 DOI: 10.35382/TVUJS.13.3.2023.2430

ANALYZING GIT LOG IN AN CODE-QUALITY-AWARE AUTOMATED PROGRAMMING ASSESSMENT SYSTEM: A CASE STUDY

Bao-An Nguyen[1], Thuy-Vi Ha Thi[2*], Hsi-Min Chen[3]

Abstract – *Automated programming assessment systems have transformed the evaluation of programming assignments, providing detailed feedback and reducing instructors' workload. This paper explores the benefits of Git log analysis in ProgEdu, a code-quality-aware automated programming assessment system. ProgEdu was utilized for assessing Java homework assignments and web programming projects over two semesters. The integration of Git log analysis in ProgEdu highlights its potential in tracking student progress, predicting performance, determining student groups based on submission behaviors, identifying inequality in group projects, and facilitating instructors' intervention. The study emphasizes the importance of enhancing software industrial practices in programming courses, including code version control, static code quality checking, unit testing, and automation tools. By incorporating these practices, students benefit from hands-on learning and situated learning experiences. Embracing these practices enhances the learning experience, improves student performance, and fosters a collaborative programming environment. It highlights the benefits for students and instructors, urging institutions to invest in software industrial practices and demonstrating the potential impact on programming education.*

Keywords: APAS, code quality, data analytics, programming education.

[1,2]Tra Vinh University, Vietnam
[3]Feng Chia University, Taichung City, Taiwan
*Corresponding author: hattvi201084@tvu.edu.vn
Received date: 12th July 2023; Revised date: 28th September 2023; Accepted date: 29th September 2023

I. INTRODUCTION

In recent years, the field of programming education has witnessed a significant transformation with the introduction of Automated Programming Assessment Systems (APASs). These systems have revolutionized the way programming assignments are evaluated, providing educators and students with numerous advantages and opportunities. APASs leverage the power of automation and technology to streamline the assessment process, offering efficient and objective evaluation of code submissions. By automating the grading process, APASs reduce the burden on instructors, freeing up valuable time that can be dedicated to other aspects of teaching and learning. Moreover, APAS provides consistent and fair evaluations by applying predefined criteria consistently to all submissions, minimizing subjective biases [1]. The incorporation of APASs also enables the provision of immediate and detailed feedback to students, allowing them to understand their mistakes, learn from them, and enhance their coding skills. Additionally, APAS facilitates the tracking of student progress over time, providing educators with valuable insights into individual learning trajectories and allowing for personalized interventions when needed [2]. By integrating APAS into programming courses, students have the opportunity to enhance their programming skills through repetitive practice, fostering a sense of mastery. Furthermore, the incorporation of APAS can stimulate students' enthusiasm for learning programming through the introduction of peer competition [3]. Fruitful results were obtained from implementing APASs effectively, such as the integration of code-quality analysis, scalability, and adaptability to different programming languages and assignment types

[4]. Overall, APAS represents a powerful tool that has the potential to enhance programming education by automating assessment processes, providing timely feedback, and optimizing the learning experience for both educators and students.

The analysis of log data APAS plays a crucial role in gaining insights into student performance, behavior, and progress. The log data collected by the APAS captures valuable information about students' coding activities, interactions with the system, and submission history. One key aspect of log data analysis is the ability to predict student performance. By examining patterns in coding behavior, such as frequency of submissions, code quality, and revision history, the APAS can make informed predictions about a student's overall performance [5, 6]. This predictive capability allows instructors to identify struggling students early on and provide targeted interventions to support their learning journey. Another valuable application of log data analysis is the identification of submission behaviors and patterns. By analyzing the timestamps and frequency of submissions, instructors can gain insights into student engagement and adherence to deadlines. This information can be used to identify students who consistently submit their work late or exhibit erratic submission behavior, enabling instructors to address these issues and promote better time management skills [7]. Furthermore, log data analysis in APAS facilitates the identification of group project dynamics and inequalities. By analyzing the contributions made by individual group members, the APAS can uncover disparities in effort and participation, highlighting potential issues of free-riding or unequal workload distribution [8]. This insight allows instructors to address these inequalities and promote a more equitable and collaborative group work environment. Lastly, log data analysis enables the tracking of student progress over time. By examining the evolution of students' code submissions, instructors can assess their growth, identify areas of improvement, and provide targeted feedback. This longitudinal view of student progress helps instructors personalize their teaching approaches and support individual student needs effectively [9]. This paper explores the benefits of incorporating Git log analysis in ProgEdu, a code-quality-aware automated programming assessment system. Over two identical semesters, ProgEdu was utilized to assess Java homework assignments and web programming projects. The integration of Git log analysis within ProgEdu highlights its potential in tracking student learning progress, predicting student performance, determining student groups based on submission behaviors, identifying inequality in group projects, and facilitating timely intervention by instructors. This study also emphasizes the importance of institutions enhancing software industrial practices, such as code version control, static code quality checking, unit testing, and automation tools, in programming courses. By implementing these practices, students not only learn through hands-on experience but also benefit from situated learning.

The results of this study strongly support the broader implementation of code-quality-aware automated programming assessment systems and the integration of Git log analysis as an essential component. This study emphasizes the importance of institutions investing in software industrial practices and highlights how Git log analysis enhances the effectiveness of programming educational tasks.

II. BACKGROUND AND RELATED WORKS

A. The ProgEdu System

ProgEdu is an innovative APAS that simulates the workflow of DevOps, incorporating essential software industrial practices into the learning process of programming [10]. The system has been designed with two primary rationales. Firstly, it aims to reduce the workload of instructors by automating the assessment process and promoting iterative learning, allowing students to learn from their mistakes and improve through a feedback-driven approach. Secondly, ProgEdu aims to promote situated learning by providing

students with early exposure to software industrial practices, including code version control, coding conventions, static code quality analysis, unit testing, continuous integration/continuous deployment (CI/CD), and DevOps principles.

ProgEdu is a docker-based application that combines several components to fulfill its objectives. It includes a front-end website that acts as a learning management system (LMS), where students can access their assignments, submit their code, and receive feedback. Additionally, ProgEdu integrates with GitLab, serving as an instance of a code version control service, enabling students to manage their code changes and collaborate on group projects effectively. Furthermore, an instance of Jenkins is utilized as a CI server, where students' source code is built, and various quality checks, vulnerability scans, security assessments, and code smell detections are performed. The results of these assessments are then provided back to the LMS, allowing both students and instructors to review and learn from the outcomes.

By combining the functionalities of LMS, GitLab, and Jenkins, ProgEdu provides a comprehensive and seamless platform for students to engage in hands-on learning, practice industry-standard coding practices, and receive timely feedback on their work. This approach empowers students to develop their programming skills while gaining practical experience with the tools and processes commonly utilized in real-world software development environments.

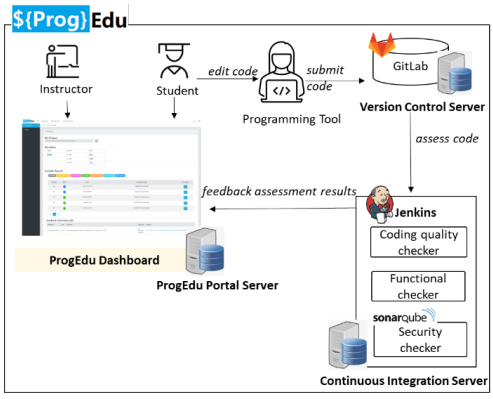

Fig. 1: ProgEdu Architecture

B. *Literature review*

Standalone Integrated Development Environments (IDEs) like Visual Studio and NetBeans or text editors like Visual Studio Code and Sublime Text have traditionally been the primary tools used in programming education; however, they fail to provide insights into student learning progress. Code version control systems, such as Git, have emerged as viable options due to their ability to incrementally track source code changes and offer valuable information to educators. The incorporation of Git as a submission tool within APASs opens up a new long longitudinal view enabling the identification of patterns, tracking of improvement, and targeted feedback provision to enhance the learning experience [11]. Moreover, Git-based APASs foster collaborative learning environments by leveraging Git's collaboration features. Students can engage in group projects, track their contributions, and learn from each other's code [12, 13].

Git-based assessment systems, which are drawing inspiration from collaborative practices prevalent in the software industry, have garnered attention in evaluating students' performance in collaborative learning, [14, 15]. Neyem et al. [16] introduced an empirical approach for software engineering (SE) courses, which involved weekly coursework consultation sessions facilitated by a project tracking tool. This tool employed GitHub as a hosting platform to enhance project traceability and aid students in understanding the implementation and testing of requirements. Gary et al. [17] proposed a Continuous Integration (CI) and Test platform to provide continuous assessment and feedback on students' activities, supporting their collaborative work in SE projects. Raibulet et al. [18] utilized GitHub, Microsoft Project, and the code inspection tool SonarQube to foster collaboration among students in SE projects. In a different context, Eraslan et al. [19] utilized the GitLab repository as a hosting platform for project artifacts and

incorporated GitLab metrics, such as commit count and the number of assigned and closed issues, as fundamental information for coursework consultation sessions in their SE courses. While these studies demonstrated the viability of integrating industrial tools and practices into student software projects, the learning data collected in the learning progress were not appropriately considered. Our proposed scheme aims to address this research gap by analyzing the Git log to bring better insights to educators.

III. METHODOLOGY

ProgEdu was implemented in two identical courses at Feng Chia University in Taiwan over the span of two semesters. The first course focused on Object-Oriented Programming using Java, while the second course centered around Web Programming involving HTML, CSS, and JavaScript. To foster iterative learning and encourage students to learn from their mistakes, an unlimited resubmission policy within ProgEdu was implemented. This policy allowed students to receive feedback from the system and make improvements to their programming skills. In order to familiarize students with industry practices early on, the research emphasized the importance of adhering to coding conventions by conducting static code quality checks on each submission. Consequently, students were required to ensure that their submitted code passed syntax checks, code quality assessments, and unit testing before reaching a successful state.

A. Java homework assigments

In the first course, ProgEdu was employed to evaluate six Java homework assignments. The assessment outcomes for each submission encompassed Compile Failure (CPF), Unit Test Failure (UTF), Check Style Failure (CSF), and BuildSuccess. To analyze the students' homework performance, their efforts based on the frequency of each result and tracked their time spent in specific statuses was measured. Additionally, the study considered factors such as the total number of submissions, the number of on-time submissions, and the duration taken to start and complete the assignments.

B. Web programming team projects

In the second course, ProgEdu was utilized as the submission system for team projects focused on web programming. Students were tasked with designing webpages using HTML, CSS, and JavaScript. As a result, the status of the source code may fall into values such as HtmlCSF, CssCSF, or BuildSuccess. To assess the individual contributions, an evaluation of the transitions was made by each team member between different source code statuses. These transitions are depicted in Figure 2. Correspondingly, contributions of an individual student are represented by features listed in Table 2.

The team project aims to measure the balance in contributions among members, thus the Gini index, a popular metrics to measure the inequality in economics was employed for this purpose. Average values of the team on these measures were also computed. We used the prefixes Gini- and Avg- to denote these mesures.

In addition, to compare the contributions among members, the Contribution Indexwas employed to calculate for each feature in Table 2 (Feature names have the prefix CI-). The CI of a measure M of student S is calculated as the ratio of contributions of student S per average contributions of the team regarding metric M:

$$CI.M_S = \frac{M_S}{M_T}$$

C. The analysis workflow

To explore insights into students' learning progress using ProgEdu, this research established a data analytics workflow as depicted in Figure 3. The first step involved extracting data from the log database and computing the values of relevant metrics as described in this section. Subsequently, features were formulated for individual students or teams based on the computed metrics. The data was then analyzed using descriptive statistics and visualizations to enable

Table 1: List of behavioral features of students in homework submissions

ID	Group	Partial-level feature name *(eg. for homework i)*	Overall-level feature name	Description
1	Homework effort	$CPF.Count_i$	*Total.CPF.Count*	#submissions given CPF
2		$CSF.Count_i$	*Total.CSF.Count*	#submissions given CSF
3		$UTF.Count_i$	*Total.UTF.Count*	#submissions given UTF
4		$Sub.Count_i$	$Total.Sub.Count_i$	#submissions
5		$Ontime.Sub.Count_i$	*Total.Ontime.Sub. Count*	#on time submissions
6	Homework time *(in minutes)*	$Time.To.Start_i$	*Avg.Time.To.Start*	Time to first submission
7		$Time.To.End_i$	*Avg.Time.To.End*	Time to last submission
8		$CPF.Dur_i$	*Avg.CPF.Dur*	Duration staying in CPF
9		$CSF.Dur_i$	*Avg.CSF.Dur*	Duration staying in CSF
10		$UTF.Dur_i$	*Avg.UTF.Dur*	Duration staying in UTF

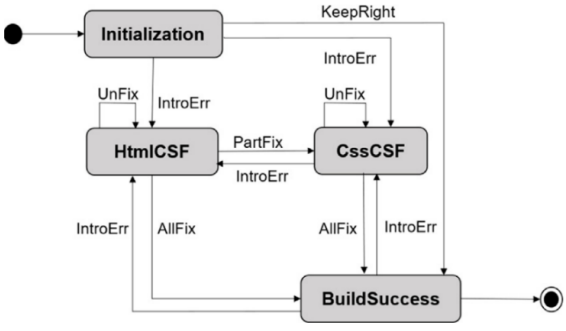

Fig. 2: Transitions between project statuses

Table 2: Individual features of students in team project

ID	Features	Description
1	*SubmTotal*	Number of submissions
2	*BuildSucc*	Number of *BuildSuccess*
3	*HtmlCSF*	Number of CSF in HTML code
4	*CsssCSF*	Number of CSF in CSS code
5	*IntroErr*	Number of *IntrErr* transitions
6	*KeepRight*	Number of *KeepRight* transitions
7	*UnFix*	Number of *UnFix* transitions
8	*PartFix*	Number of *PartErr* transitions
9	*AllFix*	Number of *AllFix* transitions

instructors to monitor students' learning progress effectively. In the next stage, machine learning techniques were employed for predictive analysis. Unsupervised learning algorithms were utilized to identify groups of students or teams exhibiting similar behavioral learning patterns. Additionally, supervised learning was employed to identify at-risk students or free riders within teams.

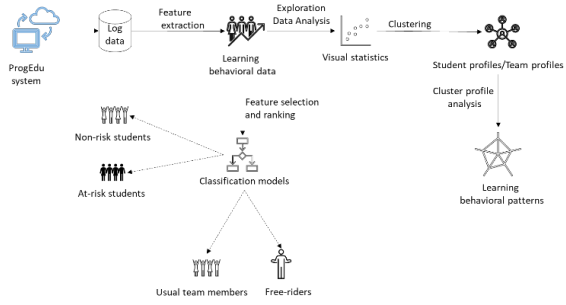

Fig. 3: The data analysis workflow

IV. RESULTS AND DISCUSSIONS

A. *Tracking submission progress*

Using the log data extracted from the system, it becomes straightforward to track students' learning behaviors in both homework assignments and team projects by plotting their submissions on a dot-plot with a time axis, as illustrated in Figure 4 and Figure 5. These visualizations serve as a reflection tool for students and provides valuable insights for instructor interventions.

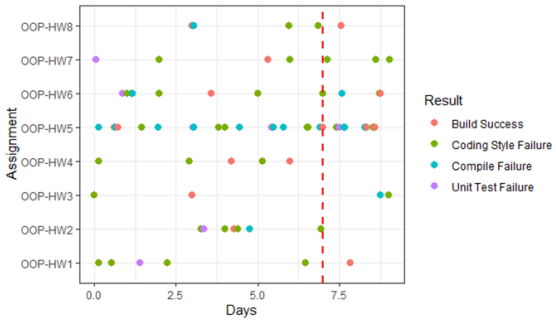

Fig. 4: Visualization of homework submission trace and assessment results of a student (the red vertical line denotes the deadline)

In both scenarios, instructors can easily observe situations where students encounter difficulties, such as issues with check style or multiple failures in unit tests, allowing them to provide timely support. Additionally, the dot plots reveal patterns of late or infrequent submissions, which further facilitate interventions from teachers.

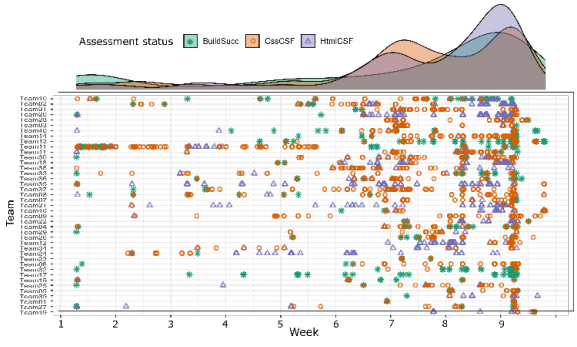

Fig. 5: Visualization of teams' submissions and assessment results in web programming projects

The dot plots generated from the log data also highlight potential areas for improvement in students' learning processes. By analyzing the distribution of submissions and identifying clusters or gaps in the timeline, instructors can identify patterns of procrastination or inconsistent engagement. Moreover, the visual representation of students' learning behaviors through the dot plots offers a comprehensive overview of the entire class's progress. Instructors can identify common challenges faced by the majority of students and adjust their teaching methods accordingly. It also allows for the identification of high-performing students who may serve as mentors or peer leaders, promoting a collaborative and supportive learning environment.

Overall, the utilization of dot plots based on log data in the ProgEdu system provides instructors with valuable insights into students' learning experiences, enabling them to offer timely support, identify areas for improvement, and foster a more productive and engaging learning environment.

B. Identifying students' profiles with unsupervised learning

In programming homework assignments, with behavioral features extracted from the log data, the authors performed unsupervised learning tasks to identify distinct groups of students with varying learning patterns in programming. An example of this analysis is depicted in Figure 6, where k-means clustering was applied to two dimensions: submission efforts (total number of submissions) and learning achievement (final scores). The names of the clusters were assigned based on the relationship between learning styles and corresponding outcomes as follows:

C1. Effective learners: These learners are positioned at the top-left corner of the figure. They demonstrate exceptional programming problem-solving skills, achieving high scores with minimal iterations. They exhibit proactive behavior by starting and completing their homework promptly. Moreover, they do not submit additional attempts once they have achieved their first success.

C2. High-effort learners: Positioned in the top-right of the figure, high-effort learners display comparable final grades to effective learners. They make a considerable number of submissions and exhibit determination in studying and iteratively solving problems. Despite taking longer to achieve success compared to effective learners, they persevere and eventually solve the problems, albeit not in the most efficient manner.

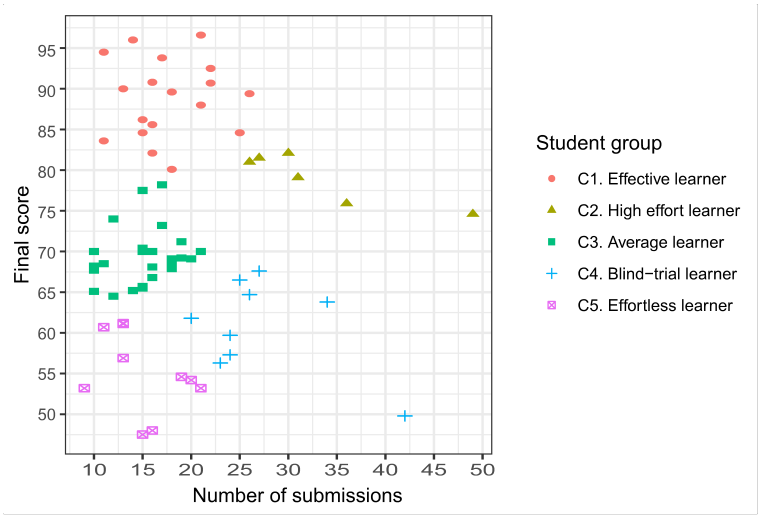

Fig. 6: Five clusters of student based on learning behaviors and learning achievement

C3. Average learners: Found in the middle-left of the figure, average learners exhibit a moderate number of submissions, falling between effective and high-effort learners. Their programming effort is at an average level, and they appear to be content with their performance. While their final results are acceptable, they are lower than those of effective and high-effort learners.

C4. Blind-trial learners: Positioned at the bottom-right of the figure, blind-trial learners rely heavily on the submission system, making numerous attempts. However, despite their extensive trials, they do not achieve results comparable to effective and high-effort learners. Some of them even fail to pass the course.

C5. Effortless learners: Plotted at the bottom-left corner of the figure, effortless learners receive the lowest scores with a minimal number of iterations. They tend to give up early and refrain from resubmitting their code after encountering failures. Consequently, they have a high likelihood of failing the course.

By examining these distinct learner groups, the study gains valuable insights into their learning behaviors, effort levels, problem-solving approaches, and corresponding outcomes, which can inform instructional interventions and support individual student needs.

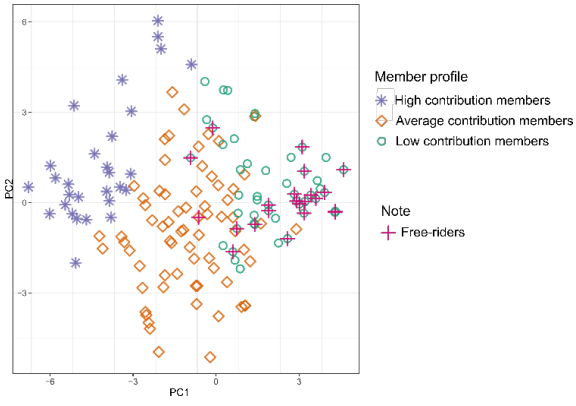

Fig. 7: Distribution of member profiles and free riders in 2D-space (reduced by using PCA)

In a similar vein, the study employed a segmentation approach to group students based on their levels of contribution to the team project. By utilizing the plain values and contribution index of metrics presented in Table 2, the Ward's minimum variance in an agglomerative hierarchical clustering method was utilized to identify three distinct clusters: high contribution members, average contribution members, and low-contribution members, as depicted in Figure 7. It is important to note that the

instructors had already identified the free riders during the final presentation, and these individuals were classified as low-contribution members in the clustering analysis. Through this data analysis framework, instructors are equipped with comprehensive information regarding the contributions of individual students, allowing for more informed grading decisions rather than solely relying on interviews or peer reviews.

C. Identifying at-risk students using supervised learning

In programming courses, the correlation between learning patterns and academic outcomes holds significant value in assisting both educators and students. To address this, the prediction models that enable the classification of students based on their learning behaviors during the early stages of the course have been constructed. The targeted variable in this model is the final result of students, which encompasses two possibilities: pass or fail. Students identified within the fail group are classified as at-risk students and are notified to make adjustments to their learning behaviors to enhance their learning outcomes. In this experiment, the data from 69 students of which 11 could not pass the course. Using just three homework assignments before the midterm examination, the author managed to attain a predictive accuracy of 70% utilizing the K-Nearest Neighbors (KNN) algorithm. The prediction evaluation by 10-fold cross-validation KNN, Naive Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and AdaBoost (AB) is presented in Table 3.

Table 3: Cross validation results for prediction models of at-risk students

Method	**AUC**	**Acc**	**Non-risk**			**At-risk**		
			F1	***Pr***	***Re***	***F1***	***Pr***	***Re***
AB	0.73	0.80	0.86	0.86	0.86	0.61	0.61	0.61
KNN	**0.87**	**0.87**	**0.91**	**0.89**	**0.94**	**0.73**	**0.80**	0.67
NB	0.81	0.75	0.81	0.93	0.73	0.64	0.52	**0.83**
RF	0.85	0.78	0.85	0.86	0.84	0.60	0.58	0.61
SVM	0.68	0.71	0.82	0.77	0.86	0.33	0.41	0.28
DT	0.80	0.83	0.88	0.88	0.88	0.67	0.67	0.67

The results demonstrate the effectiveness of the prediction models in identifying and classifying at-risk students. The high accuracy and balanced values of precision, recall, and F1-score indicate the reliability and validity of the models in predicting at-risk students and facilitating timely interventions.

D. Discussions

The primary objective of conducting Git log analysis in ProgEdu is to uncover concealed insights from the learning progress and present them to the relevant stakeholders. These analyses serve as the foundation for designing and enhancing a learning dashboard within the system. Reflection is a central aim of any learning analytics tool, and the dashboard effectively informs both students and instructors about the current status of individuals and teams. Key metrics such as SubmCount, CSF sequence length, TE, TCI, and others provide valuable information that empowers students to regulate their learning behaviors and enables instructors to determine when and how to intervene. In addition to fostering reflection, prediction capabilities enable early interventions or warnings that alert students about critical situations, such as the risk of failing the course or receiving low final grades, and the potential consequences of unaddressed technical debt in the final product. Moreover, the features extracted for prediction models facilitate student modeling, leading to recommendations that can be presented through visualizations like radar plots. Leveraging the level of student engagement depicted in the dashboard, interventions can be automatically generated or manually provided by instructors.

For instructors, the Git log analysis in ProgEdu provides a means to track student learning outcomes and identify areas where additional support may be required. It enables them to monitor individual and team performance, identify patterns and trends, and make data-driven decisions to enhance the teaching and learning experience.

Moreover, students can benefit from Git log analysis in ProgEdu by gaining insights into

their own programming journey. They can track their progress, identify areas of improvement, and receive timely feedback on their coding practices. This analysis serves as a reflection tool, allowing students to self-regulate their learning behavior and make adjustments to achieve better learning outcomes. Overall, the Git log analysis task within ProgEdu contributes to a more effective and tailored approach to programming education. By leveraging code-quality-aware analytics, instructors and students can collaborate in a structured manner, leading to improved programming skills and a deeper understanding of software development principles.

V. CONCLUSIONS

The current study delves into the Git log analysis task within ProgEdu, a code-quality-aware Assessment and Programming Assistance System (APAS). The primary objective of this analysis is to provide assistance to both instructors and students in the realm of programming learning. By leveraging Git log analysis, ProgEdu offers valuable insights into the learning process, allowing instructors to gain a deeper understanding of students' programming skills and progress. This analysis takes into account code quality metrics, such as syntax checking, code quality checking, and unit testing, to provide a comprehensive overview of students' coding abilities.

Based on the findings of this study, a strong case can be made for higher educational institutions to prioritize the tracking of students' learning progress through the utilization of digital tools, rather than relying solely on traditional term exams. By incorporating a range of tools and technologies, educators can gather valuable insights into students' behavioral patterns, so that, educators can gather valuable evidence, provide targeted feedback, and deliver effective interventions, ultimately enhancing the quality of education and supporting student success.

REFERENCES

[1] Alemán JLF. Automated assessment in a programming tools course. *IEEE Transactions on Education*. 2011;54(4): 576-581. DOI: 10.1109/TE.2010.2098442.

[2] López-Pernas S, Saqr M, Viberg O. Putting it all together: Combining learning analytics methods and data sources to understand students' approaches to learning programming. *Sustainability*. 2021;13(9): 4825. https://doi.org/10.3390/su13094825.

[3] Cheng LC, Li W, Tseng JC. Effects of an automated programming assessment system on the learning performances of experienced and novice learners. *Interactive Learning Environments*. 2021: 1–17. https://doi.org/10.1080/10494820.2021.2006237.

[4] Mekterović I, Brkić L, Milašinović B, Baranović M. Building a Comprehensive Automated Programming Assessment System. *IEEE Access*. 2020;8: 81154–81172. https://doi.org/10.1109/ACCESS.2020.2990980.

[5] Chen HM, Nguyen BA, Yan YX, Dow CR. Analysis of learning behavior in an automated programming assessment environment: A code quality perspective. *IEEE Access*. 2020;8: 167341–167354. https://doi.org/10.1109/access.2020.3024102.

[6] Karavirta V, Korhonen A, Malmi L. On the use of resubmissions in automatic assessment systems. *Computer Science Education*. 2006;16(3): 229–240. https://doi.org/10.1080/08993400600912426.

[7] Rico-Juan JR, Sánchez-Cartagena VM, Valero-Mas JJ, Gallego AJ. Identifying student profiles within online judge systems using explainable artificial intelligence. *IEEE Transactions on Learning Technologies*. 2023: 1–14. https://doi.org/10.1109/tlt.2023.3239110.

[8] Nguyen BA, Chen HM, Dow CR. Identifying nonconformities in contributions to programming projects: from an engagement perspective in improving code quality. *Behaviour & Information Technology*. 2023;42(1): 141–157. https://doi.org/10.1080/0144929X.2021.2017483.

[9] Pereira FD, Oliveira EH, Oliveira DB, Cristea AI, Carvalho LS, Fonseca SC, et al. Using learning analytics in the Amazonas: Understanding students' behaviour in introductory programming. *British Journal of Educational Technology*. 2020;51(4): 955–972. https://doi.org/10.1111/bjet.12953.

[10] Chen HM, Chen WH, Lee CC. An automated assessment system for analysis of coding convention violations in Java programming assignments. *Journal of Information Science and Engineering*. 2018;34(5): 1203–1221. https://doi.org/10.6688/JISE.201809_34(5).0006.

[11] Kelleher J. Employing git in the classroom. In: *Proceedings of 2014 World Congress on Computer Applications and Information Systems (WCCAIS), 17-19 January 2014, Hammamet, Tunisia*. TX, USA: IEEE; 2014. p.1–4. https://doi.org/10.1109/WCCAIS.2014.6916568.

[12] Gitinabard N, Okoilu R, Xu Y, Heckman S, Barnes T, Lynch C. Student Teamwork on Programming Projects: What can GitHub logs show us ?. To be

published in EDM 2020. *arXiv*. [Preprint]. 2020. https://doi.org/10.48550/arXiv.2008.11262.

[13] Beckman MD, Çetinkaya-Rundel M, Horton NJ, Rundel CW, Sullivan AJ, Tackett M. Implementing version control with git and github as a learning objective in statistics and data science courses. *Journal of Statistics and Data Science Education*. 2021;29: 132–144. https://doi.org/101080/1069189820201848485.

[14] Haaranen L, Lehtinen T. Teaching git on the side - Version control system as a course platform. In: *ITiCSE '15: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. New York, USA: Association for Computing Machinery; 2015. p.87–92. https://doi.org/10.1145/2729094.2742608.

[15] Zagalsky A, Feliciano J, Storey MA, Zhao Y, Wang, W. The emergence of github as a collaborative platform for education. *CSCW '15: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. New York, USA: Association for Computing Machinery; 2015. p.1906–1917. https://doi.org/10.1145/2675133.2675284.

[16] Neyem A, Benedetto JI, Chacon AF. Improving software engineering education through an empirical approach: lessons learned from capstone teaching experiences. In: *Proceedings of the 45th ACM technical symposium on Computer science education*. New York, USA: Association for Computing Machinery; 2014. p.391–396. https://doi.org/10.1145/2538862.2538920.

[17] Gary KA, Xavier S. Agile learning through continuous assessment. In: *Proceedings of IEEE Frontiers in Education Conference (FIE)*. TX, USA: IEEE; 2015. p. 1–4. https://doi.org/10.1109/fie.2015.7344278.

[18] Raibulet C, Fontana FA. Collaborative and teamwork software development in an undergraduate software engineering course. *Journal of Systems and Software*. 2018;144: 409–422. https://doi.org/10.1016/j.jss.2018.07.010.

[19] Eraslan S, Kopec-Harding K, Jay C, Embury SM, Haines R, Ríos JCC, Crowther P. Integrating GitLab metrics into coursework consultation sessions in a software engineering course. *Journal of Systems and Software*. 2020;167: 110613. https://doi.org/10.1016/j.jss.2020.110613.