IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 October 2024, accepted 2 December 2024. Date of publication 00 xxxx 0000, date of current version 00 xxxx 0000.

Digital Object Identifier 10.1109/ACCESS.2024.3514843

Detecting Learning Behavior in Programming Assignments by Analyzing Versioned Repositories

JOHN CHEN[1], SERENA LAU[1], JUHO LEINONEN[2], VALERIO TERRAGNI[1], AND NASSER GIACAMAN[1]

[1]Department of Electrical, Computer, and Software Engineering, The University of Auckland, Auckland 1010, New Zealand
[2]Department of Computer Science, Aalto University, 02150 Espoo, Finland

Corresponding author: Nasser Giacaman (n.giacaman@auckland.ac.nz)

This work was supported in part by the Special Interest Group on Computer Science Education (SIGCSE) Special Projects Grant, in May 2021; and in part by The University of Auckland Scholarship of Teaching and Learning Grant, in October 2022.

ABSTRACT Computing education plays a significant role in shaping the calibre of future computing professionals; hence, improving its quality is a valuable endeavour. A promising approach to enhance computing education is leveraging student data from version control systems (VCS). While previous studies have utilised VCS data to predict academic performance, there remains a gap in harnessing this data for learning analytics to understand student learning behaviours in real time. In this research, we introduce the Polivr ecosystem, a comprehensive platform designed to address this gap by utilising VCS data for learning analytics in computing education. The Polivr ecosystem comprises three key modules: Polivr Anonymiser, which ensures data privacy by anonymising student identities; Polivr Core, which mines learning metrics from Git repositories; and Polivr Web Viewer, which transforms the raw metrics into insightful visualisations for educators. We evaluated Polivr using anonymised repositories collected from undergraduate computing courses. The resulting visualisations revealed trends and patterns in student learning behaviours, such as coding habits and progression over time. These insights provide valuable information for educators to enhance teaching strategies and potentially identify at-risk students. This research demonstrates the potential of version control systems as a rich source of learning analytics, contributing to improving computing education by enabling data-driven decision-making in instructional design and student support.

INDEX TERMS Computing education, Git, learning analytics, learning behaviour, mining software repositories, version control.

I. INTRODUCTION

The software industry is one of the fastest growing markets internationally, and a similar rise in computing education is needed to support and reflect the increasing demand for computing professionals [1]. The quality of computing education plays an important role in supporting the software industry; it has a significant impact on the skill, productivity, and success of computing professionals, evident in the industry's pattern of favouring highly rated universities when recruiting [1].

One key way to cultivate high quality computing education is utilising the increasingly prevalent concept of learning analytics. Learning analytics is a methodology for capturing student learning engagement and processes, whereby learning process data is continuously collected [2], [3]. This data could be collected as students progress through programming assignments and can be used to better understand student struggles. Ultimately, computing educators can then be better provisioned for designing instruction and learning interventions tailored to student needs [3].

The goal then is to equip academic institutes with the means to capture learning analytics to enhance computing education. Currently, methodologies of collecting such analytics include formative assessment techniques such as online discussion

The associate editor coordinating the review of this manuscript and approving it for publication was S. Chandrasekaran.

forums [4], [5], [6] and automatic plagiarism detection programs such as MOSS and JPLAG [7], [8]. However, there is a lack of a standardized, quantitative means of evaluating the learning behavior of students.

More recently, the rise of AI tools, such as ChatGPT, makes this especially more difficult. ChatGPT, which stands for 'Chat Generative Pre-trained Transformer', is a highly advanced chatbot tool capable of generating human-language responses for a vast range of domains. Thus, its complexity, accuracy, and ease of access online poses it as a significant threat to the education system [9], [10]. In particular, ChatGPT's powerful ability to generate accurate source code mixed with the low performance of current Artificial Intelligence Generated Content (AIGC) detectors on code-related content makes the evaluation of computing assessment final submissions extremely difficult [11]. This is also where the existing tools mentioned, namely MOSS and JPLAG, struggle in performance. These systems focus on similarity detections, while ChatGPT and other similar AI tools can generate original code that would go unnoticed if utilised [10].

This leads to the motivation to explore new ways to capture learning analytics and student behaviours. An important element of computing education that could be leveraged is the use of version control systems (VCS), a tool often used to simplify change management in software projects [12]. VCSs, e.g. Git and Mercurial, have been widely used and encouraged in computing education for their benefits of making team collaboration and instructor administration easier, and for building student skills with industry tools [13]. As VCSs are able to monitor the activities of individual students throughout the course of a computing assignment, it has a strong potential to be used to capture a novel perspective on student learning analytics in computing education [12]. Data from student software repositories could be collected and analysed using various techniques; this field is known as mining software repositories [14], [15].

Our research project aims to investigate technologies and techniques for utilising version control systems to capture learning analytics and behaviours. The goal is to equip computing education instructors with a novel perspective on their students' learning as they progress, so that they can tailor learning interventions to best enhance their education. This would ultimately improve the quality of computing education and the calibre of graduating students, thus better supporting the computing industry.

The contributions of this report include:

- The development of a framework for mining, processing, and visualising metrics from a Git repository,
- The development of the Polivr[1] ecosystem to actualise the aforementioned framework and equip instructors with learning analytic insights,
- An evaluation of the learning pattern insights procured by the Polivr ecosystem.

[1] polivr.digitaledu.ac.nz

The structure of this report is outlined as follows. A review of the existing literature is discussed in Section II. Next, the conceptual and practical contributions made in this study are explored in Section III. Section IV presents an evaluation of the findings, followed by a discussion of key implications in Section V. Finally, conclusions are drawn and future work is outlined in Section VI.

II. LITERATURE REVIEW

A. LEARNING ANALYTICS IN COMPUTING EDUCATION

Learning analytics, in the context of higher education, involves collecting and analysing student data to optimize learning [4], [16]. With the increasingly pervasive issue of student retention in computing programs, learning analytics offers a promising solution [2], [3] given its capabilities to capture programming progress with a fine level of granularity.

One of the main—and arguably one of the most important—applications of learning analytics is enabling the quality improvement of teaching and instructional interventions. By analyzing engagement in programming tasks, educators gain insights into student challenges, enabling more personalized teaching approaches [3], [17]. Examples of interventions include tailored feedback, instructor meetings, additional topic-specific resources, and achievement badges [17], [18]. Continuous monitoring of student progress through learning analytics fosters timely feedback and adaptive teaching, which enhances student motivation, reduces anxiety, and improves performance outcomes [4], [5], [16]. Another application of learning analytics is predictive modelling, [19] a technique used for the early flagging of at-risk students and personalised learning technologies [20], [21].

To analyze student learning data in a way that allows meaningful extraction of learning behavior patterns, key metrics must first be defined to guide data collection. Cardell-Oliver [22] suggests five software metrics for both diagnostic and formative assessment: program size, functional correctness, execution time, number of program style violations, and client validation using acceptance testing. Other potential metrics include the number of artifacts produced, time spent on a task, lines of code written, number of errors obtained, number of methods written, average method length, and external resource usage [2]. These metrics can then be mapped diagnostically to different problem solving approaches, assisting instructors in adapting course delivery or design learning interventions better catered to student needs [22].

B. ETHICAL CONSIDERATIONS IN LEARNING ANALYTICS

The rapid growth of learning analytics has raised increasing concerns about the ethical use of student data [23], [24], [25]. These concerns have led to the development of guidelines for ethical considerations in this field. As the ethics within learning analytics has evolved, the focus has shifted from recognizing its importance to pinpointing which principles to apply [24], [26]. A further challenge also exists where

economic pressures motivate institutes to make ethical compromises due to the high value of learning data [24].

A key guideline is to avoid misinterpretation of student data to prevent faulty diagnoses or behaviours that obstruct learning [23]. This underscores the need to recognize the limitations of systematic and temporal models in learning analytics and avoid mistreating students based on past algorithmic categorizations [24]. Privacy concerns regarding the use of student data in research have also been highlighted [27], with emphasis placed on obtaining proper consent before data collection and research [25], [28]. Additionally, the importance of data *deidentification* - retaining unique identifiers whilst removing personal identities - has been underscored [23].

Discussion on these principles emphasize that student data should primarily benefit students, avoid reductionist and stereotyped interpretations, and ensure predictive models are unbiased [29]. Transparency in data collection, regular algorithm reviews, and providing students with an option to opt-out are also vital [30]. Additionally, the handling of outliers, protection of student progress during data collection, and limiting data use to its intended purpose are central to ethical considerations [24].

C. VERSION CONTROL SYSTEMS

Version control systems (VCS) are essential in the software industry [13], [31]. Its importance has led educators to incorporate it into student assessments [32], and some have enhanced its appeal through gamification [33]. The availability of cloud-based Git services with educational support makes its integration into computing education straightforward [34], and both educators and students have responded positively to its use in the classroom [13], [35].

VCS offer several advantages in computing education. For instructors, they simplify task distributions, code reviews, and assessment management [13]. Students benefit directly from enhanced collaboration in team assignments [31] and secure code storage. Indirect benefits include the acquisition of in-demand VCS skills [31], [36] and exposure to industry-standard tools and best practices [13]. Students demonstrate notable improvements in their overall learning and appreciation of teamwork and collaboration when using VCS [37].

A valuable application of VCS in computing education is predicting student performance. Strong correlations exist between student interaction levels with VCS and their resulting academic grades [36], [38], and hence accurate prediction models can be built using VCS activity [12]. Data variables that could be collected as inputs for the prediction model include total number of commits, number of days with at least one commit made, average quantity of commits per day, number of line additions, number of line deletions, total number of issues opened, and total number of issues closed [36]. Guerrero-Higueras et al. [36] found that the number of days where at least one commit is made had a greater predictive importance than the total number of commits, and the number of issues opened or closed during the assignment had very low importance.

D. MINING SOFTWARE REPOSITORIES

The field of mining software repositories (MSR) is vast in research. MSR research entails the creation of techniques, known as mining techniques, to extract useful information from software repositories [15]. Applications of this research include predicting the occurrence of bugs, understanding the evolution of a software project, and providing insight into team dynamics [15], [39], [40], [41].

One common area of MSR research is the mining of Git logs in a software repository [14], [39], [41], [42]. A common mined artifact is the number of commits made over time as a metric for commit consistency. Some systems extend this by correlating this behaviour against assignment milestones. One study has extended this research by analysing the size and types of commits made over the lifecycle of a software project [43]. Others have delved deeper into the owner of commits to demonstrate and visualise the collaboration of team members and project team dynamic [15], [41].

Some research has taken a different angle by looking at the quality of software across versions. Metrics utilised include the presence of code smells as well as heuristics such as cyclometric complexity [14], [44]. In this way, one can ascertain the quality of a codebase across time, often with module granularity.

E. RELATED WORK

Mangaroska et al. [45] conducted a study comparing the performance of multimodal learning analytics (MMLA) with solely IDE-based learning analytics, where MMLA is a collection of techniques that utilise various data sources to analyse learning within various, naturalistic learning settings [46]. Example data sources for MMLA include data collected from video, audio, text, gestures, and biosensors [45]. Their research aimed to explore the potential of MMLA to extract data in a way that effectively measures key constructs influencing learning. The emphasis is on constructs that cannot be easily measured purely using programming process data, such as IDE-based logs. Examples of such constructs include cognitive load, frustration, confusion, and selective attention [45].

The experiment conducted for this study involved the collection of eye-tracking data, physiological data using a wristband sensor, facial expressions using video data, and IDE log data. Data was captured while Computer Science majors completed a series of debugging activities and analyzed using eight predictive models developed for the study. The first model included only IDE-log data, and the following seven models included both IDE-log data and at least one other data stream. It was found that the first model was notably outperformed by the other seven models, which forms strong evidence for the value of MMLA in capturing

and investigating learner behaviour. Additionally, the highest performing model was defined with a combination of eye-tracking, facial, and physiological data on top of IDE-log data; this is an interesting result that could guide the effective implementation of MMLA in computing education. Thus, it can be concluded from this study that IDE-log data may not be sufficient in comprehensively capturing computing student behaviours, and exploring different data streams offers promising potential for learning analytics in computing education.

A different predictive model built by Guerrero-Higueras et al. [36] used data collected from student interactions with version control systems (VCS). Their study found a strong relationship between VCS activity and academic performance as the highest performing model in the experiment had a reasonably high accuracy score of 0.78 for predicting students' academic performance [36]. Git4School is a web application allowing teachers to visualise student progress by analysing data from their Git repositories. It provides dashboards with graphs that help instructors identify struggling students and seeing how students are progressing through their activities [47]. Pons et al. investigated what indicators about student behaviour could be extracted from Git repositories and acknowledge the challenges of analysing Git data [48].

F. SUMMARY

The central aim of our research is to investigate how version control systems can be used to identify and analyse patterns of student learning behaviour in computing education. Existing research gives evidence to the compelling benefits of harnessing learning analytics, including the improvement of teaching quality and consequently student performance, attitudes, and retention in computing degrees. There currently exist a range of different means to leverage learning analytics in computing education, such as IDE-based data, multimodal data collection, or the measurement of software metrics on final submissions. However, methods for continuously capturing data to maximize the benefits of learning analytics—such as the early flagging of at-risk students—remains largely unexplored. With the research evidence that points toward a strong relationship between student performance and metrics collected from version control system usage, this area has a strong potential to fill the aforementioned gap.

This study explores the following research questions:

RQ1: What metrics of student learning can be identified from version control in software submissions?

RQ2: How can metrics of student learning indicate student uptake with course content?

III. DETECTING LEARNING BEHAVIOUR IN SOFTWARE REPOSITORIES VIA GIT ANALYSIS

A. GIT MINING AND ANALYSIS

In order to actualise the potential of harnessing version control systems (VCS) for learning analytics, this study focuses on Git, owing to its widespread adoption in the computing education landscape.

1) MINING AND PROCESSING GIT METRICS

Git tracks changes in software repositories via snapshots known as commits, which record the differences between file versions as code additions and deletions. Commits also contain metadata such as the author, timestamp, and commit message. Thus, the rudimentary data encapsulated within each commit of a student's repository for a given assignment offers a high-level view into the student's progression. Table 1 summarises these metrics and their extraction method in the rudimentary category.

The repository snapshot at each commit can also be examined, providing insight into the codebase evolution across all commits. In particular, inspecting the abstract syntax tree (AST) representation of code at each commit provides insight into file structure. The AST analysis also facilitates identification of the code entities making up the project - such as classes, methods, and fields - and how the project was modified across the commits for the different levels of granularity. This study's approach to quantifying this modification is discussed in Appendix VI. Compiling the code at each snapshot and executing any number of provided tests also forms a key factor in assessing the progression of correctness and code quality in a student's codebase. Table 1 summarises these metrics and their extraction methods in the AST snapshot and state snapshot categories, respectively.

Furthermore, amalgamating the data across all commits yields key metrics for each repository as a whole, such as the total number of commits or the total number of additions made. Fitting a distribution to the per-commit ''snapshot'' metrics and calculating the standard deviation also offers insights into the variability of metrics within each repository. These aggregate metrics then enable comparison among students undertaking the same assignment, facilitating the identification of trends and outliers within the cohort. Table 1 summarises these metrics and their extraction method in the aggregate analytics category.

2) METRIC DATA VISUALISATION

To maximise the value of the procured Git metrics in the realm of computing education, the raw data must be transformed into comprehensible insights presented in an intuitive manner. Data persistence and user-friendly navigation through the data should also be offered to enhance the overall utility and accessibility for instructors.

To optimally illustrate the different metric categories delineated in Table 1, the following three visualisation types are proposed:

- **Repository level:** Visualisations here could include dot plots and line graphs to represent each individual commit and its associated metrics.
- **Assignment level:** These visualisations should facilitate the comparison of repository wide metrics across students. A histogram could be used to depict the

TABLE 1. Summary of the metrics obtainable by mining data from each Git commit in a software repository. These metrics are categorised into three high-level extraction methods.

Category	Extraction Method	Metrics
Rudimentary	Git logs can be inspected to extract metadata and diff metrics recorded on each commit.	– Commit author – Commit timestamp (UTC epoch) – Commit message – Commit type (bot commit, first commit, last commit, regular commit) – Number of additions made (no. lines) – Number of deletions made (no. lines)
AST Snapshot	The repository can be checked out at each commit to inspect the abstract syntax tree (AST) representation.	– File structure – Modification score at project, file, class, method, and field granularity
State Snapshot	The repository can be checked out at each commit to attempt compilation of the code and execution of any provided tests.	– Whether the code compiles – Number of test cases passed (if provided)
Aggregate Analytics	The rudimentary and snapshot metrics can be amalgamated and the standard deviation of per-commit metrics can be calculated to identify the respective metric variability.	– Total number of commits – Final number of additions (no. lines) – Final number of deletions (no. lines) – Final modification score – Final number of test cases passed – Timestamp of first commit (UTC epoch) – Timestamp of last commit (UTC epoch) – Timestamp of first user (i.e., non-bot) commit (UTC epoch) – Duration spanning first user commit and last commit (UTC epoch)

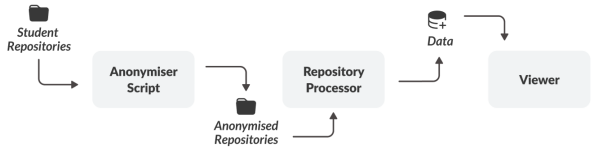


FIGURE 1. Conceptual pipeline diagram. Key modules are shown as shaded rectangles and are expanded upon in the following section. Modules are connected by generic data interchange representations.

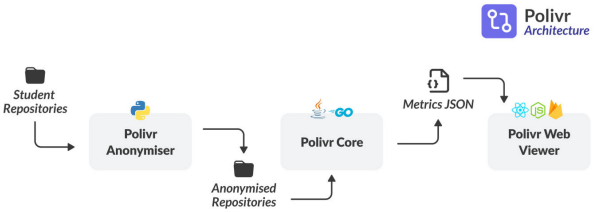


FIGURE 2. The Polivr ecosystem architecture. Key modules are shown as shaded rectangles and are expanded upon in the following text. Modules are connected by data interchange representations.

distribution of these repository wide metrics across students, and a scatter plot could be used to illustrate and identify any relationships between such metrics.

3) CONCEPTUAL APPROACH

To establish the means of mining and processing Git data to ultimately visualising the insights, a conceptual approach is devised as shown in Figure 1.

This approach consists of a pipeline, headed by an anonymiser module which is important for this study as it ensures ethical adherence and data privacy of student repositories. This is followed by the repository processor module, responsible for producing the various Git metrics outlined in Table 1. Finally, we conclude with the viewer component, responsible for transforming the Git metrics into effective visualisations.

B. THE POLIVR ECOSYSTEM

Given the promising prospects of using version control for learning analytics, this study brings forth the key practical contribution of the Polivr ecosystem (Polivr stands for 'progression of learning in versioned repositories'). In this way, Polivr implements the conceptual blueprint outlined in the previous section. This implementation is portrayed in Figure 2. Note that our implementation currently only works in the context of Java repositories.

1) POLIVR ANONYMISER

The anonymiser is a series of Python scripts used to protect the identities of past students via anonymising their identities in Git logs. This way, one can safely use their repositories when performing evaluation without risk of exposing private data. The anonymiser is complex in implementation as it has to cater for anonymising at scale and preserving 'pseudo-identities' for the same students, which may have multiple Git authorships across different repositories and/or assignments. To do this, the anonymiser works in three distinct stages as shown in Figure 3.

In the initial *mapping* stage, the anonymiser reads Git authors from all repositories and aims to associate them

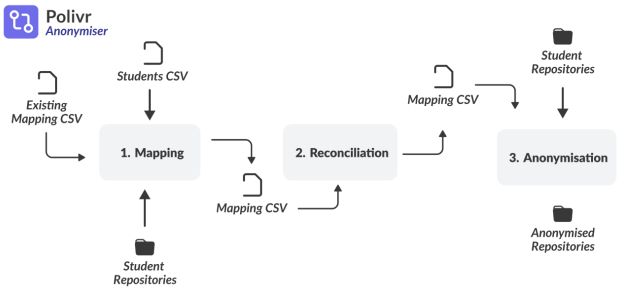


FIGURE 3. Stages of the Polivr anonymiser. Key modules are shown as shaded rectangles, connected by data interchange representations.

with unique identities from the students csv file, aided by other identifiers like UPI, GitHub username, or student ID. Each identified entity is then linked to a unique random pseudonym, with this data saved in an output mapping csv file. An existing mapping csv can be optionally inputted, preserving previous identity-to-pseudonym links, ensuring consistent pseudo-identities across all processed student repositories.

It is common that not all Git authors can be automatically mapped to identities. The *reconcilliation* stage resolves this issue through prompting and establishing manual intervention to map the remaining authors to identities. The csv can then be reprocessed so that pseudonyms are consistent across all Git authors found. Finally, stage three occurs: *anonymisation*. Here, using the complete mapping csv file, all repositories are anonymised. This is done by replacing all existing Git authors with those of their corresponding 'pseudo-identities'.

2) POLIVR CORE

Polivr Core implements the repository processor phases as given in Figure 1 in a three-layer architecture as shown in Figure 4. Each layer abstracts away the complexity of inner implementation from modules in the outer layers.

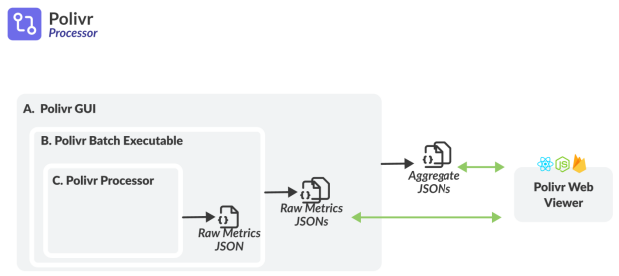


FIGURE 4. Layered architecture of Polivr Core. Each module outputs JSON files which are then uploaded and persisted to Polivr Web Viewer.

a: POLIVR

At the heart of Polivr Core, is the Polivr Processor (shortform Polivr). Polivr is built with Java and is responsible for taking in a single Git repository and producing a series of 'learning behaviour' metrics. These correspond to the first three phases.

- **Git Log Inspection:** JGit is used to process the Git logs of a repository and extract the specified rudimentary metrics.
- **State Traversal:** JGit is again used to check out the codebase at each commit and then run both Maven builds and test cases at each commit check out.
- **AST Traversal:** JavaParser was used to process the Java files and extract AST representations of the various entities. The modification score metrics are then attained as per the discussion in Appendix VI.

b: POLIVR BATCH EXECUTABLE

Alongside Polivr, a batch executable has been created using Golang. The executable is effectively a wrapper and allows one to run Polivr on many repositories both sequentially and in parallel, ultimately producing a collection of data JSONs. Experimental verification on a machine with 20 OS threads showed that parallel processing reduces the average processing time for a single repository from roughly 3 minutes to 1 minute, achieving an approximate 3 times speed-up. This is significant as processing 100 repositories would only take 5 hours as opposed to 15. This speed up is achieved by using Golang's renowned Go coroutines which operates on green threads. The executable also takes on the responsibility of setting various configurations such as memory allocation and batch sizes. It also ensures that processing is performed on a copy of each original repository such that the state of the original repository is maintained even in the event of unexpected termination.

c: POLIVR GUI

The Polivr GUI, constructed using C#.NET6, represents the final abstraction layer. A Blazor MAUI application was chosen for its cross-platform compatibility and robust developer community support. The GUI takes on two core responsibilities. The first is calculating the aggregate metrics from the 'commit-level' metrics ascertained earlier. The second is that the GUI negotiates all the data into a format that can be persisted and ultimately displayed by the web viewer. This involves mapping each repository to the appropriate student for the course, constructing various API requests, and handling client authentication appropriately. The GUI also carries the benefit of ensuring that the local environment is 'fit' to run the inner modules and makes processor configuration simpler.

3) POLIVR WEB VIEWER

After processing the student repositories with the Polivr Core, the outputted metric JSON files can be uploaded to the Polivr Web Viewer. This platform serves as the primary interface for instructors, providing an intuitive presentation of the raw metrics and simplifying the interpretation of complex repository data. Coupled with user-friendly functionality design and efficient data management, the web viewer enables instructors to conveniently draw meaningful inferences about student learning behaviour. Trends and outliers within the cohort are also easily visualised, facilitating

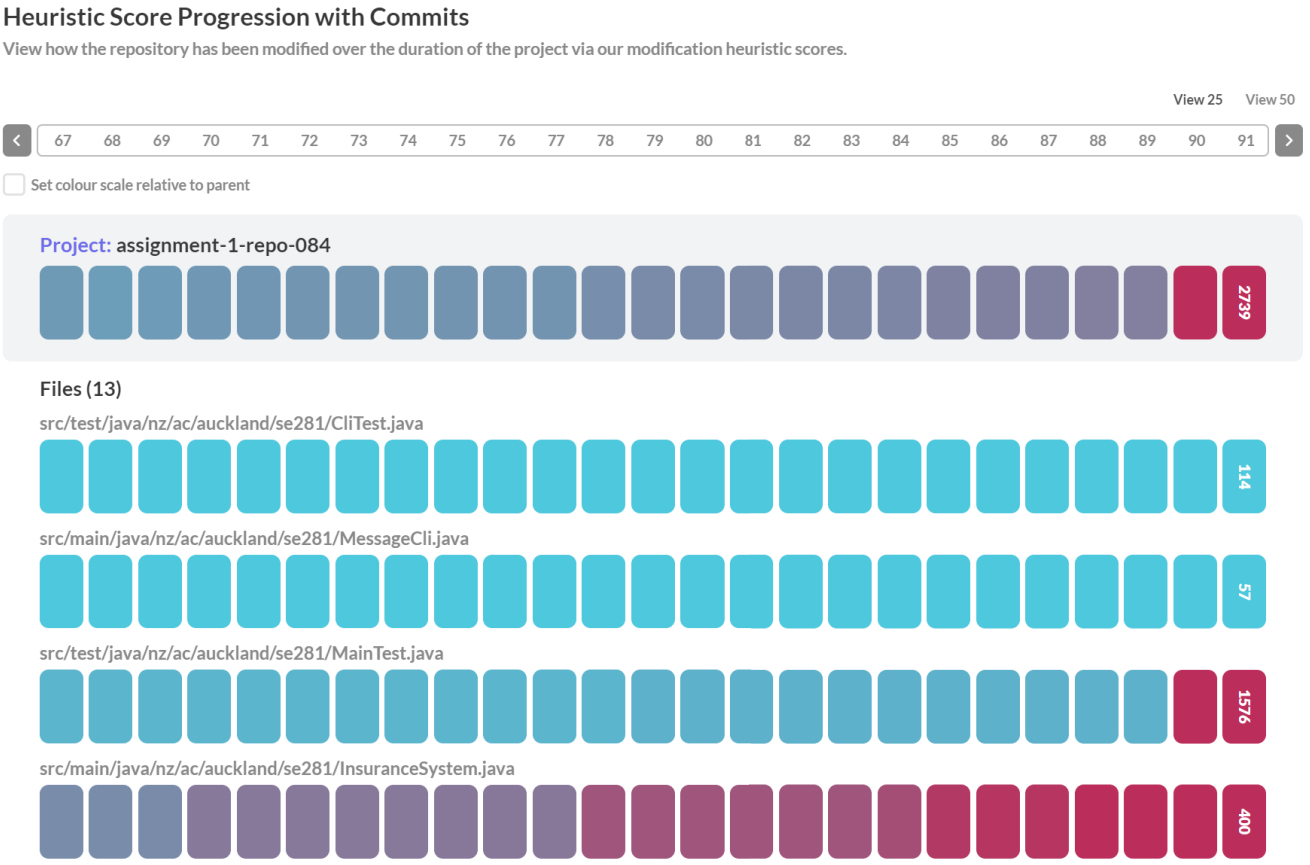


FIGURE 5. **The Polivr Web Viewer modification drilldown view. This visualisation lists each file in the repository (viewed top to bottom) and each commit across the project is represented by a tile (viewed horizontally). The number of modifications within the commit for each respective file is represented by the colour of the commit tile. Commits with larger amounts of modification appear more red and those with less modification appear more blue.**

the identification of students who may need additional attention from instructors.

Data persistence and management on the Polivr Web Viewer is facilitated through a REST API, developed using Express.js, which interfaces with the Firebase database. This API also orchestrates other backend services, including JWT token encryption and dispatching email notifications to users. The frontend user interface, built with React.js, is structured such that student repositories are grouped into assignments, and assignments into courses. This design mimics typical computing education course structures, ensuring intuitive integration and organisation of actual student data. The visualisations offered by the interface are tailored to provide a comprehensive and intuitive representation of the different Git metrics generated by Polivr Core. These visualisations can be grouped into repository, assignment, and student levels as discussed in Section III-A2.

a: REPOSITORY LEVEL

Each repository is assigned a dedicated page featuring three different tabs, with each housing a distinct set of visualisations. The 'Overview' tab displays all commits for the repository, represented both in a dot plot and a line chart format. Users can select one of the seven key commit metrics to dynamically adjust the size of the dots in the dot plot and change the y-axis measurement in the line chart.

The 'Modification Drilldown' tab features a dynamic visualisation of the depicting the composition and evolution of code entities within the project, illustrated in Figure 5. Each entity's modification over successive commits is quantified through a modification score, detailed in Appendix VI, and visually represented using a color-coded heatmap. Users can click into any parent entity to view the child entities, facilitating a layered, 'drilldown' exploration of the repository's structural transformations.

The 'Timeline' tab, illustrated in Figure 6 provides a granular view of the project's development over time, allowing users to visualise the codebase's evolution in detail. Users can navigate through the repository's commits in a sequential manner, utilising either the manual skips or the auto-play feature. The repository's file directory tree is displayed, enabling users to see the high-level file structure evolve over the commits. Additionally, newly added or modified files in each commit are visually highlighted. Users can also select individual files to view a simplified representation of the code. This representation includes the class, field,

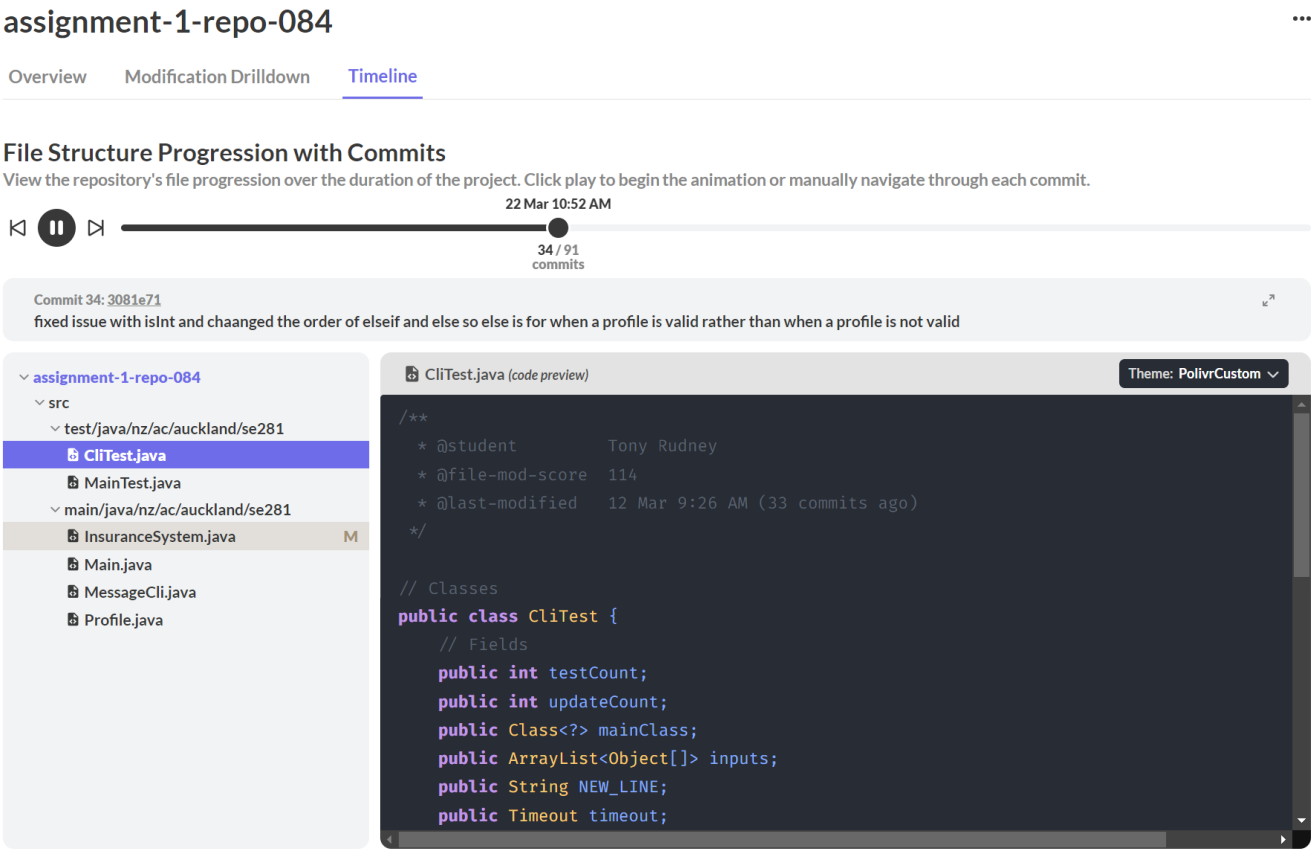


FIGURE 6. The Polivr Web Viewer timeline view. This visualisation showcases the file structure for a given commit in the left panel. Clicking on each file displays the 'skeleton' of selected code, displaying key classes, methods, and members.

and method names, accompanied by line counts for each method.

b: ASSIGNMENT LEVEL

At the assignment level, an overview of all repositories under the assignment is displayed using two charts, illustrated in Figure 7. A donut chart portrays the distribution of repository progress statuses as a proportion of the total student count. Alongside this, a line graph charts student engagement levels, quantified by the daily commit frequency throughout the assignment's duration.

To analyse the aggregated repository-wide metrics, both a histogram and a scatter plot are employed to facilitate comparison across the student cohort. Similar to the repository overview, users can choose an aggregate repository-wide metric for the histogram's x-axis and select two metrics for comparison on the scatter plot.

The histogram view offers insight into the distribution of the chosen metric, emphasising outlier repositories. Adjacent to the histogram, a detailed view lists repositories with the highest and lowest values for that metric. The scatter plot view illustrates the relationship between two selected metrics, enabling users to discern correlations effectively and efficiently. Trends are vividly presented, and consequently, outlier repositories are easily identifiable. The detailed view beside the scatter plot highlights repositories identified as outliers based on their high Mahalanobis distances [49].

The user can also select a histogram bin to view the list of repositories within, and selecting a repository either from the detailed view or directly on the scatter plot reveals a comprehensive breakdown of its metric values. Each value is placed contextually within the box plot distribution of the respective metric across the entire student cohort, offering comparative insights anchored in broader trends.

IV. EVALUATION

A. COURSE CONTEXT

In this study, the course being analysed is SOFTENG281, a second-year, 12-week "CS2-like" course that uses Java as the programming language. The first six weeks teach object-oriented programming (OOP) concepts, while the final six weeks focus on design patterns, data structures, and simple algorithms in the final six weeks. The course had an enrollment of 309 students and is a mandatory requirement for three engineering specialisations: computer systems engineering, electrical and electronic engineering, and software engineering. Before taking this course, students completed a first-year

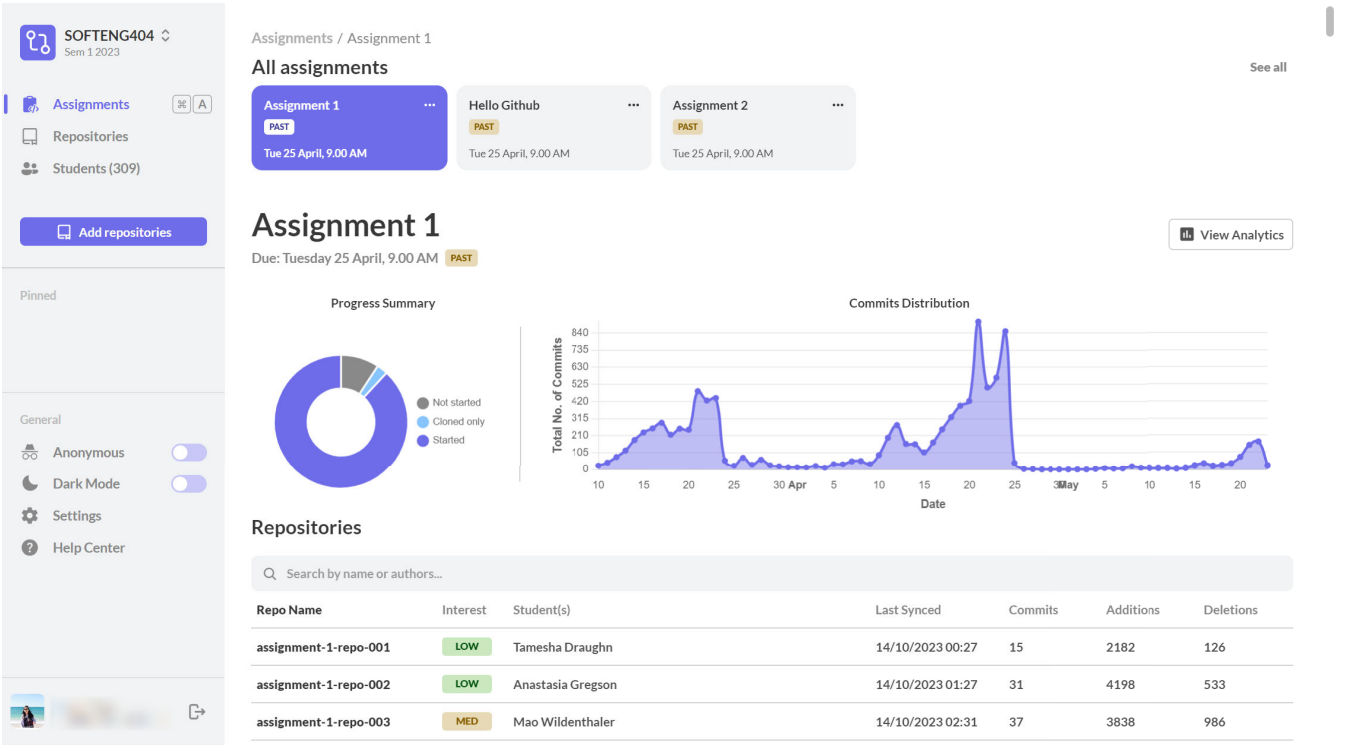


FIGURE 7. The Polivr Web Viewer assignment overview. This visualisation displays an overview of the selected assignment, notably a summary of the progress of the cohort in the form of a pie chart and the aggregate number of commits for this assignment across time. The table at the bottom of the screen lists all the repositories for the selected assignment.

CS1 course, which covered programming fundamentals using Matlab and C, as part of their engineering curriculum.

The assessment structure of SOFTENG281 includes three major take-home assignments worth 20% each, an invigilated practical test worth 30%, and some small exercises totalling 10%. This evaluation specifically focuses on the first assignment (Assignment 1), which assesses students' ability to apply the OOP concepts covered in the initial six weeks of the course. The assignments used Maven for build management, GitHub Classroom was used to distribute the assignment starter code to students. Assignment 1 aimed to test students' understanding and application of OOP concepts, such as encapsulation, inheritance, and polymorphism. Students were provided with half of the JUnit tests to help guide some of their development, while the other half were hidden to encourage students to write their own test cases.

B. METHODOLOGY

The student repositories were anonymised using a script that removed authorship details. Following anonymisation, Polivr Core processed the repositories using one of the dedicated lab machines, utilising its computing power of 20 OS threads. After processing, the output metrics were persisted to the web client hosted on Heroku. The subsequent findings and insights are centred around the graphs derived specifically from the SOFTENG281 2023 cohort case study. Section III provides a more in-depth understanding of how Polivr was used to process the student repositories.

C. REPOSITORY LEVEL

To evaluate per-commit metrics at the repository level, Repository 51 by psuedo-student Sandie Seaforth and Repository 158 by pseudo-student Brian Pouk were selected for further analysis given their distinct characteristics identified through manual inspection. The defining features of these repositories are observed through the differing progression journeys of the number of additions and deletions as well as the number of test cases passed across all commits.

1) ADDITIONS AND DELETIONS

Figure 8 illustrates a generally steady increase in the number of additions and deletions for Repository 158. This suggests a gradual progression through the assignment, with only small to moderate changes being performed within each commit. This is contrasted with Figure 9, which depicts the progression of additions and deletions for Repository 51. Here, a significant jump in the number of additions is exhibited at commits 11 and 13, and a significant jump in the number of deletions is exhibited at commit 13. This increase is also inconsistent with the general trend of very minimal additions and deletions for all other commits in Repository 51.

2) TEST CASES

Figure 10 captures Repository 158's consistent and gradual progression towards passing all test cases, both provided and hidden. This is contrasted with Figure 11, which illustrates Repository 51's more erratic progression to passing all test

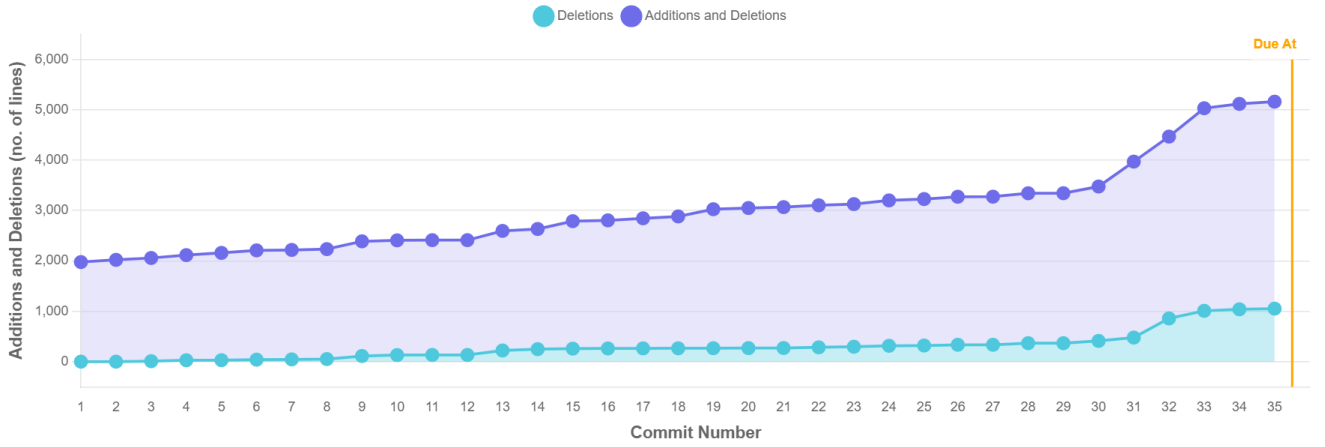


FIGURE 8. Line chart of the cumulative number of additions and deletions for each commit in Repository 158. The cumulative sum increases steadily overall, with slightly larger increases observed around commits 30 to 32. This suggests a gradual progression through the assignment, with only small to moderate changes being performed within each commit.

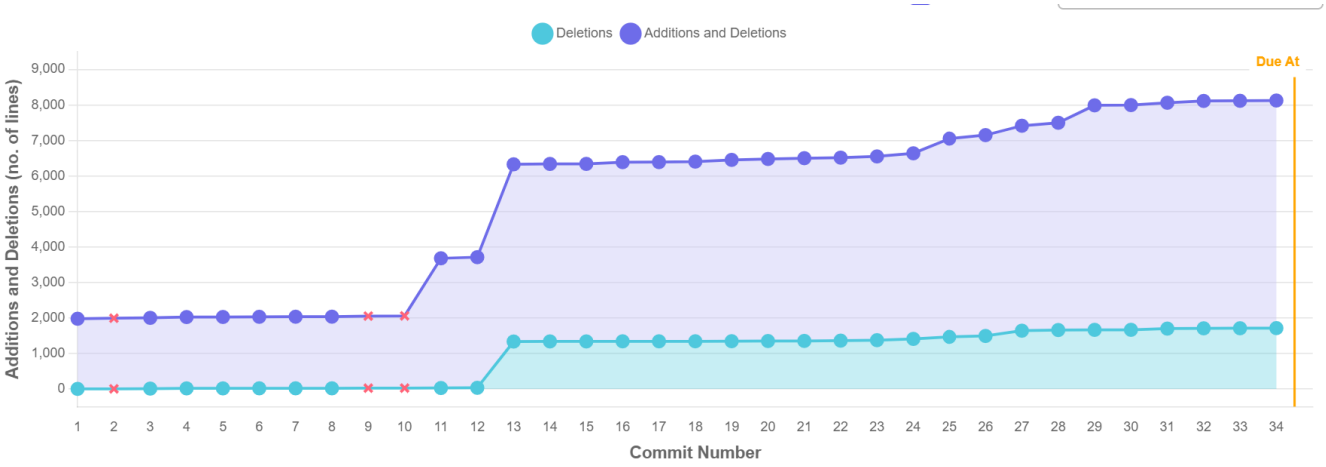


FIGURE 9. Line chart of the cumulative number of additions and deletions for each commit in Repository 51. A sharp jump in additions and deletions is observed in commits 11 and 13. This increase is inconsistent with the general trend of very minimal additions and deletions for all other commits.

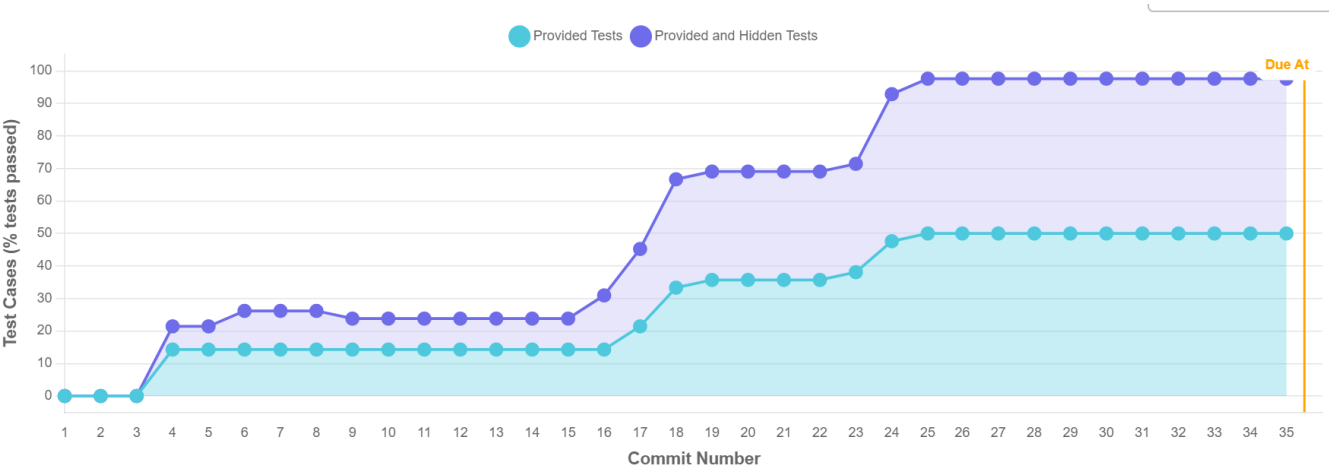


FIGURE 10. Line chart of the cumulative number of provided and hidden test cases passed for each commit in Repository 158. The student of this repository seems to make significant progress in test cases in commits 18 and 24. This suggests a consistent and gradual progression towards passing all test cases.

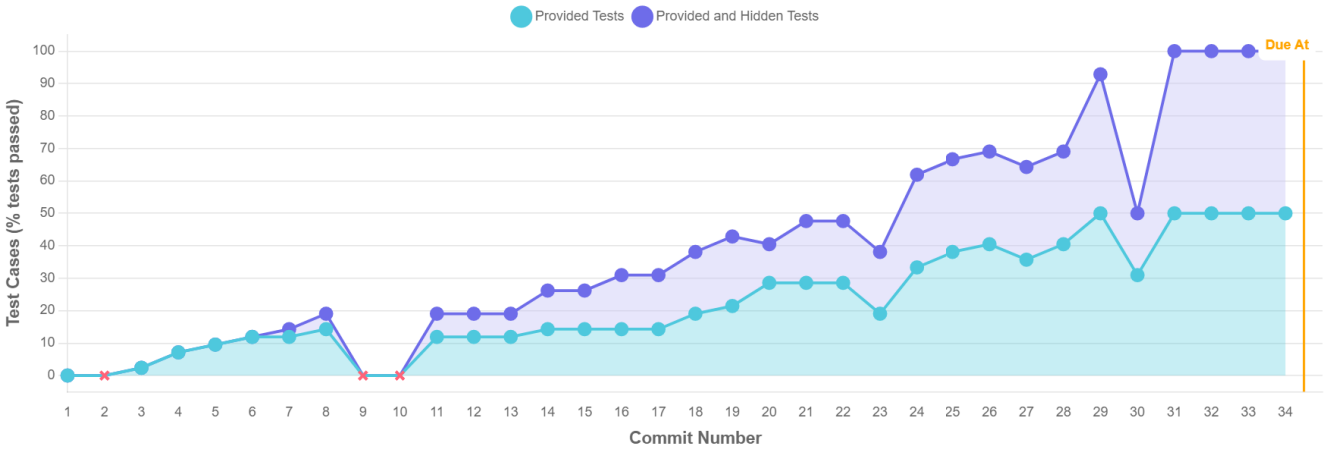


FIGURE 11. Line chart of the cumulative number of provided and hidden test cases passed for each commit in Repository 51. The student of this repository makes a few commits (9, 23, and 30) where the number of test cases dips in comparison to the previous commit. This suggests a more erratic progression to passing all test cases.

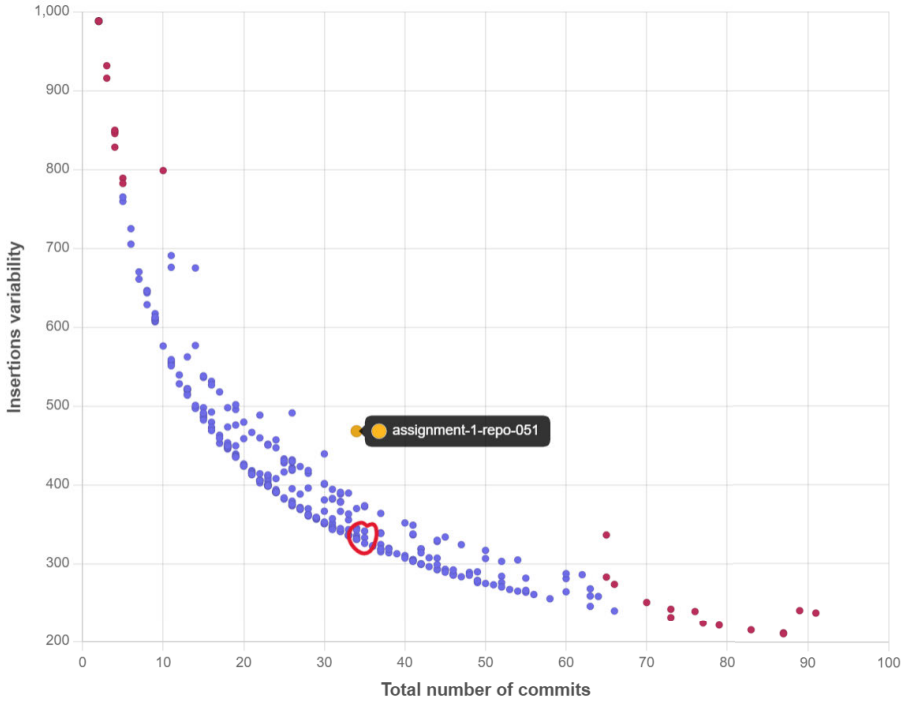


FIGURE 12. Scatter plot of the total number of commits against the insertions variability. The yellow dot highlights an outlier (Repository 51) that seems to have higher insertion variability than other repositories with a similar number of commits. This suggests that repositories with fewer commits often exhibit more significant variations in their insertion counts per commit.

cases, marked by sharp and irregular dips in the percentage of passed test cases across various commits.

D. ASSIGNMENT LEVEL

For assignment level evaluation, repository-wide metrics were compared across all repositories within Assignment 1 of the SOFTENG281 course. Additionally, each metric was plotted against other metrics using a scatter plot to identify any correlations between metrics and visualise any present trends in the data. A selection of outlier repositories were also highlighted on the plot, determined based on their high Mahalanobis distances, a metric used for determining the distance between a data point and the distribution [49]. After a rudimentary inspection of the aggregate metric graphs, the following trends emerged as being of notable interest.

1) TREND A: TOTAL NUMBER OF COMMITS AGAINST INSERTIONS VARIABILITY

The insertions variability metric gauges the fluctuation in the number of insertions for each commit, quantified using the

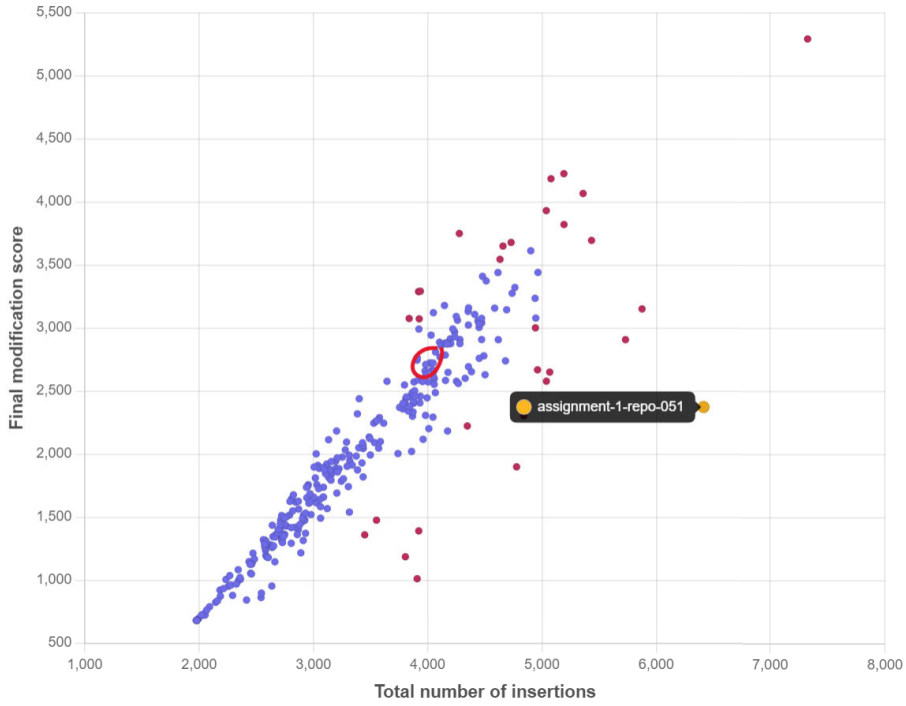


FIGURE 13. Scatter plot of the total number of insertions against the final modification score. The yellow dot highlights an outlier (Repository 51) that seems to have a higher number of insertions than other repositories with the same modification score. This suggests that repositories with more insertions often exhibit greater modification activity.

standard deviation. The plot in Figure 12 suggests a strongly negative, non-linear correlation between the total number of commits and the insertions variability of the repositories in Assignment 1. In essence, repositories with fewer commits often exhibit more significant variations in their insertion counts per commit. This relationship also appears to be more pronounced at lower commit numbers, indicated by the steeper slope, and gradually levels off as the total commits increase.

2) TREND B: TOTAL NUMBER OF INSERTIONS AGAINST FINAL MODIFICATION SCORE

The plot in Figure 13 illustrates the strongly positive, linear correlation between the total number of insertions and the final modification score. This trend suggests that repositories with more insertions often also exhibit greater modification activity, as indicated by a high modification score. A few outlier points on the plot, particularly in the upper-right region, represent repositories with disproportionately high modification scores or insertions, diverging from the general trend exhibited by most repositories.

3) TREND C: TOTAL NUMBER OF COMMITS AGAINST COMMIT TIME VARIABILITY

The commit time variability metric gauges the fluctuation in the duration of the interval between adjacent commits. The plot in figure 14 highlights a weak negative, non-linear correlation between the total number of commits and the commit time variability of the repositories. In essence, repositories with more commits often have a lower fluctuation in commit time intervals. This relationship also appears to be more pronounced at lower commit numbers, indicated by the steeper slope, and gradually levels off as the total number of commits increase.

4) TREND D: TOTAL NUMBER OF TEST CASES PASSED

The histogram in figure 15 illustrates a strongly left-skewed distribution, suggesting that most students manage to pass a high number of test cases. Additionally, the prominent spike at the highest histogram bin indicate that a significant portion of repositories passed all, or nearly all, test cases. The lower end of the histogram depicts a small number of repositories failing all test cases—these repositories often did not compile at the final commit.

V. DISCUSSION

The trends and patterns observed in the metric graphs can be interpreted with respect to the wider computing education course, facilitating more meaningful inferences about student learning behaviours. The trends identified from the assignment level evaluation in Section IV-D are discussed below. This is followed by a holistic discussion of both repository and assignment level evaluations, in light of the research objectives and the existing literature.

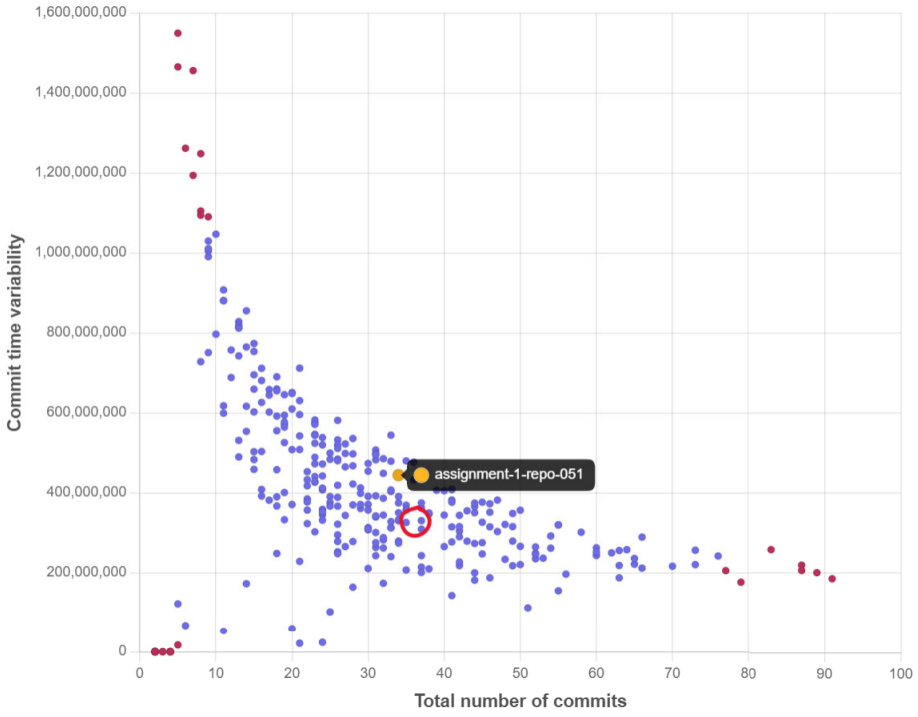


FIGURE 14. Scatter plot of the total number of commits against the commit time variability. The yellow dot highlights a data point (Repository 51) that seems to be in line with the general correlation observed. This suggests that repositories with more commits often have a lower fluctuation in commit time intervals.

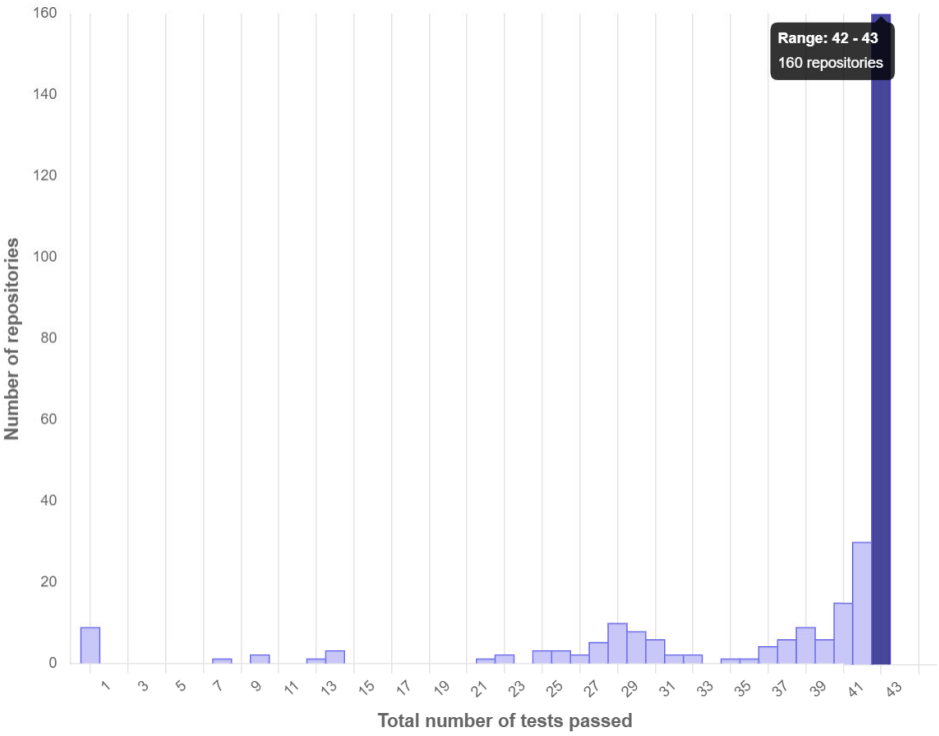


FIGURE 15. Histogram of the total number of test cases passed. We see the vast majority of repositories lie in the range 42 to 43. Given there were 42 test cases for this assignment, this indicates that most students passed all test cases for this assignment. This suggests that the test cases were not sufficiently challenging to differentiate between students.

A. TREND INSIGHTS

- **Trend A:** The insertions variability of a repository can be interpreted as a measure of the student's consistency with their commit behaviour, characterised by the size of the commit. For example, a repository with many small commits—that is the number of insertions for that commit is low—and a few large commits would result in a high insertions variability for that repository. Thus, Trend A makes sense intuitively, as a lower commit count offers less opportunity for insertions to be consistently distributed across the commits. The outliers highlighted in the top-left and central regions of the plot indicate repositories with higher insertions variabilities than most repositories, with respect to the number of commits. This suggests that these repositories contain larger changes grouped into less commits by the student.
- **Trend B:** The modification score is a quantified measure of how extensively the codebase has been altered, surpassing the simplistic insertions and deletions metrics by considering only credible changes, as explained in the Appendix. For instance, a commit solely involving comments and formatting alterations would have numerous insertions but a modification score of zero. Thus, repositories exhibiting high insertion counts yet low modification scores indicate that the student made numerous changes that did not enhance the project's functionality. This behaviour could be loosely inferred as suspicious, as frequent non-credible changes are unlikely to serve a genuine purpose. Repositories in this category are easily identifiable as the outliers below the positive correlation line. As Trend B emphasizes that the majority of repositories follow the positive correlation, it suggests that most students make code changes that predominantly have an effect on the codebase functionality.
- **Trend C:** The commit time variability of a repository can be interpreted as a measure of the student's consistency with their commit behaviour, characterised by the time intervals between each commit. For instance, a student who regularly makes commits every day would exhibit low commit time variability in their repository. Conversely, a student committing infrequently at the beginning of an assignment but very frequently toward the end would have a high commit time variability for their repository. Thus, it makes logical sense that Trend C displays a negative correlation, as a higher number of commits tend to result in more evenly spaced commit time intervals, thereby reducing commit time variability. Outlier repositories with a disproportionately high commit time variability but a low number of commits may be of concern, as they may have crammed significantly large chunks of work at one time. This behaviour could, for example, be loosely inferred as either suspicious conduct or indicative of a student struggling to complete their required tasks within the allotted time frame.
- **Trend D:** This trend illustrates a phenomenon commonly referred to as the 'ceiling effect', which occurs when a large proportion of a participant cohort achieves the highest value for some measured variable [50]. This phenomenon poses challenges, particularly in research, as it prevents drawing conclusions about the impact of an intervention on students' learning outcomes when they have already achieved the highest evaluation metric. In the context of the SOFTENG281 case study, Trend D underscores the difficulty of solely relying on test cases to evaluate student learning progress. Since tests are frequently used as the primary means of assessing student learning in assignments, this insight highlights the need to explore more comprehensive metrics for effectively assessing and understanding student learning.

B. RESEARCH IMPLICATIONS

> **RQ1:** What metrics of student learning can be identified from version control in software submissions?

A large suite of metrics have been identified, along with the different methodologies to extract these metrics from version control in software submissions. Section III-A1 outlines these metrics in further depth. Furthermore, Polivr Core successfully procured these metrics, followed by their visualisation on the Polivr Web Viewer.

> **RQ2:** How can metrics of student learning indicate student uptake with course content?

Insights from the Git learning metrics were captured as trends in the data distribution and correlations between two metrics, as discussed in Section V-A. Although student uptake of course content cannot be confidently indicated using the current metrics, these metrics still illustrate valuable insights into general behavioural patterns. Additionally, students of concern can be easily identified and further intervention to gauge uptake could be conducted.

C. PRACTICAL IMPLICATIONS

This research carries substantial practical implications. Polivr presents a novel tool for computing education instructors to assess their students' progress in Git-based assignments. With its user-friendly and intuitive interface, it not only facilitates easy access but also delivers deep insights. This equips instructors with the necessary data to refine course delivery and provide tailored learning interventions where required. Utilising Polivr while an assignment is still in progress also enables early detection of students who might be struggling, characterised by outlier repositories or abnormal metric patterns. With plans to release Polivr to the public soon, there's an ambition to reshape learning analytics for computing educators. Polivr stands poised to elevate the teaching and learning experience in computing and pave the way for more research in this domain.

D. THREATS TO VALIDITY

There are a number of factors that may compromise the validity of this study, which are important to acknowledge when drawing insights and conclusions from the findings.

- **Sampling Bias:** Our study was based solely on data from the SOFTENG281 2023 cohort, which limits the breadth of our insights and as such, the procured insights may not be necessarily generalisable to other computing education courses and student cohorts. Thus, we intend to include data from various courses and cohorts to enhance the general applicability of our findings. Moreover, as our dataset exclusively features Java repositories, further investigations should explore whether the identified patterns remain consistent across Git repositories developed in other programming languages.
- **Cloud Hosting Constraints:** The constraints of cloud hosting, such as potential downtimes or latencies, may introduce inaccuracies during data extraction for the visualisations. Additionally, the 1 MB document size limit set by our cloud storage provider, Firebase, impacts the size of repositories that can be processed. This limitation could potentially hinder the extrapolation of our findings to substantially larger projects.
- **Individual Assignments:** The assignments used in our study were solely individual assignments and did not include any group assignments. Thus, insights and trends identified may not be fully applicable to computing education courses with group assignments.

VI. CONCLUSION

Understanding student learning behaviours in the field of computing education is often a complex and challenging task, which motivates the search for novel ways of capturing learning analytics. A promising research opportunity has been identified whereby we aim to investigate the prospect of mining Git-based student repositories for 'learning metrics' with the hope of utilising such metrics to better understand student learning. To address this, we have developed the Polivr ecosystem, which is designed to extract and process a suite of learning metrics, transforming them into informative visualisations that offer valuable insights into student learning. Our evaluation of these insights revealed many interesting trends in student behaviour. This reaffirms the potential Polivr has to revolutionise the way educators perceive and harness student data, ultimately enhancing the educational experience and outcomes.

Future work for this research includes expanding the applicability of our insights by analysing more extensive historical data. We intend to broaden our scope by assessing group assignments and assignments written in different programming languages. The use of the metrics and visualisations generated by Polivr to detect academic misconduct behaviour is another intriguing area to be explored.

APPENDIX
POLIVR MODIFICATION SCORE CALCULATION

The modification score aims to be a better metric of repository progression that is tied more closely with 'credible' changes in the code with every commit. In this way, it supersedes looking at the total number of additions and deletions in a commit because it effectively sifts through these line changes and only counts credible changes to lines. The general idea here is that as entities are more and more modified, this is reflected in the score correlated to this entity.

The score is derived via a series of transitions functions, w, x, y, z. It is simpler to undertake this from a bottom up approach whereby we work from the entities with finer granularity up to a notion of score across the project.

For methods, their scores can be calculated as follows. A similar case can be made for fields. We see the score simply is the previous score plus some contribution calculated by the function δ for going from $\mathbb{M}^{i-1}_{j,k,l}$ to $\mathbb{M}^{i}_{j,k,l}$

$$\begin{aligned} ||\mathbb{M}^{i}_{j,k,l}|| &= z(||\mathbb{M}^{i-1}_{j,k,l}||, \mathbb{M}^{i}_{j,k,l}, \mathbb{M}^{i-1}_{j,k,l}) \\ &= ||\mathbb{M}^{i-1}_{j,k,l}|| + \delta(\mathbb{M}^{i}_{j,k,l}, \mathbb{M}^{i-1}_{j,k,l}) \end{aligned} \tag{1}$$

$\delta(\mathbb{M}^{i}_{j,k,l}, \mathbb{M}^{i-1}_{j,k,l})$ is the number of 'credibly' changed lines for that given method. We define a 'credible' line change as a line addition, deletion, or modification where a change has occurred, excluding annotations, package imports, whitespace changes and alterations due to commenting or formatting. Doing so would render changes due to linters as practically invisible.

If the method did not exist in the previous commit, then $||\mathbb{M}^{i-1}_{j,k,l}|| = 0$ and δ collapses to simply counting the number of credible lines in the newly added method. This reasoning is consistent across all transition functions.

Rising up a level, for classes, their scores can be calculated as given in Equation 2. The δ function encapsulates 'credible' changes to a class across two commits. For all substructures (field or method) $\mathbb{J}^{i}_{j,k,l}$ in the class $\mathbb{C}^{i}_{j,k}$, we define $\delta = \sum_l ||\mathbb{J}^{i}_{j,k,l}|| + \epsilon$ where such constituent scores $||\mathbb{J}^{i}_{j,k,l}||$ are defined above. For substructure deletions between commits $i-1$ and i, the sum of the scores of the removed substructures at the previous commit are issued to ϵ (colloquially, `/dev/null`) and, thus, changes due to deletion of substructures are still preserved in indicating modification.

Of note, method and field renaming is handled appropriately whereby scores continue to propagate if renames take place. We classify a rename where the contents of the entity has a similarity of above 50%. 50% was chosen for consistency with Git's internal rename detector [51]. Similarity comparison was performed using Greedy String Tiling due to the promise it has shown in code similarity analysis [52], [53].

$$||\mathbb{C}^{i}_{j,k}|| = y(||\mathbb{C}^{i-1}_{j,k}||, \mathbb{C}^{i}_{j,k}, \mathbb{C}^{i-1}_{j,k}) = ||\mathbb{C}^{i-1}_{j,k}|| + \delta(\mathbb{C}^{i}_{j,k}, \mathbb{C}^{i-1}_{j,k}) \tag{2}$$

Rising a further level, for files, their scores can be calculated as given in the transition equation, Equation 3. The δ function encapsulates 'credible' changes to a file across two commits.

For all structures (interfaces, enums, or classes) $\mathbb{J}^i_{j,k}$ in the file $\mathbb{F}^i_j$, we define $\delta = \sum_k ||\mathbb{J}^i_{j,k}|| + \epsilon$ where such constituent scores $||\mathbb{J}^i_{j,k}||$ are defined above. As before, ϵ stores the scores of deleted structures. Again, renames of structures are appropriately handled by a similar analysis as that described in the preceding paragraph.

$$||\mathbb{F}^i_j|| = x(||\mathbb{F}^{i-1}_j||, \mathbb{F}^i_j, \mathbb{F}^{i-1}_j) = ||\mathbb{F}^{i-1}_j|| + \delta(\mathbb{F}^i_j, \mathbb{F}^{i-1}_j) \tag{3}$$

Finally, the scores of a given project state can be calculated by the transition function w as given in Equation 4. For all files, we define $\delta = \sum_j ||\mathbb{F}^i_j|| + \epsilon$ where such constituent scores $||\mathbb{F}^i_j||$ are defined above. As before, ϵ stores the scores of deleted files. The renaming of files is catered for when performing the transition function x through the use of Git's internal rename detector heuristic [51]. This ensures that the scores are appropriately propagated.

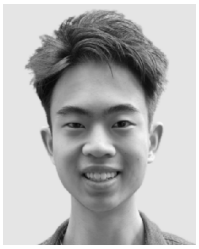
$$||\mathbb{P}^i|| = w(||\mathbb{P}^{i-1}||, \mathbb{P}^i, \mathbb{P}^{i-1}) = ||\mathbb{P}^{i-1}|| + \delta(\mathbb{P}^i, \mathbb{P}^{i-1}) \tag{4}$$

This way, we are able to ascertain the project's modification score over commit time whilst also being able to drill down into exactly what constituent entities changed at each commit.

REFERENCES

[1] E. Roberts, "Computing education and the information technology workforce," *ACM SIGCSE Bull.*, vol. 32, no. 2, pp. 83–90, Jun. 2000.
[2] D. Olivares, "Exploring learning analytics for computing education," in *Proc. 11th Annu. Int. Conf. Int. Comput. Educ. Res.*, Aug. 2015, pp. 271–272.
[3] D. M. Olivares and C. D. Hundhausen, "Supporting learning analytics in computing education," in *Proc. 7th Int. Learn. Anal. Knowl. Conf.*, Mar. 2017, pp. 584–585.
[4] N. Sclater, A. Peasgood, and J. Mullan, *Learning Analytics in Higher Education*. London, U.K.: Jisc, 2016. Accessed: Feb. 8, 2017, p. 176.
[5] Z. G. Baleni, "Online formative assessment in higher education: Its pros and cons," *Electron. J. e-Learn.*, vol. 13, no. 4, pp. 228–236, Apr. 2015.
[6] J. W. Gikandi, D. Morrow, and N. E. Davis, "Online formative assessment in higher education: A review of the literature," *Comput. Educ.*, vol. 57, no. 4, pp. 2333–2351, Dec. 2011.
[7] A. A. Pandit and G. Toksha, "Review of plagiarism detection technique in source code," in *Proc. Int. Conf. Intell. Comput. Smart Commun. (ICSC)*. Cham, Switzerland: Springer, 2020, pp. 393–405.
[8] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 76–85.
[9] M. Farrokhnia, S. K. Banihashem, O. Noroozi, and A. Wals, "A SWOT analysis of ChatGPT: Implications for educational practice and research," *Innov. Educ. Teaching Int.*, vol. 61, no. 3, pp. 460–474, May 2024.
[10] M. Khalil and E. Er, "Will ChatGPT get you caught? Rethinking of plagiarism detection," 2023, *arXiv:2302.04335*.
[11] J. Wang, S. Liu, X. Xie, and Y. Li, "Evaluating AIGC detectors on code content," 2023, *arXiv:2304.05193*.
[12] Á. M. Guerrero-Higueras, V. M. Olivera, G. Esteban, C. Fernández, F. J. Rodríguez-Sedano, and M. Á. Conde, "Model for evaluating Student performance through their interaction with version control systems," in *Proc. Learn. Anal. Summer Inst. Spain*, Jan. 2018, pp. 104–112.
[13] L. Haaranen and T. Lehtinen, "Teaching git on the side: Version control system as a course platform," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Jun. 2015, pp. 87–92.
[14] G. Robles and J. M. Gonzalez-Barahona, "Mining student repositories to gain learning analytics. An experience report," in *Proc. IEEE Global Eng. Educ. Conf. (EDUCON)*, Mar. 2013, pp. 1249–1254.
[15] A. E. Hassan, "The road ahead for mining software repositories," in *Proc. Frontiers Softw. Maintenance*, Sep. 2008, pp. 48–57.
[16] O. Viberg, M. Hatakka, O. Bälter, and A. Mavroudi, "The current landscape of learning analytics in higher education," *Comput. Hum. Behav.*, vol. 89, pp. 98–110, Dec. 2018.
[17] C. D. Hundhausen, D. M. Olivares, and A. S. Carter, "IDE-based learning analytics for computing education: A process model, critical review, and research agenda," *ACM Trans. Comput. Educ.*, vol. 17, no. 3, pp. 1–26, Sep. 2017.
[18] A. J. Stimpson and M. L. Cummings, "Assessing intervention timing in computer-based education using machine learning algorithms," *IEEE Access*, vol. 2, pp. 78–87, 2014.
[19] C. Brooks and C. Thompson, "Predictive modelling in teaching and learning," in *Handbook of Learning Analytics*. Society for Learning Analytics Research, May 2017, pp. 61–68. [Online]. Available: https://www.solaresearch.org
[20] M. Asiah, K. N. Zulkarnaen, D. Safaai, M. Y. N. N. Hafzan, M. M. Saberi, and S. S. Syuhaida, "A review on predictive modeling technique for Student academic performance monitoring," *MATEC Web Conf.*, vol. 255, Jan. 2019, Art. no. 03004.
[21] M. Ekowo and I. Palmer, *Predictive Analytics in Higher Education*. Washington, DC, USA: New America, 2017.
[22] R. Cardell-Oliver, "How can software metrics help novice programmers," in *Proc. 13th Australas. Comput. Educ. Conf.*, vol. 14, Jan. 2011, pp. 55–62.
[23] S. Slade and P. Prinsloo, "Learning analytics: Ethical issues and dilemmas," *Amer. Behav. Sci.*, vol. 57, no. 10, pp. 1510–1529, Oct. 2013.
[24] P. Prinsloo and S. Slade, *Ethics and Learning Analytics: Charting the (Un) Charted*. SoLAR, 2017. [Online]. Available: https://www.solaresearch.org
[25] J. E. Willis, S. Slade, and P. Prinsloo, "Ethical oversight of student data in learning analytics: A typology derived from a cross-continental, cross-institutional perspective," *Educ. Technol. Res. Develop.*, vol. 64, no. 5, pp. 881–901, Oct. 2016.
[26] D. Tzimas and S. Demetriadis, "Ethical issues in learning analytics: A review of the field," *Educ. Technol. Res. Develop.*, vol. 69, no. 2, pp. 1101–1133, Apr. 2021.
[27] J. C. C. Rios, K. Kopec-Harding, S. Eraslan, C. Page, R. Haines, C. Jay, and S. M. Embury, "A methodology for using GitLab for software engineering learning analytics," in *Proc. IEEE/ACM 12th Int. Workshop Cooperat. Hum. Aspects Softw. Eng. (CHASE)*, May 2019, pp. 3–6.
[28] Y.-S. Tsai and D. Gasevic, "Learning analytics in higher education—Challenges and policies: a review of eight learning analytics policies," in *Proc. 7th Int. Learn. Anal. Knowl. Conf.*, 2017, pp. 233–242.
[29] S. Slade and A. Boroowa, *Policy on Ethical Use of Student Data for Learning Analytics*. Open Univ. U.K.: Milton Keynes, 2014.
[30] A. Rubel and K. M. L. Jones, "Student privacy in learning analytics: An information ethics perspective," *Inf. Soc.*, vol. 32, no. 2, pp. 143–159, Mar. 2016.
[31] D. Rocco and W. Lloyd, "Distributed version control in the classroom," in *Proc. 42nd ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2011, pp. 637–642.
[32] R. Glassey, "Adopting Git/GitHub within teaching: A survey of tool support," in *Proc. ACM Conf. Global Comput. Educ.*, May 2019, pp. 143–149.
[33] L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," in *Proc. 2nd Int. Workshop Games Softw. Eng., Realizing User Engagement With Game Eng. Techn. (GAS)*, Jun. 2012, pp. 5–8.
[34] D. M. Case, N. W. Eloe, and J. L. Leopold, "Scaffolding version control into the computer science curriculum," in *Proc. Int. Workshop Distance Educ. Technol. (Conjunct With 22nd Int. Conf. Distrib. Multimedia Syst.)*, Nov. 2016, pp. 175–183.
[35] J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proc. 44th ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2013, pp. 639–644.
[36] Á. M. Guerrero-Higueras, C. F. Llamas, L. S. González, A. G. Fernández, G. E. Costales, and M. Á. C. González, "Academic success assessment through version control systems," *Appl. Sci.*, vol. 10, no. 4, p. 1492, Feb. 2020.
[37] P. Patani, S. Tiwari, and S. S. Rathore, "The impact of GitHub on students' learning and engagement in a software engineering course," *Comput. Appl. Eng. Educ.*, vol. 32, no. 5, pp. 1–23, Sep. 2024, doi: 10.1002/cae.22775.
[38] A. Karakaš and D. Helic, "Relation between student characteristics, git usage and success in programming courses," in *Responsive and Sustainable Educational Futures*, O. Viberg, I. Jivet, P. Muñoz-Merino, M. Perifanou, and T. Papathoma, Eds., Cham, Switzerland: Springer, 2023, pp. 133–148.

[39] F. Z. Sokol, M. F. Aniche, and M. A. Gerosa, "MetricMiner: Supporting researchers in mining software repositories," in *Proc. IEEE 13th Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2013, pp. 142–146.
[40] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *J. Softw. Maintenance Evolution: Res. Pract.*, vol. 19, no. 2, pp. 77–131, Mar. 2007.
[41] M. Mittal and A. Sureka, "Process mining software repositories from student projects in an undergraduate software engineering course," in *Proc. Companion 36th Int. Conf. Softw. Eng.*, May 2014, pp. 344–353.
[42] L. Glassy, "Using version control to observe student software development processes," *J. Comput. Sci. Colleges*, vol. 21, no. 3, pp. 99–106, Feb. 2006.
[43] M. Macak, D. Kruzelova, S. Chren, and B. Buhnova, "Using process mining for git log analysis of projects in a software development course," *Educ. Inf. Technol.*, vol. 26, no. 5, pp. 5939–5969, Sep. 2021.
[44] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining software repositories with iSPAROL and a software evolution ontology," in *Proc. 4th Int. Workshop Mining Softw. Repositories (MSR: ICSE Workshops)*, May 2007, p. 10.
[45] K. Mangaroska, K. Sharma, D. Gašević, and M. Giannakos, "Multimodal learning analytics to inform learning design: Lessons learned from computing education," *J. Learn. Anal.*, vol. 7, no. 3, pp. 79–97, Dec. 2020.
[46] P. Blikstein, "Multimodal learning analytics," in *Proc. 3rd Int. Conf. Learn. Anal. Knowl.*, Apr. 2013, pp. 102–106.
[47] J.-B. Raclet and F. Silvestre, "Git4School: A dashboard for supporting teacher interventions in software engineering courses," in *Addressing Global Challenges and Quality Education*, C. Alario-Hoyos, M. J. Rodríguez-Triana, M. Scheffel, I. Arnedillo-Sánchez, and S. M. Dennerlein, Eds., Cham, Switzerland: Springer, 2020, pp. 392–397.
[48] M. Pons, J.-M. Bruel, J.-B. Raclet, and F. Silvestre, "Finding behavioral indicators from contextualized commits in software engineering courses with process mining," in *Frontiers in Software Engineering Education*, A. Capozucca, S. Ebersold, J.-M. Bruel, and B. Meyer, Eds., Cham, Switzerland: Springer, 2023, pp. 56–68.
[49] G. J. McLachlan, "Mahalanobis distance," *Resonance*, vol. 4, no. 6, pp. 20–26, Jun. 1999.
[50] N. Staus, K. O'Connell, and M. Storksdieck, "Addressing the ceiling effect when assessing STEM out-of-school time experiences," *Frontiers Educ.*, vol. 6, Jul. 2021, Art. no. 690431, doi: 10.3389/feduc.2021.690431.
[51] *Git Diff Documentation*. Accessed: Dec. 11, 2024. [Online]. Available: https://git-scm.com/docs/git-diff
[52] A. B. Kleiman and T. Kowaltowski, "Qualitative analysis and comparison of plagiarism-detection systems in student programs," Sao Poalo, Brazil: Instituto de Computacao Universidade Estadual de Campinas, 2009.
[53] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *J. Univers. Comput. Sci.*, vol. 8, no. 11, p. 1016, Jan. 2002.

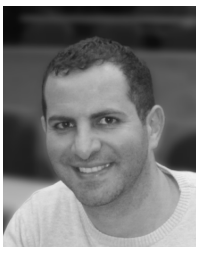


JOHN CHEN received the Bachelor of Engineering degree (Hons.) in software engineering from The University of Auckland, New Zealand. He is currently a Software Engineer with Vivienne Court Trading, Sydney, Australia. His research interests include high performance computing, software architecture and practices, and tooling and methodologies in computing education.

SERENA LAU received the Bachelor of Engineering degree (Hons.) in software engineering from The University of Auckland, New Zealand. She is currently a Software Engineer with Sandfield Associates Ltd., Auckland, New Zealand. Her research interests include human–computer interaction and tooling and methodologies in computing education.

JUHO LEINONEN received the Ph.D. degree in computer science from the University of Helsinki. He is currently an Academy Research Fellow with Aalto University, Finland. His research explores how to best support and engage diverse learner populations using educational technology and artificial intelligence.

VALERIO TERRAGNI received the Ph.D. degree in computer science from The Hong Kong University of Science and Technology. He is currently a Senior Lecturer and the Director of software engineering with The University of Auckland, New Zealand. His current research interests include automated software testing and program analysis.



NASSER GIACAMAN received the Ph.D. degree in engineering from The University of Auckland. He is currently a Senior Lecturer in software engineering with The University of Auckland, New Zealand. His current research interests include educational technologies, computing education, and immersive technologies.