An Eclipse-Based Course Project Snapshot and Submission System

Jaime Spacco, David Hovemeyer, and William Pugh
Dept. of Computer Science, University of Maryland
College Park, MD, 20742 USA
{jspacco,daveho,pugh}@cs.umd.edu

Abstract

Much research has been done on techniques to teach students how to program. However, it is usually difficult to quantify exactly how students work. Instructors typically only see students' work when they submit their projects or come to office hours. Another common problem in introductory programming courses is that student code is only subjected to rigorous testing once it has been submitted. Both of these problems can be viewed as a lack of feedback between students and instructors.

We have built an Eclipse plugin to address this lack of feedback. The plugin has two main functions. First, it captures, to a central CVS repository, the complete state of a student's project every time he or she saves, adds, or removes a file. This produces a fine-grained history of the evolution of each student's project. Second, the plugin allows the student to submit his or her project to a central server. The submit server automatically compiles and performs limited testing of the student's submission, providing feedback on how close the project is to fulfilling the project requirements.

Our goal is to provide instructors and researchers with far more detailed information about how students learn and work, and provide feedback to students that will help them focus on achieving the goals of the projects we assign.

1 Introduction

The process by which students learn to write software is not understood nearly as well as it should be. Based on evidence from grades, course and instructor evaluations, students' write-ups and reports from group projects, and the experiences of teaching assistants and instructors during office hours, we know that the learning experience for some students is far from ideal.

For example, some students spend hours making seemingly arbitrary changes to code until it works—programming by "Brownian motion". We know that other students spend hours pulling "all-nighters" in the lab. But because we only catch brief glimpses of in-progress code during office hours and see the final, submitted version of the code, we have comparatively little data about how a student actually developed that piece of code.

To gain more insight into how students actually work, some instructors use a version-control system such as CVS [2] in their courses to track changes to students' code [5]. However, this approach suffers from a number of flaws. First, it's often difficult to compel students to use a version control system; many students don't want to be bothered with the extra overhead of learning an unfamiliar and potentially unintuitive system. Next, the students who choose to use a version control system will commit their files at different intervals, which can hinder or confuse efforts to reconstruct their work habits.

As instructors, we need fine-grained, comprehensive, quantitative data for each student over the entire lifecycle of a project: in order to improve the learning process for students, we need to know what kinds of problems they are having, how they are trying to address them, and what we can do to help.

Our Eclipse-based solution consists of two major components: the AutoCVS plugin and the SubmitServer[1].

The AutoCVS plugin commits a student's files to a central repository every time the file is saved. In addition, students submit their code to the SubmitServer which will run a battery of tests on the submitted code, returning feedback to the student. Thus, we provide automated feedback in both directions: from students to instructors through AutoCVS, and from instructors back to students through the SubmitServer.

[1] Actually, for scalability and maintenance purposes, the SubmitServer is split up into a SubmitServer and a BuildServer, but the distinction is an implementation detail

2 Implementation

We now discuss AutoCVS and SubmitServer in more detail.

2.1 AutoCVS

The Eclipse AutoCVS plugin is the backbone of our framework. AutoCVS maps Eclipse `IResourceChangeEvent`s to CVS commands in order to provide fine-grained snapshots of code during development. Adding or removing a file triggers a cvs add or cvs remove command, respectively, while cvs commits are triggered whenever a file is saved. It is possible that commits at this granularity will miss some activity between saves. For example, a student might add a large function but then delete it before saving the file. However, we believe snapshots at the granularity of save operations to be sufficient for gathering comprehensive data on how students develop code.

2.2 SubmitServer

The AutoCVS plugin provides each project with a "submit" menu option that tags the CVS repository with a timestamp, zips up that project's files, and submits them to the SubmitServer along with the generated timestamp.

The SubmitServer is connected to a database that records additional information about each submission, both to facilitate data analysis later and to provide a historical archive of project submissions that allow comparisons between semesters.

The SubmitServer works with BuildServers to build student submissions and record testing information about each submission. Testing feedback is broken down into three major parts: build tests, quick tests and release tests.

A build test simply checks whether the submitted code compiles. Quickly notifying students that their submitted version of a project does not compile can save a substantial amount of time and effort for both the student and the instructor or TA in charge of grading the submission.

The build server also runs JUnit tests on each submission. There are three categories of tests:

- quick tests - These tests are distributed to students. The results of quick tests are made immediately available to students, but the results should not come as a surprise since the students already have the tests.

- release tests - These test results are not made immediately available to students. However, a student can request "release testing".

  If a student requests release testing, they are told the number of tests that they passed and the number that they failed. They are also told the names of the first two tests that failed.

  There are restrictions on how frequently students may request release testing. Each student has 3 tokens for release testing, and performing release testing consumes a token. Once a token is used to perform release testing, it regenerates in 24 hours.

- secret tests - The results from these tests are not made available to students until after the project deadline. At the moment, we do not anticipate using secret tests in project grading, but the framework supports it

The restrictions on frequency of release testing are slightly baroque, but are carefully designed to give students the incentives to develop good study and coding habits. Students are encouraged to start working on a project early, because it will provide them with more opportunities to perform release testing. Students are encouraged to think carefully about whether or not their software is ready for release before asking for release testing, since doing so consumes a limited resource.

Many aspects of this can be modified or configured differently. For example, students can be given a different number of tokens, or tokens could take a different amount of time to regenerate. The names of the tests can be as cryptic or as edifying as desired.

The database will contain the result of each test on each submission and this information will be available to the instructor. For example, 24 hours before the project is due, an instructor can check how many students have passed each of the tests. This could be used to guide discussion in class if there are some tests that very few students have successfully passed. It might also help to identify faulty tests.

3 Discussion

In this section, we discuss the of the problems we hope to address in more detail.

3.1 Feedback to Instructors

As instructors, we would like better information about what kind of progress students are making on their projects, and where they are having trouble.

For example, if a student is programming by "Brownian Motion", what parts of the code is he

or she changing? Are students making mistakes that automated bug checking tools could find easily? If a student is in the lab all night, what is he or she doing? Once students have a working version of their code, how often do they test it? Can we determine whether the major impediments to successfully completing a programming assignment are primarily confusion over core concepts, implementation details, or something else?

Our Eclipse framework will give instructors better information in two ways. First, having detailed revision histories for each student will allow instructors to find out where students are spending the most time. Second, as students use the submission system to request a release test, that gives instructors an indication that they have reached a milestone in the development cycle for their project.

3.2 Feedback to Students

One of the major advantages of AutoCVS is the ability of instructors to pull up charts and graphs of students' progress on the current programming project. Using this information, instructors can identify particularly troublesome issues in the assignment and guide lectures or discussion sections accordingly. Ideally, the instructors and TAs should be able to provide more targeted help as the submission deadline approaches based on their observations of students' progress on the project.

Another goal is to provide students with more interactive help on their programming assignments. Time spent with faculty and teaching assistants debugging code is invaluable. However, maximizing this sort of "quality" time for large numbers of students is often infeasible. Furthermore, the sampling of students that attend office hours is not always random; thus, the issues these students bring to the attention of instructors are not necessarily representative of the concerns of the class as a whole.

Students often submit a project confident that they completed the assignment only to find out days or weeks later when they receive their grade that they misunderstood a fundamental concept of the project's requirements. This problem is especially frustrating when a small amount of feedback could have alerted the student to the area of the assignment that needs attention.

The SubmitServer will provide students with more feedback on programming assignments as they're working on them than is typically available.

3.3 Tools and the Learning Process

The environment in which students learn to program is changing rapidly. The proliferation of powerful Integrated Development Environments (IDEs) such as Eclipse has fundamentally changed the way students write code. Instead of the traditional edit-compile-debug cycle, modern programming environments offer continuous feedback on the code being developed, as it is developed. For example, Eclipse offers immediate, real-time feedback about syntax errors and other problems in the code. Ideas such as continuous testing [7] and application of static bug checking tools [3] have shown promise in finding bugs earlier, and making programmers more productive.

We assume that more powerful IDEs enhance the learning process. However, there is evidence that having too many features available confuses introductory students. Plugins like Dr Java [6] fill this important niche by simplifying the IDE. What is the right balance between features and simplicity for introductory programming students?

Finally, the number of tools available to aid in programming has increased. In addition to pedagogical tools such as the DrJava plugin for Eclipse, there are code coverage plugins such as Clover, static bug checkers such as FindBugs, unit testing plugins for JUnit, and many more. What effects, if any, do these tools have on the learning process?

We need comprehensive, quantitative data on how students use these new tools and what effects they have on the learning process and student productivity. We hope that our snapshot and submission system will help us better understand these issues.

4 Results

[Note to reviewers: The Fall semester at the University of Maryland starts August 30, so we don't have any data yet. However, we successfully used an earlier version of our plugin, which supported the automatic CVS snapshots but not the extended project submission system, in Spring 2004. We will have data about the Fall 2004 semester, including instructor and student experiences, by the time of the workshop.]

5 Related Work

In [7], Saff and Ernst describe an experiment in which students were given access to software that continuously performed unit tests in the background as they completed two programming assignments. The experiment found that the feedback from continuous testing had a statistically significant positive effect on the likelihood that students would complete projects on time. With the auto-

matic testing feature of our SubmitServer, we hope that students will realize similar benefits, although the testing feedback is more limited and coarse-grained.

In [5], Liu et al analyze CVS repositories and student self and peer evaluations for the semester project of a third-year programming course. Their work primarily focuses on understanding how teams of programmers learn to develop large software projects together. While their work dealt with students at the junior or senior level, our work focuses on students in their first or second semester of computer science. Also, their experiment found that the commit granularities differed widely between teams and between members of the same team. Because AutoCVS transparently commits file at every save operation, we believe we can standardize the commit granularity and achieve a more standardized view of the software development lifecycle.

JRefleX [8] is an Eclipse-based tool for supporting small student software teams, and in broad terms has similar objectives to the AutoCVS plugin. However, JRefleX focuses more on understanding how team members collaborate than on monitoring the working behavior of individual students. Also, JRefleX is based on passive collection, analysis, and presentation of data from the development process. In our work we employ active feedback techniques to help steer students towards successful completion of their projects.

BlueJ [4] is an educational IDE designed for introductory programming. It uses an interactive, objects-first approach, and uses visual class diagrams to help students understand object-oriented concepts. It includes a built-in project submission system supporting a variety of submission mechanisms. In our work we have made the submission system interactive, allowing students to get automatic feedback on their work early in the development process.

In [1], Adrianoff et. al. describe an implementation of a testing-based environment for programming contests. Students participating in a programming contest are given test classes, which are similar to JUnit tests. An Eclipse plugin allows them to run the tests, in order to find out how close their work is to meeting the requirements of the problem. The automatic testing feature of our submit server is very similar, although our system is centrally controlled, so that requests for system tests are logged, and instructors can choose how much or little information is revealed about the tests performed.

6 Future Work

Our system will be used for CS I and CS II at Maryland in Fall 2004. The first order of business will be to analyze the data collected and begin to answer the questions posed in this paper.

We have a "launch logging" mechanism that hooks into the `LaunchManager` and records when students run or debug their code. Unfortunately this feature is not yet stable enough to unleash on 300 introductory students. We plan to solidify and enhance this feature.

A major challenge for the future will be developing techniques for presenting, visualizing and analyzing the data captured by AutoCVS and SubmitServer.

We would like to answer the questions: What Java API methods are students calling? It would be interesting to see how students accomplish similar project requirements.

We would also like to compare the code coverage among the submissions of different students as well as the code coverage of the various versions of the software as it develops.

References

[1] S. K. Adrianoff, D. B. Levine, S. D. Gewand, and G. A. Heissenberger. A testing-based framework for programming contests. In *Proceedings of the Eclipse Technology Exchange Workshop*, Anaheim, CA, USA, October 2003.

[2] CVS. http://www.cvshome.org, 2004.

[3] D. Hovemeyer and W. Pugh. Finding bugs is easy. *SIGPLAN Notices*, December 2004.

[4] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg. The BlueJ system and its pedagogy. *Journal of Computer Science Education*, 13(4), December 2003.

[5] Y. Liu, E. Stroulia, K. Wong, and D. German. Using CVS historical information to understand how students develop software. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburgh, Scotland, May 2004.

[6] C. Reis and R. Cartwright. A friendly face for eclipse. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 25–29. ACM Press, 2003.

[7] D. Saff and M. D. Ernst. An experimental evaluation of continuous testing during development. In *ISSTA 2004, Proceedings of the 2004 International Symposium on Software Testing and Analysis*, pages 76–85, Boston, MA, USA, July 12–14, 2004.

[8] K. Wong, W. Blanchet, Y. Liu, C. Schofield, E. Stroulia, and Z. Xing. JRefleX: Towards supporting small student software teams. In *Proceedings of the Eclipse Technology Exchange Workshop*, Anaheim, CA, USA, October 2003.