Measuring Team Members' Contributions in Software Engineering Projects using Git-driven Technology

Reza M. Parizi
Department of Software Engineering and Game Development
Kennesaw State University
Marietta, GA 30060, USA
rparizi1@kennesaw.edu

Paola Spoletini
Department of Software Engineering and Game Development
Kennesaw State University
Marietta, GA 30060, USA
pspoleti@kennesaw.edu

Amritraj Singh
Department of Software Engineering and Game Development
Kennesaw State University
Marietta, GA 30060, USA
amritra@students.kennesaw.edu

***Abstract*—Software engineering is inherently a human-centric and collaborative process and this reflects in its teaching programs, as most of the courses comprise projects and team efforts. In order to fairly evaluate students, there is the problem of quantifying the amount of work contributed to the team development project by each of its members. Most commonly, in order to estimates student contributions, instructors use arbitrary and subjective judgment derived from observations and evaluations. The currently used process is not a complete picture and is time consuming since it requires numerous observations and extensive paperwork's review. Emerging decentralized systems (such as git) and their widespread applications in all realms of development which capitalize on team-aware metrics, are worthwhile and can provide a solution to the problem. In this work we support a solution that utilizes git-driven technology, and its related features, to measure a team member's contributions objectively, based not only upon the completion of the project, but also at any time during progression development. Such performance assessment could generate more productive team-based learning with higher-quality graduates for better meeting software industry's expectations.**

Keywords—Software engineering education, Team performance measurement, Team-based learning, Contribution metrics, Git technology

I. INTRODUCTION

The majority of courses in software engineering programs involve developmental team-based projects. Teamwork skills are crucial and even essential for software engineering professionals [1]. However, it is always a challenge to capture and quantify the amount of effort and work contributed to a project by its members (students) in software engineering courses, mainly due to the unavailability of free quantitative performance tools within a team [2], [3], [4]. In addition to the information collected through the interactions with the team, subjective and non-quantitative approaches by instructors have largely been the way to estimate contributions and performance of data provided by student documentations [5], [6], [7]. To complicate matters, data which is provided may sometimes be inaccurate or biased in favor of certain team members which can lead to unfair assessment and distribution of grades.

The emergence of decentralized version control systems, (e.g., git [8] – the most widely used technology among educators and industry fellows), and their widespread applications in all realms of software development, has created a gold mine of resources and potential data for mining team and project-specific analytics and insights. Inspired by this opportunity, we propose to build a solution that utilizes git-driven technology and its related features to measure a team member's contributions objectively, based not only upon the completion of the project but also during the development at any given time by involved stakeholders. This kind of objective monitoring and measurement could create more productive team-based learning [9], as it may help team leaders and supervisors to provide in-time feedback to student teams who might be struggling or not exerting enough effort for the successful completion of the software project. Moreover, this tool-supported approach will assist instructors to fairly evaluate each individual of a team on the basis of quantitative measurement data instead of subjective estimation. As an indirect and complementary result, this approach will encourage and motivate students to be more actively engaged by providing meaningful contributions and efforts, and ultimately benefit universities to produce high quality and competent software engineering graduates for the industry.

The rest of this paper is structured as follows: Section II presents related work and background. Section III describes the details of the proposed solution. Section IV discusses the preliminary empirical assessment and results. Section V mentions the conclusion and future work.

II. RELATED WORK

Le et al. [10] proposed a solution to generate formative assessment feedback automatically by using text data mining techniques. By utilizing the students' individual weekly logs and the team's project plan, their proposed solution was built upon different text similarity techniques to match work done against work planned. Similarly, Lima et al. [11] designed a suite for the repository mining-based metrics for the assessment of developer contribution, based on evidence gathered from the project and team leaders. Nguyen and Chua [12] created a rubric that assesses the progress information of each team member and provides a formative performance feedback on how each

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

member is contributed to the project. Chen et al. [1] suggested a heuristic approach that assesses a student team performance in software engineering undergraduate projects. Yin et al. [13] proposed a 3-dimensional performance measurement model to help project managers improve team collaboration by indicating strengths and weaknesses of team members during the project development process. Despite some positive results, none of these works embeds Version Control System (VCS)-driven and decentralized technology-strength measurements in their techniques to provide automated assessment.

On the market side, GitPrime [14] is mainly a git-based tool for software engineering leaders/managers to understand and advocate for their team with a visual service using graphs and charts. Although scalable and promising, the service provided by this tool is not suitable for academic environments due to its high monthly cost and complexity. Moreover, since the service targets large-scale environments, it is hard for a small academic team to get personal demos from the service provider.

Our work aims to mitigate existing work limitations by eliminating the excessive cost of usage of such service, e.g., GitPrime, by being open source. In fact, the software engineering research and education community has long relied on Free and Open Source Software (FOSS) projects and tools with which to conduct and handle a wide variety of tasks. Moreover, our work is the first of its kind that considers the difficulty aspect of the problem a team member/developer is working on while evaluating performance and contribution to a project (discussed in Section IV.B). Another positive point of our proposed solution is that it would provide an estimate of time spent on a project by each individual team member which can prove beneficial during performance evaluation and especially in an academic environment. Finally, our solution is also flexible: the weightage provided on the derived metrics (See section III and IV) can be calibrated according to the nature of the software project, and the difficulty of the problem a team member is working on, will work to analyze performance.

III. Proposed Solution

The goal of our work is to realize a multi-dimensional service in which users provide a version control repository-specific information (i.e., the URL of a Git repository for a project). The system then generates the contribution reports (based on a set of metrics) with rendered visual graphs relevant to each project team member/developer or the entire team itself.

The basic model of our proposed solution and the data extraction process for team contribution and performance evaluation is shown in Fig. 1. The key questions regarding the build of the model, as shown in Fig.1, are discussed as below:

A. *Where input/initial data obtained?*

We use data from GitHub, GitLab, BitBucket sources – literally any git-based code repository.

B. *What are the input data?*

We use the git logs as inputs for a given project repository to extract the performance evaluation analytical data.

C. *How are software project related data extracted?*

We associatively mine the git log for a repository of each member/author and derive metrics, as discussed in III.D, from the associated git VCS.

D. *What are the resulting metrics?*

In our context, metrics are measurements of the contributions and efforts in the software project repositories. Metrics get created from logs and each one represents a data dimension. In our initial configuration, we propose five metrics, including, *number of commits*, *number of merge pull requests*, *number of files* and *total lines of code*. Additionally, we calculate the *time spent* on a project each day.

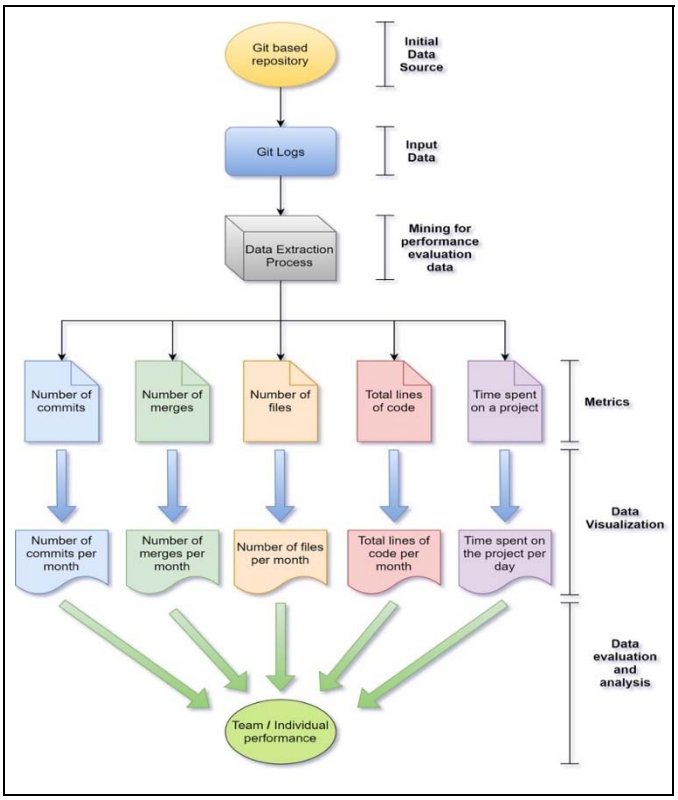

Fig. 1. The component-view model of the proposed solution

E. *How is time spent on the project measured?*

We use a simple algorithm for estimating time [15]. The algorithm for the calculation process is presented in Fig. 2. The process starts by going through all commits and comparing the difference between them in time. If the difference is smaller or equal to a given threshold, it groups the commits to a single coding session. If the difference is larger than a given threshold, the coding session is considered "*complete*". In our context, by default, a good threshold is two hours, as longer coding sessions might be unrealistic due to fatigue of a developer. Additionally, to compensate for the initial commit whose work is unknown, it adds an extra one hour to the coding session. The process continues until all coding sessions have been determined and the hours completed by individual developers have been summed up.

F. What to visualize and display as outputs?

The extracted data, based on the metric suit, can be plotted on several graphs for visualization and performance reviews. These graphs can be printed for the entire team or for an individual team member relevant to the time, including:

- Number of commits per month
- Number of merges per month
- Number of files per month
- Total lines of code per month
- Time spent on the project per day

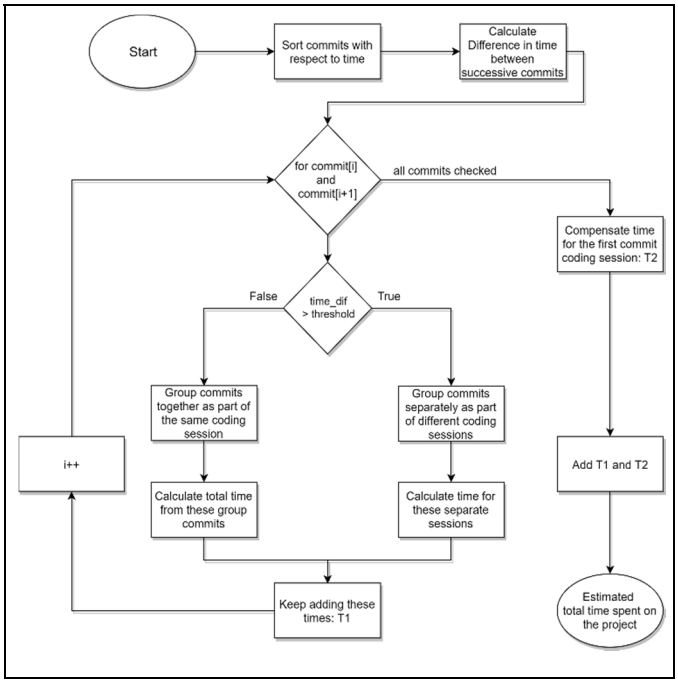

Fig. 2. Flowchart for time calculation algorithm

TABLE I. WEIGHTAGE SCHEME ON EXTRACTED DATA

Commits /month (C)[a]	Merges/ month (M)[b]	Files/ month (F)[c]	LoC/month (L)[d]	Time spent/ day (T)[e]	Weight age
70+	22+	25+	1k+	8+ hrs.	1.0
[60 – 70)	[20 - 22)	[22 – 25)	[0.9k – 1k)	[7.5 – 8)	0.9
[50 – 60)	[18 - 20)	[20 - 22)	[0.8k – 0.9k)	[7 – 7.5)	0.8
[40 – 50)	[15 - 18)	[17 - 20)	[0.7k – 0.8k)	[6.5 – 7)	0.7
[30 – 40)	[12 - 15)	[15 - 17)	[0.6k – 0.7k)	[6 – 6.5)	0.6
[25 – 30)	[10 - 12)	[13 - 15)	[0.5k – 0.6k)	[5 - 6)	0.5
[20 – 25)	[8 - 10)	[10 - 13)	[0.4k – 0.5k)	[4 – 5)	0.4
[15 – 20)	[6 - 8)	[8 - 10)	[0.3k – 0.4k)	[3 – 4)	0.3
[10 – 15)	[4 - 6)	[5 - 8)	[0.2k – 0.3k)	[2 – 3)	0.2
Below 10	Below 4	Below 5	Below 0.2k	Below 2 hrs.	0.1

[a] C = Total Number of Commits per month, [b] M = Total Number of Merges per month
[c] F = Total Number of Files added per month, [d] L = Total Number of Lines of Code per month,
[e] T = Time spent on the project per day

With the extracted metrics-based data, the system will additionally categorize/rank the developers based on their performance and contribution on the project into five classes (*Excellent*, *Good*, *Satisfactory*, *Poor* and *Unacceptable*). Table I provides our proposed weightage information for different metric used by the system. When more than one metric is chosen for performance analysis, the overall performance categorization is based on the average score of all the individual metrics weightages. Additionally, Table II provides the information regarding the score range for estimated performance categorization mentioned above.

TABLE II. ESTIMATED PERFORMANCE ON AVERAGE SCORE

Estimated Performance	Score
Excellent	0.8+
Good	[0.6 – 0.8)
Satisfactory	[0.4 - 0.6)
Poor	[0.2 – 0.4)
Unacceptable	Below 0.2

The weightage and performance classification in Table I, II and IV are provided based on careful assessment of authors' experiences, feedback from professional developers, student developers and professors within academic environments. It should be noted that this weightage distribution is just a preliminary setup of a developer's performance and contribution. It can be adjusted depending on the application nature or other project-specific factors at any time before having it fed to the tool. For now, we will be using these weightages to provide a sample performance analysis of an open-source project.

IV. ANALYSIS AND DISCUSSION

To illustrate the benefits of our proposed solution, this section presents an empirical evaluation and results.

A. Initial Evaluation

We performed a preliminary experiment on a fairly large open-source project (to advocate the scalability and flexibility of our work) named 'Atom' [16], [17] to report the project related commitment extracted data as discussed in the previous section. Following the model and metrics, data were extracted from the project's git repository for five randomly chosen contributors. For the sake of privacy of the contributors, their names will remain anonymous. We will instead replace contributor names by Author A, B and so on. Moreover, to produce an unbiased initial performance categorization, the authors were not in contact with these contributors.

Table III provides information regarding the Average *commits per month* (a single dimension) and an estimate of the initial performance categorization of each author for the month, considering the references in Table I and Table II.

TABLE III. INDIVIDUAL PERFORMANCE DATA

Authors	Average Commits /Month	Estimated Performance
A	74	Excellent
B	120	Excellent
C	29	Satisfactory
D	35	Good
E	13	Poor

Fig. 3 shows the total number of commits per month since April 2017 by the entire team for the 'Atom' project. All the graphs presented in this work are low-fidelity prototype samples and are not generated by our work in progress tool at this time. We will be discussing these graphs in explicit detail in the definitive version of this paper.

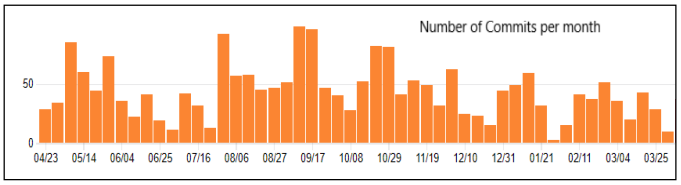

Fig. 3. Total number of commits each month since April 2017

Similarly, Fig. 4 shows the total number of commits for the author A since 2012. The figure suggests that Author A is making an excellent contribution to the project. Moreover, Table II shows that Authors A, B, C, and D are all making decent contributions to the project.

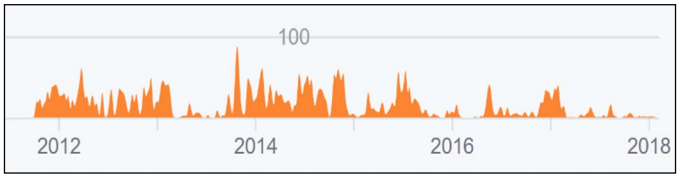

Fig. 4. Author A's commits since 2012

However, from the results in Table III, one can also see that author E has a significantly lower number of monthly commits as compared to the other contributors. It makes sense to conclude that author E is not contributing to the project as required. In most cases, the derived analytics show the real side of the contributor's performance of a software project. In some cases, however, it might not be entirely true, leading to a phenomenon called *premature analysis*. This phenomenon refers to the situations where data provided by the tool might require further analysis before a consistent performance evaluation is made (as discussed in IV.B).

B. Discussion

A casual look at Table III might suggest that author A is working significantly harder on the 'Atom' project than Author E who has 'Poor' estimated performance on the project. But what might not have been considered in this analysis is that there is a possibility that author E could be working on a significantly harder engineering problem than author A. In which case, the lower number of monthly commits by author E can be considered as a completely normal behavior. Hence, the average number of commits over time (or any other specific metrics for that matter) might not tell the entire story about a team member's effort.

To mitigate this threat and provide more consistent evaluation, we propose to include an additional component, named "*difficulty gauge*" to the model. In this component, we strongly advocate that the performance should always be analyzed based on *all* extracted data (not just the number of commits per month or any other single metric) while taking into consideration the *difficulty* of the problem a particular team member/developer is assigned.

TABLE IV. DIFFICULTY WEIGHTAGE AND PERFORMANCE FORMULA

Difficulty level	Weightage	Performance Formula
Extremely Easy	1	(C + M + F + L +T) / 5
Easy	1.5	((C + M + F + L +T) / 5) * 1.5
Normal	2	((C + M + F + L +T) / 5) * 2
Challenging	2.5	((C + M + F + L +T) / 5) * 2.5
Extremely Challenging	3	((C + M + F + L +T) / 5) * 3

In light of the difficulty gauge, Table IV provides our proposed information on preliminary weightage assignment to the difficulty level of the problem a developer is working on, along with the adjusted formula for performance evaluation based on this weightage. Similar to the weightage distribution in Table I, the weightage in this table can be adjusted by the project leader/supervisor as per the project nature and requirements.

Consequently, combined with this "*difficulty gauge*" approach, the extracted data can provide a more thorough and comprehensive performance evaluation of a team member. In an academic environment, students can keep track of their performance and can provide help or feedback to teammates struggling to complete an assigned task related to the project. Additionally, the tool can help professors evaluate student contributions to the assigned project and depending less on current ways of evaluation based on peer reviews and student documentation. Moreover, our work can also provide critical data in regard to a team's performance in professional environments. By analyzing which who needs it, the project managers could then assist a team or key individuals to mitigate/eliminate concerns and risks. As a result, managers would be able to see where the greatest impact is being made, identify areas to give concrete feedback, and help teams understand how the process changes impact the team's effectiveness.

V. CONCLUSION AND FUTURE WORK

In this work, we proposed an architectural model of a git-driven tool that establishes several performance evaluation metrics, including the number of commits, merges, files and lines of code over time, and time spent on a project per day for team members in software development projects. The extracted data are meant to be plotted on several graphs for visualization and performance reviews while categorizing each individual class alongside. In the evaluation of the performance of an individual team member, it is extremely important to keep in mind the "difficulty" of the problem an individual team member is working on and combine this information with all other extracted data to make a thorough and comprehensive analysis of the performance of that team member. The consideration of difficulty of a project has not been discussed in any of the state of the art work.

In future work, we plan to implement the FOSS version of the proposed solution using pluggable architecture to accommodate a dynamic pipeline of performance metrics, including code reuse statistics. With roots in the open source, we believe the future tool would able to evolve more quickly by the interested users. It would be also considered more trustworthy with the source code open and available to the community.

REFERENCES

[1] J. Chen, G. Qiu, L. Yuan, L. Zhang, and G. Lu, "Assessing Teamwork Performance in Software Engineering Education: A Case in a Software Engineering Undergraduate Course," in *18th Asia-Pacific Software Engineering Conference*, 2011, pp. 17–24.

[2] J. E. Sims-Knight, R. L. Upchurch, T. A. Powers, S. Haden, and R. Topciu, "Teams in software engineering education," in *32nd Annual Frontiers in Education*, 2002, p. S3G–17–S3G–22.

[3] R. F. Gamble and M. L. Hale, "Assessing individual performance in Agile undergraduate software engineering teams," in *2013 IEEE Frontiers in Education Conference (FIE)*, 2013, pp. 1678–1684.

[4] M. Hale, N. Jorgenson, and R. Gamble, "Predicting individual performance in student project teams," in *24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, 2011, pp. 11–20.

[5] J. B. Main and M. Sanchez-Pena, "Student evaluations of team members: Is there gender bias?," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–6.

[6] S. Northrup and D. Northrup, "Multidisciplinary Teamwork Assessment: Individual Contributions and Interdisciplinary Interaction," in *36th Annual Frontiers in Education*, 2006, pp. 15–20.

[7] P. K. Imbrie, J. C. Immekus, and S. J. Maller, "Work In Progress - A Model to Evaluate Team Effectiveness," in *35th Annual Frontiers in Education*, 2005, p. T4F–12–T4F–13.

[8] S. Chacon and B. Straub, *Pro Git*, 2nd Editio. Apress, 2014.

[9] M. A. Busseri and J. M. Palmer, "Improving teamwork: the effect of self-assessment on construction design teams," *Des. Stud.*, vol. 21, no. 3, pp. 223–238, May 2000.

[10] K. Le, C. Chua, and R. Wang, "Mining Software Engineering Team Project Work Logs to Generate Formative Assessment," in *24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, 2017, pp. 78–83.

[11] J. Lima, C. Treude, F. F. Filho, and U. Kulesza, "Assessing developer contribution with repository mining-based metrics," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 536–540.

[12] T. Nguyen and C. Chua, "Predictive Tool for Software Team Performance," in *23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016, pp. 373–376.

[13] Yuanyuan Yin, Shengfeng Qin, and Ray Holland, "Development of a project level performance measurement model for improving collaborative design team work," in *12th International Conference on Computer Supported Cooperative Work in Design*, 2008, pp. 135–140.

[14] "GitPrime." [Online]. Available: https://www.gitprime.com/. [Accessed: 09-Apr-2018].

[15] K. Brunfeldt, "Git Hours." [Online]. Available: https://github.com/kimmobrunfeldt/git-hours. [Accessed: 04-Apr-2018].

[16] Atom, "Atom." [Online]. Available: https://atom.io/. [Accessed: 08-Apr-2018].

[17] "Atom on GitHub." [Online]. Available: https://github.com/atom/atom. [Accessed: 08-Apr-2018].