ICLF: An Immersive Code Learning Framework based on Git for Teaching and Evaluating Student Programming Projects

Anonymous Author(s)

Abstract

Programming projects are essential in computer science education for bridging theory with practice and introducing students to tools like Git, IDEs, and debuggers. However, designing and evaluating these projects—especially in Massive Open Online Courses (MOOCs)—can be challenging. We propose the Immersive Code Learning Framework (ICLF), a scalable Git-based organizational pipeline for managing and evaluating student programming project. Students begin with an existing code base, a practice that is crucial for mirroring real-world software development. Students then iteratively complete tasks that pass predefined tests. The instructor only manages a hidden parent repository containing solutions, which is used to generate an intermediate public repository with these solutions removed via a templating system. Students are invited collaborators on private forks of this intermediate repository, possibly updated throughout the semester whenever the teacher changes the parent repository. This approach reduces grading platform dependency, supports automated feedback, and allows the project to evolve without disrupting student work. Successfully tested over several years, including in an edX MOOC, this organizational pipeline provides transparent evaluation, plagiarism detection, and continuous progress tracking for each student.

CCS Concepts

• **Social and professional topics** → **Computing education**; • **Software and its engineering** → *Software configuration management and version control systems.*

Keywords

project, teaching, test-driven development, grade, java, git, extreme code immersion

ACM Reference Format:

Anonymous Author(s). 2018. ICLF: An Immersive Code Learning Framework based on Git for Teaching and Evaluating Student Programming Projects. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Programming projects are a fundamental component of students' education to bridge theoretical knowledge with practical applications. Ideally during programming projects the students should also acquire a set important skills for a computer scientist like the ability to write clean code, test and debug it, integrate into an existing code bases, and collaborate with others using version controls systems. Unfortunately, many computer science programs still focus on theory or small projects starting from scratch, leaving graduates ill-equipped to work directly on more complex software projects.

Designing high-quality implementation projects and evaluating them effectively remains a significant challenge for educators, particularly in contexts such as Massive Open Online Courses (MOOCs), where a centralized manual correction is impractical.

Our motivation question is: *How can we effectively teach students to work with large, real-world codebases while providing transparent, scalable, and automated evaluation in programming courses, particularly in contexts like MOOCs?*

To address the question we propose the *Immersive Code Learning Framework* (ICLF) built around Git for teaching and evaluating student programming projects. Instead of starting from scratch, students are placed in a scenario mirroring real-world software development. Every student is invited as a collaborator on a git repository already containing code as a starting point. This is the immersive aspect, where students engage with an existing code base rather than starting a project from scratch. Students are then required to iteratively modify and extend this code base, completing tasks designed to meet predefined and graded unit tests. The progress is thus continuously assessed, with students receiving feedback on their performance after each submission.

As illustrated on Figure 1, the instructor owns the students' repositories, which are private forks from another intermediate public repository that is itself derived from a hidden parent repository containing the original source code with solutions. The instructor only pushes changes to this parent repository. Whenever the instructor makes a push to the parent directly, the solutions are removed to generate the intermediate repository using a templating system embedded in the source code comments. Students are then invited to pull the updates from the intermediate repository to ensure their source code remains up-to-date.

ICLF minimizes the instructor's dependency on a grading platform, ensuring that the code provided to students is well tested, functional, and the project remains feasible. Additionally, it allows instructors to dynamically evolve the project by adding tests or fixing bugs throughout the semester, even as students work on their initial tasks. The approach supports automated grading, plagiarism detection, and detailed tracking of individual progress without disrupting students' workflow.

This methodology was successfully tested over several years, including in an edX MOOC. Anonymous evaluations of our courses

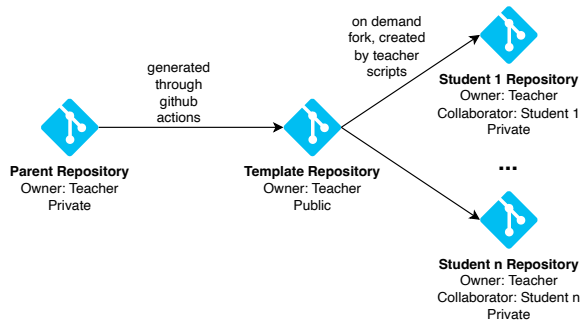

Figure 1: Set of repositories involved

highlight that students appreciate this teaching process for its structured and transparent evaluation approach. They value the flexibility to run tests locally and, when confident in their code, to validate their progress within the secure grading environment.

ICLF is especially suitable for teaching related to systems development. This includes topics such as operating system development, compiler design, optimization, machine learning libraries, and more.

Our contribution is a detailed explanation of the framework. We also provide a small complete open-source instantiation of it for a java project so that anyone interested can reuse all the related scripts and source code. We share our experience at using ICLF in two courses, including a MOOC.

Section 2, gives the related work. Section 3 gives a comprehensive explanation of the architecture, covering both functional and architectural aspects from the perspectives of students and teachers. Section 4 then describes how the framework is used to grade exercises within a secured environment. Additionally, this section features a detailed "hello-world" open-source example of the framework using a Java library tailored for precise assessment of student code. In the final section 5, we discuss our experiences using ICLF across various courses, including a MOOC on edX. This section also provides insights into how students engage with the framework.

2 Related Work

Peer review in computer science education offers significant benefits for both students and educators [7]. Regarding grading, students can be evaluated based on their peers' assessments of their code. However, peer review alone has limitations, such as students struggling to determine whether their work meets expected goals and the delay in receiving feedback. These limitations are addressed by ICLF. In our view, peer review can serve as a complementary approach to ICLF.

In *Inquiry-Based Learning* [9] learners actively engage through questioning, information gathering, and solution exploration. The goal is to encourage problem-solving skills. One such an approach is the so called *creative problem solving* teaching method [12] used for a MOOC on discrete optimization. In this approach, students are challenged to solve problems using any means necessary to achieve the desired outcome. Our approach differs somewhat, as students are tasked with integrating into an existing source code simulating the experience of joining a project in progress but, our approach is not mutually exclusive.

There are also similarities between our teaching framework and the one of *Extreme Apprenticeship* (XA) [13]. XA also allow students to run and execute tests locally as often as they wish. XA aims to guide students in using a working process similar to that of professional programmers. Our method mainly differs in that students interact with exercises through a git repository, rather than a plugin specifically developed in an IDE. Also, XA does not automatically generate a template repository from the solution of the teacher and therefore need the so-called "Alpha-Beta-Open" release process to verify test thoroughness that our continuous deployment pipeline automates completely.

GitHub Classroom is a tool offered by GitHub sharing some similarity with our teaching framework and to some extent could be used to instantiate our framework. Students can join a classroom and work within a skeleton project. However, it does not allow them to collect automatically get the grades resulting from testing. Additionally, there is no guarantee that the students have kept the provided unit tests intact, resulting in additional work from the teacher to validate the grades.

Web-CAT [4] is a Web tool for automated testing and grading of programming assignments. It is not Git-centric and requires students to upload their files manually or use a dedicated IDE plugin for submission. While Web-CAT is easier to use at the bachelor level, it is less aligned with the workflows of a professional development project compared to ICLF.

Autograder is a framework used to grade Java exercises [6]. However, unlike JavaGrader, this tool does not rely on Junit5 [2] standard testing library. Other graders exist for Java using Junit [5, 8], but these web-based tools are targeted for bachelors students.

3 Description of the framework

We explain in this section the teachers and students main interactions with the system. First, we discuss how students enroll in a course and begin working as a collaborator on a private fork of a intermediate template repository. Next, we explain how this public intermediate template repository is automatically generated from the private teacher parent repository using an operation called "stripping" removing and replacing parts of the source code.

We use GitHub as an example of a code hosting platform for the remainder of this paper; other similar platforms can be used, such as GitLab or Bitbucket. The only requirement for those platforms is to provide a rest API enabling users to create and interact with repositories [1] as well as a continuous delivery pipeline such as GitHub Actions to automate the execution of the scripts.

3.1 Enrollment

From the student's perspective[1], enrolling in the course is simple - they only need to enter their GitHub username on a grading platform[2] and accept the invitation to collaborate on a private Git repository. The git repository the student receives is a *private* (i.e. not accessible to others, excepting the teaching staff, the owner of

[1]The description can easily be extended for groups of students working a same project and sharing the source code.
[2](INGInious [3] in our case).

all the repositories) fork of template project, created by a GitHub bot account, that contains gaps representing missing implementations which they will need to complete over the course. By preventing students from forking the template themselves and keeping the teacher as the owner, the teaching staff avoids solution sharing and ensures that students' repositories remain private. The student is then ready to work and commit on its repository, filling the gaps and passing the graded unit tests. Figure 2 provides a sequence diagram of this process.

This methodology enables the teaching team to access all activities performed by the students on the repository. Since all repositories belong to the GitHub bot account owned by the teacher, it allows to retrieve all repositories and perform various tasks on them. This includes collecting analytics, check potential plagiarism issues as well as manually reviewing the code submitted by students.

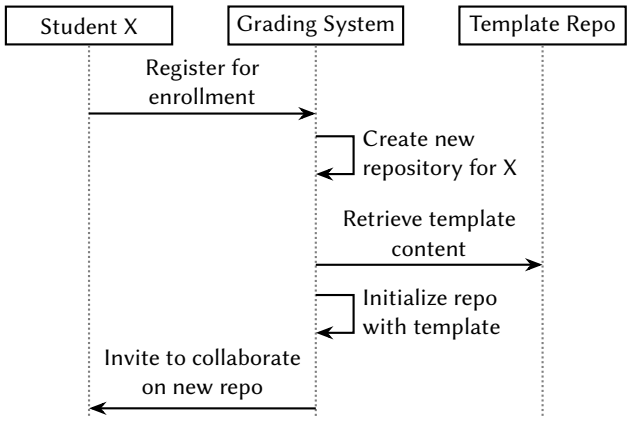

Figure 2: Enrollment of a new student.

3.2 Student interactions on the repository

The work performed by a student on its repository consists of filling the missing implementations, in an adequate order communicated by the teaching staff to the students. Unit tests are provided for the missing implementations, allowing students to assess the correctness of their code locally. Whenever an implementation has been successfully written by the student, the trainee can commit and push the changes on its repository. This is the only way for the student to test it fully, on the remote grading platform (we describe this process in section 4). This has the benefit of forcing the student to interact with version control systems.

Moreover, the student must use git to interact with the template repository as well. Indeed, the teacher can possibly update the template to fix a potential issue or to add another exercise to complete during the semester. In such cases, the student must run git commands to integrate the changes from the template within its own project. This results in merge operations, after which the student repository is up-to-date with respect to the template.

3.3 Teacher update

To generate the template repository, a teacher must have a project containing all solutions, with specific parts marked by delimiters to indicate which code needs to be stripped. The stripping operation involves removing the code written between these delimiters. An example of this operation is illustrated in Figure 3. All the code between BEGIN STRIP and END STRIP comments disappears in the template. The code after STUDENT is injected to ensure that the student receives a code that compiles. The teachers can use any such delimiters in their project, which is minimally intrusive and ensures that they have a clear view of the level of difficulty of the exercises proposed to students.

```
public int increment(int x) {
    // STUDENT throw new
        RuntimeException("Not implemented");
    // BEGIN STRIP
    return x + 1;
    // END STRIP
}

public int increment(int x) {
    throw new RuntimeException("Not
        implemented");
}
```

Figure 3: Example of a strip operation on a Java source file. The code on the top is the teacher (solution) implementation. The code below corresponds to the public template. It is generated automatically by parsing the solution file and processing the strip tokens. We rely on a small utility tool for this: https://pypi.org/project/amanda/

The teacher can indicate which parts of the source code need to be stripped by adding tokens to the relevant files in the solution. This allows the teacher to have a clear overview of the gaps that need to be filled by the students. The combination of all the stripped files then creates the template repository, with gaps for the missing implementations that students must complete.

The continuous deployment capabilities of platforms like GitHub Actions or Gitlab CI/CD, enable to automate entirely the generation and update of the template repository. Upon a push to the solution repository by the teacher, a script is executed to strip the solution and create a new version of the template repository. The updated version is then pushed to the template repository, making it instantly accessible to all users. Figure 4 illustrates this automated process. The relation between the teacher, the template and a student repository is represented on Figure 5.

The teacher can also utilize the same approach to selectively hide certain unit tests from students. This feature enables a portion of the tests to be visible in the template, while keeping the rest hidden. The ability to hide tests can be useful in situations where a few simple tests are made available only for students to understand the expected behavior, while the remaining tests are hidden to prevent brute-forcing and encourage students to develop their own tests. One can also keep the tests consistent between the teacher and the template repository to ensure a fully transparent evaluation of student projects.

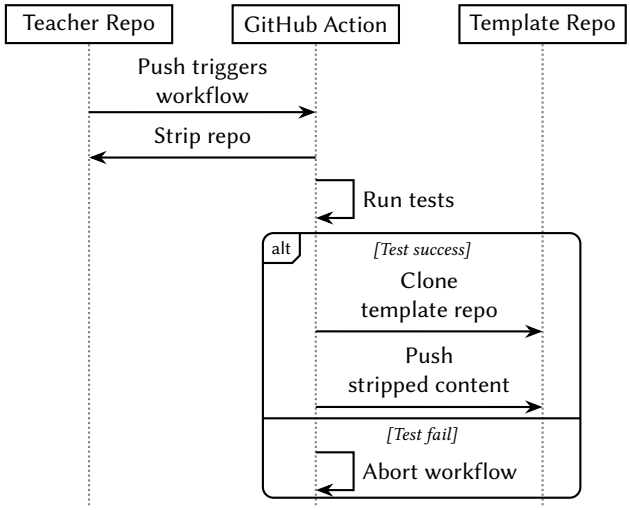

Figure 4: The update of the template repository. A GitHub action is triggered at each push on the solution repository, stripping the code and pushing it to the template project.

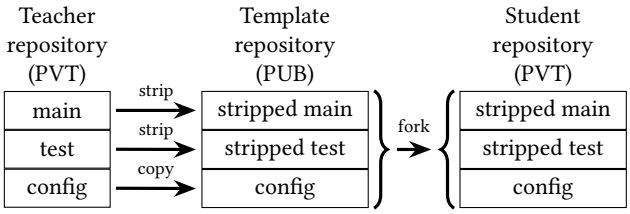

Figure 5: Creation of the template repository through stripping and of the student repository through forking. Only the template repository is public, the teacher and student ones are private.

4 Grading

The unit tests in a project should provide a way to determine a student's grade when executed. However, certain modules in a project may be more crucial than others, and teachers need to be able to assign weights to relevant tests accordingly. To address this, we use JavaGrader, a Junit5 extension that we have developed that enables Java unit tests to output a graded report. Similar functionalities could be achieved with other programming languages such as Python or C++. We also discuss how to set up an automated grading system that securely runs tests in a controlled environment—ensuring the original tests are execute, guaranteeing the validity of the final grade.

4.1 Grading the tests

To benefit from features introduced by Junit5 [2] (such as parametric testing, parallel execution, tagging and filtering of the tests, etc.), we rely on an extension called JavaGrader. This extension adds additional functionalities to the tests in the form of annotations, similarly to python decorators. Here are the most important annotations provided by the extension:

- `Grade` is the core annotation from JavaGrader. It says that a unit test is graded and might contain optional parameters such as a maximum run time or a weight. The maximum run time can be specified as either CPU timeout (how much time the current thread has been spent running the test) or as a wall-clock timeout (how much time has passed since the beginning of the test).
- `GradeFeedback` can be used to provide an additional message depending on the outcome of a unit test.
- `Forbid` prevents the use of a given library when executing the test. This is useful for exercises where a student is asked to write code without relying on a specific library or data structure. It is implemented by overriding the class loader from the thread running the test, ensuring that all classes loaded by the student are allowed.

An example usage can be found in Listing 1. More annotations are provided in the library. They cover some cases, such as conditional tests, that are only run if the previous was successful. This can be used for testing the time complexity of an algorithm, which should only be run if previous tests have first validated the correctness of the program. JavaGrader can be used alongside other extensions when executing the test suite.

```
@Grade
public class MyTests {
    @Test
    @Grade(value = 5, cpuTimeout = 1)
    @Forbid("java.lang.Thread")
    void mytest1() {
        //this works
        something();
    }
    @Test
    @Grade(value = 3)
    @GradeFeedback(message = "You forgot to
        consider this particular case
        [...]", on = FAIL)
    void mytest2() {
        //this doesn't
        somethingElse();
    }
}
```

Listing 1: Transforming unit tests into graded tests by using JavaGrader. `mytest1` benefits from the forbidding of the `java.lang.Thread` library, and adds a cpu timeout on the test. Indeed, the cpu timeout would have been meaningless if the student run its code in a spawned thread, as it is not related to the initial thread running the test.

4.2 Student self-assessment

As a fraction of tests from the teachers are provided in the template (and thus in the student) repository, the trainees can use them to their advantage. They can locally run (a part of) the graded tests, and obtain an immediate feedback, along with debugging possibilities,

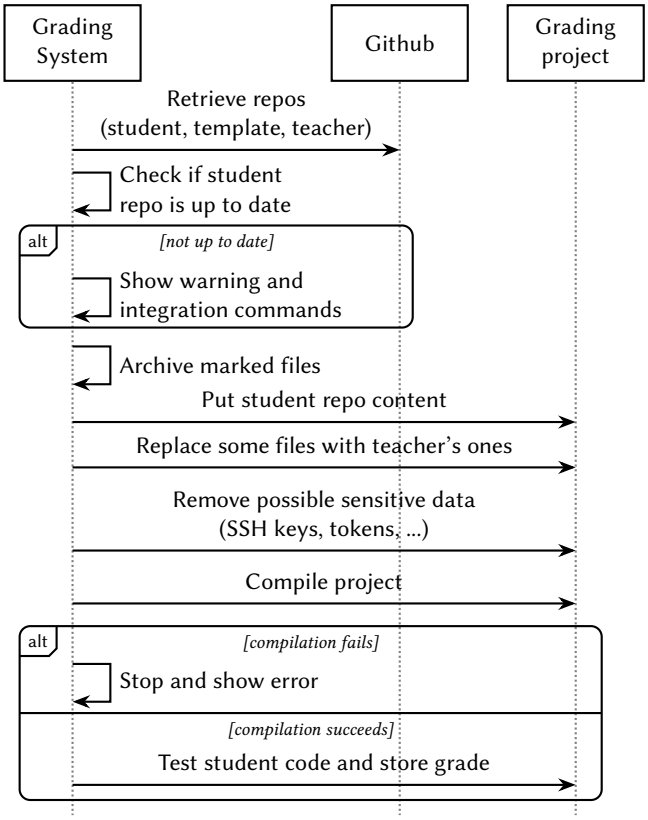

Figure 6: Grading a task within a grading system. The feedback is constructed by the grading system and contains the information related to the grading and notes for the students.

directly on their computer, even when offline. The students are then able to iterate more efficiently on their code; this reduces both the overall effort of the code-evaluate-fix loop and the computation load on the online grading system. More advanced tests, requiring the usage of a secret or specific computing capabilities not present on trainees' computers, can be stripped and made available on the online grading system only (as detailed in the next section).

4.3 Running the tests in a secure environment

Truly assessing the grades from the students requires particular caution. Simply retrieving a student repository and running its tests does not guarantee that their output is valid: nothing prevents a student from pushing a modified version of the tests. Furthermore, it might be a good practice to let the students modify the tests: they should be free of adding more tests cases if they wish, to convince themselves of the correctness of their code.

Figure 6 depicts the sequence diagram for grading a task, with the key steps outlined next.

(1) The student, template and teacher repositories are retrieved.
(2) It is ensured that the last commit from the template appears within the student repository. If not, a warning will be given as feedback letting the student know that the repository need to be up-to date to be graded. Some commands are prompted to help the student integrate the changes from the template.
(3) Some marked files and directories from the student repository are archived. The archived files can be used for further analysis by the teaching team such as plagiarism detection between students with a tool like JPlag [10].
(4) The grading project is prepared by replacing files or directories from the student to put the ones from the teacher instead (typically the tests, including hidden ones, and configuration but possibly additional files such as a sensitive dataset that must be hidden to students).
(5) The sensible files that are mandatory for cloning the repositories are removed (ssh keys, token files, etc.)
(6) The grading project is compiled. Failure to compile them stops the evaluation with an error message corresponding to the error printed by the compiler.
(7) Test the student code and give feedback to the student. The grades are stored.

To prevent any memory corruption and fully automate the grading, we use a grading platform such as INGInious [3][3]. INGInious essentially run these steps in a *jailed* environment ensuring, among other things, that the code cannot interact with anything other than the grader, cannot access the internet, and cannot read or write files on the file system except where they are specifically allowed to. This platform is also used with a special task for creating the repository, using the methodology presented in section 3.1.

The feedback given back to the student in step 7 consists of the grade for the task, and any possible information proving useful feedback to the student. In the case of JavaGrader that we use, an example output related to the tests from Listing 1, is presented on Figure 7.

Test	Status	Grade	Comment
MyTests	❌ **Failed**	0.62/1	
→ mytest1()	✅ Success	0.62/0.62	
→ mytest2()	❌ **Failed**	0/0.38	You forgot to consider this particular case [...]
TOTAL		**0.62/1**	
TOTAL WITHOUT ABORTED		**0.62/1**	

Figure 7: Output of the tests from Listing 1. As the class `MyTests` is also annotated with `@Grade` without specifying optional parameters, the maximum grade is 1. This value can be overridden by changing the class annotation.

Setting up a grading task can be done in a matter of minutes. Except for the homework statement, only a few parts need to change between two grading tasks. The procedure remains the same across the tasks for one course, with the exception of steps 3 and 4, where different files or directory can be specified, and for the tests that needs to be run.

As an example, we provide a detailed "hello-world" open-source implementation of the whole pipeline. The example is accessible

[3]Other plateforms such as GradeScope [11] could also be used.

through https://inginious.org/course/iclf-example. It uses INGInious as the grading system. The related repositories can be found at https://github.com/OneAnonymizedUser/teacher-repository for the teacher repository and at https://github.com/OneAnonymizedUser/template-repository for the template. For demonstrating purposes, the teacher repository is public, but it should remain private in a real scenario.

5 Use cases

The teaching framework presented in this paper was successfully used in two courses for more that three years: A Discrete Optimization Course and a Constraint Programming Course on edX.

5.1 A Discrete Optimization Course

This course focuses on advanced algorithms for discrete optimization, with theoretical content similar to that of [12]. Each task is related to a different optimization technique, one of these is the well-known branch-and-bound method. Students are provided with code for cumbersome tasks such as instance parsing, along with an incomplete skeleton of a generic algorithm (e.g., branch-and-bound) that they need to complete and then instantiate on a problem (e.g., the traveling salesman problem). Some of the graded tests are hidden from the students, primarily the most challenging instances, making it difficult to achieve the maximum grade, in line with the method introduced in [12]. The complete teaching framework, as described in this paper, was utilized over three years, with about 100 students each year.

Each year, our teaching team prepared the projects just in time. As a result, students had to pull the next project, which appeared as a new package in the template source code, every two weeks. Anonymous feedback from students demonstrated their appreciation for the teaching framework and course format.

As the framework enables the collection of global statistics, we computed the average number of commits over time by the students in a typical year, as shown in Figure 8 to get insights on how and when students interact with the framework. We observe that, as expected, students tend to work closer to deadlines, with a peak in the number of commits before each due date. Interestingly we observed that not all commits correspond to a grading request, indicating that students prefer to assess their code locally before submitting their work.

5.2 MOOC on Constraint Programming on edX

We have taught a Constraint Programming (CP) course on edX for three years to an audience of approximately 300 students per year (100 students from our university, and 200 online external students) using the proposed methodology[4]. Most university courses on constraint programming teach it at the user level of a given library. In contrast, ICLF enabled us to have students take charge of the development of an entire constraint programming solver library. For students, a basic constraint programming solver is a fairly large piece of software, consisting of around 10,000 lines of code, excluding tests. The sequence of proposed tasks starts at the lower levels of the solver and gradually moves toward applications, ensuring there is no "dark magic" for students from beginning to end. There

[4]Inginious enables the Learning Tool Interoperability (LTI) connection with edX

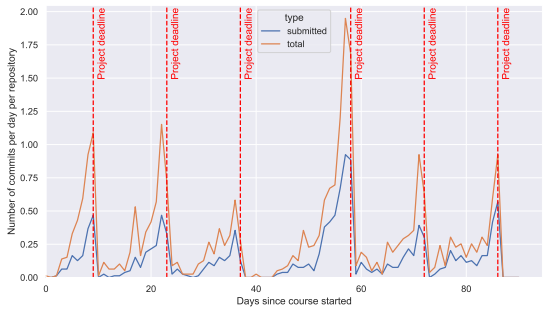

Figure 8: Average number of commits (per student-repository) over time for the Discrete Optimization course. We report the total number of commits and the number of commits that were submitted to the grading system. The commits related to merges from the template, fetching the new projects instructions, are omitted.

are no fixed deadlines for the projects, but students are strongly encouraged to stay up-to-date with the exercises, as some require code and algorithms from earlier projects to complete. Although not included here due to space constraints, we collected data on the commits made by students each day during the semester. We observed a higher number of commits on Fridays, corresponding to on-site practice sessions organized at our university. Additionally, we noted that the number of commits gradually increased as the semester progressed, peaking as the final deadline approached.

6 Conclusion

We have presented the Immersive Code Learning Framework (ICLF), a scalable, and effective approach based on git to teaching and grading programming projects. By placing students in an environment with pre-existing codebases, the framework simulated real-world software development scenarios, fostering skills such as debugging, version control, and iterative development.

The framework's ability to provide instant feedback, automate grading, and dynamically evolve projects was successfully demonstrated in diverse educational settings, including a Discrete Optimization course and a Constraint Programming MOOC. Anonymous evaluations from students underscored the value of ICLF in offering a structured, transparent, and rewarding learning process.

References

[1] 2023. GitHub REST API documentation - GitHub Docs. https://docs.github.com/en/rest [Online; accessed 22. Mar. 2023].
[2] 2023. JUnit 5. https://junit.org/junit5 [Online; accessed 21. Mar. 2023].
[3] Guillaume Derval, Anthony Gego, Pierre Reinbold, Benjamin Frantzen, and Peter Van Roy. 2015. Automatic grading of programming exercises in a MOOC using the INGInious platform. *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15)* (2015), 86–91.
[4] Stephen H Edwards and Manuel A Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*. 328–328.
[5] Olly Gotel, Christelle Scharff, and Andy Wildenberg. 2007. Extending and contributing to an open source web-based system for the assessment of programming problems. In *Proceedings of the 5th international symposium on Principles and practice of programming in Java*. 3–12.

[6] Michael T Helmick. 2007. Interface-based programming assignments and automatic grading of java programs. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*. 63–67.

[7] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. 2020. A review of peer code review in higher education. *ACM Transactions on Computing Education (TOCE)* 20, 3 (2020), 1–25.

[8] Ashrita Kunchala and Maneesh Gunnala Ranga Rao. 2016. Java auto grader. (2016).

[9] Margus Pedaste, Mario Mäeots, Leo A Siiman, Ton De Jong, Siswa AN Van Riesen, Ellen T Kamp, Constantinos C Manoli, Zacharias C Zacharia, and Eleftheria Tsourlidaki. 2015. Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational research review* 14 (2015), 47–61.

[10] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. 2000. *JPlag: Finding plagiarisms among a set of programs*. Univ., Fak. für Informatik.

[11] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. 2017. Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work. In *Proceedings of the fourth (2017) acm conference on learning@ scale*. 81–88.

[12] Pascal Van Hentenryck and Carleton Coffrin. 2014. Teaching creative problem solving in a MOOC. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 677–682.

[13] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. 2012. Multi-faceted support for MOOC in programming. In *Proceedings of the 13th annual conference on Information technology education*. 171–176.