Towards Using Continuous Integration Tools to Teach Programming Courses

Erik Kral, Petr Capek
Faculty of Applied Informatics
Tomas Bata University in Zlin
Nad Stranemi 4511, 760 05 Zlin, Czech Republic
{ekral,capek}@fai.utb.cz

***Abstract* — This work proposes a study focused on Continuous Integration (CI) tools in teaching programming courses as well as for automatically grading student assignments. There are many automatic grading tools for students´ programming assignments which share most of their functionalities with CI tools. We would like to provide a methodology on how to implement this tool in teaching programming languages. We will evaluate and compare common CI tools and run experiments with students in programming courses.**

Keywords—Continuous Integration; Software Engineering; Education; Programming Course; Student´s Programming Assignments

I. Introduction

In the programming languages teaching field, we have a problem with how to balance teaching basic to advanced topics using professional tools such as source code control, build servers, and automated building, testing and deployment. These tools are typically targeted at advanced developers and large groups of developers working long-term on a large, single-source code repository. We currently teach basic and advanced programming courses at our university but we do not use any advanced form of Continuous Integration (CI) tools that are the industry standards nowadays in any of these courses. At Tomas Bata University (TBU) in Zlin, we currently use assignment evaluations based on Docker Technology [1] which is sufficient - but only supports a limited number of languages and build tools, and it is not integrated with source control tools or advanced automated tests. There are many automatic assignment grading software tools [2], built mostly by universities. However, in our experience, it is very hard to keep such software up-to-date with new languages and related software (e.g. compilers, frameworks). Most of these Automatic Grading Tools (AGT) share functionality with Continuous Integration (CI) methodology [3] - which is a part of the Extreme Programming (XP) software development methodology. Studies on using XP for teaching programming courses [4] already exist. This study [5] covers using CI to teach Software Engineering and suggests a method to use it by which a large group of students work on a large legacy code base - as opposed to an approach in which a small, isolated group of students work on a project that is not large enough to show the benefits of CI professional practices. However, the approach of using a large group of students to work on a legacy code is not suitable for programming courses for beginners. Both AGTs and CI share similar methodologies, like unit tests, code quality metrics, and code reviews. Our goal is to use CI tools as an AGT and to begin continual use of this for beginner courses through to advanced courses; i.e. from basic isolated student´s assignments through to shared, long-term tasks for large groups of students.

II. Problem Definition

Both AGTs and CI tools are commonly-used and mature tools, but AGTs are not in general, a professional standard since they are built by universities and not software companies. Our goal is to use professional grading software as soon as possible so that the students can gradually get used to working with this tool.

Most AGTs have the following features:

- Assignment planning and coordination between student and teacher. Teacher´s reviews and communication between teachers and students
- A single-source code repository; but, the code is not shared between students
- Plagiarism detection
- Automatic and manual grading of functional and nonfunctional requirements - similar to the requirements on commercial software
- Sandboxed runtime to prevent student hacks on servers
- Integration with other teaching tools -like Moodle

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Program, Project No.: LO1303 (MSMT-7778/2014); as well as the European Regional Development Fund, under CEBIA-Tech Project No.: CZ.1.05/2.1.00/03.0089.

978-1-4673-9795-7/15 $31.00 © 2015 European Union
DOI 10.1109/CSCI.2015.78

Most CI tools have the following features:

- Task planning, team coordination, code reviews
- Single-source code repositories, and the code is intensively shared among developers
- Automated build, tests and deployment
- Code quality metrics – static or dynamic code analysis
- Integration with 3rd-party tools

While the features of AGTs and CI tools mostly overlap, there is an important difference in their plagiarism detection, product deployment and the level of code sharing characteristics. The process workflow in programming courses has been analyzed. First the student´s workflow is described followed by the teacher´s workflow

A. *Student´s workflow*

In the beginner courses, there is no single source code repository (mainline code) and there are no sharing; or, only small groups of two or three students share the code. Overall, students cannot access one another's solutions. But, on the other hand - in advanced courses, the mainline may be shared by large groups of students, and students work on the same mainline.

1) Students can access a current assignment description, its goals, subtasks and grading metric

2) Students can copy the current source code (mainline) into their local development machine as a working copy using a source code management system from the prepared mainline -which may include automated tests

3) Students can alter the source code in a working copy and change or add automated tests and students can build a working copy and run automated tests

4) Students can update a working copy with possible changes from teachers or collaborating students, fix problems if it is their responsibility, or communicate fixes with collaborators; and, if it passes tests, commit changes to the mainline

5) Students' new commits are detected on the server and students receive the results of another set of automated tests, code quality measurements, plagiarism tests – and optionally, may get teacher´s code reviews and a final grade.

B. *Teacher´s workflow*

Students are organized by subject, and there can be more student groups in each subject. As in beginner courses, students may work independently from one another, teachers have to prepare copies of the main repositories and grant access to each student.

1) Teachers prepare current assignment descriptions, their goals and grading metrics and also create automated tests for students using interfaces or dummy classes where each assignment can be split into several, monitorable subtasks

2) Teachers create one or more repositories (mainlines) for groups of students and assign students rights to work on these mainlines - since this can be time-consuming, it should be automated

3) Teachers can update mainlines including automated tests

4) Teachers do code reviews and final grading; after that, the student is not allowed to commit changes or get additional tasks

III. Methodology

In this work, we will evaluate and compare common CI tools based on the student´s and teacher´s workflows - presented herein. A methodology will be created on how to use CI tools and integration servers to automatically grade students. Experiments will be run in the following semester with students in the Beginner´s Programming course and the more advanced Object-oriented course. In total, more than 100 students are enrolled each year in this course, and 4 teachers run these courses.

IV. Expected Contributions

The main contributions of our research will be review of CI tools and their implementation by means of teaching the Programming course, a methodology for using CI tools for teaching and automatic grading and qualitative observations based on students' and teachers' feedback

V. Conclusion

In this work, we describe future research on using a Continuous Integration tool - usually in the form of a server for teaching. We have introduced working workflows and presented the methodology for our research.

References

[1] G. F. Špaček, R. Sohlich and T. Dulík, "Docker as Platform for Assignments Evaluation", Procedia Engineering, vol. 100, pp. 1665-1671, 2015.

[2] J.C. Caiza, J.M. Del Alamo, "Programming Assignments Automatic Grading: Review of Tools and Implementations", in Inted 2013 Proceedings, 2013, pp. 5691–5700

[3] M. Fowler, "Continuous Integration", martinfowler.com, 2006. [Online]. Available: http://www.martinfowler.com/articles/continuousIntegration.html. [Accessed: 10- Oct- 2015].

[4] M.M. Muller, W.F. Tichy, "Case study: extreme programming in a university environment," in Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on Software Engineering, IEEE

[5] J.G. Sus, W. Billingsley, "Using continuous integration of code and content to teach software engineering with limited resources", in Software Engineering (ICSE), 2012 34th International Conference on Software Engineering, vol., IEEE