Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses

Jaime Spacco*, David Hovemeyer†, William Pugh*, Fawzi Emad*,
Jeffrey K. Hollingsworth*, and Nelson Padua-Perez*

*Dept. of Computer Science
A. V. Williams Building
University of Maryland
College Park, MD 20742 USA

†Dept. of Computer Science
Vassar College
124 Raymond Ave.
Poughkeepsie, NY 12604 USA

{jspacco,pugh,fpe,hollings,nelson}@cs.umd.edu

hovemeyer@cs.vassar.edu

ABSTRACT

We developed Marmoset, an automated submission and testing system, to explore techniques to provide improved feedback to both students and instructors as students work on programming assignments, and to collect data to perform detailed research on the development processes of students.

To address the issue of feedback, Marmoset provides students with limited access to the results of the instructor's private test cases using a novel token-based incentive system. This both encourages students to start their work early and to think critically about their work. Because students submit early, instructors can monitor all students' progress on test cases, helping identify challenging or ambiguous test cases early in order to update the project specification or devote additional time in lecture or lab sessions to the difficult test cases.

To study and better understand the development process of students, Marmoset can be configured to transparently capture snapshots to a central repository everytime students save their files. These detailed development histories offer a unique, detailed perspective of each student's progress on a programming assignment, from the first line of code written and saved all the way through the final edit before the final submission. This type of data has proven extremely valuable many uses, such as mining new bug patterns and evaluating existing bug-finding tools.

In this paper, we describe our initial experiences using Marmoset in several introductory computer science courses, from the perspectives of both instructors and students. We also describe some initial research results from analyzing the student snapshot database.

ITiCSE'06, June 26–28, 2006, Bologna, Italy.

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: Computer-assisted instruction (CAI); K.3.2 [**Computer and Information Science Education**]: Computer Science Education; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Experimentation, Human Factors, Measurement

Keywords

Testing, project submission, code snapshots

1. INTRODUCTION

In project-based programming courses, communication between students and instructors is a valuable and scarce commodity. Students benefit from feedback about how well they are progressing toward meeting the project objectives. Instructors need feedback about where in projects students are having problems in order to make the best use of time in lectures, discussion sections, and office hours. Unfortunately, the amount of time instructors can spend interacting with students directly is limited. Due to large enrollments, finding time to help students can be especially challenging in introductory programming courses. The problem is compounded by the fact that many students wait too long before starting assigned projects, or flounder because they have hit a conceptual roadblock they can't steer around. Such students are at a high risk for dropping or failing the course.

Automation can help address these problems. Providing controlled feedback to students about their progress encourages them to start work early and think critically about their work. This feedback can also direct students towards areas of their project which need attention, and help them ask the right questions in discussion sections and office hours. Providing feedback about student progress to instructors helps them decide how best to use time in lectures and office hours.

We have implemented a system called MARMOSET to support automatic project submission and testing, and provide students, instructors and researchers feedback about stu-

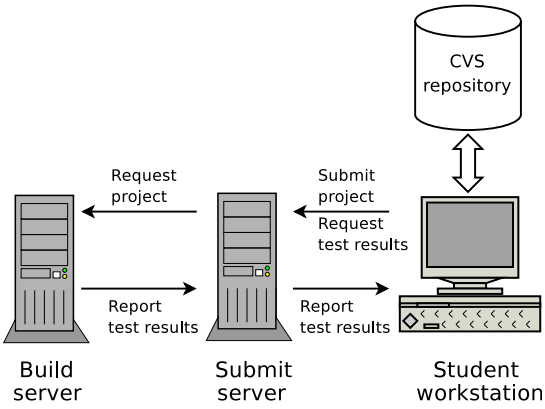

Figure 1: Architecture of MARMOSET

dent work on the programmign projects. Automatic testing of student projects is not a new idea [9, 3, 4, 1, 7, 10], nor is using version control repositories to record a history of student work [6]. However, we believe that our approach is different from previous systems in two important ways. First, MARMOSET uses a novel technique to reward students for starting early and thinking critically about their projects. Second, we are not aware of any other teaching environments that capture fine-grained snapshots of student work. These snapshots help instructors understand students' work habits.

In this paper, we describe experiences from using MARMOSET in several introductory programming courses. We have found it to be valuable for both students and instructors, and the student snapshot data we have collected has increased our understanding of typical student programming errors.

2. DESIGN AND FEATURES

MARMOSET is a programming project submission and testing system (with optional components described in Sections 2.4 and 2.5). Figure 1 shows the architecture of the system. Students can submit their projects either by using a command line tool or by using a menu option within an IDE such as Eclipse (the menu option being provided by a plugin). The system that receives these submissions is referred to as the *submit server*.

The submit server hands the submission off to a separate *build server* for building and testing, and a *test setup* for the project describes how the project should be tested. The test setup contains all of the auxiliary resources needed to build the project and execute the tests, such as libraries, data files, and compiler directives. Because the build servers run independently of the submit server, many build servers can be configured to provide better throughput and redundancy.

The submit and build server can execute and record the results of four different kinds of test cases:

- *Student tests*: test cases written by students
- *Public tests*: test cases provided to students
- *Release tests*: the instructor's confidential test cases, with results selectively made available to students before the project deadline
- *Secret tests*: additional confidential instructor test cases, with results not disclosed to students until after the project deadline

Once a project has been compiled and tested by MARMOSET, students have immediate access to the results of the student and public test cases. Since students have direct access to these test cases, the results should not come as a surprise. However, if students have written code in a way that is platform dependent, this can be caught by observing that the student and public tests on the submit server yield different results than the student sees when they test their software.

Secret tests are the traditional bulwark of programming project grading. Students don't see anything about the results of running secret test against their submission until after the deadline.

2.1 Release Tests

Release tests are one of the more unique and innovative features of the MARMOSET. If a student submission passes all of the public tests, then the student has the option of *release testing* that submission.

If a student requests release testing of a submission, the system reveals to the student only the number of release tests the submission failed and the names of the first two failed release tests (if any). For example, for a Poker game project, students might be told that their submission failed 4 release tests, and that the names of the first two failed test cases are `testFullHouse` and `testFourOfAKind`.

Students are limited in how many release tests they can perform. Each release test consumes a *release token*, which eventually regenerates. The parameters can be configured on a per-project basis, but in the standard configuration students have 3 tokens, each of which regenerates 24 hours after being used. This configuration allows a student to perform 3 release tests of her project during a 24 hour period.

2.2 Instructional monitoring

The instructional staff can monitor student progress on a programming assignment at any time. For example, instructors can access views such as a table detailing how many students have passed each of the test cases, a list of the best submission for each student, a list of students who have not yet submitted anything, and so on. Instructors can also drill down into a particular student's submission, revealing the details of a particular test case. This feature helps instructors tailor lectures and office hours towards concepts that students are having trouble with.

2.3 Build and Test Features

MARMOSET handles both Java and makefile-based projects. The makefile-based projects can handle any programming language that can be compiled and tested using the `make` [8] utility. We have used MARMOSET with projects in C, Ruby, and Objective Caml.

For Java projects, MARMOSET implements a number of Java-specific features. Java test cases are specified using JUnit [5] and are run under a security manager. The security manager allows relatively fine-grained control of what student code is and is not allow to do at runtime; we use it to prevent student code from doing things like launching new processes, opening network connections which might compromise test implementations or data files, etc. We also run FindBugs (a static analysis tool that looks for Java coding defects) on each submission and report the warnings we be-

lieve appropriate back to students. Finally, we use Clover, a code coverage tool provided under an academic license, to collect code coverage data from the test runs on Java projects.

Non-java projects are run as an unprivileged user, which substantially limits the damage that could be done, accidently or malicously, by student code.

2.4 Automatic CVS synchronization

An optional component of the MARMOSET Eclipse plugin is automatic CVS synchronization. If enabled, this feature performs an automatic CVS update every time Eclipse is started and an automatic CVS commit on every save. The Eclipse plugin is intended for use only on single student projects, where the same student can commit without fear of their changes conflicting with anyone else's. The intent is to transparently keep a student's files synchronized with a CVS repository stored on a central server, regardless of whether the student is working on their project in their dorm room, on their laptop, or at a university computer lab.

2.5 Research Data Collection

In most of the courses using MARMOSET we also ask students to participate in a research study. Participation allows us to study anonymized versions of data from their programming programs for research purposes. We ask participating students to fill out an optional demographic survey, but other than the optional survey, students in the research study do not experience the course any differently than students who do not participate in the study.

When Automatic CVS synchronization is used in a course from which we are collecting research data, we get snapshots of student code at the granularity of every save of every file. We collect one additional bit of data for research purposes, which is to record which snapshots were actually run or debugged. This allows us to distinguish frequent saves being performed to ensure no work is lost from frequent saves being performed as part of a edit-compile-run cycle.

3. INSTRUCTOR EXPERIENCE

As instructors, we had a number of expectations and goals for the MARMOSET system:

- By making release tests a limited resource, we hope to encourage students to both start projects early (and therefore have more opportunities for release testing), and think about their code carefully and test their code thoroughly before expending a precious release token.
- By encouraging students to submit projects early, our system can provide instructors with feedback about whether test cases are testing the desired material early enough for the project to be adjusted as necessary.
- Even projects that do not use release testing should benefit from MARMOSET because the detailed feedback provided well in advance of the project deadline allows instructors to adjust the test cases early enough to give students their grades hours—not days or weeks—after the deadline.
- We hope MARMOSET will provide a framework around which we can design research studies that focus on what steps students take—and the mistakes they make—as they develop their programming assignments.

Question	1	2	3	4	5	NA
Q1	0	5	12	32	21	0
Q2	5	3	9	14	39	0
Q3	4	13	6	31	16	0
Q4	3	10	8	34	15	0
Q5	6	20	20	19	5	0
Q6	10	7	10	17	25	1

Table 1: Student Survey Results

In this section, we present narratives of our experiences using MARMOSET in several courses, and discuss how well it met our expectations.

3.1 William Pugh, Object Oriented Programming II

As one of the lead PIs on the MARMOSET project, I've been an enthusiastic backer of the release testing approach. However, there were a number of things I didn't anticipate.

One point is that using MARMOSET really front-loads the programming assignment development process. You really have to get the project entirely done, including the test cases, well in advance of handing it out to students. This is something we should do even with standard project grading techniques, but MARMOSET really forces you to do it.

Initially we had a naïve assumption that once you finished designing the project and made it available to students, you wouldn't have to make any more changes to the test cases. What we found is that despite our best efforts, we often have to update the test cases after the project is made available to students, and we had to add an entire work flow system to MARMOSET for loading and validating new test setups before they are made available to students.

I think that MARMOSET has encouraged some students to start work earlier than they might have otherwise. However, some students are very stubborn and it is very difficult to get them to develop good work habits. The MARMOSET system allows us to see, for example, which students haven't made submissions for a project 48 or 24 hours before the project deadline. I've tried to talk to some of those students about their study habits, but I also worry about creating a big brother atmosphere.

Before the use of MARMOSET, when programming assignments were not tested against the instructors test data until after the programming assignment, the project descriptions were very detailed and tried to contain every detail needed to implement the projects correctly. With release testing, things do not need to be specified at this level of detail, since students get a chance to test their interpretations of the project description against the release tests. There has been some discussion among the instructors as to whether having less detailed specifications is a good idea. On one hand, extremely detailed specifications are good and useful. On the other hand, having specifications that are potentially ambiguous in places and getting feedback from users (e.g, release tests) is more representative of the real world.

3.2 Jeffrey Hollingsworth, Introduction to Low-Level Programming Concepts

MARMOSET allows TAs to spend their time helping students and evaluating code for style rather than doing testing of programs. The key to effectively using it is to have all

of the tests cases done and ready at least a week before the project is handed out. This gives the TAs a chance to try the projects and debug any confusion about the rules for what will be tested. The main challenge in developing assignments to work well with MARMOSET is ensuring that the test harness for each test case will handle any valid implementation of the project specification, not just the reference copy created by the instructor. Having the TAs test the assignment before the students helped greatly with this.

The overall implementation is very flexible and allowed us to use projects where the "tests" were quite different. For example, borrowing an idea from Edwards [1], one programming assignment involved the students writing test cases for an API we provided. Using several implementations of the API each seeded with a different bug, we were able to use MARMOSET to evaluate the effectiveness of the students' test cases.

3.3 Nelson Padua-Perez, Object-Oriented Programming I and II

My experience with MARMOSET has been a positive one. Although it requires some up front effort to set the environment, the benefits outweighs the initial effort. Among the main advantages of the system we can mention:

- MARMOSET promotes better projects. MARMOSET forces you to think about the grading process, as you consider a project idea. Thinking how to grade a project as you design a project, promotes a clear definition of project objectives, and how those objectives will be verified during the testing phase.
- MARMOSET improves the student's learning experience. By monitoring students' tests a teacher can identify concepts students seem to have problems understanding. These concepts can be revisited in lecture even before a project is due.
- MARMOSET improves the quality of the grading process. By generating automatic tests results, teachers can now devote more time to code style and documentation.

3.4 Fawzi Emad, Object-Oriented Programming I and II

Before using MARMOSET, I always had to make difficult choices about whether or not to make project test cases publicly available to the students. While public tests provide students with valuable feedback, students sometimes rely too heavily upon them, resulting in "trial-and-error" programming. The alternative, "secret" tests that are completely hidden from the students, requires students to think hard about different input cases, encouraging them to plan more carefully and work harder to anticipate various possibilities. The obvious downside to "secret" testing is that students receive no feedback at all about how their project is performing until after it is too late for them to make changes.

In contrast, "release tests" give students enough feedback to keep them motivated, but not so much feedback that they resort to a trial-and-error approach. I have personally found the release testing concept to be extraordinarily valuable, as my students are starting projects earlier, are working harder to produce a finished product, and are still learning to think for themselves about complex input cases.

The ability to obtain a quick visual summary of the rates of success that students are having with various test cases at any given moment has also been very useful. In any situation where most of my students are failing a particular test case, I know that either something is wrong with my test, or that I need to go back to the classroom to re-visit a concept.

4. STUDENT EXPERIENCE

Directly assessing the pedagogical impact of MARMOSET has been a challenge. One possibility would be a controlled experiment: however, it would be very difficult ethically to allow some students in a course to use the system while denying it to others. Another possibility would be to compare different semesters of the same course before and after the introduction of MARMOSET. We were not able to perform such a comparison because the initial deployment of MARMOSET at the University of Maryland coincided with a major restructuring of the introductory curriculum that changed the language used in the first two semesters from C/C++ to Java, and introduced an entirely new sequence of projects.[1]

Although a direct assessment of MARMOSET's impact on student achievement has not possible so far, we did want to gain some understanding of how students perceived the system, and whether or not they felt it enhanced their experience or detracted from it. To this end, we conducted a survey that asked the following questions:

Q1 Is your overall impression positive? (1=negative, 5=positive)

Q2 Do you prefer release testing over traditional post-deadline testing? (1=post-deadline, 5=release)

Q3 Were you able to make good use of feedback from release tests? (1=no, 5=yes)

Q4 Did release testing encourage you to start projects earlier that you might have otherwise? (1=no, 5=yes)

Q5 Did release testing make you feel more relaxed and confident (or, conversely, more tense and unsure)? (1=tense and unsure, 5=relaxed and confident)

Q6 For projects with secret test cases, did you keep working after you had passed all of the release tests? (1=no, 5=yes)

The students who took the survey had used MARMOSET in Object-Oriented Programming I, and were currently enrolled in Object-Oriented Programming II. We solicited responses on a scale from 1 to 5, where 1 is the least positive outcome, 3 is neutral, and 5 is the most positive outcome.

The survey results are shown in Table 1. In general, students had a positive impression of the system. According to the student responses, the feedback from the release tests was useful, and were a motivation to start work early. Surprisingly, even given the positive reaction to the system overall, the students were split evenly on the question of whether or not release tests increased their confidence. We speculate that some students found themselves in situations where their projects failed one or more release tests, but

[1]In the future, we may be able to make such a comparison by using MARMOSET in a different course without making other simultaneous curriculum changes.

they either did not understand the reason for the failure, or they did not understand how to fix the problem.

5. RESEARCH EXPERIENCE

As mentioned earlier, in most courses in which MARMOSET is used, we invite students to participate in a research study where anonymized versions of their project snapshots are recorded in a database. The volume of data we have collected to date is remarkable. For example, in a single semester of Object-Oriented II, we collected over 30,000 unique, compilable snapshots of student code. In these snapshots, we recorded approximately 400,000 unit test results, approximately 50,000 of which were failures due to an easily recognizable runtime exception, such as null pointer exceptions and class cast exceptions. We used the runtime exception data to measure the accuracy of the FindBugs static analysis tool at detecting the underlying bugs for several types of runtime exceptions. By identifying cases that were *not* identified by FindBugs, we were able to improve the accuracy of the static analysis to find more bugs with fewer false positives. For example, after improvements to the detector for bad type casts, FindBugs was able to find between 8% and 30% of all class cast exceptions in student submissions for two programming projects.

Even after this initial success, we still feel that we have only scratched the surface of the data collected, and that trying to extract all the interesting information from the data is like trying to drink from a fire-hose.

6. RELATED WORK

Many systems exist to automatically collect and test student submissions: some examples are [9, 3, 4, 1, 7, 10]. Our contribution is to control students' access to information about test results in a way that provides incentives to adopt good programming habits.

In [6], Liu et. al. study CVS histories of students working on a team project to better understand both the behavior of individual students and team interactions. They found that both good and bad coding practices had characteristic ways of manifesting in the CVS history. Our goals for the data we collect with our automatic code snapshot system are similar, although we consider individual students rather than teams. Our system has the advantage of capturing changes at a finer granularity: file modification, rather than explicit commit.

In [1], Edwards presents a strong case for making unit testing a fundamental part of the Computer Science curriculum. In particular, he advocates requiring students to develop their own test cases for projects, using project solutions written by instructors (possibly containing known defects) to test the student tests.

Ellsworth et al describe Quiver [2], an automated QUIz VERification tool that provides a closed-lab environment where students complete a small programming assignment in a limited amount of time. Because Quiver was designed for an interactive, timed environment, it requires that students use a specific client machine and editor, does not provide limited feedback as in release tests, and does not capture fine-grained snapshots of student code.

Praktomat [12] is an automated testing and code review system for introductory students that allows students to submit versions of their code, view and provide feedback on other students' code, receive feedback on their own code, and then re-submit their code. The data collected by Praktomat suggests that students benefited greatly from providing and receiving feedback on their code; however, the added overhead of detecting or preventing plagiarism in such a system is a major drawback.

The Environment for Learning to Program (ELP) [11] is a static analysis framework for grading student projects and providing feedback about code quality. ELP is a complementary static approach to our dynamic testing system.

7. REFERENCES

[1] S. H. Edwards. Rethinking computer science education from a test-first perspective. In *Companion of the 2003 ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Anaheim, CA, October 2003.

[2] C. C. Ellsworth, J. James B. Fenwick, and B. L. Kurtz. The quiver system. *SIGCSE Bull.*, 36(1):205–209, 2004.

[3] D. Jackson and M. Usher. Grading student programs using ASSYST. In *Proceedings of the 1997 SIGCSE Technical Symposium on Computer Science Education*, pages 335–339. ACM Press, 1997.

[4] E. L. Jones. Grading student programs - a software testing approach. In *Proceedings of the fourteenth annual consortium on Small Colleges Southeastern conference*, pages 185–192. The Consortium for Computing in Small Colleges, 2000.

[5] JUnit, testing resources for extreme programming. http://junit.org, 2004.

[6] 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.

[7] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 55–59, New York, NY, USA, 2002. ACM Press.

[8] A. Oram. *Managing Projects with Make*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1992.

[9] K. A. Reek. A software infrastructure to support introductory computer science courses. In *SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, pages 125–129, New York, NY, USA, 1996. ACM Press.

[10] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 133–136, New York, NY, USA, 2001. ACM Press.

[11] N. Truong, P. Roe, and P. Bancroft. Static analysis of students' java programs. In *Proceedings of the sixth conference on Australian computing education*, pages 317–325. Australian Computer Society, Inc., 2004.

[12] 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*, pages 89–92. ACM Press, 2000.