The Journal of Systems & Software 188 (2022) 111273

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Code-quality evaluation scheme for assessment of student contributions to programming projects[☆]

Hsi-Min Chen [a], Bao-An Nguyen [b,*], Chyi-Ren Dow [a]

[a] Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan
[b] Department of Information Technology, Tra Vinh University, Tra Vinh, Viet Nam

ARTICLE INFO

Article history:
Received 19 July 2021
Received in revised form 10 January 2022
Accepted 15 February 2022
Available online 21 February 2022

Keywords:
Quality
Project-based learning
Cooperative/collaborative learning
Programming education
Automated programming assessment system

ABSTRACT

Project-based learning is the most common approach to software engineering education, due to its emphasis on the teamwork skills essential to real-world collaborations. This study developed an automated programming assessment system (APAS) featuring a code-quality evaluation scheme to overcome difficulties in assessing the contribution of individual team members. Team participation is visualized on a weekly basis to provide insights pertaining to team dynamics, and metrics based on code quality allow the segmentation of students by level of contribution. Insights provided by the proposed system were also shown to facilitate interventions aimed at improving code quality. Empirical analysis based on submission data from 146 students (41 teams) demonstrated the feasibility of the proposed system in monitoring group-based learning projects at the university level, particularly in detecting free-riders.

1. Introduction

Collaboration and teamwork are key elements of software development (Raibulet and Arcelli Fontana, 2018), and project-based learning (PBL) is the most influential instruction method aimed at conveying collaboration skills (Cico et al., 2021). Most students completing courses in computer science will participate in at least one group project (Wilkins and Lawhead, 2000). The key purpose of these group projects is to provide experience with the software development process as it plays out in the real world.

The widespread adoption of Agile software development in universities (Cico et al., 2021) has prompted numerous educators to base software engineering projects on hands-on agile practices (Masood et al., 2018). This facilitates the iterative development of products with the ability to make incremental improvements. Educators have recently been promoting Git/GitHub as a versatile collaborative platform for PBL in software engineering courses (Kelleher, 2014; Zagalsky et al., 2015). The version control function and shared repository model provide a flexible infrastructure for the management, distribution, and hosting of project materials. The integration of Git/GitHub with open-source software also enables educators to add extensive functions to create computer-supported collaborative learning platforms ideally suited to their needs. Hosting projects on GitHub makes it easy for instructors to track the overall progress of a project as well as the contributions of individual team members based on the collaboration history. The use of industrial processes and tools also provides students with the ability to participate in authentic situations replicating real practices in the workspace (Heckman and King, 2018).

This paper describes the application of our Git-driven automated programming assessment system (APAS), referred to as ProgEdu (Chen et al., 2018) aimed at extending monitoring capability for team programming projects. In this study, Git was used as the submission tool and GitLab was used as the code repository. Continuous integration practices (Fowler, 2006) were used to automate assessments using open-source tools. The proposed system differs from other APASs in two important features:

(i) Support mechanism for assessment of code quality in collaborative programming projects: Many novice programmers lacking familiarity with the principles of clean code (Martin, 2009) are unable to provide code that is comprehensible to their teammates (Athanasiou et al., 2014). Thus, we developed an automated code-quality assessment function to monitor compliance with established coding styles. Linters is used for static code analysis, as well as SonarQube[1] is used to assess code quality and code security. Coding style failure (*CSF*) feedback warnings are returned to students immediately after submission. Students are encouraged

☆ Editor: Shane McIntosh.
* Correspondence to: Department of Information Technology, Tra Vinh University, 126 Nguyen Thien Thanh Street, Ward 5, Tra Vinh City, Tra Vinh Province, 87000 Viet Nam.
E-mail addresses: hmchen@mail.fcu.edu.tw (H.-M. Chen), annb@tvu.edu.vn (B.-A. Nguyen), crdow@mail.fcu.edu.tw (C.-R. Dow).

1 https://www.sonarqube.org/.

to submit their contributions to facilitate agile development and iterative improvements.

(ii) Mechanism for assessment of individual contributions to code-quality improvement: The collaborative nature of programming makes it difficult to assess the contributions of individual students (Buffardi, 2020). Peer evaluation is the most commonly applied approach to measure individual contributions; however, the resulting assessments are highly subjective. Git-based metrics by which to measure the contributions of individual members could conceivably be computed from Git logs (Parizi et al., 2019); however, a lack of unit tests makes it impossible to evaluate the functional correctness of submitted code. This opens the door to cheating through the submission of meaningless blocks of code. In this study, we sought to extend qualitative measurements using metrics based on code quality to prevent this behavior.

The monitoring efficacy of the proposed scheme was evaluated in a Web programming course with 147 undergraduate students. At the end of the course, we analyzed system log data to explore the efficacy of using ProgEdu in monitoring students' programming projects at both team level and individual level. Therefore, we aim to address the following two research questions:

RQ1: At the team level, to what extent does ProgEdu facilitate the monitoring of team projects?

RQ2: At the individual level, do measures pertain to code-quality improvements provide an accurate assessment of the contributions of students?

Results of the empirical study show the feasibility of applying ProgEdu in monitoring programming projects. Responses to RQ1 demonstrate that through learning analytics on ProgEdu log data, instructors can monitor the degree of teams' participation in completing the programming projects and fixing code quality errors. The answers to RQ2 show that the proposed measurements on code quality improvements can be incorporated into the assessment of students' contributions to their teams, especially in the detection of free-riders. Within the study, we also provide notes that can facilitate instructors to improve teams' collaboration and code quality by offering proper interventions.

The remainder of this paper is organized as follows: Section 2 outlines related works and key concepts used in the development of the ProgEdu system. Section 3 describes the ProgEdu system in detail. Section 4 presents data mining experiments used to assess the efficacy of the proposed system and the metrics used to monitor team programming projects. A brief discussion and conclusions are presented at the end of the paper.

2. Literature review

Besides providing a collaborative platform to monitor programming projects, the ProgEdu system also enhances student code quality and facilitates the evaluation of individual contributions. This review outlines related works to clarify our motivations and highlight our contributions to the field.

2.1. Automated programming assessment systems and collaborative platforms

Over the last twenty years, more than 100 automated assessment tools have been developed providing a variety of functions for a range of programming languages and running environments (Keuning et al., 2018). Some educators have developed standalone APASs for assessment and grading (Blumenstein et al., 2008; Fernández and Luis, 2011; Hundt et al., 2017; Robinson and Carroll, 2017). Others have integrated automated assessment functions into learning management systems such as Moodle to create virtual programming laboratories in which to practice and evaluate programming (Kaya and Özel, 2015; Rodríguez-del Pino et al., 2012). Automated assessment tools have also been integrated with intelligent tutoring systems aimed at guiding students as they develop programming skills via real-time feedback (Grivokostopoulou et al., 2017). However, there have been relatively few reports on the lifespan and efficacy of such tools for the promotion of advancing programming skills. Many researchers have suggested collaborations among the authors of these tools aimed at simplifying the adoption process and producing measurable improvements in terms of learning (Pettit and Prather, 2017).

Many of the resulting systems are equipped with functions aimed at facilitating all phases of collaborative learning, including administration procedures, delivery of digital learning materials, assessments, and the collection of student responses (Chua et al., 2019). There have been few reports on leveraging automated assessment systems in the context of collaborative learning.

Git-based assessment systems, which inherit the collaborative working paradigm from the software industry, have proven highly effective for the assessment of student performance in collaborative learning (Haaranen and Lehtinen, 2015; Zagalsky et al., 2015). Neyem et al. (2014) adopted an empirical approach to software engineering courses involving weekly consultation sessions under the guidance of a project tracking tool. GitHub was used as a hosting platform to facilitate tracing and make it clear to students how the requirements were selected and tested. Gary and Xavier (2015) proposed a continuous integration and test platform allowing continuous assessment and feedback on student activities to support teamwork in software engineering projects. Raibulet and Arcelli Fontana (2018) combined GitHub with Microsoft Project and the code inspection tool SonarQube to stimulate collaborative activities among students involved in software engineering projects. Eraslan et al. (2020) employed the GitLab repository to host project artifacts and used GitLab metrics (e.g., the number of commits and the number of assigned and closed issues) as basic information for coursework consultation sessions for software engineering courses. These studies illustrate the feasibility of using industrial tools to facilitate academic software projects. However, none of the above studies addressed the issue of assessing the contributions of individual students engaged in team projects. The scheme proposed in this paper is meant to fill this research gap.

2.2. Code quality and automated programming assessment

Source code quality can have a profound effect on the cost of maintaining software products. Poor code quality and static errors can hinder code comprehension, bug detection, refactoring, maintainability, and reusability. In the long term, software projects with poor code quality might be subsequently at risk of *technical debt*. In software engineering, technical debt refers to contingent compromises that are occasionally accepted by developers (Brown et al., 2010), such as bugs and design issues (Black et al., 2009), which must eventually be addressed to maintain the health of the project. Debt is not necessarily "bad", considering that a limited amount of debt can help developers to accelerate development over the short term (Guo and Seaman, 2011). However, a failure to address technical debts promptly leads to the accumulation of 'interest', which can have a profoundly negative effect on the long-term success of software development (Tom et al., 2013).

Educators at the university level have pointed out the difficulties associated with training students to produce high-quality code (Cardell-Oliver, 2011; Chen et al., 2018; Patton and McGill, 2006). This is a significant obstacle within the context of programming courses, as Kirk et al. (2020) suggested that code quality is a key criterion of computing competence. There is every indication that code quality can be gradually improved if instructors make it a priority. Researchers had attributed this improvement to the resubmission policy and feedback systems. In an analysis of over a million Java source files submitted to the APAS Web-CAT in five semesters by 3691 students, Edwards et al. (2017) discovered that the number of static errors in the final submissions was six times lower than in the initial submissions. It is also identified that code quality is an indicator of learning performance, as indicated by the association between the presence of static errors in the final submissions and low final grades. On the contrary, if code quality assurance is not encouraged by the instructors, this ability would be not improved over the years, as reported by Breuker et al. (2011): when instructors fail to emphasize code quality, there is no assurance of improvement between the first and second years.

Few educators at the university level are able to assess the iterative progress of students in terms of static code (Edwards et al., 2017). In many cases, code quality is assessed only in the final version of the students' programs (Raibulet and Arcelli Fontana, 2018), thereby precluding the ability to improve code quality through the ongoing correction of static errors. Code-quality inspection tools are not built into common programming languages; therefore, professional workflows (e.g., continuous integration) are required to assess code quality on the fly.

The continuous assessment of code quality can be facilitated by APASs. Keuning et al. (2018) reported that 37 out of 101 (36%) automated feedback tools for programming exercises were equipped with static analysis functionality. Correlatively, Kirk et al. (2020) found that 30% of programming courses directly address the issue of code quality. Among tools for static code analysis, CheckStyle[2] and PMD[3] are the most widely used tools in APASs (Keuning et al., 2018). SonarQube is another powerful tool employed by many educators to analyze code quality and code security (Lu et al., 2018; Nguyen et al., 2020; Raibulet and Arcelli Fontana, 2018). Nonetheless, the efficacy of these systems in terms of improving code quality has not been adequately assessed.

ProgEdu[4] is an APAS that was designed based on the continuous integration practice to automatically assess students' Java programming assignments personally (Chen et al., 2018). The continuous integration server, i.e., Jenkins, and the build tool Maven were employed to facilitate continuous assessment with three plugins: plagiarism checking, unit test, and coding convention checking. Iterative learning was promoted in ProgEdu, in which students have unlimited opportunities to practice clean code via free resubmissions. ProgEdu has shown the usability in supporting the instructors in students' code assessment and making students familiar with good programming concepts and practices such as version control (Git/GitLab) and coding convention adherence.

Code quality assessment in ProgEdu has made it possible to discover useful insights in programming the learning behaviors of students via personal assignment data (Chen et al., 2020). Via an analysis of 1320 Java programming submissions to ProgEdu, five different student profiles had been detected based on the effort and the time that students paid for code quality improvement, named *Effective learners, High-effort learners, Average learners, Blind-trial learners,* and *Low-effort learners*. It also showed that the instructor can predict early the learning result of students using data of only three assignments given before the midterm. Utilizing iterative submissions and feedback, the likelihood of a student producing a failure at the end of the course was lower than at the beginning. Different from the previous studies focusing on personal programming assignments, in this research, we extended ProgEdu to enable team projects in Web programming courses and facilitate instructors to understand students' contributions and learning behaviors in teams.

2.3. Measuring contributions of individual members in team programming projects

In any type of team-based learning, the process of assessing the contributions of individual team members and converting this assessment to numeric grades is fraught with difficulties (Johnston and Miles, 2004). The grading schemes adopted by instructors generally fall into one of two cases: (i) All team members are assigned the same grade based on the quality of project output or (ii) individual team members are assigned different grades based on a combination of the project output and their contributions (Wilkins and Lawhead, 2000). The second approach is preferable since it rewards students for their contributions and prohibits "free riders" (i.e., individuals who make a minimum contribution and rely on teammates to perform the bulk of the work) from taking advantage of others. However, few instructors are able to access information by which to evaluate individual contributions to collaborative projects.

Peer evaluation is a common strategy adopted by educators. In multiple studies, CATME (Layton et al., 2007) has been applied with success to team formation and peer evaluation. CATME provides an online questionnaire using a five-point scale for peer- and self-evaluations based on the five essential aspects of teamwork: contributing to achieving the team goals, interacting with teammates, keeping the team on track, delivering quality work, and providing task-related knowledge/skills/abilities. Note however that peer evaluations are subject to the "halo effect", in which ratings are influenced by the charisma of different members (Loughry et al., 2007).

Professional software engineering tools allow for the continuous recording of working logs to facilitate evidence-based assessments at the individual level. Git logs for example contain evidence by which to infer member contributions. Parizi et al. (2019) recommended using data from GitHub, GitLab, Bitbucket, or any Git-based code repository to evaluate team members' contributions on a monthly basis. They posited the following five metrics: number of commits, number of merges, number of files, the total number of lines of code, and time spent on the project. Eraslan et al. (2020) sought to reveal the contributions of members using a variety of GitLab metrics, including the number of commits and number of issues assigned and closed by which to assemble a weekly report for consultation sessions. Buffardi (2020) used a combination of GitHub repositories and GitHub Projects in conjunction with Kanban boards to manage team-based software engineering projects. They recommended augmenting Git-driven metrics with transitions in story status to measure the contributions made by individual members. Git-based metrics are data-driven and objective; however, this type of quantitative measurement is susceptible to manipulation by unscrupulous students. For example, students can artificially increase their apparent contributions by repeating multiple commits or adding and then removing multiple lines of code. In an attempt to reduce reliance on Git logs, De Bassi et al. (2018) recommended using quality metrics to measure the contributions of developers. The metrics are used to inspect code quality from the perspective of complexity, inheritance, size, and coupling. Not all of these metrics are appropriate to the context of student projects; however, they do provide code-quality metrics that overcome the limitations of Git-based metrics. In the current study, we employed a combination of Git-based and code-quality-based metrics to perform assessments at the individual level.

[2] https://checkstyle.sourceforge.io/.
[3] https://pmd.github.io/.
[4] https://github.com/ProgEdu-Organization/ProgEdu.

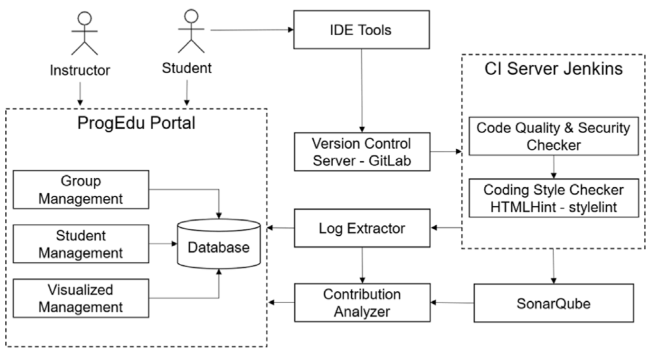


Fig. 1. ProgEdu architecture.

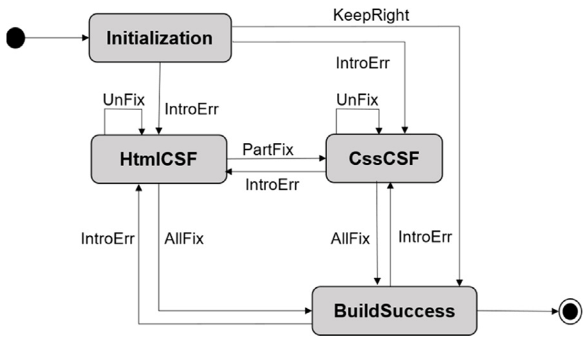


Fig. 2. Transitions associated with changes in project status.

3. Proposed ProgEdu system

To address the need for groupware to administer team projects in a Web programming course, we extended ProgEdu with a number of additional features: (i) we integrated code analysis tools for Web programming languages (HTML, CSS, and JavaScript) and (ii) we developed a strategy for the evaluation of individual contributions based on improvements in code quality.

3.1. ProgEdu architecture

Fig. 1 illustrates the architecture of ProgEdu, which was built as a distributed system serving two actors: the instructor and the student.

ProgEdu portal: The portal was developed as a learning management system to facilitate team administration and project monitoring. The student management module and group management module make it easier for instructors to manage students and assemble teams. The visualization management module provides visualized reports to keep instructors and students up to date on the progress of the project. Instructors can track the progress of all groups, whereas students can track only their own group.

Version control server — GitLab: We employed GitLab[5] as a source code repository receiving student source code committed by Git commands. The GitLab repository is linked to a continuous integration server to enable immediate code analysis.

Continuous integration and assessment server: Triggered by code commits to GitLab, the continuous integration server Jenkins[6] sends committed code to assessment modules. As this version of ProgEdu is devoted to assessing Web programming, we employed *HTMLHint*[7] and *stylint*[8] as static code analysis tools respectively for HTML and CSS code. Code quality status and feedback information related to coding style failures are displayed after each submission in the team interface. To familiarize students with industry metrics of code quality (e.g., bugs, vulnerabilities, and code smell), we also leveraged SonarQube to analyze code quality and code security. Not all issues reported by *SonarQube* are easy for undergraduate students to resolve; therefore, we adopted the project status based on the results of *HTMLHint* and *stylint* as representative data for the analysis of individual contributions.

Log extractor: All data pertaining to students committing project artifacts recorded in the logs of GitLab and Jenkins are continuously collected by the log extractor and sent to the portal.

5 https://about.gitlab.com/.
6 https://www.jenkins.io/.
7 https://htmlhint.com/.
8 https://stylelint.io/.

Table 1
Individual contribution measures.

Metric	Type	Description
SubmTotal	Event	Number of submissions
BuildSucc	State	Number of *BuildSuccess*
HtmlCSF	State	Number of *CSF in HTML* code
CssCSF	State	Number of *CSF in CSS* code
IntroErr	Transition	Number of *IntrErr* transitions
KeepRight	Transition	Number of *KeepRight* transitions
UnFix	Transition	Number of *UnFix* transitions
PartFix	Transition	Number of *PartErr* transitions
AllFix	Transition	Number of *AllFix* transitions

Contribution analyzer: Data from the log extractor is then used by the contribution analyzer to compute the contribution of each team member based on their effort in committing source code and improving project code quality. The metrics used to evaluate contributions are presented in the next subsection.

Embedding continuous integration within the system makes it possible to provide real-time feedback pertaining to changes in project status. Following each submission, the student can access the system portal to determine whether the new code conforms to coding style requirements. Project status and member contributions are presented visually to facilitate the monitoring of team progress.

3.2. Code-quality assessment and project status

Submitted source code is analyzed by code-quality checking modules implemented in *HTMLHint* and *stylelint*. Based on the assessment results, the status of the project is designated as *Initial*, *HtmlCSF*, *CssCSF*, or *BuildSuccess*. *Initial* indicates that the project is in the preliminary stages of development. Check style failures (*CSF*) returned by *HTMLHint* and *stylelint* respectively earn the designations *HtmlCSF* and *CssCSF*. Note that the *CssCSF* designation can be only reached if *HtmlCSF* is resolved. When all *CSF*s are resolved, the project status changes to *BuildSuccess*.

Corresponding to possible changes in project status, we defined five transitions in the state transition diagram in Fig. 2:

(1) *IntroErr*: A transition that introduces new *CSF*s from a clean state or pulls the project back to *HtmlCSF* from *CssCSF*;
(2) *KeepRight*: A transition that maintains the project status as *BuildSuccess* or *Initial*;
(3) *PartFix*: A transition that addresses *HtmlCSF*, such that the project status changes to *CssCSF*;
(4) *UnFix*: A transition that does not lead to the resolution of the *CSF* of interest;
(5) *AllFix*: A transition that leads to *BuildSuccess* from any state of *CSF*.

Table 2
Team contribution measures.

Metrics	Calculation	Description
Contribution index (CI)	$CI.M_S = \frac{M_S}{\overline{\overline{M_T}}}$	The ratio of contributions of student S *per average contributions* of the team regarding metric M
Contribution max-difference (CM)	$CM.M_S = \max_{i \in T}(CI.M_i) - CI.M_S$	The difference in contribution index *between* the student S and *the most contributor* in the team regarding metric M

3.3. Measuring team participation

The level of team participation each week is crucial for the success of teamwork. We provide the instructors two measures for weekly team participation which are *Team Submission Count (TSC)* and *Team Participation Ratio (TPR)*. *TSC* refers to the total number of submissions made by all team members each week. Team coordination was measured by *TPR*, which refers to the number of active members each week normalized by the size of the team. *TSC* and *TPR* of team *T* are calculated for week *w*, as follows:

$$TSC_T^w = \sum_{i \in T} SubmTotal_i^w$$

$TPR_T^w = |C|/|T|$, where $C = \{i | i \in T \wedge SubmTotal_i^w \geq 1\}$.
Average values of *TSC* and *TPR* overall weeks, denoted as *Avg.TSC* and *Avg.TPR*, can be inferred from weekly measures to illustrate an overview of how a team performed in their project. We employed these measures of team participation in visual data analysis to provide insights into team dynamics to the instructors.

3.4. Measuring member contributions

Every submission of code by a student is a contribution to the group project; however, the value of student contributions extends far beyond the total number of submissions. Supplementary to this count, we consider the effort that the student made to improve the quality of the code. Code-quality improvements are reflected in changes in project status; therefore, we combined the three states and the five transitions with the total number of submissions as measures of individual contributions (Table 1). The contributions of individual team members can be calculated in various ways. We developed two metrics in this study. Given team *T*, team-contribution metrics of student *S* on metric *M* are calculated using the schemes outlined in Table 2.

The proposed metrics reveal the degree to which students shoulder their responsibilities and uncover instances of free riders (Maiden and Perry, 2011). Note that differences can have a profound impact on student performance; however, we opted not to reveal these values to the students to avoid the effects of Campbell's Law (Campbell, 1979), Campbell's Law stipulates that in cases where team assessments rely heavily on quantitative measures, team members are prone to changing their behavior adaptively, even when the behavior is of no benefit to the ultimate outcome (Buffardi, 2020). In the current study, we used these metrics as features for the clustering of students based on differences in their contributions. Note that the distribution of CM values might vary among teams, it was normalized by all teams in further use using z-scores ($z = (x - \bar{x})/S$, where z is the normalized CM value, x is the original CM value, $\bar{x}$ and S are respectively mean and standard deviation of CM values of all teams). These clustering results can be used to supplement individual assessment tasks, as illustrated by the empirical case study presented in the following section.

4. Empirical case study

In this section, we present an empirical case study on the application of ProgEdu in a Web programming course. First, we address descriptive data pertaining to the engagement of students during the course in order to determine the extent to which ProgEdu facilitates the monitoring of team projects (i.e., the first research question). We then perform cluster analysis of student profiles based on their contribution in order to determine whether improvements in code quality provide an accurate assessment of the contributions of individual students (i.e., the second research question).

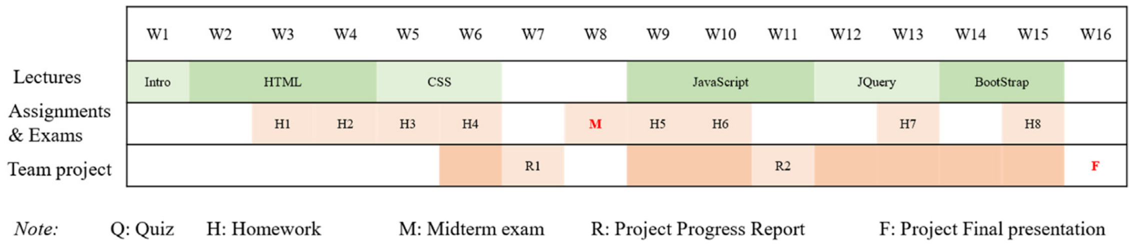
4.1. The course and data

Course: The proposed system was implemented in a second-year Web programming course in the Department of Information Engineering and Computer Science, Feng Chia University. The course objectives were designed to provide key knowledge and techniques of the development of responsive web applications to undergraduate students in computer science, who have passed the prerequisite course Introduction to programming. The course knowledge is delivered in 15 weeks, as well as the corresponding syllabus of lectures and assessments are detailed in Fig. 3.

Assessment scheme: Students' academic performance is assessed in three aspects namely homework, midterm exam, and final project with weights of 0.3, 0.3, and 0.4, respectively. Eight homework assignments were corresponding to the weekly lectures. In the midterm exam, students were required to write HTML and CSS code to design and format a simple webpage given by the instructor. The final assessment is conducted via a team project starting two weeks before the midterm.

Team project: The 146 students were grouped into 41 teams of three or four students and were tasked with building small-scale Web applications over a period of 9 weeks. Teams were required to report the project progress twice, in the seventh and eleventh weeks. Under the paradigm of Agile practices, teams were required to make iterative improvements by submitting changes in source code to ProgEdu and addressing failures reported by the system. Note that students can also receive feedback from SonarQube. However, SonarQube addresses advanced issues such as code smell, potential bugs, and vulnerabilities. Fixing these issues was optional for students.

Project assessment: Team projects were assessed in a final presentation at the end of the semester. Project grades were on a scale of 1–100. Prior to the presentation, the students were required to fill out a peer-evaluation questionnaire pertaining to the contributions made by their teammates. In the questionnaire, each student was asked two questions about their peers: (i) the jobs that each member participated in and corresponding frequencies; (ii) the percentage of contributions made by each member (see Appendix A for details). Peer-evaluation results were then validated and adjusted by the instructor through individual interviews conducted at the time of the final presentation. Responses to the first question are not used to grading but provide insights about team dynamics and facilitate the instructor in the individual interview. The assignment of individual grades



	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16
Lectures	Intro	HTML			CSS				JavaScript			JQuery		BootStrap		
Assignments & Exams			H1	H2	H3	H4		M	H5	H6			H7		H8	
Team project							R1				R2					F

Note: Q: Quiz H: Homework M: Midterm exam R: Project Progress Report F: Project Final presentation

Fig. 3. Syllabus of lectures and assessments of the Web programming course.

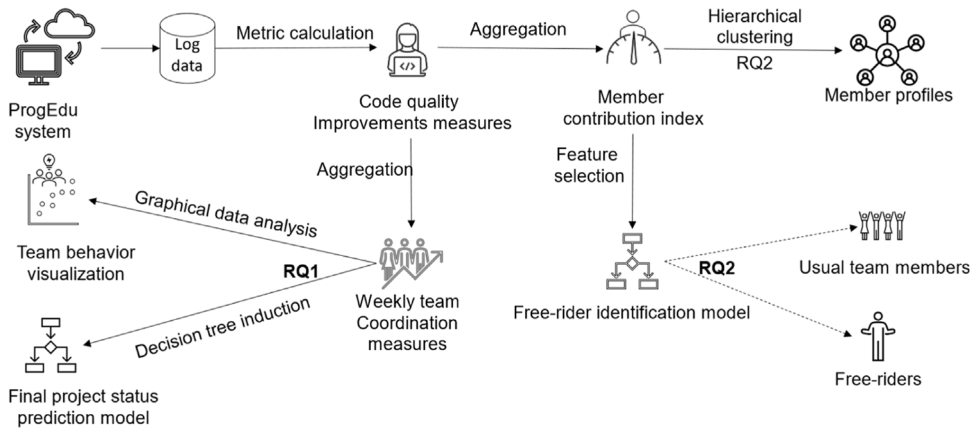


Fig. 4. The learning analytics workflow adopted in the empirical study.

was based on the project product and their individual contributions following the scheme recommended by Fellenz (2006) (see Appendix B for details).

As the students were notified that the evaluation results cannot be changed once submitted and will be validated by individual interviews in the final presentation, "benign collusion", the situation that all team members agreed to give each other the same credit (Fellenz, 2006), did not happen in our data. There is only one group in which all members were rated as 100% of contributions, and the instructor also agreed with these ratings. Among 146 students, 50 students (34.2%) were rated as 100% contributed, 68 students (46.5%) were rated contributed less than they should do (less than 100%), and 28 students (19.11%) credited as high contributors (over 100%). Notably, 11 students (8%) were evaluated as having no contribution.

Free-rider labeling: Peer reviews and individual interviews made it possible for the instructor to identify free riders. Among the 146 students in the study, 20 (13.7%) were identified as free riders, who earned a mean grade of 60.7 (*sd* = *7.9, max* = *77, min* = *50*). The average grade of free riders was 23.9 points below that of the highest-scoring member on their team (*max difference* = *42, min difference* = *12*).

4.2. Methodology

We established a learning analytics workflow on ProgEdu log data to address the research questions, as illustrated in Fig. 4. In this workflow, we employ learning analytics as "the measurement, collection, analysis, and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" (Siemens and Baker, 2012). After the log data is extracted, measures on individual and team contributions were inferred. To demonstrate the efficacy of ProgEdu in monitoring projects at the team level, descriptive data analysis, correlation analysis, visual data analysis, and decision tree induction were adopted on the way to finding the responses to RQ1. At the individual level, educational data mining techniques consisting of clustering and classification were performed on measures of member contributions to address RQ2.

4.3. Results

4.3.1. RQ1: At the team level, to what extent does ProgEdu facilitate the monitoring of team projects?

This research question was addressed by conducting a learning analytics experiment on submission log data. Our results revealed that ProgEdu gave instructors a meaningful understanding of a project's progress, applicable to the assessment of individual students as well as to the fine-tuning of group interactions. We sought to further elucidate these insights by posing the following sub-research questions:

RQ1.1. *How did students participate in their projects?*

On a weekly basis, the visualization of weekly measures of *TSC* and *TPR* can provide a deep insight into the level of effort and participation that each team made in submitting project source code. Fig. 5 illustrates the participation-related information made available to instructors during the project.

Most of the teams were less active in the first five weeks, and more active in the latter four weeks. Prior to the fifth week, few team members were participating in code submissions, as indicated by low values of weekly *TSC* and *TPR* values. Starting from the sixth week, most of the team members made active contributions. Since the sixth week of the project corresponds to

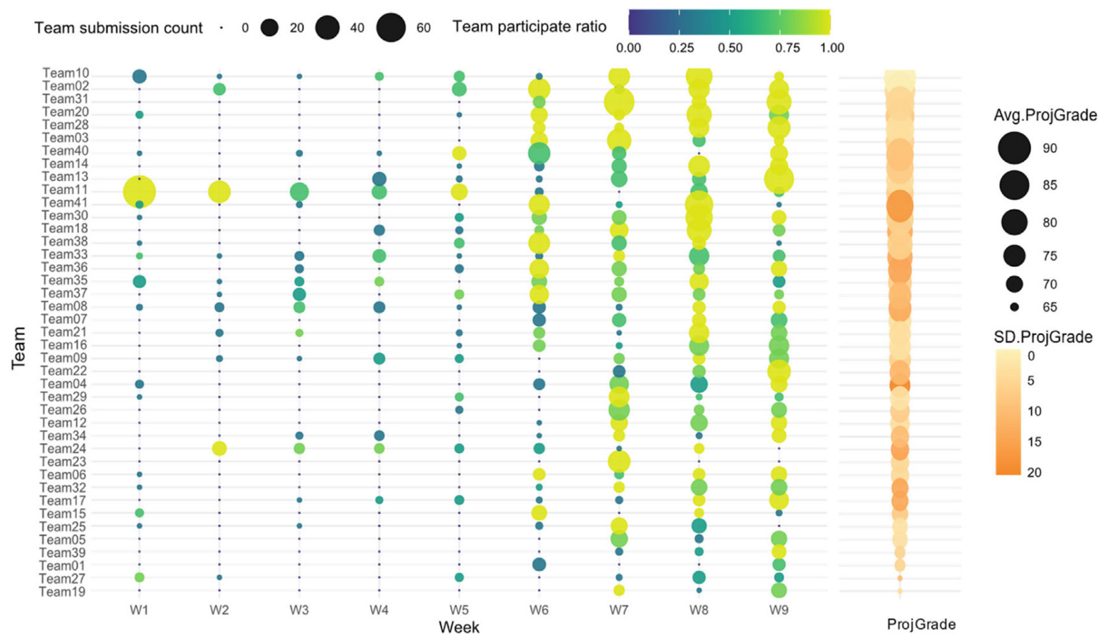


Fig. 5. Visualization of weekly team participation: markers' size denotes weekly submission count; markers' color denotes weekly team participation ratio. Teams appear in descending order of project grade (*ProjGrade*): markers' size denotes average project grade (Avg.*ProjGrade*); markers' color denotes within-team variation of project grade (SD.*ProjGrade*). . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

week 10 of the semester, there are three reasons subsequential triggering teams to be more active from week 6. First, students were asked to work on the requirement engineering and user interface design in the first five weeks; second, all lessons related to writing HTML and CSS code were taught from that week; third, the instructor' comments in the second project progress report warn student teams to pay dedicated time for the projects. Note that a small number of teams made little effort and were poorly coordinated throughout the project duration. On average, the amount of weekly submissions made by a team is more than twice the team size *(Avg.TSC; mean* $= 7.27$*; sd* $= 3.74$*; min* $=$ *2.22; max* $= 21.56$*)*, whereas less than a haft of team members participated in the teamwork *(Avg.TPR; mean* $= 0.40$*; sd* $= 0.12$*; min* $= 0.11$*; max* $= 0.68$*)*.

Both TSC and TPR in the four last weeks have had a strong impact on teams' performance. Notably, week 6 was a pivotal week in which teams were triggered to be active from this week can achieve better project grades and vice versa. These patterns posed another sub-research question about the relationship between team coordination and project grades. The answer to these questions can be used by the instructor to guide teamwork.

RQ1.2. *Does better team coordination manifest higher project grades?*

Visual data analysis using scatter plots and correlation analysis are good tools to address this sub-question. We employed the average values of TSC and TPR as measures of teams' coordination and plot the data in 2D-scatter plots. As shown in Fig. 5 and Table 3, the teams that earned higher grades appeared to share the workload more effectively in the last four weeks, as denoted by their high TPR values. The lower values of TPR in these weeks, especially in the final week, were also indicators of the large variations in grade among team members. Beginning in the fifth week, team effort and team coordination were both moderately correlated with project grades. We observed a significant correlation between TPR and Avg. ProjGrade in weeks 6 and 8, indicating that these were critical weeks in which team collaboration had a greater impact on the success of projects. Identifying critical points in which to conduct interventions could conceivably lead to improvements in student grades.

Table 3
Correlation between weekly TPR, TSC and *Avg.ProjGrade*.

Week	1	2	3	4	5	6	7	8	9	Avg.
TPR	0.09	0.18	0.07	0.21	0.38*	0.51*	0.28	0.44*	0.31*	0.66*
TSC	0.21	0.18	0.14	0.22	0.4*	0.48*	0.3	0.64*	0.27	0.82*

Note: *: $p < 0.05$.

In an overall view, average values of both TSC and TPR are significantly correlated with Avg. ProjGrade $\neg(\rho = 0.82$ and $\rho = 0.66$, $p < 0.01$, respectively), as shown in Table 3. The scatter plots in Fig. 6 are meaningful responses to this research question. It is conspicuous that the poor-coordinated teams might result in lower project grades, whereas teams with better coordination can earn higher grades. Especially, average numbers of weekly submissions are strongly associated with given grades (Fig. 6a). Besides, high numbers of active members each week can also be a potential indicator of high grades (Fig. 6b). From these insights, the instructor can offer policies to encourage students' awareness of their projects, either in submission frequency or in maintaining the balance in member contributions.

RQ1.3. *How was the projects' code quality improved?*

Our code-quality assessment scheme in conjunction with a policy of resubmission resulted in a large number of submissions (2687 by the 41 teams). Table 4 shows the descriptions of metrics on individual' submissions and the correlation between them. The first impression was that students who made more submissions might have higher grades, as denoted by the correlation between *SubmTotal* and *ProjGrade* *(*$\rho = 0.58$*,* $p < 0.05$*)*.

In normal behavior, most of the students focused on meeting functional requirements, and the instructors did not enforce the repair of stylistic errors after each assessment. As a result, most of the students paid little heed to compliance with coding conventions, as illustrated by the fact that 83.9% of submissions resulted in *CSF*s, and 71% of submissions failed to trigger an improvement

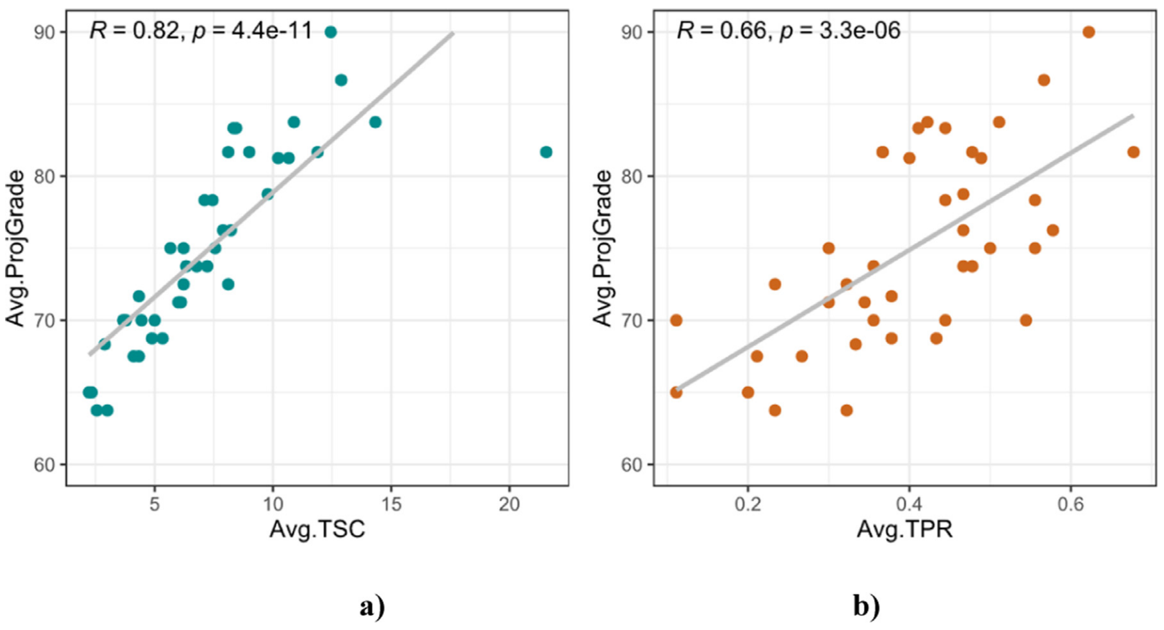


Fig. 6. Illustrations of the associations between team coordination and project grades.

Table 4
Statistic on all submissions, mean(sd) values per individual, and Pearson correlations between metrics.

	Total	Mean	SD	1	2	3	4	5	6	7	8	9
1. SubmTotal	2687	18.4	16.4									
2. BuildSucc	431	3.01	5.83	0.66*								
3. HtmlCSF	**749**	5.1	6.83	0.45*	−0.04							
4. CssCSF	**1507**	10.3	11	**0.87***	0.48*	0.07						
5. IntroErr	223	1.68	2.76	0.5*	0.36*	0.18*	0.44*					
6. KeepRight	322	2.21	4.83	0.57*	0.97*	−0.1	0.38*	0.19*				
7. UnfixErr	1922	13.1	11.5	0.92*	0.35*	0.58*	**0.83***	0.35*	0.27*			
8. PartFix	111	0.8	1.29	0.45*	0.17*	0.25*	0.41*	0.81*	0.07	0.35*		
9. AllFix	109	0.77	1.61	0.68*	0.7*	0.08	**0.59***	0.55*	0.5*	0.48*	0.29*	
10. ProjGrade		74.5	9.84	**0.58***	0.26*	0.34*	**0.52***	0.18*	0.22*	**0.59***	0.2*	0.34*

in code quality (*UnFix*). Note however that some of the teams paid attention to code quality, as indicated by their careful adherence to fixing all *CSF*s.

The statistics in Table 4 reveal notable differences between HTML and CSS. Among the total of submissions, 28% were *HtmlCSF* (749 per 2687) and more than 56% were *CssCSF* (1507 per 2687). Please note that while the volume of CSS code is very small in comparison to that of HTML code, the *CssCSF* count was double *HtmlCSF* one. This means that the students paid more effort to write clean HTML than fixing *CssCSF*s. At the individual level, we did not observe a significant correlation between causing *HtmlCSF* and fixing code-quality errors. We observed a relatively strong correlation between *CssCSF* and *AllFix* values ($\rho = 0.59$, $p < 0.05$), indicating that students who made a greater submission effort resulting in *CssCSF* also made a greater contribution to fixing code-quality errors, leading to higher grades ($\rho = 0.52$, $p < 0.05$).

There are two reasons for a member who caused more *CssCSFs* should be considered as a better contributor. First, when the project status was being *HtmlCSF*, then a student shifted it to *CssCSF* by fixing all *HtmlCSFs* and introducing new *CssCSF*s, the student has contributed as a *PartFix*. Second, when the project status was being *CssCSF*, then a student submitted new HTML code letting the new status remain in *CssCSF*, indicating the student has contributed clean HTML code, even though the transition was *UnFix*. With a *PartFix* count of only 111 out of 2678 submissions, it is obvious that the second reason is dominant over the first one. This gives us an explanation of why *CssCSFs* caused by *UnFix* transitions are positive contributions. Correspondingly, it also explain the significant correlation between *CssCSF* and UnFix ($\rho = 0.83$, $p < 0.05$), between *UnFix* and *ProjGrade* ($\rho = 0.59$, $p < 0.05$), and why *UnFix* is also an important indicator of contributions.

We employed visual data analysis to explore insights in weekly statistics on code quality improvements and distribution of assessment logs, as presented in Figs. 7 and 8, respectively. The numbers of teams that reached *BuildSucc* status were as follows: week 6 (15), week 7 (11), week 8 (12), and week 9 (20). By the end of the semester, 19 out of 41 teams (46.3%) succeeded in making submissions with no *CSF*s, as depicted in Fig. 7a. We also observed a gradual increase in the number of students achieving *BuildSucc*, reaching 40 in the final week (Fig. 7b). It appears that ProgEdu was effective in making students aware of code quality, as indicated by the fact that nearly 50% of the final products were free from *CSF*s.

Teams' submission efforts and corresponding results in the temporal dimension are visualized in Fig. 8. High numbers of submissions and *HtmlCSF* in week 8 and week 9 denote that multiple teams only work hard in the two final weeks. Despite all needed lessons being delivered, a large amount of HTML code was submitted in the very late weeks, as depicted by the high density

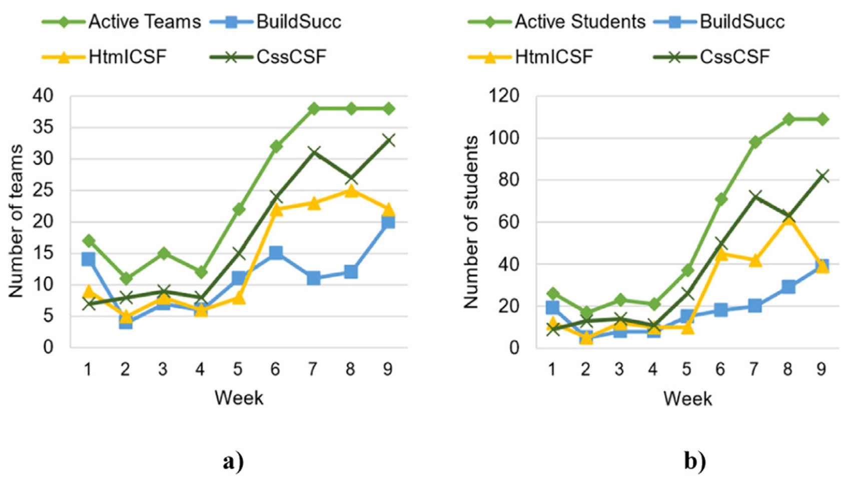


Fig. 7. Weekly statistics on code quality improvements: (a) counted at team level; (b) counted at the individual level.

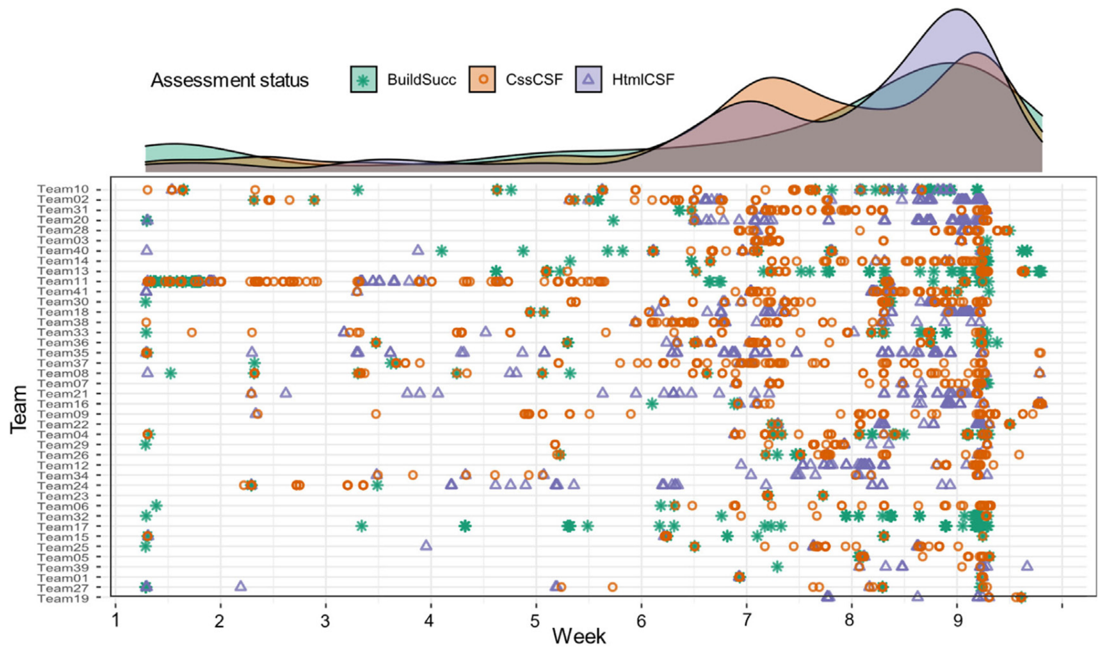


Fig. 8. Visualization of teams' submissions and corresponding assessment status.

of *HtmlCSF* in the final weeks. As a result, some teams cannot output high-quality products free of code quality failures. This poses a warning to the instructors to have earlier interventions to combat students' procrastination.

The fact that there are too many *CssCSFs* while the volume of CSS code is so small compared to that of HTML code reveals an inappropriate submission style that can be easily remedied. That is, students often initialized the website with some lines of low-quality CSS code, then they iteratively wrote and submitted new HTML code to realize the webpage design without any CSS change. As a result, most submissions containing new clean HTML code received the same responses *CssCSF*s. We believe that the instructors can easily mitigate this by offering interventions. That is asking the team to fully format the web pages with clean CSS code before implementing its functionalities. Consequently, project status would not stay in *CssCSF* and be directly shifted from *HtmlCSF* to BuildSuccess. This simple operation can reduce the burden of "technical debts" discussed in the next sub-question.

RQ1.4. *What is the indicator of a team giving up improving code quality?*

Identifying the indicators for giving up on code correction could help instructors to overcome this issue. We employed the concept of "technical debt" (Cunningham, 1992) to explain delays in fixing programming issues.

Iterative submissions throughout the project produced a sequence of accumulated assessment results ordered temporally, such as {*HtmlCSF, BuildSucc, CssCSF, HtmlCSF, BuildSucc,* ...}. Let S^T be the sequence of code-quality assessment results of team *T*. *BuildSucc* splits S^T into a set of multiple subsequences $C^T = \{s_i\}$, in which s_i is a subsequence of S^T, subsequently terminated by

Table 5
Comparison between BS. Final group and CSF. Final group.

	BS. Final N = 19 (46.3%)	CSF. Final N = 22(53.7%)	Wilcoxon test (W)
SubmTotal			
Median	85	62	139.5*
Mean (SD)	89.89(51.81)	64.09(30.24)	
CSF sequence length (all)			
Median	5.00	26.00	1767.5***
Mean (SD)	15.86(22.96)	34.18(31.35)	
Longest CSF sequence length			
Median	34.00	51.50	260.5
Mean (SD)	41.57(28.48)	52.72(28.50)	
Latest CSF sequence length			
Median	5.00	51.50	335***
Mean(SD)	20.26(26.37)	52.13(29.40)	

Note: *: $p < 0.1$; **: $p < 0.05$; ***: $p < 0.001$.

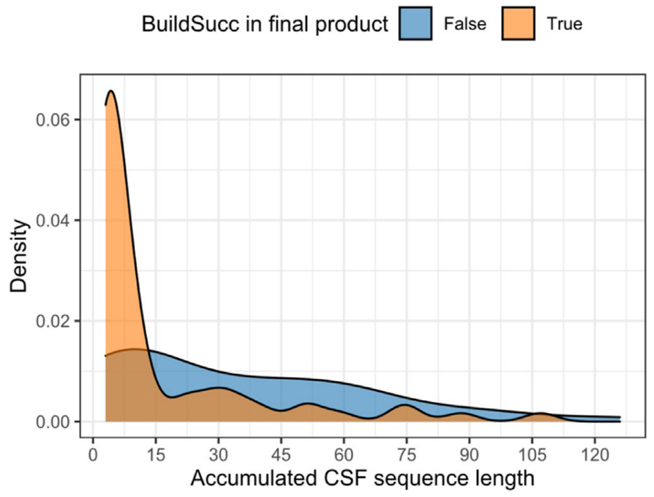


Fig. 9. Density plot of length of *CSF* sequences generated by two groups of teams discriminated by the status of the final product.

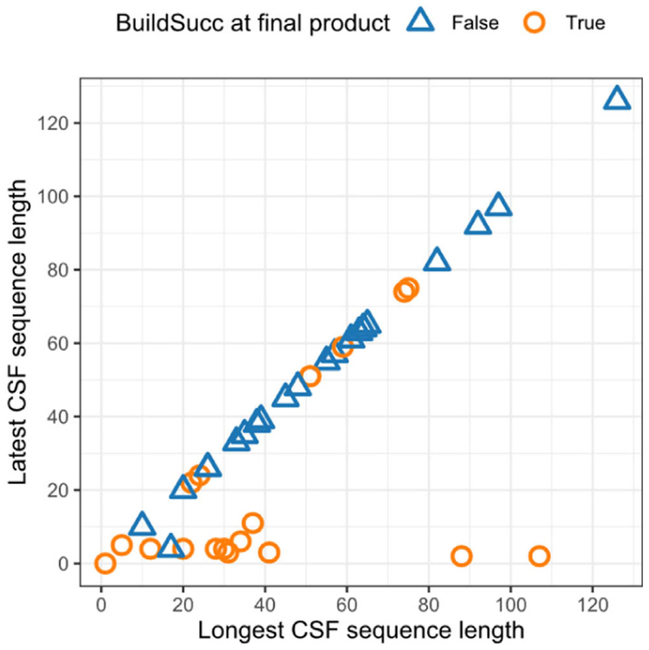


Fig. 10. Scatter plot represents the length of the latest and the longest *CFS* sequences of 41 project teams.

BuildSucc. The longest *CSF* sequence in C^T is a sequence $s_l \in C^T$ where $s_l = agrmax(len(s_i))$ with $s_i \in C^T$. The latest *CSF* sequence of S^T is a sequence s_n, which appears latest in C^T, such that $s_n \in C^T$, with $n = \|C^T\|$.

We hypothesized that the excessive accumulation of technical debt leads to students abandoning debt repayment. In other words, the team may choose to ignore the debt when the length of the *CSF* sequence exceeds a certain threshold. In other words, the effort required to clear up the summed debts exceeds the capacity of the student under current project demands and time constraints. Thus, we inspected the lengths of the debt sequences to identify this threshold by dividing the teams into two groups based on the final status of their projects: *BS. Final* (i.e., *BuildSucc* in the final product; 19 teams) and *CSF. Final* (i.e., *CSF* in the final product; 22 teams).

The data in Table 5 and the density plot in Fig. 9 reveal that *BS. Final* teams made more frequent attempts to fix all of their *CSF*s. For the most part, they kept the number of accumulated *CSF*s below 15, allowing it to reach 30 only in urgent situations. *CSF. Final* teams paid little to no attention to *CSF*s, as indicated by a failure to reduce the length of *CSF* sequences to any lower bound. Overall, the length of *CSF* sequences exceeded that of their more diligent counterparts. The mean probability of reaching *BuildSucc* status was as follows: *CSF. Final* (0.06 per submission) and *BS. Final* (0.28 per submission)

The scatter plot in Fig. 10 illustrates the relationship between the longest *CSF* sequence and the latest *CSF* sequences of the 41 teams. Most of the *BS. Final* teams maintained the accumulated technical debt below 40 and rarely delayed fixing all *CSF*s until the due day. The behaviors in *CSF. Final* teams were precisely the opposite, as indicated by the fact that their longest CSF sequences were also their latest *CSF* sequences. The length of *CSF* sequences confirmed our hypothesis pertaining to the weight of technical debt and the abandonment of debt repayment. Based on the data in Fig. 10, it appears that the threshold is roughly 40, considering that only 33% of the teams made an effort to fix *CSF* sequences comprising more than 40 submissions. We therefore recommend that instructors initiate intervention at around this point. For instance, if a team has not reached *BuildSucc* status after more than 30 submissions, then they should be encouraged to dedicate

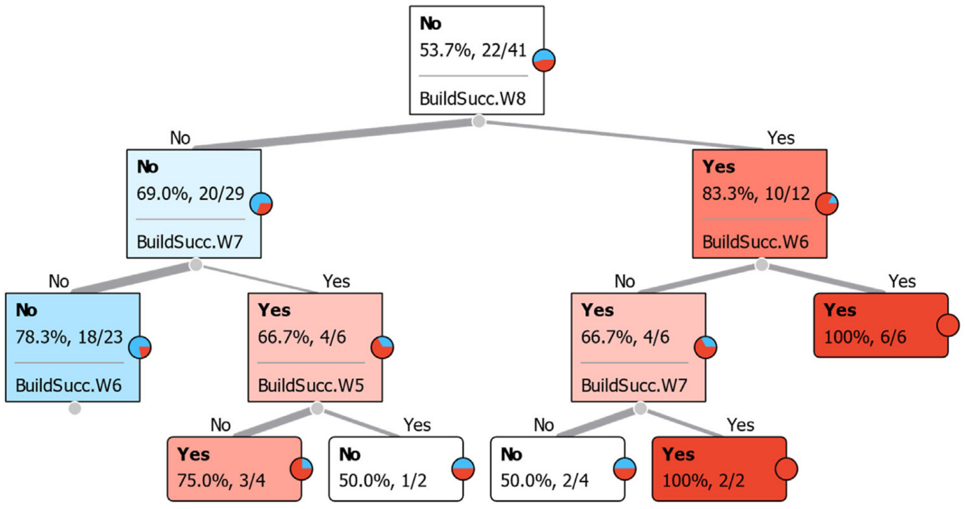


Fig. 11. Visualization of the decision tree that predicts the final project status based on prior weeks' success.

themselves to code quality issues before adding any more source code.

RQ1.5. *What is the most appropriate time to repay all technical CSF debts?*

To elucidate the impact of promptly addressing *CSF*s, we built a prediction model to estimate the final status based on weekly success in fixing all *CSF*s. The prediction target was a binary attribute denoting whether a team reaches *BuildSucc* in the final product. We omitted the first weeks, due to the fact that all of the teams reached *BuildSucc* in their initial submissions. We also eliminated data from the final week because it actually contained the final status. Using data from the remaining 7 weeks (weeks 2 to 8), we encoded the record of each team as a binary vector in which each value (Yes/No) denoted *BuildSucc* status in the corresponding week. Decision tree induction was used to find patterns determining the final success of teams in addressing all coding style issues.

The decision tree in Fig. 11 shows that the effort exerted in week 8 presented the highest predictive value for final *BuildSucc*, as indicated by its position at the root of the decision tree. Success in weeks 6 and 7 was equally indicative of final success. Furthermore, every team that achieved *BuildSucc* in week 8, as well as week 6 or week 7, achieved *BuildSucc* in the final product. Conversely, 78% of the teams that failed to repay their technical debt in week 8 and week 7 obtained *CSF* in the final week. The absence of the first four weeks in the decision tree indicates that code polishing could be postponed until 50% of the available time has passed. Beyond the fourth week, any delay in fixing *CSF*s brings with it the risk of accumulating a critical amount of technical debt, which cannot be corrected prior to the project deadline.

The above analysis shows how ProgEdu can be used to facilitate the monitoring of *student teams* engaged in programming projects. Visualizations of GitLab log data revealed (in real-time) a number of interesting patterns pertaining to team coordination, such as critical weeks in which group cooperation appears to have a more pronounced effect on project outcomes. Code assessment results revealed the strategies (or lack thereof) aimed at dealing with code quality. As shown in Fig. 7b, many members were not expected to write clean code, as indicated by the fact that less than one-third of the students achieved *BuildSucc* every week. Indeed, it appears that specific members were responsible for fixing CSFs. Nonetheless, this kind of behavior should be regulated by instructors, due to the fact that code quality is crucial to productivity and is considered a crucial soft skill. Our analysis of *CSF* sequences also revealed patterns that appeared to affect the success of teams in enhancing code quality. We recommend that instructors intervene midway through a project when dealing with teams that have been unable to eliminate code-quality issues, and/or teams that are consistently receiving *CSF*s after more than 30 submissions.

Responses to the first research question demonstrate an application of ProgEdu in assisting instructors to monitor student projects at the team level through the advancement of learning analytics. The instructors can observe weekly team efforts, team participation, teams' submission logs, and code-quality improvements during the progress via visualizations of team tracking metrics. Further analysis of the system data also provides good hints to elucidate the associations between team behavior and project grade from which three lessons would be drawn. First, teams who were negatively engaged in project developments in the latter four weeks would not have good achievements. Second, teams who let the length of *CSF* sequences be longer than 30 would be failing to repay the technical debt. Third, if a team failed to repay all technical debts two weeks before the deadline, they would never do that. Based on these experiences, the instructors can determine how and when to command interventions by warning teams to increase project engagement and to correct code quality issues.

4.3.2. RQ2: At the individual level, do measures pertain to code-quality improvements provide an accurate assessment of the contributions of students?

This research question addresses the effectiveness of the proposed metrics used to assess individual contributions. We converted the issue into a problem of clustering, in which students making similar contributions are assigned to the same group. Here, we use the quality of clustering results to assess the applicability of metrics used to quantify individual contributions to improving code quality.

We used nine metrics of individual contributions to calculate a contribution index (CI) and contribution max-difference (CM) for each member vs. their team. The nine metrics are listed in Table 1 and the formulas used to derive CI and CM are listed in Table 2. Thus, individual records are represented by a vector comprising 18 measures: 9 for CI and 9 for CM. Differences between individuals were based on Euclidean distance.

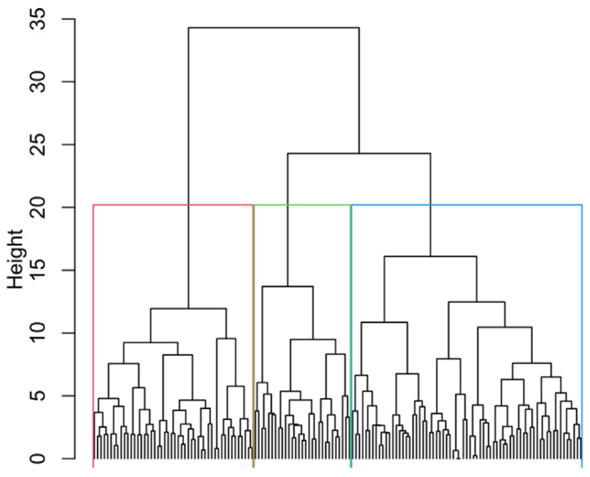


Fig. 12. Dendrogram generated by agglomerative hierarchical clustering.

We employed agglomerative hierarchical clustering with Ward's minimum variance (Ward, 1963) to identify the clusters. The best number of clusters can be clarified by observing the dendrogram generated by the clustering algorithm. Based on the heigh of the dendrogram in Fig. 12, three clusters with significant differences can be determined. Fig. 13a illustrates the distribution of clusters in a two-dimensional space, which was reduced from the original 18-dimensional space using principal component analysis (PCA, Jolliffe, 1986). We also plotted free riders on a scatter plot, in which 19 of the 20 free riders ended up in a single cluster.

Based on individual contribution levels, the three clusters were named *High contribution (C1)*, *Average contribution (C2)*, and *Low contribution (C3)*. A comparison of the three is presented in the form of a boxplot in Fig. 13b. The boxplot clearly indicates how the limited contributions of team members in *C3* yielded low project grades; however, they do not clearly indicate the differences between the remaining groups, *C1* and *C2*. We therefore performed a detailed cluster analysis to elucidate these differences.

We observed clear differences in participation patterns in the temporal domain, as shown in Fig. 14. Individuals in *C1* were more active from the beginning and their participation remained consistent throughout the project. Specifically, they began submitting code from the onset of the project and gradually increased their contributions from the fifth week until the due date. Individuals in *C2* and *C3* were less active during the first two-thirds of the project period, gradually becoming more active as the deadline approached. Individuals in *C2* were active earlier and had two peaks of contributions in the seventh and final weeks, whereas individuals in *C3* remained unresponsive until the last few days prior to the deadline.

Mean value comparisons were conducted to elucidate differences between the three group profiles. Kruskal–Wallis test results revealed significant differences between C2 and C3 in terms of the mean of contributions. Post-hoc analysis was used to clarify between-profile differences. As depicted in Table 6, our results made it clear that individuals in *C1* made greater contributions to the projects, as indicated by their high number of submissions (*SubmTotal*, $m = 30.72$). Intuitively, we expected that students who make more submissions would also produce fewer errors in their code; however, we found that they actually produced a larger number of errors. This can be attributed to the fact that the number of errors was proportional to the number of submissions. Nonetheless, these active members also made a greater contribution to fixing *CSFs*, as indicated by higher mean numbers of *PartFix* and *UnFix*. Note that this behavior also earned them higher project grades and the esteem of their teammates, as indicated by high ratings.

As shown in Table 6, the posthoc analysis revealed that many of the differences in contribution between *C1* and *C2* were insignificant. Significant differences in *HtmlCSF*, *IntroErr*, and *ParFix* suggest that individuals in *C1* made more of a contribution to modifying code and resolving code quality issues than did their counterparts in *C2*. This indicates that in cases where two or more members make roughly the same number of submissions, instructors can differentiate between them based on their contributions to improving code quality. Higher scores should go to those who make submissions that lead to improvements in code quality. The contributions of individuals in *C3* were significantly less than those of their counterparts in all measures. Individuals in C3 contributed only one-third as many submissions as *C1*, and half as many as *C2* (*SubmTotal*, $m = 9.54$ vs. $m = 30.72$ and 19.44). Furthermore, their contribution to addressing code quality issues was negligible (*PartFix*, $m = 0.27$; and *AllFix*, $m=0.25$). As a result, those individuals earned poor assessments from their teammates and the instructor.

As this was a programming course, the instructor asked all the students to serve as a programmer role who must contribute code to their team projects. Ideally, each team member should contribute system features equally. However, please note high contribution members would not always be the member with the highest number of submissions. There are two findings in such a phenomenon: (i) high contribution members can complete assigned features with a stable pace of code commits, whereas another two kinds of members tend to start developing assigned features late before the deadline; (ii) high contribution members are usually team leaders who are active in discussing, designing, planning, and assigning tasks to others but less frequent in making submissions in the last weeks, therefore, their measures in the system would be lower than other members. In such a case, the instructors can count these management tasks in the peer-evaluation questionnaire, i.e., Appendix A, to determine their level of contributions manually via individual interviews.

We performed a crosscheck to validate the clustering scheme against peer evaluation results. Overall, the mean value of contribution percentage of all students was 80.82% (sd = 38.84) which illustrates that, from peers' perspective, students generally contributed less than they should. By member profiles, contribution of students in *C1* are highly rated by their peers *(*$m = 106.89$*,* $sd = 26.64$*)* than students in *C2 (*$m = 92.39$*,* $sd = 29.48$*, not significant difference*). In contrast, the peer evaluation results of students in *C3* were significantly lower than those of the other two groups. The distribution of peer-evaluation results among the three groups depicted in Fig. 15 illustrates the feasibility of our individual assessment scheme.

Our data-driven approach to evaluation was highly consistent with the results of peer evaluations, and the proposed scheme proved particularly effective in identifying free riders, as shown in Fig. 13a. This should make it far easier for instructors to deal with free riders and formulate accurate assessments at the individual level.

The detection of free riders is crucial to the assessment process. Thus, we sought to determine the impact of the proposed metrics on the problem of identifying free riders using machine-learning models. We used as predictive features the same 18 features previously used for clustering. We also selected four well-known classification algorithms for model construction: support vector machine (SVM), logistic regression (LR), K-nearest neighbors (KNN), and random forest (RF). We randomly selected 66%

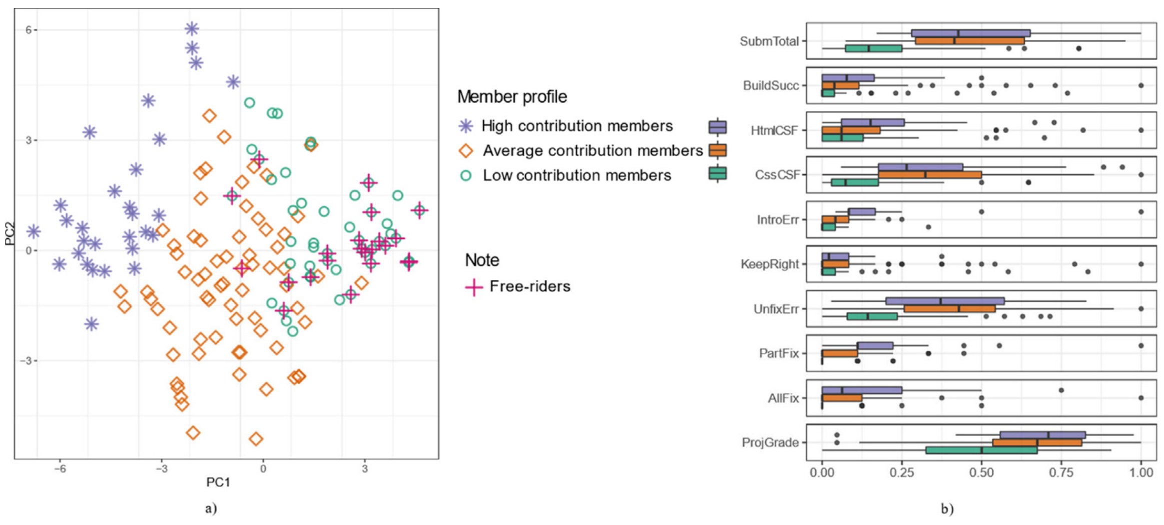


Fig. 13. (a) Distribution of member profiles and free-riders in 2D-space; (b) Comparison between member profiles in boxplots.

Table 6
Descriptions of three-member profiles.

	C1. High contribution member	C2. Average contribution member	C3. Low contribution member	Kruskal Wallis	Between profile posthoc analysis
	N = 29(20%)	N = 69(47%)	N = 48(33%)	χ^2	
SubmTotal	30.72 (28.42)	19.44 (8.72)	9.45 (8.02)	46.94**	1, 2 > 3
BuildSucc	5.20 (8.97)	3.04 (5.02)	1.62 (3.93)	13.83**	1, 2 > 3
HtmlCSF	8.24 (8.67)	4.79 (6.76)	3.62 (4.94)	9.752**	1 > 2, 3
CssCSF	17.20 (17.50)	11.60 (7.94)	4.20 (5.30)	37.50**	1, 2 > 3
IntroErr	4.48 (4.71)	1.20 (1.27)	0.68 (1.30)	50.03**	1 > 2 > 3
KeepRight	3.17 (6.61)	2.37 (4.55)	1.37 (3.82)	7.39*	
UnfixErr	19.86 (18.16)	14.60 (8.20)	6.87 (6.39)	33.59**	1, 2 > 3
PartFix	2.17 (1.87)	0.59 (0.92)	0.27 (0.53)	47.95**	1 > 2, 3
AllFix	1.86 (2.70)	0.66 (1.22)	0.25 (0.70)	16.15**	1 > 3
ProjGrade	79.17 (9.05)	76.63 (8.03)	68.47 (9.99)	26.93**	1, 2 > 3

Note: Between-profile comparisons were performed using the non-parametric Kruskal–Wallis test. Post-hoc comparisons indicate that the differences are significant. *: $p < 0.05$; **: $p < 0.01$.

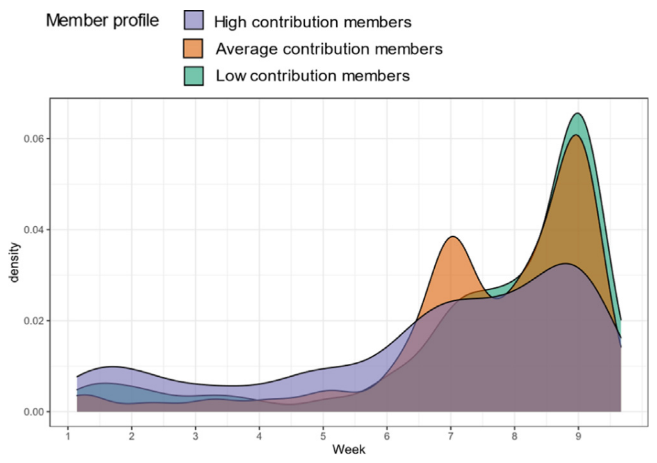


Fig. 14. Density plot of submissions made by the three-member profiles.

Table 7
Performance of free-rider prediction.

Model	Acc	AUC	F1	Pre	Rec
SVM	0.90	0.94	0.62	0.74	0.54
LR	0.87	0.91	0.63	0.60	0.66
KNN	0.90	0.93	0.66	0.73	0.60
RF	**0.93**	**0.96**	**0.77**	**0.82**	**0.73**

of the data for training and used the remainder for testing. To evaluate prediction performance, we repeated the train/test procedure in ten iterations and reported the test scores in terms of *Accuracy (Acc)*, *Area Under the Curve (AUC)*, and three metrics derived from the confusion matrix (*Precision (Pre)*, *Recall (Rec)*, and *F1-measure (F1)*). The predictive performance of free-riders prediction is listed in Table 7.

Our testing results led us to conclude that measures of individual contributions derived from student submissions can be used to build models for the identification of free riders with an accuracy of more than 90%. Recall was identified as the most important metric, based on the fact that our primary objective was to detect as many free riders as possible. The test scores in Table 7 show that RF outperformed the other methods in all test metrics, with a recall of 73%. This means that nearly 75% of free riders can be identified using supervised learning models. The top-6 features and their impact on RF prediction are illustrated in Fig. 16. It is easy to explain why the features that are most effective in detecting free riders are related to the total number of submissions. Low CI values in generating *CssCSF* and

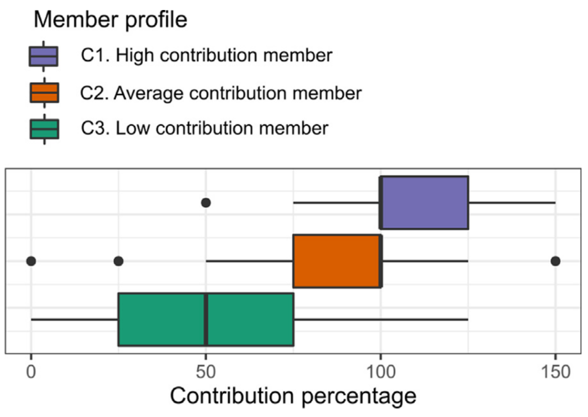


Fig. 15. Distributions of peer-evaluation results differentiated by member profiles.

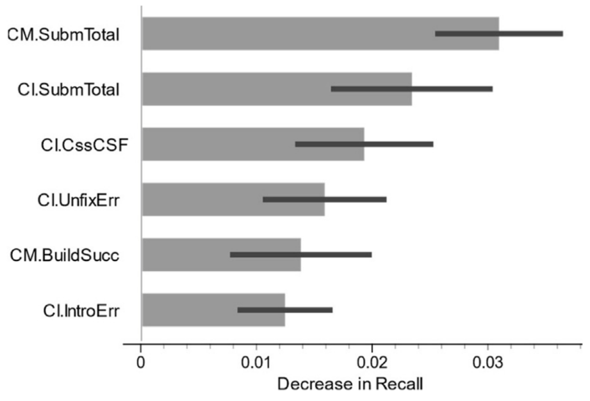


Fig. 16. Importance of top-6 features used in RF prediction model.

its consequent transition *UnFix* also appear to be indicators of free riders. Note that the appearance of *CM. BuildSucc* and *CI. IntroErr* in the top-6 features indicates that the ability to solve existing code quality errors could help to discriminate between normal contributors and free riders.

The acceptable rate of free-rider identification, again, promotes the impact of code quality assessment in evaluating individual contributions. Our code-quality-based metrics helped to clarify the contributions of individual team members and identify free riders. Statistical analysis based solely on the number of submissions is easily manipulated simply by making unsubstantial submissions with meaningless source code.

Analysis results for RQ2 show that measures pertain to code-quality improvements provide a feasible assessment of the contributions of students at the individual level. Using the proposed measures, both supervised and unsupervised algorithms provide feasible hints to the instructors in individual assessments. However, please note that *free-riders* or *low contribution members* found in this step should not be identified as actual free-riders in the final evaluation of instructors. The detection results just can be used as one of important evidence for the instructors to individual assessment since there are multiple reasons for free riding in teamwork. For example, the appearance of "lone-wolves" or "poor drivers" in a team might encourage remaining members to be free-riders (Ferguson et al., 2020). In teamwork, lone-wolves are those students who feel overconfident about their academic performance, lack confidence in other members' abilities, and therefore, need to demonstrate their superiority in the team (Barr et al., 2005). In our data, we recognized a lone wolf appeared in Team11 with a high frequency of submission during the first three weeks, as shown in Fig. 8. As a result, such behavior boosts the values of CM measures of other members and made them be identified as low contribution members. Since the measures of CM are sensitive with such submission-intensive behavior, we suggest that not only the free-riding should be combat, but also the hard-working of *diligent isolates* should be warned to maintain the team collaboration (Pieterse and Thompson, 2010).

4.4. Discussions

The objectives of this empirical study are to examine the applicability of utilizing a Git-based assessment system as a collaborative platform for team programming projects. Study results show that the proposed system and monitoring scheme can gain actionable insights at the team level and the individual level, as well as can facilitate instructors in project assessment. Further, this study also provides useful lessons learned to the instructors to improve the effectiveness of the system in future courses.

The monitoring data show that the system capable to keep track of team dynamics during the progress. The system log data facilitated learning analytics tasks in which measures and visualization demonstrated their efficacy in providing insights and actionable information to system users. On a weekly basis, both team productivity and team participation were well measured and visualized by TSC and TPR, respectively. Visualizations on the measures can play as a self-reflection tool to the team as well as a monitoring tool to the instructors, as illustrated in Fig. 5. Correlations between average values of TSC, TPR, and the project grades reveal that the monitoring scheme can help the instructor determine factors influencing project outputs. Those teams with a better effort in submissions and collaboration can receive higher grades. Note that almost all teams became active after the progress report in week 6, which led to some teams were not able to finish the project with high-quality products and clean code. Therefore, a higher frequency of progress reports and reminders from the instructors can enhance team engagement for better results.

From the perspective of code quality, although nearly half of the teams were not successful in presenting the final product with high coding quality, there were important lessons learned from students' behaviors in improving code quality. Since we only want to test the system capability in monitoring team projects without strict policy or intervention, too many submissions with no code quality improvement were made. In future courses, the instructor can enforce students to correct all *CSF*s after a limited number of submissions, and the system offers warning messages to the teams that accumulate long *CSF* sequences. The teams are also recommended to repay their technical debts frequently, e.g., weekly, or bi-weekly, to avoid the burden of accumulated debts. Especially, two weeks before the deadline is an appropriate time for teams to fix all their *CSF*s before any continuation. We believe that both team collaboration and code quality can be improved via appropriate interventions from instructors.

Through learning analytics, individual contributions can be assessed effectively using hierarchical clustering. Students were segmented into three clusters accordingly to their levels of contributions. Based on the measures of *CI* and *CM*, instructors have more evidence in individual grading, besides the peer-evaluation results which are sensitive to the hallow effect and collusion. In general, the high contribution members submitted more code and fixed more *CSF*s leading to high values of *CI* and *CM*. However, it is interesting that they also generated a large number of *CSF*s, which contradicts the findings of Chen et al. (2020), which indicated that code quality was proportional to the number of submissions. This discrepancy can be explained by a difference in context. The scale of individual programming assignments is far smaller than that of team projects, such that students do not rely on incremental improvements. The compact code submitted in most small-scale projects allows for greater care in terms of coding style, whereas larger projects impose a wide range of demands, such that coding style is often given lower priority. High contribution members in this study made considerable efforts to resolve *CSF*s, which is consistent with the behavior of high-performance learners in previous research (Chen et al., 2020).

The identification of free-riders shows another application of learning analytics on system data. Please note that, although low-contribution members and free-riders could be detected effectively, the instructors should not rely on only these results for individual assessment. It is much better to boost the level of engagement of these students during the project progress using the weekly values of *CI* and *CM*. Besides, considering the excessive efforts of high-contribution members also benefit to team dynamic through combating both lone-wolves and free-riders.

The findings of this study provide a starting point to develop a learning analytics dashboard for the APAS. Visualizations on weekly activities of teams not only put the team projects into a reflexive learning loop but also provides a useful basis for developing a group awareness tool, where students are guided to pay more attention to self and teammates' participation and self-adjust their behaviors (Li et al., 2021). We leave these remarks to the future versions of ProgEdu.

4.5. Limitations and threats to validity

This study has several limitations. First, only GitLab log data and assessment results were employed, which only elucidate the engagement history of team members. There is a great deal of available data, e.g., the *number of added/removed lines of code, static code analysis feedback, bugs, vulnerabilities, and code smells from SonarQube*, which may give a better understanding of coding behaviors and frequent errors that students often encountered. Second, the efficacy of code quality assessment in enhancing collaborative programming using ProgEdu has not been evaluated yet. Comparative research can be conducted to elucidate how better novice programmers can participate when the code quality is ensured. Results of this study can make a significant voice in promoting the teaching of code quality assurance skills in higher education. Third, the lack of functional validation concerning user requirements is also a limitation. Due to the variety of project topics, it is difficult for the instructors to build all test cases. However, the test cases can be developed for common use cases such as login, user registration, etc. These notes can be sent to the instructors to improve in future courses.

Besides, it is important to note some threats to the validity of reported results. First, the analyzed data was extracted from a programming course in a university in central Taiwan. Consequently, the reported results are not general. Second, all proposed metrics in the studies were subjective to the specific languages taught in the course which were HTML and CSS. Hence, they cannot be reused in those courses teaching other languages. In such a case, the metrics to evaluate students' effort should be refined to the current language. Third, the data obtained in this study might be governed by the course' instructors, who are also the authors of this article. In the current stage of the system, we only aim to examine its monitoring capability for team projects, so no interventions were offered. Under the influence of any changes in the project instructions, students' behaviors might significantly shift causing some behavioral patterns in the current paper, such as free-riders and lone-wolves, might be no longer exist. Finally, the individual project grades were subjectively given by the instructors based on the results of the individual interview during the final presentations. Although peer review results were used, they were not the only data source for individual assessment in this course. Therefore, the same evaluation results might not appear in other scenarios.

5. Conclusions

This paper introduces a novel APAS with code-quality awareness aimed at facilitating the management of team projects in the field of programming. The infrastructure of the proposed system is based largely on Git-driven technologies in conjunction with a continuous integration server. We also leveraged open-source static code analysis tools to build automated assessment features. Code-quality-based metrics were used to measure the contributions of individual team members to their respective projects. System logs were extracted for learning analytic tasks, by which team collaboration can be visualized and presented to instructors for team regulation. Individual members were then categorized according to their levels of contributions. Our use of unsupervised learning proved particularly effective in identifying free riders, identifying close to 75% using the RF classifier.

In conclusion, our study demonstrates the efficacy of leveraging an APAS as a collaborative platform for team projects following an agile development approach. The capture of iterative improvements in project source code gives instructors a great deal of data by which to characterize the behavior of students, detect nonconformities in teamwork, and perform assessments at the individual level. We also demonstrate the effectiveness of learning analytics in facilitating efforts to manage student projects. This method is easily replicated by other educators in other programming languages. In the future, we aim to analyze the effect of feedback on improving code quality over time.

CRediT authorship contribution statement

Hsi-Min Chen: Conceptualization, Methodology, Software, Writing – review and editing. **Bao-An Nguyen:** Writing – original draft, Visualization, Data curation. **Chyi-Ren Dow:** Supervision, Resources.

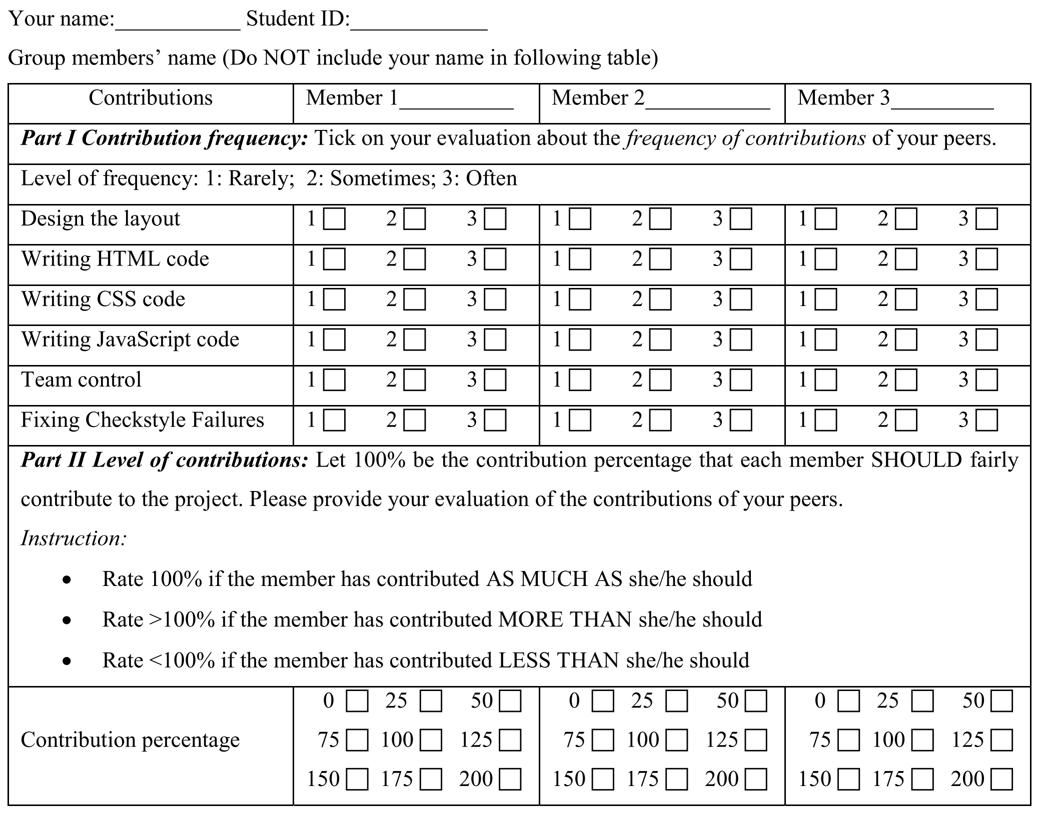
Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported in part by Ministry of Science and Technology, Taiwan R.O.C., under grants no. 109-2221-E-035-055 -MY3.

Appendix A. Peer-evaluation questionnaire



Your name:___________ Student ID:____________

Group members' name (Do NOT include your name in following table)

Contributions	Member 1__________	Member 2___________	Member 3_________
Part I Contribution frequency: Tick on your evaluation about the *frequency of contributions* of your peers.			
Level of frequency: 1: Rarely; 2: Sometimes; 3: Often			
Design the layout	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Writing HTML code	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Writing CSS code	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Writing JavaScript code	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Team control	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Fixing Checkstyle Failures	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐	1 ☐ 2 ☐ 3 ☐
Part II Level of contributions: Let 100% be the contribution percentage that each member SHOULD fairly contribute to the project. Please provide your evaluation of the contributions of your peers. *Instruction:* • Rate 100% if the member has contributed AS MUCH AS she/he should • Rate >100% if the member has contributed MORE THAN she/he should • Rate <100% if the member has contributed LESS THAN she/he should			
Contribution percentage	0 ☐ 25 ☐ 50 ☐ 75 ☐ 100 ☐ 125 ☐ 150 ☐ 175 ☐ 200 ☐	0 ☐ 25 ☐ 50 ☐ 75 ☐ 100 ☐ 125 ☐ 150 ☐ 175 ☐ 200 ☐	0 ☐ 25 ☐ 50 ☐ 75 ☐ 100 ☐ 125 ☐ 150 ☐ 175 ☐ 200 ☐

Appendix B. Grading procedure

Contribution Percentage (CP) of a member is calculated by taking average values of contribution percentages evaluated by other team members.

Data smoothing by bin boundary then was applied to replace CP values by multiples of 25. As a result, CP can receive one of seven values of contribution percentage used in the questionnaire.

Final individual grade (FIG) = Project product grade (PPG) + Individual contribution grade (ICG)

- PPG is given equally for all members ($60 \leq \text{PPG} \leq 90$)
- ICG = 20 ∗ (CP−100)/100

For example: Given a team wit h the project was given a PPG as 80. Three team members A, B, C were evaluated with CP values 75%, 100% and 150%, respectively. FIG of each member is calculated as follows:

$$FIG(A) = 80 + 20 * (75 - 100)/100 = 75$$

$$FIG(B) = 80 + 20 * (100 - 100)/100 = 80$$

$$FIG(C) = 80 + 20 * (150 - 100)/100 = 90$$

References

Athanasiou, D., Nugroho, A., Visser, J., Zaidman, A., 2014. Test code quality and its relation to issue handling performance. IEEE Trans. Softw. Eng. 40, 1100–1125. http://dx.doi.org/10.1109/TSE.2014.2342227.

Barr, T.F., Dixon, A.L., Gassenheimer, J.B., 2005. Exploring the Lone wolf phenomenon in student teams. J. Mark. Educ. 27, 81–90. http://dx.doi.org/10.1177/0273475304273459.

Black, S., Boca, P.P., Bowen, J.P., Gorman, J., Hinchey, M., 2009. Formal versus agile: Survival of the fittest. Computer (Long. Beach. Calif) 42, 37–45. http://dx.doi.org/10.1109/MC.2009.284.

Blumenstein, M., Green, S., Fogelman, S., Nguyen, A., Muthukkumarasamy, V., 2008. Performance analysis of GAME: A generic automated marking environment. Comput. Educ. 50, 1203–1216. http://dx.doi.org/10.1016/j.compedu.2006.11.006.

Breuker, D.M., Derriks, J., Brunekreef, J., 2011. Measuring static quality of student code. In: ITiCSE'11 - Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science. ACM Press, New York, New York, USA, pp. 13–17. http://dx.doi.org/10.1145/1999747.1999754.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N., 2010. Managing technical debt in software-reliant systems. In: Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010. pp. 47–51. http://dx.doi.org/10.1145/1882362.1882373.

Buffardi, K., 2020. Assessing individual contributions to software engineering projects with git logs and user stories. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education. ACM, New York, NY, USA.

Campbell, D.T., 1979. Assessing the impact of planned social change. Eval. Program Plan. 2, 67–90. http://dx.doi.org/10.1016/0149-7189(79)90048-X.

Cardell-Oliver, R., 2011. How can software metrics help novice programmers? In: Proceedings of the Thirteenth Australasian Computing Education Conference-Vol. 114. pp. 55–62.

Chen, H.-M., Chen, W.-H., Lee, C.-C., 2018. An automated assessment system for analysis of coding convention violations in Java programming assignments. J. Inf. Sci. Eng. 34, 1203–1221. http://dx.doi.org/10.6688/jise.201809_34(5).0006.

Chen, H.-M., Nguyen, B.-A., Yan, Y.-X., Dow, C.-R., 2020. Analysis of learning behavior in an automated programming assessment environment: A code quality perspective. IEEE Access 8. http://dx.doi.org/10.1109/access.2020.3024102.

Chua, Y.H.V., Rajalingam, P., Tan, S.C., Dauwels, J., 2019. EduBrowser: A multimodal automated monitoring system for co-located collaborative learning. Commun. Comput. Inf. Sci. 1011, 125–138. http://dx.doi.org/10.1007/978-3-030-20798-4_12.

Cico, O., Jaccheri, L., Nguyen-Duc, A., Zhang, H., 2021. Exploring the intersection between software industry and software engineering education - A systematic mapping of software engineering trends. J. Syst. Softw. 172, 110736. http://dx.doi.org/10.1016/j.jss.2020.110736.

Cunningham, W., 1992. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger 4, 29–30.

De Bassi, P.R., Puppi, G.M., Banali, P.H., Paraiso, E.C., 2018. Measuring developers' contribution in source code using quality metrics. In: Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design, CSCWD 2018. pp. 869–871. http://dx.doi.org/10.1109/scswd.2018.8465320.

Edwards, S.H., Kandru, N., Rajagopal, M.B.M., 2017. Investigating static analysis errors in student Java programs. In: ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research. Association for Computing Machinery, Inc, New York, NY, USA, pp. 65–73. http://dx.doi.org/10.1145/3105726.3106182.

Eraslan, S., Kopec-Harding, K., Jay, C., Embury, S.M., Haines, R., Cortés Ríos, J.C., Crowther, P., 2020. Integrating GitLab metrics into coursework consultation sessions in a software engineering course. J. Syst. Softw. 167, 110613. http://dx.doi.org/10.1016/j.jss.2020.110613.

Fellenz, M.R., 2006. Toward fairness in assessing student groupwork: A protocol for peer evaluation of individual contributions. J. Manage. Educ. 30, 570–591. http://dx.doi.org/10.1177/1052562906286713.

Ferguson, E., Shichman, R., Tan, J.H.W., 2020. When lone wolf defectors undermine the power of the opt-out default. Sci. Rep. 10, 1–12. http://dx.doi.org/10.1038/s41598-020-65163-1.

Fernández, A., Luis, J., 2011. Automated assessment in a programming tools course. IEEE Trans. Educ. 54, 576–581. http://dx.doi.org/10.1109/TE.2010.2098442.

Fowler, M., 2006. Continuous integration [WWW Document]. URL https://martinfowler.com/articles/continuousIntegration.html (accessed 11.20.20).

Gary, K.A., Xavier, S., 2015. Agile learning through continuous assessment. In: Proceedings - Frontiers in Education Conference, FIE. Institute of Electrical and Electronics Engineers Inc., http://dx.doi.org/10.1109/fie.2015.7344278.

Grivokostopoulou, F., Perikos, I., Hatzilygeroudis, I., 2017. An educational system for learning search algorithms and automatically assessing student performance. Int. J. Artif. Intell. Educ. 27, 207–240. http://dx.doi.org/10.1007/s40593-016-0116-x.

Guo, Y., Seaman, C., 2011. A portfolio approach to technical debt management. In: Proceedings - International Conference on Software Engineering. pp. 31–34. http://dx.doi.org/10.1145/1985362.1985370.

Haaranen, L., Lehtinen, T., 2015. Teaching git on the side - version control system as a course platform. In: Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE 2015-June. pp. 87–92. http://dx.doi.org/10.1145/2729094.2742608.

Heckman, S., King, J., 2018. Developing software engineering skills using real tools for automated grading. In: SIGCSE 2018 - Proc. 49th ACM Tech. Symp. Comput. Sci. Educ. 2018-Janua. pp. 794–799. http://dx.doi.org/10.1145/3159450.3159595.

Hundt, C., Schlarb, M., Schmidt, B., 2017. SAUCE: A web application for interactive teaching and learning of parallel programming. J. Parallel Distrib. Comput. 105, 163–173. http://dx.doi.org/10.1016/j.jpdc.2016.12.028.

Johnston, L., Miles, L., 2004. Assessing contributions to group assignments. Assess. Eval. High. Educ. 29, 751–768. http://dx.doi.org/10.1080/0260293042000227272.

Jolliffe, I.T., 1986. Principal component analysis and factor analysis. In: Principal Component Analysis. Springer New York, New York, NY, pp. 115–128. http://dx.doi.org/10.1007/978-1-4757-1904-8_7.

Kaya, M., Özel, S.A., 2015. Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments. Comput. Appl. Eng. Educ. 23, 363–373. http://dx.doi.org/10.1002/cae.21606.

Kelleher, J., 2014. Employing git in the classroom. In: 2014 World Congress on Computer Applications and Information Systems (WCCAIS). IEEE, pp. 1–4. http://dx.doi.org/10.1109/wccais.2014.6916568.

Keuning, H., Jeuring, J., Heeren, B., 2018. A systematic literature review of automated feedback generation for programming exercises. ACM Trans. Comput. Educ. 19. http://dx.doi.org/10.1145/3231711.

Kirk, D., Crow, T., Luxton-Reilly, A., Tempero, E., 2020. On assuring learning about code quality. In: ACE 2020 - Proceedings of the 22nd Australasian Computing Education Conference, Held in Conjunction with Australasian Computer Science Week. pp. 86–94. http://dx.doi.org/10.1145/3373165.3373175.

Layton, R., Ohland, M., Pomeranz, H., 2007. Software for student team formation and peer evaluation: CATME incorporates Team-Maker. In: Proceedings of the American Society for Engineering Education Annual Conference & Exposition. Honolulu.

Li, Y., Li, Xiaoran, Zhang, Y., Xin, Li, 2021. The effects of a group awareness tool on knowledge construction in computer-supported collaborative learning. Br. J. Educ. Technol. 52, 1178–1196. http://dx.doi.org/10.1111/bjet.13066.

Loughry, M.L., Ohland, M.W., DeWayne Moore, D., 2007. Development of a theory-based assessment of team member effectiveness. Educ. Psychol. Meas. 67, 505–524. http://dx.doi.org/10.1177/0013164406292085.

Lu, Y., Mao, X., Li, Z., Zhang, Y., Wang, T., Yin, G., 2018. Internal quality assurance for external contributions in GitHub: An empirical investigation. J. Softw. Evol. Process. 30, e1918. http://dx.doi.org/10.1002/smr.1918.

Maiden, B., Perry, B., 2011. Dealing with free-riders in assessed group work: Results from a study at a UK university. Assess. Eval. High. Educ. 36, 451–464. http://dx.doi.org/10.1080/02602930903429302.

Martin, R.C., 2009. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson Education.

Masood, Z., Hoda, R., Blincoe, K., 2018. Adapting agile practices in university contexts. J. Syst. Softw. 144, 501–510. http://dx.doi.org/10.1016/j.jss.2018.07.011.

Neyem, A., Benedetto, J.I., Chacon, A.F., 2014. Improving software engineering education through an empirical approach: Lessons learned from capstone teaching experiences. In: SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education. Association for Computing Machinery, New York, New York, USA, pp. 391–396. http://dx.doi.org/10.1145/2538862.2538920.

Nguyen, B.-A., Ho, K.-Y., Chen, H.-M., 2020. Measure students' contribution in web programming projects by exploring source code repository. In: International Computer Symposium. Tainan, Taiwan.

Parizi, R.M., Spoletini, P., Singh, A., 2019. Measuring team members' contributions in software engineering projects using git-driven technology. In: Proceedings - Frontiers in Education Conference, FIE. Institute of Electrical and Electronics Engineers Inc, http://dx.doi.org/10.1109/fie.2018.8658983.

Patton, A.L., McGill, M., 2006. Student portfolios and software quality metrics in computer science education. J. Comput. Sci. Coll. 21, 42–48.

Pettit, R., Prather, J., 2017. Automated assessment tools: Too many cooks, not enough collaboration. J. Comput. Sci. Coll. 32, 113–121.

Pieterse, V., Thompson, L., 2010. Academic alignment to reduce the presence of social loafers and diligent isolates in student teams. Teach. High. Educ. 15, 355–367. http://dx.doi.org/10.1080/13562517.2010.493346.

Rodríguez-del Pino, J.C., Rubio Royo, E., Hernández Figueroa, Z., 2012. A virtual programming lab for moodle with automatic assessment and anti-plagiarism features. In: Proceedings of the 2012 International Conference on E-Learning, e-Business, Enterprise Information Systems, & e-Government. Las Vegas, USA. pp. 80–85.

Raibulet, C., Arcelli Fontana, F., 2018. Collaborative and teamwork software development in an undergraduate software engineering course. J. Syst. Softw. 144, 409–422. http://dx.doi.org/10.1016/j.jss.2018.07.010.

Robinson, P.E., Carroll, J., 2017. An online learning platform for teaching, learning, and assessment of programming. In: IEEE Global Engineering Education Conference, EDUCON. IEEE Computer Society, pp. 547–556. http://dx.doi.org/10.1109/educon.2017.7942900.

Siemens, G., Baker, R.S.J.d., 2012. Learning analytics and educational data mining: Towards communication and collaboration. In: Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, LAK '12. Association for Computing Machinery, New York, NY, USA, pp. 252–254. http://dx.doi.org/10.1145/2330601.2330661.

Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86, 1498–1516. http://dx.doi.org/10.1016/j.jss.2012.12.052.

Ward, J.H., 1963. Hierarchical grouping to optimize an objective function. J. Am. Stat. Assoc. 58, 236–244. http://dx.doi.org/10.1080/01621459.1963.10500845.

Wilkins, D.E., Lawhead, P.B., 2000. Evaluating individuals in team projects. In: SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education). pp. 172–175. http://dx.doi.org/10.1145/331795.331849.

Zagalsky, A., Feliciano, J., Storey, M.A., Zhao, Y., Wang, W., 2015. The emergence of GitHub as a collaborative platform for education. In: Proceedings of the 2015 ACM International Conference on Computer-Supported Cooperative Work and Social Computing. pp. 1906–1917. http://dx.doi.org/10.1145/2675133.2675284.

Hsi-Min Chen received the B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, programming education, object-oriented technology, service computing, and distributed computing.

Email: hmchen@mail.fcu.edu.tw

Bao-An Nguyen received the B.S in Hanoi University of Science and Technology in 2005, the M.S. and Ph.D. degrees in Information Engineering and Computer Science from Feng Chia University, Taiwan, in 2011 and 2021, respectively. He is currently a faculty of the Department of Information Technology, School of Engineering and Technology, Tra Vinh University, Vietnam. His research interests include data mining, software engineering, and education technology.

Email: annb@tvu.edu.vn

Chyi-Ren Dow was born in 1962. He received the B.S. and M.S. degrees in information engineering from National Chiao Tung University, Taiwan, in 1984 and 1988, respectively, and the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh, PA, in 1992 and 1994, respectively. He is currently a Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include mobile computing, ad-hoc wireless networks, agent techniques, fault tolerance, and learning technology.

Email: crdow@mail.fcu.edu.tw