# Supporting On-line Direct Markup and Evaluation of Students' Projects

Hussein Vastani, Stephen H. Edwards, Manuel A. Pérez-Quiñones
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

## Abstract

Automated grading systems have been in use for several years. These systems automate part of the grading process by compiling, executing and testing student submitted source code. However, such systems often fail to include a mechanism to allow instructors or grader to provide free form comments on student work. One must resort to other methods to provide feedback to students.

This paper presents the development of a feedback mechanism that streamlines the grading process for instructors and teaching assistants. A web-based grading tool has been developed that allows course staff to enter comments for student programs directly through a web browser. This tool is tightly integrated with Web-CAT, an automated grader. The result is a one-stop web-based interface for students to receive all of their feedback.

We present the results of an anonymous survey of computer science professors from different universities on their expectations with respect to TA grading activities for programming assignments, as well as the learning outcomes these professors desire for their students. In addition, we present the results of interviews with teaching assistants in introductory programming level courses to learn about the different grading methods they use when grading programming assignments. Finally, we report on a usability evaluation of the tool itself and discuss directions for future work.

## Introduction

Automated grading systems have been in use in Computer Science education for several years. Numerous systems have been developed that automate the process of grading by compiling, executing and testing student submitted source code. However, such systems often fail to include support for free form comments provided by instructors or grading staff. Instead, instructors or teaching assistants have to resort to other methods to provide their feedback to the students.

This paper presents a web-based grading tool that allows course staff to enter comments on student programs directly through a web browser. This tool is tightly integrated with Web-CAT, an automated grader [5]. The result is a one-stop web-based interface where students receive all of their feedback. We present the results of an anonymous survey that was sent out to Computer Science professors to gather information on their grading practices when assessing programming assignments. In addition, we present the results of interviews with teaching assistants in introductory programming level courses to learn about the different grading methods they use when grading programming assignments, and the difficulties they face. Information gathered through these two channels served to form the requirements for our tool design. Finally, we present our tool and an initial evaluation of its functionality.

## Related Work

Feedback is an important part of teaching and learning. Instructors provide feedback to students to evaluate their work, to inform them of their mistakes and suggest corrections, and to help students improve their efforts. Feedback is usually provided verbally or in written form—either written by hand or typed using a computer. Price and Petre [17] compared the

nature and quality of assignment feedback provided on paper and in electronic form. Their results showed that the two methods are comparable. Using electronic marking as a medium of providing feedback did not impair students' or instructors' ability to communicate effectively. Price and Petre also showed that electronically managed assignments offer less administrative overhead and faster turnaround time.

**Web-CAT**

Many educators have used automated systems to assess and provide rapid feedback on student programming assignments (for a more in depth review of the literature see [20,4]). While these systems vary, they typically focus on compilation and execution of student programs against some form of instructor-provided test data. Virginia Tech is actively exploring an alternative approach where students write their own test cases and are graded in part on the quality of their own testing efforts. As a result, we have designed and implemented a general-purpose automated grading tool and incorporated it into Web-CAT, the Web-based Center for Automated Testing [5].

Web-CAT is a web-based application implemented using Apple's WebObjects framework [21]. It is designed to be language independent, and is currently used for grading program submissions in six languages, including Java and C++. For Java, it uses open-source tools such as Checkstyle [2] and PMD [14] to perform static analysis of coding and commenting style and to spot potential coding issues, and uses the commercial tool Clover [3] to instrument student code for coverage analysis.

In operation, students submit programming projects through a web interface, or directly through a plug-in within their IDE. The Web-CAT Grader compiles student code and tests together, executes all the student tests, optionally executes additional instructor-provided tests, and uses the results to assess the validity and completeness of student testing. Static analysis and stylistic checks are also performed when possible.

The reports produced by the various assessment tools are merged into one seamless source code markup report that is viewable on the web by the student. This unified feedback report shows overall summary information, including the score, compilation problems, and test run results, as well as detailed file-by-file feedback on the submission. The TA markup capability discussed in this paper is integrated into Web-CAT to support direct markup of code. The goal is to have instructor- or TA-provided comments presented to the student in the same unified feedback report, directly embedded in the color-coded source code view provided to students.

**Role of Automated Systems in the Grading Process**

Instructors teaching Computer Science courses, particularly introductory courses, often find themselves overburdened with work and do not have enough time to do a thorough assessment of students' programming assignments. As a solution, automated grading systems have been developed and have achieved some success in improving the grading of assignments. These systems undoubtedly provide additional benefits in terms of consistency, thoroughness and efficiency in assessing student programs [8]. Most support the electronic submission of student assignments [10,12,19], with many also supporting the automatic compilation and execution of student code against some instructor-provided tests or test data [8,18,9]. A smaller number of such systems also supports electronic markup of student code in some form [15,16,11,13] and the automatic return of results.

The biggest advantage of many automated tools is that they support the work of instructors and teaching assistants by automating various aspects of the grading process. They maintain consistency in grading and provide timely feedback to the students. The instructors and

teaching assistants can then spend their grading effort on deeper issues such as style, design and documentation.

Although such systems help instructors and teaching assistants, very few support actual markup of the student code. Some graders follow the usual practice of printing out the student code and making comments and point deductions on the paper copy itself, which is then handed back to the student. Others read student code by retrieving it electronically, and then make comments on a separate document that is then mailed back to the student. The efficiency of the grading process achieved by the use of automated systems is not maintained during the later, more important steps in grading, i.e. TA feedback and return of student assignments.

## Systems That Support Online Marking

The work of Popyack, Herrmann,Char, Zoski, Cera and Lass at Drexel University [15] is the most relevant to our work. They have developed a marking methodology that combines the electronic and paper grading approaches. They considered the advantages of both of these methods and present a solution that allows graders to write free-hand comments on a student's electronic submission using pen-based Tablet PCs. This style of feedback closely resembles traditional pen and paper grading and represents the final document in digital form for better archiving (as seen in Figure 1). They make use of Adobe Acrobat1 to markup documents.

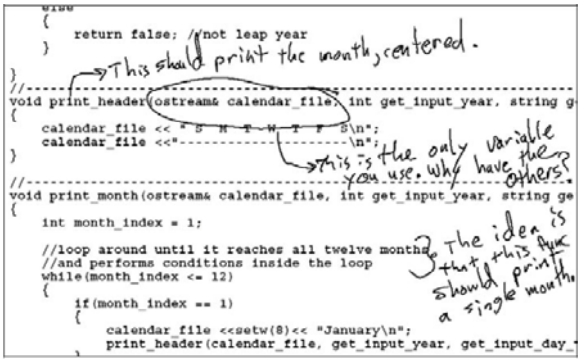


Figure 1. Electronic Markup in Popyack et al.

## Survey of CS Professors

In order to understand how Computer Science professors grade programming assignments, an online survey was prepared and announced on the mailing list of the Special Interest Group on Computer Science Education (SIGCSE) of the ACM. The survey goals were to elicit responses from professors about their grading practices, and to gather information about the expectations they have with respect to TA grading activities. Sixty two people responded. All participants had prior experience in grading student programming assignments.

## Results

On average, most graders (63%) spend 10 or more minutes on each student's assignment. Most (96%) provide 15 comments or less per student assignment. Overall, most professors were satisfied with the quality (27 agree, 16 strongly agree) and the quantity (33 agree, 13 strongly agree) of the feedback provided to students. A rubric was a common method used for grading assignments, and most respondents believe that the rubric is a useful tool to assess the student's work (34 agree, 16 strongly agree).

The survey also asked professors to rate the relative time (as a percentage) spent on providing programming assignment feedback in different categories: commenting, indentation, naming, design, control flow, correctness/bugs, and student testing. Almost 50% of the graders feel they do not spend enough time providing feedback on student testing and 36% feel they do not spend enough time grading design issues. On the other hand, around 14% of graders spend too much time on grading for correctness/bugs and seven percent spend too much time checking the quality and adequacy of student commenting. The percent distribution is shown in Figure 2. Given the time constraints on grading and the requirement that feedback be provided in a timely manner, it is no surprise to see that the two most time consuming activities (grading design and evaluating student testing of their code) were the ones that need more time devoted to them.

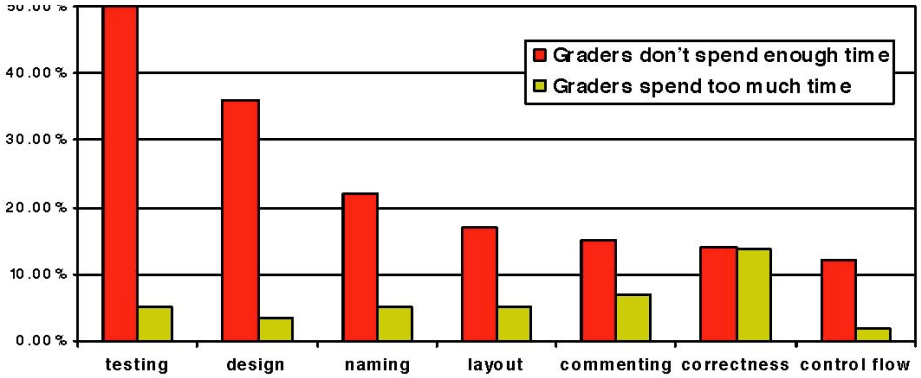


Figure 2. Percentage of time spent on different categories.

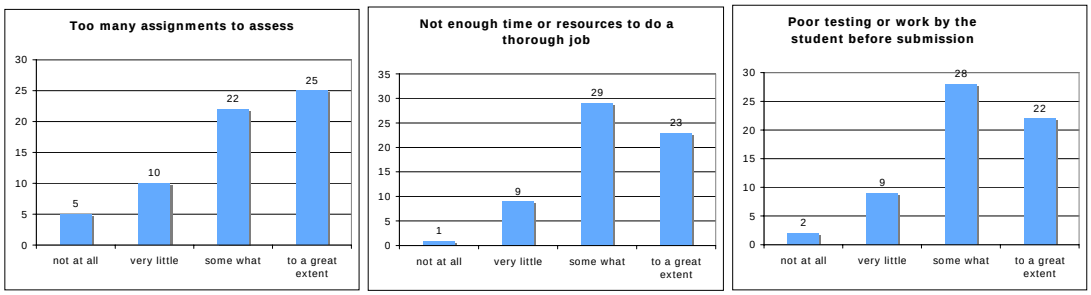


Figure 3. Impediments faced while providing feedback.

The survey also asked professors about the impediments that graders face when providing feedback to students. The question used a scale with the options: "Not at all", "Very Little", "Somewhat", "To a Great Extent." The three impediments that are relevant for this paper are: too many assignments, not enough time to do a thorough job, and poor testing done by the student before submission. The graphs below show the distribution of the responses. About 70% of the graders agree that clerical tasks of managing student assignments and returning results are obstacles in the grading process.

Thirty two out of sixty two responses stated that students turn in their source code print-out for the graders to read and assess. Almost an equal number of responses mention the use of an electronic submission system. About 65% of graders write their comments on a paper printout and primarily use plain text with occasional arrows, circles and lines to point out source code issues. One person stated that s/he provides "hand written" comments, typing them would be preferred but it is not done "simply as a time factor issue." From those that provide comments electronically, one "include[s] a link to a Web document that contains the instructor's solution at the relevant line in the instructor's solution source code." Comments are provided directly in the code in some cases, or as "short comments" as part of the student's score file that is e-mailed back to the student in many others. Some feedback was also given verbally "during the demo" of the program.

To assess student assignments for correctness, many graders either hand-execute the assignments against instructor provided data or use support software to execute the file(s) and compare the results with the professors' output. Some use a combination. One professor stated, "I have the automatic grader send all output to a big file which I examine manually. When appropriate I use diff to compare student output to correct output."

Almost half of the responses stated that the biggest disadvantage of the grading method was that their grading process was time consuming and sometimes "inconsistencies can creep in

even though a rubric is used." One person felt that the process was "time consuming and subjective ~ high variability."

**Summary**

We have seen that despite the numerous developments in systems that automate the grading process, many instructors still use a paper-based grading method. The reason could be that most automated systems are developed in-house and only provide support for the most basic form of feedback (program execution). Professors still find it difficult to grade assignments in big classes and prefer to automate the testing process if possible. Most of the professors do code markup on paper and many feel that they do not get adequate time to grade for code design and student testing.

## Interviews with Teaching Assistants

In addition of the survey of professors, we conducted interviews with teaching assistants (TAs) to understand their perspective on the grading process. The interviews focused on getting a thorough understanding of the different grading methods used by TAs and their relative advantages and disadvantages. Six TAs at Virginia Tech who had experience using different grading methods (both electronic and paper) were selected for these interviews. Each interview took about 60 to 80 minutes. Two general findings are reported here: how the TAs organize the submission files while grading, and how markup is done.

**File Organization**

All six TAs used the same method of organizing files while grading submissions. TAs maintained a folder (submission folder) with the submitted source files for all the student assignments sorted alphabetically. They would go through the submissions in order and use appropriate software to read the source code files. As they would finish reading a student's code, they would type their comments as well as the overall score in a plain text file and save it in a different folder (results folder) using the student's e-mail ID as part of the file name. They would then move the students' source code from the submission folder to a third folder (graded folder) to keep track of their progress. After all the files were graded and moved to the graded folder, a script would be used to e-mail each student his or her results file.

The plain text feedback file would contain a few statements from the TA and a common header usually placed at the beginning or at the end of the document. This header shows a summary of the deductions taken and other information, like the TA name, course name, and assignment.

TAs sometimes wrote a header template in another text file so that they could copy and paste it in each new grade report. All they would do then is fill out the missing fields in the header. TAs also included the scores produced by any automated compilation tool as a part of the deduction summary.

The organization and movement of files in the three folders sometimes became cumbersome for the TAs. Such work can become confusing, particularly when a TA needs to go back to a previously graded assignment to make modifications. This method of grading also requires a lot of time. TAs must be meticulous while writing comments because they have to explain in a clear way where and why the points were taken off in the student's assignment. Using a single file to provide feedback also indirectly encourages a TA to provide more general, overall comments for an assignment, since it is more difficult to identify specific features of the student's submission. For example, a TA would make a comment that "you have used upper case letters for some variables" or "you could have used a do while instead of a while loop". But the TA would not point out the specific variable or file or function to which this comment referred.

**Code Markup**

Two TAs out of the six interviewed had used Adobe Acrobat for marking up code

electronically. As with the process described above, TAs maintained a folder of students' source code files in PDF format sorted alphabetically. The difference is that when using Acrobat, the TA can directly entered comments in the PDF files themselves. Thus, the TA does not have to maintain a separate folder of text file grade reports. Acrobat has a feature to enter comments in the form of a pop-up note. TAs used this feature to write comments on student files and sometimes make this comment point to a specific line in the source code for emphasis. The disadvantage here is that these pop-up notes are not embedded in the code well and tend to move away from their original location as the user scrolls through the file. TAs claimed that it took them some time to learn the markup features of this software. Other markup features used included inserting images/icons and highlighting lines with color. When done with an assignment, the TA can move the files to another folder. When all grading is complete, a script can be used to e-mail the annotated PDF files back to the students.

Even though this method of grading reduced the overall grading time of the TAs, they still did not like the way they had to organize students' files. TAs also had to manually calculate the score summary for each student's assignment file. However, they felt that the markup features of this software helped them provide better feedback to the students by making their comments more understandable and readable.

**Observations**

The interviews with TAs helped us understand the problems they face while grading students' submitted assignments. TAs usually spend a lot of time organizing students' source code and report files. They find it difficult to point out mistakes or to provide corrective solutions. They also find it tedious to copy and paste comments that point out the common mistakes made by many students. TAs stated that they prefer to have an automated way to total up the point deductions made and add the header block for every report file that is sent out to the students.

**Requirements for an Ideal Solution**

Based on our survey of Computer Science professors and our interviews with TAs, we describe in this section the "ideal" solution. This serves as a set of requirements for a tool that will allow instructors and teaching assistants to markup a student's submitted source code files. A grading tool that supports direct markup of code should have the following features:

- Insert, edit and remove comments for any line of the source code using mouse clicks or using a pen-based interface. Allow comments to be entered about a group of lines.

- Assign a category for each comment made so that types of errors can be classified easily and accounted for in the summary of points.

- Support some form of visibility of the comment by selecting who can view the particular comment. E.g. Professors only, Professors and Teaching Assistants, etc.

- Save the comments and update the overall scores based on the deductions made by the user. The grader can resume grading at any time and find the file in the same state as he/she had left.

- Track the progress of grading, so that one can see which student assignments have been completed, or which files within a submission already have been reviewed.

- If possible, this functionality should be a complement to existing testing services for program grading. This will allow the grader to focus on providing feedback based on design issues and have automated tools test for functionality and/or assignment coverage.

- The feedback provided to the student should include a combination of style, design, testing, and other feedback in a single "format" (e.g. same report).

- The commenting tool should be integrated with a tool that supports online submission of assignments, distribution of feedback back to students, and archival of student submissions.

### Direct Markup in Web-CAT

In this section, we discuss the grading tool that supports direct markup of student code. The tool is integrated with Web-CAT and uses a "What You See Is What You Get" (WYSIWYG) editor. The TA can enter comments associated with any line of code. The comments most recently entered are stored together with the points taken off for each comment in a "history list." This allows the TA to reselect the most common error from a menu and reuse the comment and the point deduction. Grading can be done in multiple sessions, and the tool will keep track of the state of the grading process. The scores are automatically tabulated and include scores from other automated grading components as well as the points deducted by the TA by hand. The tool addresses other concerns, such as security, that are beyond the scope of this article. More discussion can be found in [20].

The following example, supported by screenshots, will walk us through the steps a TA takes while grading the assignment using the direct markup interface of Web-CAT. The TA is grading an introductory programming level course CS 1705 and uses the Web-CAT system to make comments on a student's submission for Program Assignment #2. Figure 4 shows the list of students and the scores they received during the automated phase of grading. In this example, the TA has already started grading Joe Hokie's assignment and has given him a score of 41 points but has not finished grading the entire assignment yet. S/he now has the option of going back to grade the remainder of Joe Hokie's assignment or to start grading Guy Smart's submission.

Once the TA chooses to return to Joe Hokie's assignment, s/he has to select which one of the files to grade, as shown in Figure 5. The check marks next to the last three class files indicate that the TA has already finished grading those files. The table also shows the number of remarks generated by static analysis tools or by the TA, the deductions, and the percentage of code executed by the student's test cases. At the bottom of the page is a text box where the TA writes any overall comments s/he has for the assignment and edits the final score if necessary. S/he clicks on the edit icon next to the first file (TrashCollector) to enter comments for that file. This brings up the markup editor, shown in Figure 6.

Figure 6 shows the WYSIWYG interface of our grading tool. The tool supports adding new comments, editing comments, and removing comments. New comments can be created by selecting the line and creating a comment or selecting the line and then selecting a previous comment from the history list to be reused. Comments include the name of the author (usually the grader), a textual description, a category, and the point deduction. The categories available are Error, Warning, Question, Answer, Good, Suggestion, Extra Credit (in order of severity). The comment can include font style (bold, italics, and underline), and can also include a hyperlink to another web page.

Figure 7 above shows one of the comment boxes shown in the formatted source listing. These boxes can either be TA-generated or tool-generated. For boxes generated by one of the testing tools, the comments are not editable. The box mentions the source of the comment: the TA's name, the instructor's name, or the name of a static analysis tool that generated the feedback message((e.g., PMD, Checkstyle, etc.). Figure 7 is an example of a user-generated comment box. It has been inserted for line 120 which has been executed 7 times. This new

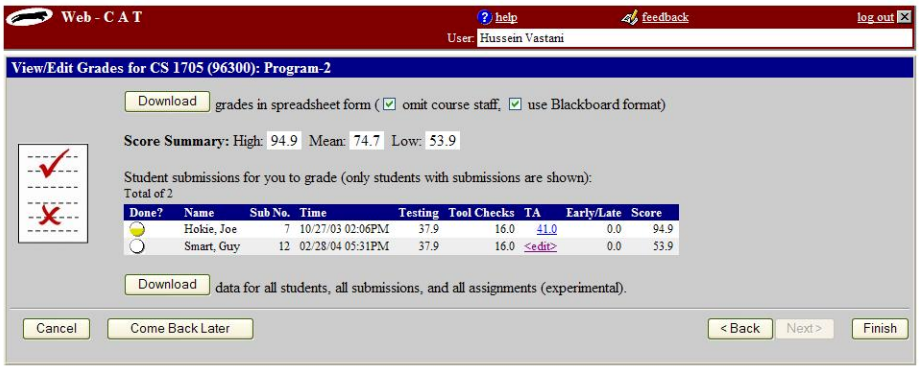


Figure 4. Selection of Students.

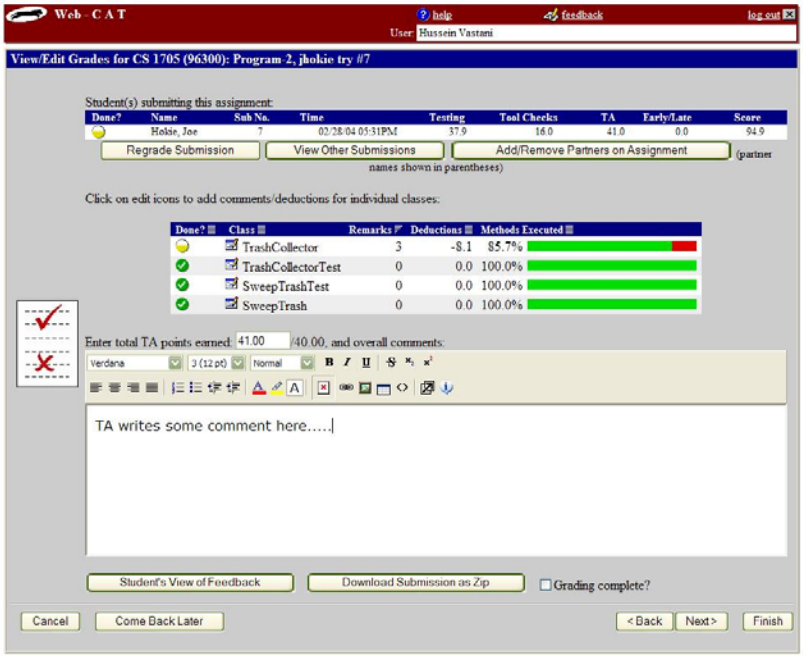


Figure 5. A students grade report, as viewed by the TA.

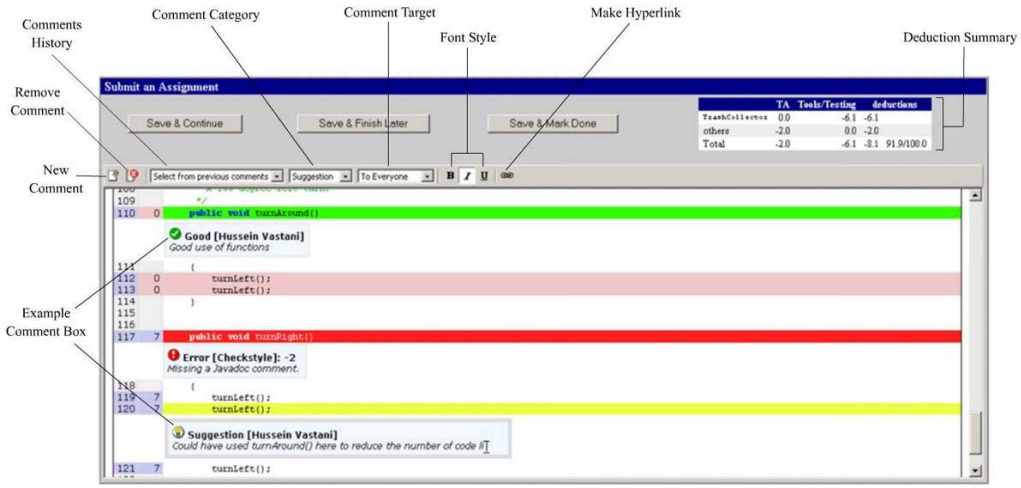


Figure 6. What You See is What You Get markup editor, based on HTMLArea [7].


Figure 7. Comment Box in Web-Cat produced formatted output.

comment is labeled as a 'Suggestion'. Each category has a color, an icon and (optionally) a point deduction associated with it. The icon here is a small light bulb and line 120 has been highlighted in yellow.

| | TA | Tools/Testing | deductions | |
|---|---|---|---|---|
| TrashCollector | 0.0 | -6.1 | -6.1 | |
| others | -2.0 | 0.0 | -2.0 | |
| Total | -2.0 | -6.1 | -8.1 | 91.9/100.0 |

Figure 8. Sample Deduction Summary Table.

Figure 8 shows the deduction summary table present at the top of the grading page in Figure 6. This table provides the TA with information about the deductions made for the current file as well as other files in this student's submission. The deductions made by the TA are shown alongside those generated through automated analysis (labeled "Tool & Testing"). In this example, there are no deductions made for the current TrashCollector file.

Once the grading is done, the TA can view the final grade report, as shown in Figure 9. This is the view of the report that the student will see. The page displays the score summary with respect to Design/Readability, Style/Coding and Correctness/Testing. Any overall comments made by the TA on this assignment appear next. This summary page also shows the list of files—the student can click on any file to view the inline comments made by the TA. Finally, this page shows the results produced by running student-written tests together with an estimate of how thoroughly the students solution covers the required behavior.

One advantage of the Web-CAT system is that the TA does not have to save and email the grade report back to the students or update any scores in a different spreadsheet. Once the TA is done with grading, s/he clicks on the Finish button. A notification e-mail is automatically sent to the student informing him or her that their assignment has been graded and feedback

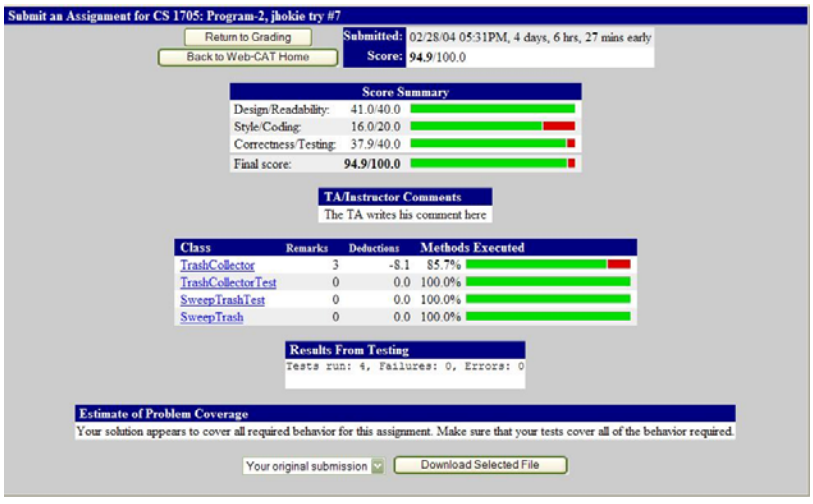


Figure 9. Final Grade Report.

is now available to view online. The student can then log into the system and view the grading report shown in Figure 9.

### Usability Evaluation of System

After our new markup interface was developed and added to the Web-CAT grading system, we interviewed four TAs that had an opportunity to use it. The TAs were asked their opinions about our new markup interface. Two TAs were clear victims of "new software syndrome" and were reluctant to use the new system. Instead, they continued to use their old method of writing comments and did not explore the new features. They assumed the new system to be complicated and imagined that it would take too much time to learn and adjust to the new way of grading in the middle of the semester. The other two TAs used the new interface to provide comments. Overall, they were very happy with this new way of grading.

There were three key features they liked and felt were advantageous in the new interface:

**Points summary/Header on top of page:** One of the biggest complaints that TAs had with the other methods of grading was the absence of some kind of a header to provide information such as file name, point deductions, the total points assigned, etc. They found the process of creating a header table and manually copying & pasting in all other report files very tedious. The header on our interface is a summary table that has the point deductions made by the TAs and the automated tools for the current source file and the other source files in a student's assignment.

**Ability to write inline comments:** TAs really liked the way our new grading interface allowed direct mark up the student's source code. During the interview, a TA commented, "Sometimes I had to copy and paste the code in the overall comment box to let students know what line of comment has the error". By providing inline comments directly in the source code file, TAs could easily point out errors in any line and provide solutions to it. TAs also felt that the inline comment boxes looked consistent and emphasized their comments.

**Accessing history of comments written earlier:** Another factor that the TAs found bothersome about the other ways of grading was the extra effort to rewrite or copy and paste the frequently used comments. "I wish I didn't have to type the same comments over and over again. What I do now is type it in a separate text file and then copy and paste it whenever needed" said a TA. The history list not only stores the twenty most recently used textual comments written by the TA, but also the point deductions, category and visibility. This helps the TA maintain consistency and fairness in grading all student assignments. TAs also felt that this feature would greatly reduce grading time since many students tend to make common mistakes.

The TAs also provided some suggestions that point areas of improvement for our work:

**Having many comment boxes affects code readability:** A TA claimed that "Extreme coloring and large number of comment boxes, makes the code look cluttered and unreadable". It was difficult to read the source code lines between the multiple comment boxes inserted by the TAs, PMD tool and Checkstyle tool. TAs found it troublesome to scroll the code window since the increased number of comment boxes took up too much space. They suggested that if the comment boxes can be minimized somehow (like the post-it feature in Adobe acrobat), the source view would look cleaner and more readable. Also TAs felt it would be convenient to include a feature to toggle on/off the comment boxes inserted by the automated tools (PMD and Checkstyle) since they sometimes found it hard to spot out their own comments.

**Providing comments for a block of code:** One TA felt that certain comments not only apply to one line of code but also to multiple lines. For example, if a student's logic in the while loop was incorrect, then the TA would want to highlight the entire loop to show the error. The TA therefore suggested that if the new interface had the feature to highlight

multiple lines of code and apply a comment box to it, he/she could provide a better solution for the error thus increasing the quality of feedback.

## Discussion and Future Work

In this paper we have presented our work on creating a code markup tool that is integrated with Web-CAT, and online automated grading system. We have shown how our tool tries to streamline the grading process by helping the instructors and teaching assistants focus more on the deeper issues of grading while the automated tools do the work of assessing for correctness as well as checking for testing coverage and other stylistic considerations. The system also automates the return of the results back to the students, and archives student assignments which can easily be accessed online.

Our work was motivated by a survey of Computer Science professors and interviews of TAs on how they do their grading of programming assignments. We tried to build a system that addressed their needs. The system has been in use at Virginia Tech for close to a year.

In the future, we want to expand our grading tool to support peer review of other student assignments by presenting a student with the same interface and features that the TA sees during grading. The only difference is that during peer review, students cannot make point deductions or modify the comments made by the professors or teaching assistants. The peer review feature is still under development and will be integrated into Web-CAT in the near future.

We are also exploring how to provide direct markup of code using a Tablet PC instead of a web based interface. We expect this functionality to be integrated with Web-CAT but not to be available for another year or so.

Finally, we have completed support for Eclipse and BlueJ to further automate the submission process. Our students develop their projects on their personal computers. The development environment has an option to automatically send their project directly from their environment to Web-CAT.

## References


1. Adobe website, website last accessed on June 30, 2004, http://www.adobe.com/.

2. Checkstyle home page, website last accessed on June 30th 2004, http://checkstyle.source forge. net/ .

3. Clover: a code coverage tool for Java, website last accessed on June 30th 2004, http://www.thecortex.net/clover/.

4. Edwards, S., "Teaching Software Testing: Automatic Grading Meets Test-first Coding", OOPSLA '03, October 26-30, 2003, Anaheim, California, USA, 2003.

5. Edwards, S., "Improving student performance by evaluating how well students test their own programs", Journal of Educational Resources in Computing, v3, n3 (September 2003), pp. 1-24, ACM Press.

6. Edwards, S., "Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action", SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA, 2004, pp.

7. HTMLArea, website last accessed on June 30th, 2004, http://www.interactivetools.co/products/ htmlarea/.

8. Isong, J., "Developing An Automated Program Checker", Proceedings of the Seventh Annual Consortium for Computing in Small Colleges Central Plains Conference on the Journal of Computing in Small Colleges, Bransom, MO, The Consortium for Computing in Small Colleges USA, 2001, pp. 218-224.

9. Jones, E. L., "Grading Student Programs – A Software Testing Approach.", Proceedings of the Fourteenth Annual Consortium on Small Colleges Southeaster Conference, Salem, VA, The Consortium for Computing in Small Colleges USA, 2000, pp. 185-192.

10. Dr. Latham, J.T., "Managing Coursework: Wringing the Stone, or Cracking the Nut?", Nikos Drakos, Computer Based Learning Unit, University of Leeds. 1995, http://www.cs.man.ac.uk/~jtl/ARCADE/huddersfield98/huddersfield98.html.

11. Joy, M and Luck M, "The BOSS System for On-line Submission and Assessment of Computing Assignments", Computer Based Assessment (Volume 2): Case studies in Science & Computing, ed. Dan Charman and Andrew Elmes, SEED Publications, University of Plymouth, pp. 39-44, 1998.

12. MacPhereson, P.A., "A Technique for Student Program Submission on UNIX Systems." ACM SIGCSE Bulletin, Volume 29, Issue 4, New York, NY, ACM Press, 1997, pp 54-56.

13. Mason, D.V. and Woit, D.M., "Providing Mark-Up and Feedback to Students with Online Marking", Proceedings of the Thirtieth Annual SIGCSE Technical Symposium on Computer Science Education, New Orleans, LA, ACM Press, 1999, pp. 3-6.

14. PMD home page, website last accessed on June 30th 2004, http://pmd.sourceforge.net/.

15. Popyack, J.L, Herrmann, N., Char B., Zoski, P., Cera C., Lass R., "Pen-Based Electronic Grading of Online Student Submissions", Drexel University, Presented at the Syllabus fall2002 Boston Area Conference on Education Technology, Newton, Massachusetts, November 4-5, 2002.

16. Preston, Jon A. and Shackelford R., "Improving On-line Assessment: An Investigation of Existing Marking Technologies.", Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education, Cracow, Poland, ACM Press, 1999, pp. 29-32.

17. Price, B. and Petre, M., "Teaching Programming Through Paperless Assignments: An Empirical Evaluation of Instructor Feedback", Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education, Uppsala, Sweden, ACM Press, 1997, pp 94-99.

18. Reek, K.A., "The TRY System – or – How to Avoid Testing Student Programs", Proceedings of the Twentieth SIGCSE Technical Symposium on Computer Science Education, Louisville, KY, ACM Press, 1989, pp. 112-116.

19. Reek, K.A., "A Software Infrastructure to Support Introductory Computer Science Courses", Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education, Philadelphia, PA, ACM Press, 1996, pp. 125-129.

20. Vastani, Hussein Kamaluddin, "Supporting Direct Markup and Evaluation of Students' Projects On-line" Master Thesis, Department Computer Science, Virginia Tech, June 11, 2004, Electronic Thesis etd-08172004-020310, available at http://scholar.lib.vt.edu/theses/available/etd-08172004-020310/.

21. WebObjects Development, Student Guide WebObjects 5.2, web page last accessed on June 30th, 2004, http://www.apple.com/webobjects .

## Biographical Information

Hussein Vastani works as a Software Engineer at Advanced Simulation Technology Inc located in Herndon, Virginia. He received Bachelor's Degree (2002) and a Master's Degree (2004) in Computer Science from Virginia Tech.

Stephen H. Edwards is an Associate Professor in Computer Science at Virginia Tech. His research interests include software engineering, automated testing, software reuse, computer science education, formal methods, and programming languages. Learn more about Web-CAT on-line at: http://web-cat.cs.vt.edu/WCWiki.

Manuel A. Pérez-Quiñones is an Assistant Professor in Computer Science at Virginia Tech. He is a member of the Center for Human-Computer Interaction. His research interests are in Human-Computer Interaction, Educational Uses of Computers, and Interaction with Portable Devices (phones, pdas,etc.).