CAT: A Contribution Analysis Tool for Team Projects

Osman Din[*]
Georgia Institute of Technology
Atlanta, GA
USA
osman.din@gatech.edu

ABSTRACT

Contribution analysis is concerned with objectively analyzing a student's quantifiable contributions to team projects. The contention motivating this paper is that accurately evaluating student contributions is an important pedagogical task that an instructor must not leave to guesswork. As part of an educational technology course, the author created *CAT* (contribution analysis tool). This paper describes the design of this Java-based tool and the underlying principles behind its operations.

CCS Concepts

•Applied computing → Distance learning; Computer-managed instruction;

Keywords

Contribution Analysis; Team Projects; Software Engineering

1. INTRODUCTION

1.1 Importance

Researchers from different academic domains have paid significant attention to the problem of contribution analysis. The domains include electronics [8], engineering [9], accounting [11], and business studies [21, 31]. For instance, "free-riding," the phenomenon where one team member does most of the hard work and *all* team members are given a good grade, is a commonly identified problem [20, 31].

Team projects are deemed important in many domains because it is often desirable to simulate real life professional situations for students. Software engineering is an engineering discipline focused on addressing all aspects of software production [22]. The author chose the domain of software engineering projects due to the special stress on team projects in software engineering courses. In addition, software engineering emphasizes factors such as process, best practices, testing and code quality, and for each of these factors, there exist automatically verifiable metrics. For example, with respect to software quality, a plethora of metrics, objective and subjective, are available or easily attainable.

1.2 Motivation

Automating contribution analysis can reap several benefits:

- Gathering insight into student behavior by looking at how acting as an effective participant relates to the final student grade.
- Consistent grading: Random or casual grading can be detrimental to student (and team) morale, after all.
- Insight into TA behavior, by examining what class graders value, for example.
- Identifying "troubled" relationships by getting alerts for teams that exhibit an imbalance in their participation score.
- Transparency: All teams can be graded uniformly, and the effective participation criteria can be made clear easily.
- Student peace of mind: A student can more easily report any "free-riders" in a project.

The proliferation of distributed workflow tools (such as Git) and hosted solutions that make available rich statistics (such as GitHub [7]) has created interesting possibilities. Using such solutions means that the instructor can make use of important tracking and analysis data. However, the challenge is to create an interface which successfully integrates these disparate tools in a seamless way and encourages (or even enforces) the use of these tools.

2. BACKGROUND

The author found no prior art directly aimed at solving the problem of analyzing team contributions. There exist a number of related solutions, however.

2.1 Non-coding contribution analysis

Khandaker and Sho (2010) discuss an assessment framework for Wiki-based collaborations to prevent problems such as free-riding [3]. Similarly, Putro, Carbone and Sheard

[*]Draft 1.0

SIGCSE '16 2016, USA
© 2016 ACM. ISBN 123-4567-24-567/08/06. . . $15.00
DOI: 10.475/123_4

(2014) discuss an "assessment framework, which can be used to evaluate the value of a student's contributions and their interaction during wiki-based group work construction" [4].

2.2 Ranking developers in projects

A software engineering tool discussed by Nagwani and Verma (2012) is concerned with developer ranking in bug repositories. "This ranking is generated from the contribution made by the individual developers in terms of bugs fixed, severity of bugs, reporting newer problems and number of comments made by the developer."[5] The project only measures developer activity (not the quality of activity), and rankings can also go stale (contributors come and go, after all). It must also be noted that CAT focuses on the contribution assessment problem from an instructor's (rather than a software engineer's) point of view and the emphasis is on functionality (creation) rather than bug fixing(maintenance).

Similarly, Open Hub, a web directory of open source projects, has a ranking system, KudoRank. Contributors rank each other by giving a thumbs-up, but a part of the score comes from the number of hosted projects ("The busiest contributions will receive most of the points.") [29]. This is different from the requirements of the educational domain, where being busy is likely not considered the best measure of contribution, and giving each other a thumbs up might indicate subjective biases. TopCoder is another project, although close-sourced (and thus not in the public domain), that ranks developers in competitions [30].

2.3 General assessment techniques

Wilkins and Lawhead (2000) survey a range of "assessment instruments" that include early variations on the peer feedback theme (e.g., keeping a log) [1]. The authors do not argue for one assessment technique against the other. Devlin, Drummond, Philips and Marshall (2008) advocate a contribution matrix to bridge the gap between process and product contributions [2]. Incidentally, researchers have also found that students sometimes do not follow proper software development processes [18].

2.4 Predicting student performance

The paper "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance" specifies a technical approach to classify students in introductory programming courses [6]. The features used to classify students are *gender, major, grade average, programming experience*. Although the tool does not analyze student contributions, it highlights the importance of early troubleshooting.

The tools discussed here do not address the problem of assessing student coding contributions. It is also unclear from the description of these artifacts if they are extensible. The design outlined in this paper allows for both extension and customization. The novelty of the system lies in pulling together various sources of data and connecting them with pedagogical goals.

3. CONTRIBUTION ANALYSIS

3.1 Use cases or intended application

The project did not assume (much less dictate) how the tool would (or should) be used. But some of the possibilities envisioned for use are: (a) as a grading tool; (b) as an evaluation mechanism; (c) as an enforcer of standards (that indirectly conveys the "message" to the free-riders that the instructor is "watching"); (d) as a fact-finding tool (that corroborates student claims); (e) or merely as an educational tool for peer reviewers, just as Kulkarni, et al. (2013) have concluded that "giving students feedback about their grading bias increased subsequent accuracy [12]."

In addition, the use of this system could be supplemented with peer feedback tools (a variety of tools exist [13, 14, 15, 16]), as it could be hard to analyze each and every facet of a team project.

3.2 Issues

As issues and tradeoffs help determine the design of a solution, the following set of questions were considered in designing the tool.

- *What metrics should be selected (and with what weights) when ranking a contribution?* Software engineering metrics should be selected to assess the quality of code in software development projects. Pedagogical criteria (possibly coming from prior research on assessment frameworks) should be used to determine what is suitable to the needs of the class. Of course, not all coding metrics are relevant to pedagogical metrics (and vice versa). Metrics should not have an equal weight, as different instructors may want to weigh things differently, and they may want to tailor things to practical needs.

- *What is code?* Sommerville (2010) defines software as computer programs and non-code artifacts, such as documentation [22]. The focus in this project is on code. But code is also concerned with automation and maintenance (e.g., via scripting). These are ancillary tasks, and there must be a way for an instructor to specify their importance in a project. The problem of fully analyzing contributions in unstructured assignments (e.g., UML class model diagrams) is a related one, perhaps to be left for future exploration.

- *What is a contribution?* Is communicating with team members, for example, part of a student's contribution? The goal is not to capture all contribution, but to make use of only that which can be objectively measured. And yet, some of the valid concerns might be: How would one account for the number of meetings attended? How about offline dynamics? The short answer is this: Directly measuring collaboration or cooperativeness is not the aim here. Assessing collaboration is not directly tackled. Collaboration is an important related concern, however. As Vivian, Falkner & Falkner (2013) point out, "measuring by 'contribution' or 'product'. . . does not capture the dynamic and complexes skills and knowledge required to operate effective as a team member [26]." Vivian, et al., also discuss how using Dickinson and McIntrye's model of teamwork [28] (analyzing through metrics such as "Team Leadership", "Monitoring", "Giving Feedback", "Backup Behavior"), helps an instructor understand how students are performing in online teamwork [26]. It must be noted, though, that one's contribution score can indirectly reflect on one's collaboration. The tool can also be used for claim verification purposes (e.g., if someone says that he or she worked consistently on a project, but team members report that the work was low quality).

- *How to tackle students gaming the system?* An instructor should be empowered to vary the parameters of the rankings to evade any students manipulating the system.

- *What are some of the implications of different project setups and engineering methodologies?* XP, for example, employs pair coding [27]. It is not clear to what extent these methodologies are adopted in Computer Science courses (and this can be a topic for future investigation). However, a practical use case is providing an interface where an instructor would override or correct the auto-generated list of participants.

3.3 Code Analysis Methodology

- A variety of popular code analysis tools can analyze important *objective* attributes of a program, such as exception handling, code coverage ratio (e.g., 75%), type of coverage (branch, line, etc.), existence and nature of test cases (unit, integration, etc.), use of object oriented principles (e.g., inheritance), code complexity (e.g., too many inter-dependencies), and the amount of dead code. Some of these metrics have a direct relation with software quality characteristics; the number of system failures, for example, can be tied to reliability [22]. Object-oriented metrics have also been proposed by Chidamber and Kemerer [24] (a Java implementation of this design is also available [25]).

- There are some *subjective* criteria, in addition. Sommerville (2010) notes that "Some quality attributes, measuring the non-functional aspects of the system (such as maintainability and usability) cannot be objectively determined and are difficult to determine directly"[22]. The goal in this project is not to measure and advocate for such subjective non-functional requirements; the relevant subjective requirements, though, can be converted into objective requirements if they are specified in sufficient detail by a professor. For example, usability could be defined as a requirement that there be no more than three broken links on a website. A product standard must be specified by the instructor, in other words. This is analogous to the practice in the software industry, where standards play an important role in quality assurance [22].

- As mentioned earlier, Khandaker & Sho (2010) and Putro, Carbone & Sheard (2014) discuss an assessment framework that can be used to evaluate the value of a student's contributions and their interactions during wiki-based group work. The underlying techniques could be applied to analyze contributions to student project documentation (such as a project plan or user manual).

4. TOOL OVERVIEW

Tables 1 and 2 list some of the plugins and metrics that *CAT* makes use of.

Table 1: A sample rubric.

Plugin	Metric	Weight ($\sum_{i=1}^{10} x_i = 1$)
Git	No. of commits	0.2
	No. of tickets created	0.1
	No. of bug fixes	0.1
	No. of pull requests	0.1
Tests	No. of passing tests	0.0
Checkstyle	Style violations	0.1
JUnit	No. of passing tests	0.1
EclEmma	Branch coverage	0.1
	Line coverage	0.1
Wiki	Number of figures added	0.1

Table 2: Select plugins

Source or interface	Role
GitHub API	Repository statistics
EclEmma Jacoco	Code coverage
Sonar	Code quality
Checkstyle	Style compatibility
OpenNLP	Readability in user manuals
PMD	Static analysis

4.1 Design

A monolithic design, or a design that hard-codes business logic decisions and dependencies, would have been inappropriate. Different instructors have different notions of relevancy, and although this project took a stance on some issues (by selecting some metrics and omitting others for practical reasons), the design was kept flexible to suit the needs of the circumstances. The score ranking mechanism was made customizable, allowing different weights to be assigned to different metrics. On an architectural level, *CAT* was implemented as a modular application, thereby making it possible to allow related contribution analysis plugins to be customized, added and removed. The program was written in Java, to allow extensibility and probability. The program relies on open source libraries and software such as PrimeFaces (JSF library), JSF (MVC), Apache Maven (Java build tool), Apache Tomcat (servlet container), Apache Derby (embedded relational database), JUnit (unit testing).

There are three important logical components: (a) Contribution analysis component; (b) Data source (e.g., a source code repository) integration component; (c) Connectors (e.g., a PMD static analysis plugin that analyzes data from a repository). The tool does not download and run Git builds itself, but relays this task to Jenkins, a popular open-source continuous-integration (CI) server. Jenkins runs various plugins as part of the build process and provides relevant statistics via a RESTful API.

Jenkins also allows various build and code analysis plugins to be customized, added, and removed, thus making it making it possible for the tool to be extended. As long as a connector is available or can be written for Jenkins, the input source can be made available as part of the tool. Table 1 highlights some of these inputs and the weights an instructor might attach to them.

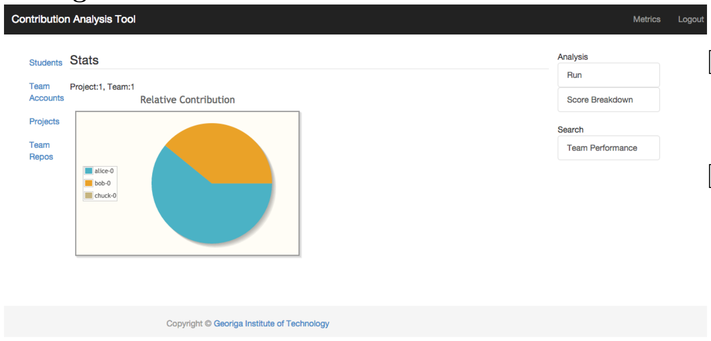
4.2 The application

To use the tool, an instructor would create (perhaps programmatically) GitHub repositories for the teams, assign students to teams, provide students access to their team repositories, and list all team repositories in a CSV file.

The tool then imports all necessary CSV files and updates the user interface with the new data. The instructor can then run the analysis on the provided data.

Some of the pertinent features of the application are: (a) integration with external build systems; (b) authentication and authorization; (c) visual comparison of student performances; (d) bulk data import; and (e) data export for external analysis. Figure 1 shows an application screenshot.

Figure 1: Screenshot: Relative contributions



5. FUTURE WORK

The author would like to add more plugins to the application. Integration with other academic systems (such as a Learning Management System) might be greatly beneficial to instructors as well. Finally, using the tool in a real class would allow a comparison between manual and automated grading.

6. SUMMARY

This paper discusses the important problem of contribution analysis and how an extensible web application that allows weighing of metrics, customization of analysis, and assimilation of statistics from multiple sources can help solve this problem. The developed tool does not require that some particular type of projects be used as input or that some metric *must* figure as a part of the student grade. Indeed, as long as relevant plugins can be developed and added to the system, the tool strives to meet the project's twin objectives of flexibility and extensibility.

7. REFERENCES

[1] Dawn E. Wilkins and Pamela B. Lawhead. 2000. Evaluating individuals in team projects. In Proceedings of the thirty-first SIGCSE technical symposium on Computer science education (SIGCSE '00), Susan Haller (Ed.). ACM, New York, NY, USA, 172-175. DOI=10.1145/330908.331849 http://doi.acm.org/10.1145/330908.331849

[2] Devlin, M., Drummond, S., Phillips, C. and Marshall, L. Improving Assessment in Software Engineering Student Team Projects. In 9th Annual Conference of the Subject Centre for Information and Computer Sciences, 26th-28th August 2008, Liverpool Hope University White, H. (ed.), Higher Education Academy, Subject Centre for ICS , pp 133-139, 2008.

[3] Khandaker, N.; Leen-Kiat Soh,"Assessing individual contributions to groupwork by a summary of tracked computer-supported collaborative activities," in Frontiers in Education Conference (FIE), 2010 IEEE , vol., no., pp.S1E-1-S1E-6, 27-30 Oct. 2010 doi: 10.1109/FIE.2010.5673505

[4] Iwan Handoyo Putro, Angela Carbone, and Judy Sheard. 2014. Developing a framework to assess students' contributions during wiki construction. In Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148 (ACE '14), Jacqueline Whalley and Daryl D'Souza (Eds.), Vol. 148. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 123-131.

[5] Naresh Kumar Nagwani, Shrish Verma. Rank-Me: A Java Tool for Ranking Team Members in Software Bug Repositories. Journal of Software Engineering and Applications Vol.5 No.4, Pub. Date: April 23, 2012.

[6] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. In Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15). ACM, New York, NY, USA, 121-130. DOI=10.1145/2787622.2787717 http://doi.acm.org/10.1145/2787622.2787717

[7] Statistics. https://developer.github.com/v3/repos/statistics/. Accessed: 2015-08-29

[8] Lockwood, J.W., "Automated team project management and evaluation through interactive web modules," in Microelectronic Systems Education, 1999. MSE'99. IEEE International Conference on , vol., no., pp.26-27, 1999 doi: 10.1109/MSE.1999.787020

[9] DeFranco, J., Neill, C., "Improving Individual Learning in Software Engineering Team Projects", American Society for Engineering Education, Atlanta Georgia, 2013.

[10] Judith L. Gersting and Frank H. Young. 1998. Contributions of the working student. SIGCSE Bull. 30, 2 (June 1998), 26-27. DOI=10.1145/292422.292434 http://doi.acm.org/10.1145/292422.292434

[11] Susan C. Lambert, Amanda J. Carter, and Margaret Lightbody (2014) Taking the Guesswork Out of Assessing Individual Contributions to Group Work Assignments. Issues in Accounting Education: February 2014, Vol. 29, No. 1, pp. 169-180.

[12] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, and Scott R. Klemmer. 2013. Peer and self assessment in massive online classes. ACM Trans. Comput.-Hum. Interact. 20, 6, Article 33 (December 2013), 31 pages. DOI=10.1145/2505057 http://doi.acm.org/10.1145/2505057

[13] TEAMMATES project. https://github.com/TEAMMATES. Accessed: 2015-08-29

[14] Tool overview. http://peerreview.cis.unimelb.edu.au/tools/tool-overview/. Accessed:

2015-08-29

[15] Crucible. https://www.atlassian.com/software/crucible/overview. Accessed: 2015-08-29

[16] PeerScholar. http://www.pearsoned.ca/highered/peerscholar/. Accessed: 2015-08-29

[17] Schneider, J.-G.; Vasa, R., "Agile practices in software development - experiences from student projects," in Software Engineering Conference, 2006. Australian , vol., no., pp.10 pp.-410, 18-21 April 2006 doi: 10.1109/ASWEC.2006.9

[18] Younessi, H.; Grant, D.D., "Using the CMM to evaluate student SE projects," in Software Engineering: Education and Practice, 1996. Proceedings. International Conference , vol., no., pp.386-391, 24-27 Jan 1996 doi: 10.1109/SEEP.1996.534025

[19] How can I assess group work? https://www.cmu.edu/teaching/designteach/design/instructionalstrategies/groupprojects/assess.html. Accessed: 2015-08-29.

[20] Louwarnoud van der Duim, Jesper Andersson, and Marco Sinnema. 2007. Good Practices for Educational Software Engineering Projects. In Proceedings of the 29th international conference on Software Engineering (ICSE '07). IEEE Computer Society, Washington, DC, USA, 698-707. DOI=10.1109/ICSE.2007.40 http://dx.doi.org/10.1109/ICSE.2007.40

[21] Tollefson, E. Evaluating Peer Contributions to Group Work. Master Teacher Program thesis. United States Military Academy, West Point, NY.

[22] Ian Sommerville. 2010. Software Engineering (9th ed.). Addison-Wesley Publishing Company, USA.

[23] Prosa UML Modeller and SA/SD/RT Modeller. http://www.prosa.fi/. Accessed: 2015-09-10.

[24] S. R. Chidamber and C. F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. IEEE Trans. Softw. Eng. 20, 6 (June 1994), 476-493. DOI=10.1109/32.295895 http://dx.doi.org/10.1109/32.295895

[25] Spinellis, D., "Tool writing: a forgotten art? (software tools)," in Software, IEEE , vol.22, no.4, pp.9-11, July-Aug. 2005 doi: 10.1109/MS.2005.111

[26] Rebecca Vivian, Katrina Falkner, and Nickolas Falkner. 2013. Analysing computer science students' teamwork role adoption in an online self-organised teamwork activity. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13). ACM, New York, NY, USA, 105-114. DOI=10.1145/2526968.2526980 http://doi.acm.org/10.1145/2526968.2526980

[27] Hedin, G., L. Bendix, and B. Magnusson, Introducing software engineering by means of Extreme Programming, in Proceedings of the 25th international conference on Software engineering. 2003, IEEE Computer Society. p. 586–593.

[28] Dickinson, T. and McIntyre, R. 2009. A conceptual framework for teamwork measurement, in Team Performance Assessment and Measurement: Theory, Methods, and Applications, M. Bannick, E. Salas, and C. Prince, (Eds). Lawrence Erlbaum Associates: Mawah, New Jersey.

[29] Kudos. http://blog.openhub.net/kudos/. Accessed: 2015-08-29

[30] TopCoder. https://www.topcoder.com/. Accessed 2015-09-11.

[31] Cinebell, S., Cinebell, J., Stecher, M. 2010, A Stapler Does Not Make it a Team Project: An Examination of Collaboration in Student Teams in Journal of the Academy of Business Education.