Continuous Publishing of Online Programming Assignments with INLOOP

Martin Morgenstern and Birgit Demuth
Technische Universität Dresden
{martin.morgenstern1,birgit.demuth}@tu-dresden.de

***Abstract*—We supplement our software engineering lectures and exercises with facultative online programming assignments. Based on our experience with previous systems, we developed INLOOP, a solution tailored to our needs. The core of the system is a test runner component that supports virtually any kind of solution assessment through a generic interface. The system supports automated plagiarism checks and grading, which we currently use to award students bonus points for written exams. In addition, it is directly integrated with a version control system to facilitate peer-reviewed development of assignments by our team of instructors and to enable continuous publishing of assignments. The resulting workflow helped to improve the quality of our assignments and tests. The architecture of INLOOP is characterized by asynchronous solution processing and takes advantage of container technology to achieve scalability, robustness and isolation, while maintaining a small footprint. So far, our approach is a success: we observed an increasing number of students who actively practiced programming, and they clearly performed better in the written exam. The poster presents the architecture of INLOOP and the implemented continuous publishing workflow.**

I. Motivation and Goals

The motivation of the INLOOP approach is to improve teaching in software engineering by taking advantage of interactive technologies. Our goals can be described from four perspectives: *Students* should be given an easy-to-use environment to practice their software development skills. They should receive immediate, detailed and user-friendly feedback for submitted solutions. *Instructors* should be given tools to manage complex programming assignments and handle increasing numbers of students, because at such a scale, individual feedback on solutions cannot be manually provided. *Operators* are primarily concerned about the robustness and scalability of the system. *Researchers and Developers* want a system that is easy to comprehend and extend.

II. Background

Since 2006, we have been supplementing our software engineering lectures and exercises with facultative online programming assignments. The course focuses on object-oriented modeling and programming using UML and Java. Our system uses public JUnit tests to automatically verify solutions submitted by the students; the task complexity ranges from basic to demanding, where students have to implement a design given by a UML diagram. We award students one exam bonus point for each demanding task solved.

III. Features and Design

In INLOOP, tasks are organized into categories and have optional publication dates and deadlines, which can be used to restrict the user's ability to view a task or submit solutions for it. The test runner of INLOOP supports virtually any kind of solution assessment, i.e., any build, test or analysis tool that is available for Linux, in a secure and robust manner. Instructors specify the test environment for a task in a declarative fashion, and the system builds a test image that is used by the test runner. By default, the test output is shown to the user as-is, but pretty-printer modules may transform it into HTML fragments that can be integrated into the user interface. Additionally, the system uses an offline version of the JPlag plagiarism checker [2] to automatically flag suspicious solutions. The plagiarism check is executed as a batch job after a task deadline has passed, in order to be able to mutually compare all submissions, including submissions from previous teaching periods.

A distinctive feature of INLOOP is its direct integration with the Git version control system: all task artifacts (metadata, description, diagrams, tests, etc.) are retrieved from a remote repository, transformed if necessary, and published. For example, such transformations include building the test image, rendering Markdown task texts, and creating ZIP archives of supplementary material. This process is triggered by a Git push directly in the IDE on the instructor's computer; a feature we refer to as *continuous publishing* (Fig. 1). Before, we had to manually edit and publish tasks in the backend of a web application, which was an error-prone process. Our approach has the following advantages: 1) the process is fully automated, 2) all task artifacts are stored in one place, 3) changes can be tracked, and 4) task artifacts can be developed in collaboration. Another key characteristic is that continuous publishing can be combined with continuous integration tests and a branching model such as Gitflow [3]. As an example of the latter, instructors can maintain a set of tasks in a protected "master" branch and a derived set of tasks in a "develop" branch, for the current and next semester, respectively, and configure INLOOP to use the "master" branch for publishing.[1] The "develop" branch could then be used to work on backward incompatible changes which potentially break existing submissions.

[1] Although INLOOP does not change the status of existing submissions when a new task revision is published, maintaining a stable task set during a semester is advisable to avoid student confusion.

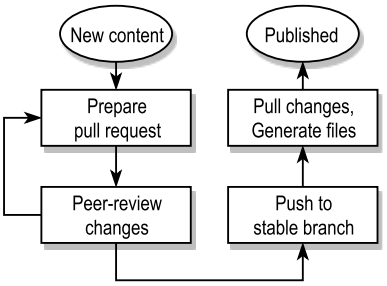

Fig. 1. Continuous publishing workflow.

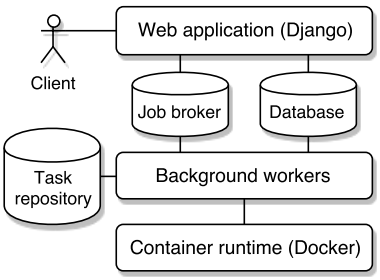

Fig. 2. INLOOP architecture.

INLOOP is designed as a multi-process application (Fig. 2). The core components are the web application and the background workers, both implemented in Python. An event-based job broker and a relational database are used to exchange data between the processes. The web application is a standard Django application, but instead of blocking on long-running test runner jobs, it submits them to a job queue. The jobs are picked up by a configurable number of background workers, and are executed inside their own, ephemeral Docker container. Apart from the necessary input and output, the containers are completely isolated from each other, the network, and the host system; in addition, strict resource limits are enforced.

IV. Experiences

Having bonus points as an incentive seems to be a good motivation for our students. Hence, we observed an increasing number who practiced their development skills with INLOOP. In the summer term 2017, 11074 solutions were submitted, up to 419 per day in the week before the written exam. Students who actively practiced programming online clearly performed better in the written exam. But in our setting, plagiarism checks remain a necessity: out of 1838 valid solutions, 127 (6,9%) have been identified as plagiarism, and some of them were complete copies.

With continuous publishing, we eliminated inconsistencies by generating all published content from a single source repository. The streamlined workflow enabled us to spend more time on actual task development, instead of time-consuming and error-prone manual publishing. Furthermore, our chosen branching model made it easy to incorporate urgent bug fixes into the current "stable" set of tasks, while at the same time being able to prepare larger source code transformations for the next semester.

V. Related Work

There are many online programming systems available. We are not aware of one that incorporates a continuous publishing workflow based on version control, but the idea has been used in the past in other publishing systems, e.g., the Open Journal System [4]. However, GitHub Pages [5] is an example of a website hosting service with a Git-driven workflow, and was the model for our approach. INLOOP's closest relative is the Praktomat [6], a system that supports a broad set of programming languages, but uses specific checkers instead of a generic interface. It offers a flexible grading system which our system does not provide, and its first version also featured mutual reviews by students [7]. But Praktomat does not use asynchronous processing, and requires the intervention of the operator to enforce resource limits and isolation. Marmoset [8] is a related system that uses asynchronous processing and build servers on separate (virtual) machines. An interesting feature is its incentive system to motivate students to start work early by giving them a daily budget of only two test runs.

VI. Future Work

We plan to integrate static code analysis and style checks into INLOOP and ultimately reject code that violates a predefined coding standard. In the same spirit, the current binary *solved/not solved* test result could also be enhanced to give grades based on the code quality metrics calculated by such tools. To make it more challenging for students to solve tasks by just satisfying static test data, our unit tests could be complemented with property-based testing. Another promising idea, inspired by Web-CAT [9], is to have programming assignments that focus on testing skills. For example, students are given a specification and a software "black box", and are then asked to find as many bugs as possible by writing a test suite.

References

[1] M. Morgenstern and D. Muhs, "INLOOP: interactive learning center for object-oriented programming." [Online]. Available: https://github.com/st-tu-dresden/inloop

[2] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *J. UCS*, vol. 8, no. 11, p. 1016, 2002.

[3] V. Driessen, "A successful git branching model." [Online]. Available: http://nvie.com/git-model

[4] J. Willinsky, "Open journal systems: An example of open source software for journal management and publishing," *Library Hi Tech*, vol. 23, no. 4, pp. 504–519, 2005.

[5] GitHub Inc., "GitHub Pages." [Online]. Available: https://pages.github.com/

[6] J. Breitner, M. Hecker, and G. Snelting, "Der Grader Praktomat," in *Automatisierte Bewertung in der Programmierausbildung*. Waxmann Verlag GmbH, 2017, no. 6, pp. 159–172.

[7] A. Zeller, "Making students read and review code," in *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education*. ACM, 2000, pp. 89–92.

[8] J. Spacco, D. Hovemeyer, B. Pugh, J. Hollingsworth, N. Padua-Perez, and F. Emad, "Experiences with Marmoset," Tech. Rep., 2006.

[9] S. H. Edwards and M. A. Perez-Quinones, "Web-CAT: Automatically grading programming assignments," *SIGCSE Bull.*, vol. 40, no. 3, pp. 328–328, Jun. 2008.