob said that each of them had done the work they agreed to do (which they thought was an even split) but that his partner started very late which made him apprehensive. To confirm this, you expand the team-view to show individual submissions made by each team member and do in fact notice that all of the submissions for the other member, Joe, were made the night before the due date and that Bobs submissions were made earlier and more consistently.

[1] http://github.com/nickbradley/autotest
[2] http://github.com

Table 1. Dataset Attributes.

Attribute Name	Attribute Type	Description
testGrade	Quantitative	Percentage of private tests that passed against student code.
coverageGrade	Quantitative	Percentage of code executed by student written tests.
finalGrade	Quantitative	Computed as 0.8*testGrade + 0.2*coverageGrade.
timestamp	Sequential	Unix time of record creation.
commitSha	Categorical	The (partial) SHA-1 hash of the submitted commit.
committer	Categorical[a]	The GitHub ID of the student making the submission.
team	Categorical[b]	The team number, stored as *teamXX*, 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.

Next, you talk to Joe. After he explains what he contributed, you mention that by starting so late, it may negatively impact the group dynamic. Joe refutes this by claiming that he started days earlier but, after you show him Teamline, agrees that he should start earlier next time. While looking at Teamline, you also notice that Bob did quite a bit more work for the previous deliverable as well. You point this out to both Bob and Joe and they are a bit surprised. You help them divide up the work for the next deliverable more equitably.

Later in the week, you decide to check how the team is proceeding. You immediately see that several submissions were made. Curious to see if your discussion helped, you expand the team-view and see that Joe has already made several submissions. You feel much more confident having scaled back Joe's grade by only 20is now contributing more.

4 Implementation Approach

We have decided to implement Teamline as a web application. We decided on this for a variety of reasons: our familiarity with web technologies, platform independence, increase likelihood of adoption in CPSC310 (and other courses that will be using AutoTest), and integration with other service used in the course. Teamline is minimally dependent on existing AutoTest infrastructure, only requiring access to the database via a REST endpoint, and is completely novel of the existing system including the dashboard.

Given the above, we will use HTML5, CSS3 and jQuery to implement our vis. With the advances in CSS, we believe we will not need a vis-specific library like D3.js but we will finalize this decision once we have started coding.

5 Expertise

We decided on this project, in part, because we are both currently TAs for CPSC310 and have experienced the challenges of assigning a fair retrospective grade. In addition, we are both excited to make use of the otherwise largely unused data.

In addition, Nick wrote and is currently managing the AutoTest system including the database. As such, he has a total understanding of the data: how it was created, its limitations, and some ways it can be meaningfully linked with other data sources like GitHub.

Finally, this project is mildly interesting from a research perspective since it is in our research area of software engineering. At a high level, it will be interesting to see how software engineering students use the git workflow to manage their project and collaborate with their team members.

6 Milestones and Schedule

We are prepared to spend about 108 hours towards this project. Table 2 provides a breakdown of the project's tasks.

7 Previous Work

References

[1] T. Munzner. *Visualization Analysis and Design*. A K Peters Visualization Series. CRC Press, 2014.

Table 2. Task Schedule.

Task	Est Time	Deadline	Description
Pitch (x2)	8	Feb. 16	Create slides, rehearse pitch.
Proposal	12	Mar. 6	Discuss project ideas, create mockups, write proposal.
Project Review 1	2	Mar. 21	Prepare slides.
Interim writeup	6	Mar. 31	Summary of progress, completed previous work section.
Project Review 2	2	Apr. 2	Prepare slides, have some version of demo ready.
Implementation	48	Apr. 7	Completed vis tool.
- Create database view	4	Mar. 14	Create view(s) of computed/derived attributes in CouchDB.
- Create tabs/buttons	4	Mar. 21	Set up project frontend. Create navigation buttons.
- Main vis (team view)	25	Mar. 31	Implement team view including fetching data, display/layout, interaction, animation.
- Main vis (student view)	15	Apr. 7	Implement student view. Some of the team view implementation should be reusable.
Presentation	10	Apr. 25	Prepare slides, demo, video(?). Rehearse.
Final paper	20	Apr. 28	Finalize paper. Draft to be written Apr. 10-18.