Automating Testing of Programming Assignments

Knut Anders Stokke
Dept. of Informatics
University of Bergen
knutandersstokke@gmail.com

Anya Helene Bagge
Dept. of Informatics
University of Bergen
anya@ii.uib.no

Håkon Heggernes Lerring
Dept. of Informatics
University of Bergen
hakon.lerring@uib.no

⟨Programming 2018 Posters⟩

Introduction Programming courses typically emphasize practical exercises, since experience is crucial to learning the craft; and to become effective developers, students need to learn not just programming, but also how to use at least some of the wide range of tools a modern programmer relies upon. At the same time, the conscientious lecturer or teaching assistant, will want to keep track of the students' work, in order to offer help, feedback and occasional nagging.

In this poster we describe our automated programming assignment management and testing system, which used for the second and third programming courses at our University.

Background Before, assignments were handled in one of two ways: weekly assignments were non-compulsory and would not be handed in at all (many students would neglect to do these), while compulsory assignments were submitted as a zip-file to the learning management system (LMS) or by email. Typically, the LMS would only allow a single submission, so the students would submit once, near the deadline. With this approach, the teaching staff would have no idea how the students were doing, or if they had even started their work at at all; and the students would easily make mistakes and miss files when bundling their submission files. Help and feedback along the way would be limited to the students who initiated contact or showed up to lab sessions and asked for help.

Continous feedback would also be hard; even without a "submit once only" limit in the system, downloading, extracting and examining the code would be quite a burde. If the students were stuck with a bug and needed debug help from a teaching assistant, they might send a code fragment or a zip file to the TA. This approach gets messy very quickly, as it's hard to keep track of code versioning.

Submission using a version control system is a natural answer to these challenges, and a natural fit with programming education. Together with version control, standard software engineering practices like continuous integration can be used to provide feedback.

Our approach has been to set up a GitLab installation, and connect it to an automated testing system. Each time a student pushes code to the server, compilation and testing will be triggered, and the results summarised in a report, so the student can identify any problems with the submitted code, and keep track of the status of all their assignments. Similarly, the test results are available to the teaching staff, and also summarised in a spreadsheet so progress for all students can be monitored at a glance.

GitLab GitLab[1] is a web portal and hosting solution for Git. It is available both as a cloud solution (free for small and personal projects, but also offering paid enterprise solutions) and as a self-hosted solution. We run our own GitLab installation[2] for our courses, so that we can adapt it to our needs.

In particular, we full administration rights in order to set up and manage accounts and repositories and such for all our students; for this reason we haven't considered using our University's central GitLab server or one of the public Git cloud services. User accounts are created automatically based on student enrollment from the University's Canvas LMS; fortunately, Canvas has an extensive API that allows for easy integration, though we have only made limited use of this so far.

[1]https://about.gitlab.com/
[2]https://retting.ii.uib.no/v18

Programming Assignments in Git Typically, our exercises also include some code (libraries, a framework, or some code that should be debugged and/or extended), so we set up each one as a Git repository, with the exercise text, the code hand-outs and the necessary project files (our students mostly use Eclipse, so we give them an already set-up Eclipse project).

This *upstream* project is hosted publicly on our server. Additionally, each student gets his or her private copy of the project, which is what they'll be working on. The setup is handled through the teacher's web interface; the system will automatically create a personal *fork* of the project for each student, and set it up so that a test job is spawned everytime the student pushes new code to the repository.

The students will download / clone their own repository through Git, and work on their exercise as normal. We try to encourage them to commit changes often, and push them to the server. After pushing, they can check their own page in the system and see their test results. In some cases, they'll mostly get the same test results as if they were running the tests themselves, but we may also add additional tests if necessary. Typical problems like missing files, wrong case in filenames and character set issues will also typically trigger a failure, so that the students have an opportunity to fix such problems before the assignment deadline.

Management Testing is managed through a web interface; the teacher can select a number of standard test setups (such as compiling a Java project, running all JUnit tests in a project, or checking whether the student has actually committed something), or define a custom test (look for a specific file, look for a merge commit in the git log, or any arbitrary command). Additionally, extra files can be added to the project before running tests, so that we may, for instance, run our own JUnit tests (or use our own test data) rather than relying just on what the students have in their own projects. The teacher can also choose a custom formula for computing the overall scoring of a test run; by default, the score is just the percentage of tests that succeeded.

Experiences We have been using the system since 2015; in the beginning only for the course INF101, which is a first-year course teaching intermediate Java programming, abstraction and object-orientation. The past two years, the students have also been using the system in their next course, INF102 (algorithms and data structures).

While we expected that the students would find the (nearly) instant feedback of the system convenient, and perhaps also find Git to be as troublesome and tricky to use as its reputation indicates, we had not fully anticipated how useful such a system would be for the course instructors. We use the system not just for compulsory assignments and work that will be graded manually, but also for non-compulsory activities. This gives us a fairly good overview over student activities, and also allows us to give targeted advice and follow-up. E.g., "You feel confused about generics? Well, I can give you a quick explanation, but you should really do Lab 4 to learn more about it." It is also much easier to see if some students are falling behind and then nudge them in the right direction.

Version control in general, and Git in particular, can be quite challenging for beginners, so we do have to spend some training the students in how to use the system, and offer help if something becomes really messed up. In practice, the extra time spent on this and on resolving Git-related issues are offset by an overall reduction in administration effort and in the effort needed to resolve programming questions. For example, a question from a student who's stuck with some strange bug is now easily resolved by just pulling the student's repository and examining the code, rather than trying to guess what's going on from a fragment sent by the student. The instructor's own working copy can easily be set up with a branch for each student, so that switching between students for grading or assistance is quick and simple.

We have also been able to train students in useful collaborative skills that will be handy for participating in open source projects or in a summer job; such as issue tracking, merge/pull requests, and commonly used version control skills. This is something they would normally only meet in a later software engineering course.