Mining the Data in Programming Assignments for Educational Research

Stephen H. Edwards and Vinh Ly
Dept. of Computer Science, Virginia Tech
660 McBryde Hall (0106)
Blacksburg, VA 24061, USA
+1 540 231 5723

edwards@cs.vt.edu, vinhly@vt.edu

ABSTRACT

In computer science and information technology education, instructors often use electronic tools to collect, compile, execute, and analyze student assignments. The assessment results produced by these tools provide a large body of data about student work habits, the quality of student work, and the areas where students are struggling. This paper reports on efforts to extract significantly more useful data from electronically collected assignments in computer programming courses. The work is being performed in the context of the most widely used open-source automated grading system: Web-CAT. We have enhanced a Web-CAT plug-in to allow collection of data about the frequency and types of run-time errors produced by students, the frequency and types of test case failures that occur during grading, basic code size metrics, test coverage metrics, and more. This information can be combined with the results of "by-hand" grading activities to form a large, rich data corpus characterizing student behavior over many assignments in one course, over many courses, and even across semesters. The data collected in this way is a valuable resource for researchers in computer science education.

Keywords: on-line education, computer science, automated grading, Web-CAT, data mining

1. INTRODUCTION

In computer science and information technology education, instructors often use electronic tools to collect and process student work. This is particularly true for programming assignments, which can be compiled, executed, and analyzed in a variety of ways. Many tools for automatically grading programming assignments exist, and the assessment results produced by these tools provide a large body of data about student work habits, the quality of student work, and the areas where students are struggling. Unfortunately, most such data goes unused, once it has been reduced to a single number: the student's assignment score.

This paper reports on efforts to extract significantly more useful data from electronically collected assignments in computer programming courses. The work is being performed in the context of the most widely used open-source automated grading system: Web-CAT. Web-CAT provides a plug-in mechanism allowing instructors to provide any sequence of customized grading, feedback generation, and data collection actions desired. We have enhanced existing plug-ins to allow collection of data about the frequency and types of run-time errors produced by students, the frequency and types of test case failures that occur during grading, basic code size metrics, test coverage metrics, and more. This information can be combined with data on "by-hand" grading results from instructors or teaching assistants (also collected electronically) to form a large, rich data corpus characterizing student behavior over many assignments in one course, over many courses, and even across semesters.

The data collected in this way is a valuable resource for researchers in computer science education. Collecting raw results for thousands or even tens of thousands of student program assignments provides access to a spectrum of measures that give broader insight into the quality and performance of student work. The resulting data permit the investigation of a wide range of educational research questions, including what kinds of errors students encounter most, what assignments are poor discriminators among student learning levels, and what course learning objectives are being met. Existing work has been carried out for programming assignments written in C++, but the techniques are applicable in other languages, including Java. The data collection approach described here is **low-cost and low-effort**, because it automates the most tedious, repetitive, and taxing aspects of collecting fine-grained data on student deliverables.

2. BACKGROUND

While many automated grading tools have been developed by various institutions over the past several decades, most have seen only localized use. One existing automated grading system has begun to see more wide-spread use: Web-CAT, the Web-based Center for Automated Testing [2][3][4]. As the only automated grading system to focus on assessing student testing performance, Web-CAT is used by nine separate institutions, with a growing user community. The Web-CAT server at Virginia Tech alone has processed over 186 thousand program submissions by 2942 students in 119 course sections since 2003. Web-CAT is available as an open-source project on SourceForge [5].

Other automated grading systems typically focus on assessing whether or not student code produces the correct output. Web-CAT, on the other hand, is typically used in a way that focuses on assessing the student's performance at testing his or her own code, and on generating concrete, directed feedback to help the student learn and improve. Such a tool allows educators to give assignments that require test suites to be submitted along with

code. Ideally, students should be able to "try out" their code-in-progress together with their tests early and often, getting timely feedback each time.

At the same time, however, Web-CAT has been engineered to support arbitrary instructor-provided plug-ins for processing and assessing student work, so virtually any grading scheme or strategy you can devise can be implemented without modifying the underlying system itself. Administrators can upload new plug-ins over the web and publish them for instructors to use. Instructors can even write their own plug-ins off-line and then upload them via Web-CAT's web interface. Such plug-ins require no code changes to the server, and are immediately available for use without an application restart.

Web-CAT's plug-in architecture provides a great deal of flexibility. The most commonly used grading plug-ins currently available are for processing Java or C++ assignments where students write their own software tests. In order to provide appropriate assessment of testing performance and appropriate incentive to improve, these plug-ins use a scoring strategy that does more than just give some sort of "correctness" score for the student's code. In addition, Web-CAT assesses the validity and the completeness of the student's tests.

3. COLLECTING FINE-GRAINED DATA

The data collection approach presented here is low-cost and low-effort, because it automates the most tedious, repetitive, and taxing aspects of collecting fine-grained data on student deliverables. Educators are keenly aware of the costs associated with various assessment approaches, which is often an important factor when selecting tools to use [46].

Further, the data collection mechanism is **always on**, in the sense that it continuously collects all grading feedback provided by course staff, for every student, for every assignment, for every course that is using the system. The data collected includes run-time errors produced by students, as well as a number of automatically measurable features of their programs, including code size, proportion of required behavior that is implemented correctly, and thoroughness of testing when students are required to test their own code, among others. This data can be archived from semester to semester over a multi-year evaluation cycle. This approach also will make it straightforward to build in regular (once per semester or once per year) reporting of performance summaries that are driven directly by instructor rubrics—and thus tied directly to the outcomes that are being assessed. These benefits will be obtained without requiring any extra data collection, recording, or reporting responsibilities of faculty members. Instead, the relatively high-cost effort of continuous collection is fully automated, and the net result is a smaller up-front cost for faculty, who must choose outcomes ahead of time and design grading schemes that speak to these outcomes.

3.1 Collecting Student Submission Data

Web-CAT uses plug-ins to process student submissions. The most popular plug-ins handle submissions of Java and C++ assignments. Details on the Java plug-in are available elsewhere [3][5]. As a result, this paper focuses on enhancements to the C++ plug-in—the Java plug-in features are similar.

We have enhanced both plug-ins to support the collection about a variety of data available when each student submission is processed. Some of this data is routinely provided by the compiler, including the number and types of compilation errors. Other data is collected during behavioral analysis, when student-written tests are executed against the student's code, or when instructor-provided reference tests are executed against the student's code. Finally, we have added additional static dynamic analysis tools to collect further data.

3.2 Code Coverage

In the effort of collecting data that are more useful to characterize student submissions, we have integrated BullseyeCoverage, a coverage analyzer for C++ and C, into Web-CAT's C++ plug-in. This tool collects code coverage data that includes method, condition, and decision coverage measures. The resulting information explicitly characterizes which parts of a student's submission have been exercised by the student's own software tests. This provides a comprehensive view of the overall execution percentage of methods, conditions, and decisions for each source file. The tool also produces a detailed source code report of which lines were not executed and why.

Function coverage is the measurement of the degree to which all methods are invoked when running a given set of tests. This information is useful for students since it shows an overall picture of wherever every method is executed when students run their own tests. Depends on course policy, this could prevent any unforeseen bugs in student submissions because of lacking of testing all functional requirements by encouraging more testing.

Modified condition/decision coverage (MCDC) is a hybrid measure based in part on condition coverage and decision coverage. Condition coverage is a measurement of the true-or-faluse outcome true of each Boolean expression or sub-expression. Sub-expression is one of multiple nested sub-expressions separated by logic-and, logic-or in one expression. Decision coverage is a measurement of all result flows of Boolean expressions tested in control structures (such as if statements, while statements, switches, and so on).

Instructors using this plug-in can choose how stringently to grade students, picking the desired level of coverage they wish to use for scoring. Regardless of the instructor's choice for grading, however, all measures are collected for analysis purposes. Web-CAT also shows these results to students, in the form of a color-coded, syntax-highlighted, web-viewable printout that explicitly points out which parts of their code are not tested as well as possible, with informative comments about why. Students can quickly scan this view to locate the non-executed parts of their solution in order to improve their own testing. The purpose of this is to help instructors to judge student submissions and students who can learn from their testing experiences and improve their code quality.

In addition to helping instructors to grade student submissions, Web-CAT also assists students to identify what they miss. Graphical user-friendly diagrams are used to represent percentage results. In addition to those diagrams, a friendly feedback system, which is based on the locations of uncovered functions and conditions/decision, is integrated into the source code pages. Those uncovered parts will be highlighted in the source code. Furthermore, a useful feedback will be provided. For function coverage, feedback is a note about what parts are not executed when running tests. For condition/decision coverage, feedback shows what Boolean result lacks of testing. If there are nested conditions, the feedback will include details for each condition's coverage as well as for the overall coverage of the nested conditions.

3.3 Error Code and Stack Trace

Finally, in addition to code coverage data, we have also enhanced the C++ plug-in to track carefully the results of each test run executed on student-provided code. Results from instructor-provided reference tests are logged in detail for analysis and later reporting. This logging provides additional useful information on the type of errors that students make in their submissions. From what we design and implement for the CxxTest infrastructure, we can identify a large ranges of errors. These errors range from memory errors, run-time errors, to assertion errors made in the student's own tests. Table 1 contains a list of what type of errors that we can collect.

With the error codes recorded during assignment processing, instructions are provided with powerful information to grade student submissions. Furthermore, they can build statistical data to evaluate, adjust, and improve the course curriculum by identify students' weakness through these error codes.

Giving back error codes and an explanation of the codes is useful. It, however, would be more beneficial to students, if we can show them where those errors are from. As a result, we have

Category	Error Code	Meaning
Basic	0	General Failure
	1	Pass
Memory Error	0	Called delete on array pointer (should use delete[])
	1	Called delete[] on non-array pointer (should use delete)
	2	Freed uninitialized pointer
	3	Freed memory that was already freed
	4	Dereferenced uninitialized pointer
	5	Dereferenced null pointer
	6	Dereferenced freed memory
	7	Checked pointers cannot be used with memory not allocated with new or new[]
	8	Memory leak caused by last valid pointer to memory block going out of scope
	9	Memory leak caused by last valid pointer to memory block being overwritten
	10	Comparison with a dead pointer may result in unpredictable behavior
	11	Indexed a non-array pointer
	12	Invalid array index (%d); valid indices are [0..%lu]
	13	Deleted pointer that was not dynamically allocated
	14	Memory before/after block was corrupted; likely invalid array indexing or pointer arithmetic
FailedAssert Error	0	TS_ASSERT(expr)
	1	TS_ASSERT_EQUALS(x, y)
	2	TS_ASSERT_SAME_DATA(x, y, size)
	3	TS_ASSERT_DELTA(x, y, d)
	4	TS_ASSERT_DIFFERS(x, y)
	5	TS_ASSERT_LESS_THAN(x, y)
	6	TS_ASSERT_LESS_THAN_EQUALS(x, y)
	7	TS_ASSERT_PREDICATE(R, x)
	8	TS_ASSERT_RELATION(R, x, y)
	9	TS_ASSERT_THROWS(expr, type)
	10	TS_ASSERT_THROWS_NOTHING(expr)
Run-Time Error	0	SIGFPE: floating point exception (div by zero?)
	1	SIGSEGV: segmentation fault (null pointer dereference?)
	2	SIGILL: illegal instruction
	3	SIGTRAP: trace trap
	4	SIGEMT: EMT instruction
	5	SIGBUS: bus error
	6	SIGSYS: bad argument to system call
	7	SIGABRT: execution aborted
	8	run-time exception

Table 1: Classification of errors that may occur while testing student-written C++ software.

developed a built-in stack trace infrastructure that can trace back the last execution line and previous calls when the errors occur. This provides a tremendous amount of information for C++ students who do not have a built-in stack trace like Java students.

For many of these errors, it is hard for students to identify the sources by themselves. There are some programs out there to identify some of these violations. However, it requires students to do many extra steps and learn how to use many new tools. This new plug-in is an attempt to ease the process and to help students accessing information that they cannot find by themselves easily.

For students, the error codes and stack trace is a tremendously useful resource for them to be able to identify errors in their code, to trace back and fix it. The most valuable thing is that students can learn from their own mistakes and improve their code quality and robustness.

4. RESULTS FROM HAND-GRADING

Course personnel typically still grade at least some aspects of programming assignments "by hand", even when using an automated grading system to score other aspects of an assignment. We expect this practice to continue. However, we can leverage existing technology to support this process, increasing speed and consistency. Currently, Web-CAT already provides course staff with the ability to mark up student work by hand on-line, using just a web browser [9][10]. This capability also provides the crucial hook needed for automated data collection for outcomes-based assessment.

While different grading strategies can be used on programming assignments, rubrics are particularly synergistic with objectives-based course assessment. By rubric, we mean a clearly defined set of guidelines that are used in assessing achievement by observing student performance [1][6]. A rubric typically defines several categories or levels of performance, often over a range from unacceptable to exceptional or exemplary. For each level of performance, the rubric clearly defines the specific behaviors or traits that demonstrate achievement of that level. Powell *et al.* discusses the value of rubrics in computer science for promoting consistency and streamlining the grading process [7]. If we consider two hypothetical learning objectives for a typical computer science course, we might use a rubric like the one shown in Table 2. Presumably, these course-level learning objectives also would be related to particular program-level outcomes. As a result, one can directly tie achievement of well-chosen course learning-objectives to program-level outcomes when desired.

We are currently in the process of extending Web-CAT's hand-grading support to include rubric-based grading. All instructor-written or TA-written comments on an assignment, together with rubric scores for every aspect of the grading criteria then will be automatically collected for later summarization, reporting, and analysis.

5. ASSESSING STUDENT LEARNING

To see how the information collected by Web-CAT as proposed here could be used for course assessment, consider the two learning objectives illustrated in the rubric of Table 2: students should be able to evaluate and validate the correctness of a programmatic solution, and be able to document programs clearly and effectively. For a course that included these objectives, the instructor might require students to write their own test cases for the code they write [2], and take advantage of Web-CAT's ability to evaluate how thoroughly students test their own work. If the course used Java, the instructor might also take advantage of the static analysis tools that Web-CAT uses to check

Objective	Unsatisfactory	Satisfactory	Good	Excellent
Evaluate and validate the correctness of a programmatic solution	No tests have been submitted by the student, or entire methods remain completely untested.	Written test cases are submitted along with the code. The test cases exercise each method of the design.	Test cases demonstrate a clear effort at comprehensive coverage of method behaviors in the corresponding code, including testing of boundary conditions and likely errors.	Student has written a comprehensive, professional quality set of test cases. In addition to boundary values and likely errors, each method is tested multiple times, including full branch-level test coverage.
Document programs clearly and effectively	No consistent or concerted effort at documentation.	All public classes and public methods have Javadoc descriptions, although some are incomplete. Internal commenting may be missing or inconsistent.	All public classes and public methods have complete Javadoc descriptions with no missing tags. Attempts have been made to describe key algorithmic decisions and tradeoffs internally	Professional-quality comments exist throughout, with complete and correct Javadoc descriptions and clear, concise writing. All tricky internal code is thoroughly and clearly documented. Rationale for design decisions is given.

Table 2: A sample rubric covering two course learning objectives.

conformance to stylistic and coding conventions. Finally, the instructor can use a comprehensive rubric that includes the two objectives shown in Table 2 for grading, instructing graders to categorize all of their free-form comments according to the corresponding facet of the rubric that applies. In this situation, Web-CAT will be able to provide the measurements shown in Table 3. This information would be available for each and every student, for each and every assignment across the course. Further, Table 3 only shows the subset of what has been collected that is relevant to the two objectives in this example; other objectives can be treated similarly. Instructors can choose the indicators most relevant to their personal goals, while assessment and accreditation committees can choose others. Web-CAT's plug-in architecture makes it easy for new measures to be dropped into place when necessary. Most importantly, however, the biggest value comes from Web-CAT's generalized reporting engine that allows various stakeholders to view summary statistics in tables and graphs that characterize aggregate data over all students, over all assignments, or progress over time. Finally, this rich and deep data collection can be archived and used for accreditation or program assessment later, without requiring any special collection actions or additional burden of the course instructor.

6. CONCLUSIONS AND FUTURE WORK

Collecting comprehensive, fine-grained information about student performance on course tasks is important for measuring the achievement of learning objectives. This level of information provides excellent detail and support for assessing a wide variety of technically-oriented outcomes [12]. Whitfield [11] suggests using this data for "*course embedded assessment*", where grading criteria for each assignment within a course are driven by the course's learning objectives, where course learning objectives are tied to degree outcomes, and where results from individual assignments can then be mapped systematically to program outcomes for assessment and accreditation. However, by-hand collection and collation of this information is expensive, time-consuming, and oppressive, yet individual instructors may see little value [46].

To address this problem, we have described basic extensions to Web-CAT plug-ins that allow for low-cost and low-effort collection of this data for student programming assignments. The approach can be used in a wide variety of computer science or information technology courses that employ programming activities. While we have focused our discussion on C++ assignments in this paper, the same techniques are equally applicable to other languages, including Java. This strategy leverages the infrastructure provided by an existing open-source automated grading tool, extending its capabilities for detailed data collection. Web-CAT also provides a general-purpose data reporting engine based on the popular open-source tool BIRT [birt].

Support for collecting code coverage measures, static code analysis measures, test case results, and instructor comments have all been implemented. We are in the process of adding explicit rubric support for hand-grading assignments, so that course staff can employ outcomes-driven rubrics as assessment tools and then collect and summarize the corresponding detailed comments to the same degree of granularity as other performance data.

While this basic infrastructure has proven useful for course-level exploration of student performance on individual assignments, the next step is to use this strategy across several courses to compile data useful in outcomes-based program-level assessment. Further, we hope to build a corpus of data on student performance in our own courses that can be used both to gauge the impact of educational changes and innovations and to further other research questions arising in our curriculum.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. DUE-0618663. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Objective	Indicator	Score
Evaluate and validate the correctness of a programmatic solution	Number of student-written test cases	17
	Number of methods covered by student tests	9 (100%)
	Number of statements covered by student tests	127 (100%)
	Number of conditionals covered by student tests	23 (84%)
	Number of student tests passed	17 (100%)
	Number of run-time errors produced	0
	Number of instructor-written reference tests passed	24 (96%)
	Number of instructor-written reference tests failed	1 (4%)
	Grader rating in rubric	Good
	Number of grader comments in rubric category (full text of all comments in this category available for qualitative analysis, if needed)	0 (0%)
Document programs clearly and effectively	Number of missing class comments	0 (0%)
	Number of missing public method comments	0 (0%)
	Number of missing required comment fields	2 (4%)
	Grader rating in rubric	Good
	Number of grader comments in rubric category (full text of all comments in this category available for qualitative analysis, if needed)	3 (38%)

Table 3: A sample of data available through Web-CAT for two course objectives.

REFERENCES

[1] Becker, K. Grading programming assignments using rubrics. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, ACM, 2003, pp. 253-253.

[2] Edwards, S. H. Improving student performance by evaluating how well students test their own programs. *Journal of Educational Resources in Computing*, 3(3):1-24, Sept. 2003.

[3] Edwards, S. H. Rethinking computer science education from a test-first perspective. In *Addendum to the 2003 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM, 2003, pp. 148-155.

[4] Edwards, S. H. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ACM, 2004, pp. 26-30.

[5] Edwards, S.H. Web-CAT Wiki, available at: http://web-cat.sourceforge.net/.

[6] McCauley, R. Rubrics as assessment guides. *ACM SIGCSE Bulletin,* 35(4): 17-18, Dec. 2003.

[7] A. Powell, S. Turner, M. Tungare, M.A. Pérez-Quiñones, and S.H. Edwards. An online teacher peer review system. In C. Crawford et al. (Eds.), *Proceedings of the Society for Information Technology and Teacher Education International Conference 2006*, AACE , pp. 126-133.

[46] Sanders, K.E., and McCartney, R. Program assessment tools in computer science: a report from the trenches. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ACM, 2003, pp. 31-35.

[9] Vastani, H. K. *Supporting Direct Markup and Evaluation of Students' Projects On-line*. Master's Thesis, Department of Computer Science, Virginia Tech, June 11, 2004, available at: http://scholar.lib.vt.edu/theses/available/etd-08172004-020310/

[10] Vastani, H., Edwards, S., Pérez-Quiñones, M.A. Supporting on-line direct markup and evaluation of students' projects. *Computers in Education Journal*, 16(3), July-Sept. 2006.

[11] Whitfield, D. From university wide outcomes to course embedded assessment of CS1. *Journal of Computing Sciences in Colleges*, 18(5): 210-220, May 2003.

[12] Winters, T. and Payne, T. What do students know?: an outcomes-based assessment system. In *Proceedings of the 2005 International Workshop on Computing Education Research*, ACM, 2005, pp. 165-172.