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

Hussein Vastani

Thesis submitted to the faculty of

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Dr. Stephen Edwards, Chair

Dr. Manuel Perez

Dr. Chris North

June 11, 2004

Blacksburg, VA, USA

Keywords: Automated grading, WYSIWYG editor

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

by

Hussein Vastani

Abstract

Automated grading systems have been researched at various universities for several years. Numerous systems have been developed that automate the process of grading by compiling, executing and testing the students submitted source code. However, such systems are mostly written as UNIX scripts and are restricted to performing one kind of activity. The instructors or teaching assistants have to resort to other methods in order to provide their feedback to the students.

The core of this thesis is to research a TA feedback mechanism which will streamline the grading process for the professors and teaching assistants. A web-based grading tool has been developed that allows course staff to enter comments for students' programs directly through a web browser. The interface provides for full direct-manipulation editing of comments, which are then immediately viewable by students when they look up assignment results. Such an interface also has potential to support peer grading of assignments.

Teaching assistants of introductory programming level courses were interviewed to learn about the different grading methods they use and were asked their opinion of our new grading interface. TAs were also asked to grade assignments using the traditional paper method as well as the computer using our new grading tool for comparison. Finally, an anonymous survey was sent out to various computer science faculties in different universities to gather information about the expectations they have with respect to TA grading activities for programming assignments and the learning outcomes that these professors desire for their students.

Acknowledgements

I would like to express my sincere gratitude to all those who have made this thesis possible for me. Firstly, I want to thank my parents for their love, support and continuous encouragement. It is my family who has helped me realize my goals in life. I am deeply indebted to my advisor, Dr. Stephen Edwards whose guidance and motivation helped me tremendously in my research and writing up this thesis. Furthermore I want to thank my committee members, Dr. Chris North and Dr. Manuel Perez for their valuable ideas and suggestions. I want to express my appreciation to the teaching assistants and others who have directly or indirectly influenced my work Lastly, I'd like to thank all my friends, my roommates and especially Anita who has always been there for me to share all the ups and downs in my graduate study.

This work is supported in part by the National Science Foundation under grant DUE-0127225. Any opinions, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of NSF.

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1 Introduction

According to Merriam Webster [Merriam-Webster, 1982], feedback is *"the transmission of evaluative or corrective information to the original or controlling source about an action, event, or process"*. Providing feedback has pedagogical value and it cannot be disputed that proper feedback increases students' performance and facilitates effective learning. Professors teaching computer science courses provide feedback to students in various ways. Some still use the traditional paper-based method of writing comments on code printouts, while some type their comments using a text editor on the computer, and email the document to students. Regardless of the kind of feedback mechanism they use, they face many challenges in providing quality feedback in a timely manner.

Paper-based markup as a means of providing feedback is the most widely used form of markup by many professors in universities. They simply write their comments directly on paper and draw circles and arrows to get special attention on certain parts in the paper. "The grader's remarks (perhaps in red ink) stand out as distinct from the original submitted work, and the context is understood from their location on the paper. The grader can work in close or cramped quarters, compare papers side-by-side, and easily flip back and forth between pages." [Popyack et al, 2002]. Paper copies are not easy to manage especially when the size of the class gets large. It also gets inconvenient if the copies need to be distributed to other teaching assistants or to make use of plagiarism detection software [Aiken et al, 2003]. The graders have no permanent record of the comments they make on students assignments (unless they make copies of the marked up assignments), since they are returned back to the students.

Electronic submission of assignments has been beneficial to professors teaching computer science courses. Use of automated grading systems not only reduces the workload of grading many student assignments, but also overcomes the deficiencies involved in paper based grading [Reek, 1989][Luck & Joy, 1998][Jackson & Usher, 1997], [Isong, 2001][Jones, 2000]. These systems focus mainly on code compilation and execution of programs against some form of instructor-provided test data. Although some aspects of the assessment can be accurately performed by the computer, human intervention is still necessary in certain areas. Such sophisticated systems tend to reduce the close interaction between students and professors. As a solution, some professors conduct interactive demonstration of students' programs to develop active relationship with them. However, due to the increasing size of the courses, individualized attention becomes difficult. Other professors in universities with abundant resources employ teaching assistants (TA) or graders to assist them in grading students' programming assignments. Nonetheless, there is a concern of fairness and consistency in grading, since the teaching assistants may not have the right experience or skills required to assess such complex programs accurately and comprehensively.

How can one provide a way to help educators manage the work load of grading multiple student assignments efficiently, without introducing the overhead of additional

installation of software, and at the same time maintain fairness and improve the quality of the feedback provided to them? One approach to addressing this problem is having an online grading system which can be used by the instructors and teaching assistants simply with the help of a browser. This system will enable them to grade the students' submitted source code files by a direct manipulation scheme. It will do so in a timely manner and include other features such as line highlighting, automated score calculations, archiving and report generation. The feedback report can be then viewable by the student online.

1.2 Problem Statement

Instructors of computer science courses are already overburdened with work and thus have little time to devote to additional assessment activities. Even the teaching assistants find it difficult to grade large number of assignments within a small time frame. As a result, the research question investigated in this thesis is

> *Can we streamline the process of grading programming assignments by allowing educators to enter comments directly in source code using a WYSIWYG interface within the web browser?*

To address this question an online grading tool was designed and developed. Such a tool includes support for inserting, editing and removing comments by graders in the color-coded display of students' source code. The graders have the ability to save the comments and resume grading at a later time. The comment boxes have properties such as category, style, color and visibility associated with them. Graders can easily change these properties to make them more understandable to the students and thus increase the quality of feedback. A list of recently used comments, which tremendously helps in reducing the overall grading time, is also available to the graders.

This tool can be integrated in the Web-CAT [Edwards, 2003] grading system developed and in use at Virginia Tech. Instructors and teaching assistants can log in, access the students submitted source code files and enter comments directly in those files. They can assign positive or negative points which will automatically be included in the overall score for that student's assignment. They can also provide overall comments in a separate text box specifically meant for that purpose. Once they are finished grading the source files of one student, they can easily navigate to the next student's assignment.

1.3 Proposed Solution

This research discusses the design and development of a WYSIWYG interface that will allow instructors and teaching assistants to markup students submitted source code files by providing a unique feature to insert comment boxes on a click of a button.

Graders can not only write their comments directly in the comment box, but also delete the comment boxes or edit them at anytime. They can also insert comment boxes by selecting from a list of previously inserted comments.

Each comment box is associated with a category and visibility for the purpose of clarity and preference. On selecting the category of a comment, the related code line will be highlighted with an appropriate color for emphasis and the TA can choose the target of the comment. Negative, positive or no points can be assigned for a comment which will take effect in the overall score of the assignment. The TA can save the comments to come back to later or mark the file as 'done'.

This tool which we will call '**our grading tool**' or '**new grading interface**' has the following novel features:

- Insert, edit and remove comments in the form of a comment box for any line of the source code by simple mouse clicks.
- Assign a category for each comment made which in turn draws an icon and color highlights the source code line for which the comment is made.
- Change the visibility of the comment by selecting who can view the particular comment. E.g. Professors only, Professors and Teaching Assistants, etc.
- Store and use flexibly the most recent comments made or edited by the user.
- Save the comments in the database and update the overall scores based on the deductions made by the user. The user can resume grading at any time and find the file in the same state as he/she had left.

This interface is added to the Web-CAT grading system to enhance the grading services it provides to instructors and teaching assistants. The feedback from this tool is combined with the feedback provided by Web-CAT grading system to prepare a single consolidated report for each student. Web-CAT is configurable to do automated error checking and help teaching assistants concentrate on other important issues of grading such as design, readability, etc.

Finally, a formative evaluation was performed to assess the effectiveness of this tool and get the teaching assistants perception on its use.

1.4 Feedback Mechanism in Web-CAT

Web-CAT is a web application that provides software testing services to students. It's most prominent service is an electronic submission and grading system that supports test-driven development [Edwards, 2004].

Students are assessed based on how well they demonstrate the correctness of their own solution, and are also given concrete, automated feedback on what code they have not tested well enough, what style or documentation issues are present in their work, and what coding patterns might indicate potential bugs or trouble spots in their work.

Assignment feedback is available in summary form, characterizing how well the student has performed in several dimensions. The student also gets detailed feedback on each class (file) submitted, as well as the results of the student's own tests.

Feedback on individual files written by the student is provided through a color-coded source code view in the student's web browser. Individual statements that reflect violations of expected stylistic conventions or potential coding errors are marked with explanations. Point deductions are provided where necessary.

The core of this thesis is the **instructor/TA feedback mechanism**, the details of which we will see in later chapters. It will allow course staff to enter comments to students directly through their web browser. The interface provides for full direct-manipulation editing of comments, which are then immediately viewable by students when they look up assignment results, just as automatically generated comments are.

Such an interface also has the potential to support student review assignments, where one student (or group) is assigned the task to critically review and comment on the work of another. Peer feedback helps promote code reading skills, and that feedback can be used by the original author to improve their own work.

1.5 Organization of Chapters

Numerous efforts have been made in the past to devise a system that would help the overall grading process in computer science courses. Chapter 2 discusses some of these efforts and identifies the novelties of the approach presented here. In order to assess the need for our tool, personal interviews with teaching assistants were conducted. Chapter 3 discusses these interviews and their results in detail. Chapter 4 discusses the use of our tool with sample screen shots and procedures while Chapter 5 discusses the detailed design, the architecture and the integration of our new grading interface with Web-CAT. The latter half of this thesis provides information about the different assessment methods and the discussion of the results. Chapter 6 discusses the opinions of the TAs regarding the new grading interface. Chapter 7 discusses the experimental evaluation of this tool using TAs by comparing traditional paper-based grading to Web-CAT grading. Chapter 8 discusses the survey given to the SIGCSE members to get their perception on the current grading process. Finally, Chapter 9 presents the conclusions, contributions and suggested extensions to this research.

Chapter 2: Related Work

2.1 Why is Feedback Necessary?

Feedback is an important part of teaching and learning. Instructors provide feedback to students to evaluate their work, inform them of their mistakes and suggest corrections, and to help students improve their efforts by not repeating the same errors. In her paper [Bull, 1999], Bull says that "feedback can be used to direct future learning, motivate students to investigate other resources and identify students who need additional support". She discusses how Computer Aided Assessment (CAA) helps to provide constructive, detailed and consistent feedback when there are large numbers of students. The major concerns with providing feedback are the instructors' abilities to provide quality feedback in small time frames and the students' willingness to read the comments made by the instructors.

Feedback is usually provided through some medium. It could be verbal or written. Feedback can be provided in writing on paper or using a computer. Few approaches have been made to compare the quality of feedback provided on paper and on the computer. In their paper, Price and Petre [Price & Petre, 1997] show the results of a study that aimed at comparing the nature and quality of feedback on paper and electronic assignments. The results of the study show that the two methods of providing feedback are comparable. They show that using electronic marking as a medium of providing feedback does not impair students' or instructors' abilities of expression. They also prove that administration gets faster and more efficient since the turnaround time reduces with electronic assignments. Students and administrators are pleased with the quality of the feedback provided and with ease of handling electronic assignments.

In this chapter we will see how instructors provide feedback to the students using various automated systems developed and the drawbacks prevalent in those systems.

2.2 Role of Automated Systems in Grading Process

Instructors in various universities teaching computer science courses are 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 found to be very successful in reducing the workload of the instructors and teaching assistants. They undoubtedly provide additional benefits in terms of consistency, thoroughness and efficiency in assessing student programs [Isong, 2001]. However, most of these systems have been developed locally by instructors and are restricted in their use. Amongst the systems developed, most of them support the electronic submission of student assignments. Few also support the automatic compilation and execution of student code against some instructor-provided test data. And very few, besides assessing for correctness, support the actual electronic markup of the student code and the automatic return of results.

The Arcade system developed by Dr. Latham [Latham, 1995] aims mainly at handling the administrative aspects of the assignment. The system automatically checks for deadlines and submission information. The actual procession of the submission is not taken into account in the system.

Macpherson's paper [McPhereson, 1997] presents a new scheme for collecting student programs and redistributing grades using simple UNIX scripts. The system transfers the control of the files that are placed by the student in his or her directory, to the professor after the program is due. The programs cannot be altered by the student until they are graded, after which the control is passed back to the student with the grade file left in the student's account and a copy mailed to the student. The actual grading however is done manually by the instructor. This system does not address the concerns about accountability and the potential for errors in transmission.

In her paper, Isong [Isong, 2001] discusses her approach to developing an automated program checker to assess the functional correctness of the student programs. Instructors are responsible to create an assignment specification and develop an extensive suite of test cases from the specification. They also develop grading plan based on the assignment specification and test cases. Isong's checker does not check for efficiency, style, complexity and test data adequacy. Like many other in-house grading systems, Isong's checker is written as a collection of UNIX shell scripts.

Reek's software developed at Rochester Institute of Technology is a Unix-based system [Reek, 1996]. It uses a file-system based organizational strategy for managing assignments and student uploads. The submission and archiving systems infrastructure is built on the TRY program [Reek, 1989]. His software focuses on compiling and executing student programs against instructor-provided test data and then assigning a grade based on comparing the actual output of the student program against expected results provided by the instructor. Instructors can assign categories to different test cases in order to control whether or not students can see the output compared to what was expected, and whether failures halt or abort the grading process.

Program Submission and Grading Environment (PSGE) is another Unix-based assignment checker [Jones, 2001] developed by Jones. It has support for semi-automatic and automatic grading of assignments. Unlike TRY and Islong's checker, PSGE defers grading until after the assignment deadline. Thus, students do not have an opportunity to correct any bugs or other errors and resubmit the assignment.

The ASSYST software developed by Jackson and Usher [Jackson & Usher, 1997] does an extensive evaluation of student performance. It not only provides the basic compile/test/compare-output features of other systems, but also includes static checks for style and code complexity metrics. The biggest advantage of this system is that it assesses student-provided test cases. Students are rewarded for the correctness of their programs and for writing test cases to test their programs. It has a graphical user interface to direct all aspects of the grading process.

Virginia Tech for long has used its own grading system called the Curator [McQuain, 2004]. This system is similar in principle to most systems that have been described. It allows a student to submit the programming assignment electronically. The Curator compiles and executes the students' as well as the instructor's reference source code on a randomly generated input file. It grades the results by comparing the students' output against the reference implementations output. The student then receives feedback in the form of a report that summarizes the score and includes the test input used, the student's output, and the instructor's expected output for reference. The unique feature of this system is that it has a user interface which gives the look of a web-based application even though the actual compilation and execution are done using shell scripts.

English and Siviter [English & Siviter, 2000] describe a module that integrates different automated assessment tools and techniques designed to assess a variety of different skill types into a common framework and share the result in a common database.

The biggest advantage of many automated tools is that they leverage 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.

Instructors that allow multiple submissions before the due-date give an opportunity to the students to learn from their mistakes without being penalized for it. The students can correct any errors they made and resubmit the assignment. This also encourages students to begin their assignments earlier.

Although such systems help instructors and teaching assistants spend more time on looking at the individual source code of the students, 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 itself which are handed back to the students. Others read student code by retrieving it from the list of electronic submissions and making comments on a separate document that is then mailed 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. In the following sections we will see the approaches people have taken to perform online marking of student code and the return of results back to the students.

2.3 Systems That Support Online Marking

Quality assessment of student assignments can be done best by instructors and teaching assistants themselves. Most automated systems assess only the programs correctness and to some extent check for style and test coverage. Checking for readability, design and efficiency can be best done by the skilled graders.

The work of Popyack, Herrmann,Char, Zoski, Cera and Lass at Drexel University [Popyack et al, 2002] is most relevant to our research. They have developed a marking

methodology that combines the electronic and paper grading approaches. They consider the advantages of both these methods and present a solution that allows graders to write free hand comments on the students' electronic submission using pen-based tablet PCs. This style of feedback has close resemblance to traditional pen and paper grading and to the representation of document in digital form for better archiving. They make use of Adobe acrobat [Adobe, 2004] to markup documents because of its supporting features. "Adobe Acrobat provides a means of marking up PDF documents with an electronic pen, so the resulting document resembles the original with the grader's notations overlaid, much like graded paper assignment" [Popyack et al, 2002]. The graders can make use of Acrobat's sticky-notes and other features like line highlighting, coloring, etc. to assist in the markup. Scripts can be embedded in PDF documents using Acrobat Javascript for the purpose of manipulating document elements. Since the document can be digitally signed and locked, graders do not have to worry about students making any changes to the comments made.

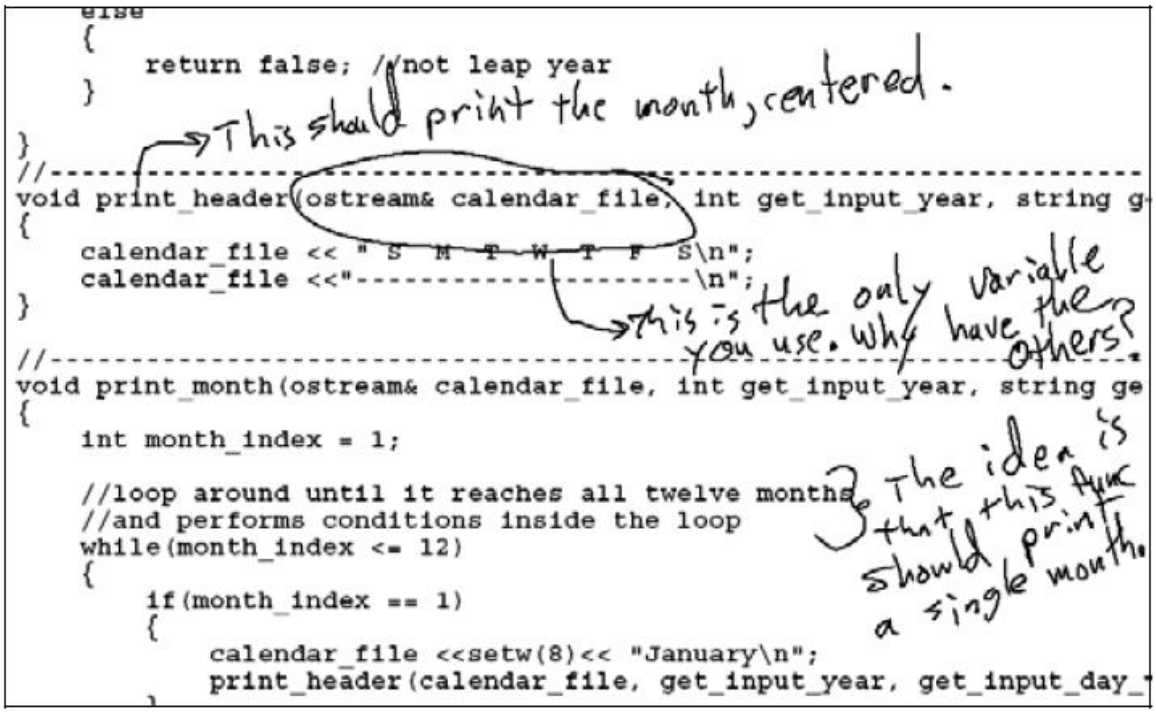

Figure 2.1 Example markup done using pen based method [Popyack et al, 2002]

Students upload their submission as a zip file using WebCT's course management system [WebCT, 2001]. Due to certain drawbacks in WebCT, the authors developed Labrador [Cera et al, 2002], a perl based client side software that interfaces with WebCT. This software provides additional services to the graders to download student submissions systematically and in the appropriate format. It also supports post processing of student submissions like plagiarism detection, style checkers and conversion of documents to PDF format for electronic pen-based grading. Labrador has an interactive command line as well as a GUI interface. Compilation, execution and testing of student code and the return of results back to the students are still not supported in the current software.

Besides being a course management system that manages students' electronic submissions, WebCT [WebCT, 2001] also has a grading interface. This interface consists of a textbox for the graders to enter the comments and another textbox to enter the grade. The graders who are used to drawing arrows, lines or circles while grading on paper using a pen are now forced to simply write comments using plain text. They fear the students may find it difficult to understand the graders comments.

Preston and Shackelford at Georgia Tech [Preston & Shackelford, 1999] are working on developing a software tool that will benefit the markers by improving their marking process in large sized classes. They investigated the existing marking methodologies of their TAs, and are now incorporating support for the same in their tool which will allow an overall view of students' work and will have the ability to hide and display the implementation details when needed. It will also allow quick and easy navigation between sections of a work. The other features they are planning to incorporate in their tool are, language syntax highlighting, enabling annotation features, upload and download of required files and the automatic submission of grades back to the students. The graders use feedback codes to categorize the errors while grading. The tool will allow separation of the assessment interface from the students' source code to hide the error codes from the graders.

Recent additions to Luck and Joy's BOSS system [Luck & Joy, 1998] include a graphical electronic marksheet developed using the Tcl/Tkl toolkit. This marksheet is constructed after the instructor provides the different categories that will be considered during grading and weight attached to these categories. The BOSS system supports automatic execution and testing of student code against several sets of input data [Luck & Joy, 1998]. The marksheet includes the marks obtained after the tests are run and the students' output compared to the expected output. It has buttons to view the details of these tests. The instructors manually grade the other aspects of the program like style, layout, commenting by moving the slider on a scale of zero to ten to indicate the marks awarded for each category. The final score, which is the addition of individual marks multiplied by the weight (if any), is displayed on the marksheet. This system allows double marking on the same marksheet by different graders and the final score is then computed as the mean of these marks. The instructors can also write comments for the student by clicking the relevant button on top of the marksheet. The marksheets are emailed back to the students and are archived in their original form for any conflicts that may arise.

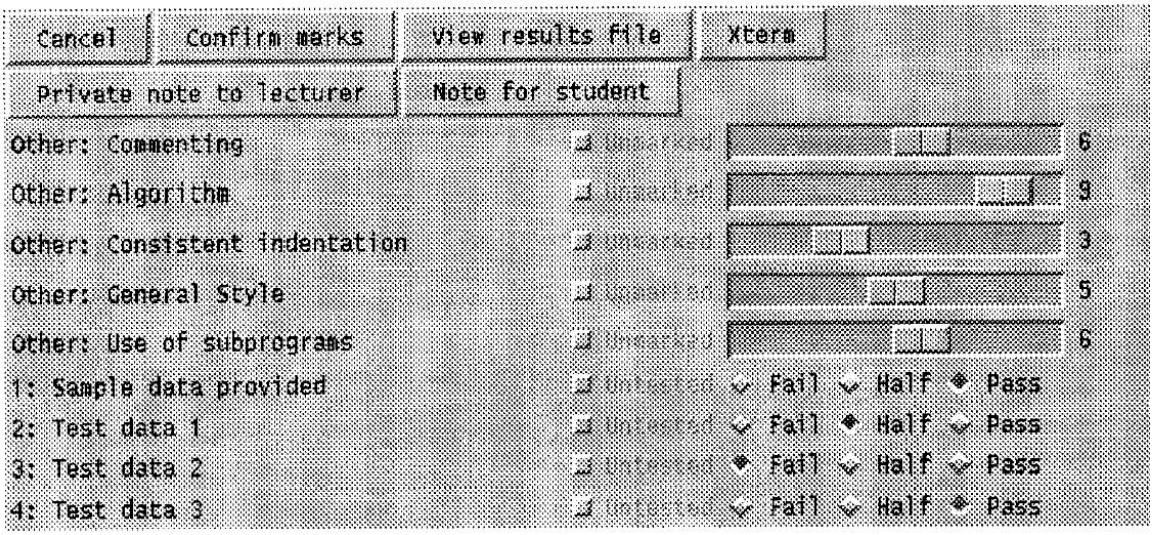

Figure 2.2 Example marksheet [Luck & Joy, 1998]

Mason and Woit [Mason & Woit, 1999] describe an environment that provides online in-context annotation of student programs before they are returned to the students. The annotation program generates a hyperlink for each significant feature of the assignment to provide marking and annotation for that section. In addition to the markable sections, there exist a comment field and assignable mark for the total assignment. When the grader clicks on the hyperlink to add a comment for a particular section, another window opens up with the relevant information such as the type of section, the actual code and a pull down menu with predefined mark/comment choices. The grader can type his comment in the box provided and save the changes. The code is underlined to show the students that a comment has been made there. The student can then click on that hyperlink to see the details. The system only saves the latest marking of the assignment.

2.4 Support for Peer Review

Peer review or assessment is a part of the student learning process. As the name suggests, students review or grade the work of other students usually in the same course. Peer assessment encourages deep learning for students by allowing them to critique and provide feedback on other students work. [AAHE, 1993][Topping, 1998]. Students can improve their own work by learning different techniques and realizing the similar mistakes they may have made while performing a review of other students work [Bhalerao & Ward, 2001]. Student doing peer assessment improve their programming style and think deeply about the quality of work [Brindley & Scoffield, 1998].

The reasons why peer review is very effective has been discussed in the literature. [Topping, 1998.][Somervell, 1993][Dochy & McDowell, 1997][Zeller, 2000]. For this research, we are concerned with systems that have been developed to support peer review of electronically submitted assignments.

Sitthiworachart and Joy at University of Warwick [Sitthiworachart & Joy, 2003] have developed a web-based peer review system. Students perform an assessment of other students' work by marking up their source code via the web interface of this system. After logging in, each student marker is presented with a menu page that requests the marker to perform three steps. These three steps include marking source code files of other students by clicking on the relevant buttons, marking the quality of other markers, and seeing the overall calculated marks awarded to the student. The actual marking of student assignments includes answering a set of nine multiple choice questions on different categories such as readability, correctness and style. For each question, the marker can select No, Yes, partial or let it remain unmarked. The marker can also write comments for a group of three questions by following the instructions mentioned in the marking criteria on the interface. In the second step, the marker grades the quality of marking done by other markers by answering if the comments made by markers are relevant, well explained and useful. Finally the student can see the details of his/her score by clicking on a link at the bottom of the interface. This shows the full mark and comments made by the markers and the grade the student received based on his/her quality of marking.

In their paper, Trivedi et al [Trivedi et al, 2003] have described an extensive Web-based system called RRAS as one which manages anonymous peer evaluation of electronically submitted assignment. RRAS has a web interface which allows students to review and assess the work of other students and to check the results of their evaluation. It also has interfaces to help instructors administer the course, set access permissions and for students to log in and submit their assignments. RRAS provides the students with a rubric designed by the professor, which they use for peer evaluation. Students can also view the evaluation of their work done by other reviewers. The rubric here is the only medium of assessment used by the students.

The Praktomat system [Zeller, 2000] is another automated system that manages online submission and uses automated testing for assessing program correctness. It also supports mutual reviewing of student programs for the purpose of improving readability of student code. The assignment that the student submits through the Praktomat system is first compiled and run against test cases to check for failures, and appropriate feedback is displayed to the student. If no failure occurs, the student retrieves another students' program, selected by Praktomat, to read and review. The review form that the student sees has the source code and a list of multiple choice questions related to assessing different areas like style, indentation, usage of identifiers, structure, etc. Students who get their assignment reviewed before the due date can make changes and resubmit. Praktomat also maintains anonymity by not disclosing the name of the author of the program to the reviewer. The system does not allow the instructors to refer to the reviews done by the students while they do their own share of final assessment.

Gehringer's peer-grading system [Gehringer, 2001] uses the World Wide Web for submission and peer review of student assignments. It is a servlet-based java application where students submit their assignment as sequential files or a single zip file. Assignments are selected at random or by the professor and given to the students for review. The system allows reviewers and authors to communicate via a shared web page

that is created for each author. The reviewer writes his/her comments on an interface that has a text box and another box to enter the grade. The authors can make changes after viewing the reviewers' comments and resubmit the assignment which is reviewed again.

Thus we have seen that very few attempts have been made to develop online software that support automatic grading as well as peer review of students assignment. These systems maintain anonymity to some extent but do not support direct markup of student code. Students usually read the source code file on some interface and answer questions or write few comments which are added to the students report as feedback. Our grading tool supports peer review where students are presented with the same grading interface that the TAs use to write inline comments in the source code.

Chapter 3: Requirement Analysis

In order to assess the need for our new grading interface and to understand why teaching assistants require a different and improved method of grading, personal interviews with TAs were conducted. These interviews focused on learning about their current and past methods of grading and the relative advantages and disadvantages of those methods. The goal of these interviews was to help us understand the requirements that would streamline the grading process of a teaching assistant and design a system/tool that would entail all the necessary features needed by the teaching assistants.

TAs who had experience using different grading methods (both electronic and paper) were selected for these interviews. This chapter describes the personal interviews with TAs and discusses their views and suggestions on the different grading methods they ever used.

3.1 TA Interviews

Six TAs of introductory programming level courses were interviewed. The purpose of these interviews was to learn how TAs grade student programming assignments and their likes, dislikes with respect to this process.

The interviews focused on getting a thorough understanding of the different grading methods used by TAs and their relative advantages and disadvantages. Other related questions attempted to gather information about,

- time it takes for them to do the grading
- barriers/difficult part in grading
- their interaction with students with respect to the students grading results
- their interaction between the professors with respect to grading criteria and issues
- their idea of quality comments
- the quantity of comments
- the different markup materials used, like colored pens (if any)
- grading criteria used
- any suggestions for improvements in the grading process

The interviews were casual and the responses to questions generated more related questions. The TAs were very co-operative, gave good feedback on their grading process and provided honest suggestions that they felt would help alleviate the problems

encountered during grading. Each interview took about 60 to 80 minutes and was held in a building on the Virginia Tech campus.

3.2 Discussion of Responses

The following sub-sections provide a detailed discussion of the different grading methods used by the TAs and a summary of their responses to most of the questions mentioned in the above section (3.1).

3.2.1 Using Notepad to Write Comments

Notepad is a basic stand-alone text editor that is used to create simple documents in Windows. The most common use for Notepad is to view or edit text (.txt) files. Notepad supports only basic formatting and the file can be saved as Unicode, ANSI, UTF-8, or big-endian Unicode [Notepad, 2001].

This method of grading had been used by all six TAs. TAs maintained a folder (say 'A') with the submitted source files of all the student assignments sorted alphabetically. They would then go in order and use the relevant 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 (say 'B') using the student's personal identification (PID) name, provided by Virginia Tech, as the file name. They would then move the students' source code from the source code folder to a third folder (say 'C') to keep track of the projects that are done grading. After all the files are graded and moved to the other folder (C), a professor written script is executed. This script goes through all the files in that folder and emails each student his/her file using the file name as the account name for the VT email server.

Besides having few statements or a paragraph as TA comments, the text file or the grade report also contained a header inserted by the TA that is placed usually at the beginning or at the end of the document. This header is needed to show the deduction summary and other information like the TA name, course name, etc., since this file is the only form of feedback to the students.

TAs sometimes wrote a header template in another text file so that they could copy and paste it in the grade report instead of typing the same header again in all files. All they would do then is fill out the missing fields in the header. TAs also included the scores outputted from any automated compilation tool as a part of the deductions summary in the header mentioned above.

The organization and movement of files in the three folders sometimes became cumbersome for the TAs. It gets confusing as well especially when the TAs need to go back to an already graded assignment to make any modifications. This method of grading also involves a lot of time to grade student assignments. TAs have to be meticulous and careful while writing the comments because they have to explain in a clear way where and why the points were taken off in the student's assignment. Having just one file to write the comments, indirectly makes TAs just provide overall general comments for the

student assignments. 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 wouldn't exactly point out the variable or file or function it occurred in. Another issue is consistency and fairness in assigning grades. TAs sometimes found it hard to remember how many points they took off for a previous student who had made the same mistake and thus land up taking off more or less points for the current students' assignment. TAs usually receives guidelines from the professor on the things to check during grading, but they still use their own discretion in assigning points to the students.

TAs suggested that it would be nice to have an online repository where the students' assignment files as well as the TA reports are saved and accessed easily. Since TAs did not like the fact that they had to manually total up the deductions and insert the headers in the text grade report, they preferred having a report file that had the header ready for them.

3.2.2 Using Adobe Acrobat to Markup Code

Adobe Acrobat is a stand-alone software used for viewing and creating pdf documents. The Portable Document Format (pdf) is the de facto standard for the reliable distribution and exchange of electronic documents and forms and it also preserves the font, images, graphics and layout of the source document [Adobe, 2004].

Two TA's out of the six interviewed, had used the Adobe method of grading before. Similar to the notepad way of grading, TAs maintained a folder of students' source code files in pdf format sorted alphabetically. The difference here is that the TA comments are entered directly in the pdf files using Acrobat features. Thus TAs do not have to maintain a separate folder of text files as grade reports. Adobe has a feature to enter comments in the form of a pop-up note. TAs used this feature to write comments on students' files and sometimes make this comment point to any line in the source code for emphasis. The disadvantage here is that these pop-up notes are not well embedded in the code 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 included inserting images/icons and highlighting lines. For example, a TA inserts a small bug icon to point out any bug in the code and highlights source lines to show warning or errors. The TA could do this for a single line as well as many lines together. Similar to the notepad way of grading, TAs have to move files to another folder as they finish grading. Finally, a script goes through all files and emails them to the students using filename as the VT user account. 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 copy and paste a header template in a pop-up note to show the score summary for each student's assignment file. However, they felt that the markup features of this software helped them provide good feedback to the students by making their comments more understandable and readable.

3.2.3 Using Web-CAT System's Text Block to Write Comments

Web-CAT, as we have seen earlier, is a web-based application that provides grading services to instructors and teaching assistants. The text block mentioned here is actually a text editor associated with a text area embedded in the Web-CAT grading subsystem (figure 3.3). It has many features for styling, coloring and formatting text.

Five TAs out of the six interviewed had used the Web-CAT text block method of grading before. TAs typed their comments in this block that got inserted as a part of the automatically generated grade report viewable by the student online. TAs made point deductions or additions by changing the point value in a small text box placed right above this text area. Any change in points made by the TAs got adjusted in the overall points obtained by the student for a particular assignment. The biggest advantage was that the TAs did not have to worry about organizing and saving students source files and report files. All they had to do was read the source code and make comments in the text block. Usually, in this kind of grading, TAs do not manually provide a summary of the overall point deductions for the assignment since the Web-CAT report displays that. They however can choose to give a summary of the TA point deductions by manually typing it in the textbox. The automated grading tools (for compilation, testing, etc) incorporated in Web-CAT and the instructor provided grading scripts greatly assisted the TAs with their grading. TAs only had to concentrate on fewer deeper issues with the students' design and readability of the code. The source code functionality and test coverage was done by these automated tools. Professors can specify other checks in the grading script like checking for naming, javadocs, style, etc. These reduced the workload of the TAs tremendously and they only had to check for efficiency and design mainly. Another good feature of this system is that TAs can look at the previously submitted assignments for a student. This helped the TAs to see if a student was making the same mistakes again and use that knowledge to decide on the point deductions to be made. But TAs found it annoying to go back and forth between the source code view and the text box view (figure 3.3, 3.4) since every request for either page went through the server. Depending on the internet connection and server load, the page loads took some amount of time. The notepad way of grading was preferred with respect to this issue since TAs were able to open up a window for the text file and for the source code simultaneously. TAs felt that simply writing textual comments did not get the students attention and thought that maybe not many students read the TA comments. In order to reduce the navigation time, TAs suggested having the source code and comment box on a single page. Some TAs enforced that the text block was important and should not be eliminated since it could be used to write overall comments for all the files in the assignment. Web-CAT's online report access feature eliminated the need to email students their grading report.

3.3 Conclusion

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, provide corrective solutions or explain things simply by writing textual comments in a separate

file. They also find it tedious to copy and paste comments that point out the common mistakes made by many students. TAs 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.

The comments and suggestions made by the TAs during the interviews proved helpful in deciding the different features that our solution system should entail. The following chapters discuss the interface and architecture of our solution system as well us the different methods used to evaluate our system.

Chapter 4: User Interface

Our grading tool or the 'new grading interface' is added as an additional grading service to the Web-CAT web application. One of the important goals of this interface is that it should be easy to use and should avoid any additional overheads in terms of learning its particulars. The biggest advantage of this interface is that it requires no installation and can be used simply by a browser. The grading process involves navigating through few wizard-like pages to select students' assignment, and using this new interface to mark up the individual source code files.

A sequence of dynamic HTML pages extensively supported by server-side java classes is presented to the user. These pages resemble the wizard sequences similar to those used during the logging and selection of assignments steps in Web-CAT. At each stage of the wizard, the user makes his/her selection that carries the task closer to completion. Status icons associated with each student's assignments is updated based on users' actions.

This chapter discusses mainly the front end, WYSIWYG interface of our new grading interface. This interface is easy to learn since it incorporates the standard features of an HTML form like buttons and drop down list. This interface currently works in internet explorer browser with JavaScript enabled. A hypothetical example is used to walk through the grading process from a TA's perspective.

4.1 Grading Process

Grading is an iterative process involving the following steps:

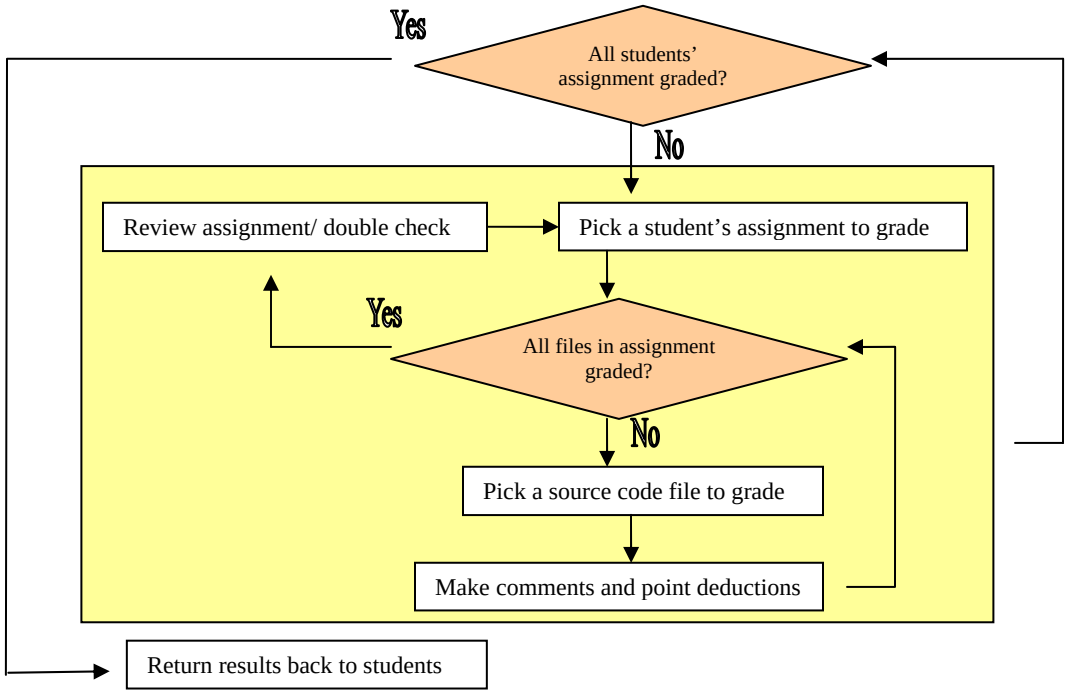

Figure 4.1 Pick a student's assignment

1. **Pick an assignment to grade**

   The grader has access to each student assignment submissions which is either made electronically or by submitting a paper printout of the source code. The assignments are stacked in order of submission, alphabetically or just randomly and the grader has to assess every assignment for correctness. The above process is repeated until all the students' assignments have been graded.

2. **Pick a file to grade**

   The assignment may have one or more source code files. The grader reads every file in the assignment either on the computer or the paper printout and makes comments on it. This process of reading the file and making comments is repeated until all the files in that assignment are completed.

3. **Make comments and point deductions**

   The comments are made by simply writing them using a pen or pencil on the paper or by typing them on a text editor or some other interface on the computer. The grader makes point deductions along with providing remarks for the assignment. This is the most time consuming process since the grader has multiple things to check for and various areas to cover. The grader then calculates the total score for the assignment and writes it somewhere to be used later.

4. **Review assignment/double check**

   This step may not be necessary but some TAs prefer to review the current students assignment after grading before he/she moves on to the next student. This helps the TA to double check and make sure that nothing has been missed out. At this time the TA can also re-calculate the score for that assignment. It helps build TA confidence.

5. **Returns assignment scores back to the students**

   Upon completion, the grader returns the graded assignments along with the individual scores back to the students. If the assignments were graded on paper, the grader usually saves student scores for records. Also, unlike the computer-way, the grader does not have a copy of the actual comments made during grading.

The goal of our method is to streamline this grading process using a grading tool that can help graders access student assignments and make comments and point deductions directly in the source files. This tool has other features to assist in solving any issues that can come up during the grading process. In a hypothetical example, we will see how our new grading interface achieves this goal.

4.2 Unique Features

The grading tool that we have developed has the following novel features:

- **WYSIWYG editor**

Our tool has a What You See Is What You Get (WYSIWYG) interface that allows TAs to click or select any source code line and insert a comment for that line on a click of a button.

- **History of comments made**

Our tool allows easy access to the twenty most recently used comments made by a TA for all assignments for all the students in multiple courses.

- **Access control for comments**

Access controls allow comments to be visible only to the instructor or only to the course staff.

- **Automatic updating of scores**

Our tool displays a summary of the point deductions and total scores assigned by the TA and other static tool checks for single and multiple files in a student assignment.

- **Resume grading at any time**

The tool has a unique feature to save the comments in the database once the TA saves and quits the application and to retrieve those saved comments when the TA returns to finish the grading.

- **Integration with the Web-CAT grading system**

Integration of our tool with the Web-CAT system makes it a comprehensive grading system that now manages TA feedback and point deductions in addition to automated testing and other grading services.

4.3 Example TA Grading Scenario

The following example supported by screenshots, will walk us through the steps a TA takes while grading the assignment using our new grading interface on Web-CAT. For the sake of this example we will call the TA, Hussein Vastani. Vastani is the TA for an introductory programming level course CS 1705 and uses the Web-CAT system to make comments on students' submitted program number 2.

4.3.1 Pick a Student Assignment

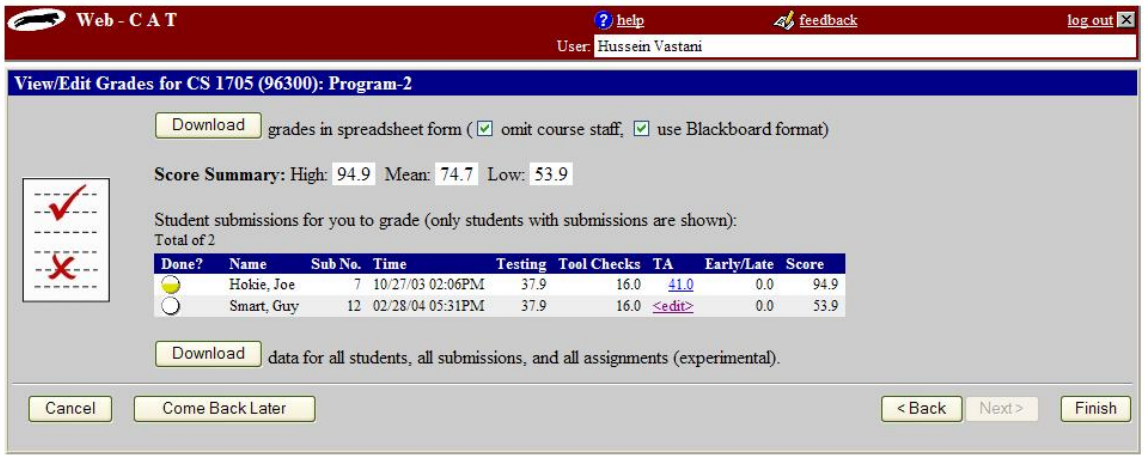

Figure 4.2 Screenshot of page with the student list

This is the wizard page that Vastani sees after logging in and selecting the right course and program number. It shows the list of students and the scores they received for the thoroughness of their code testing and by the automated tool checks. It also shows other information such as the number of submissions made so far, the time at which the last submission was made, as well as any early or late score. It shows the high, mean and low scores of all the students for that program in the course. There is also an option to download the scores in some format. In the above example, Vastani has already started grading Hokie, Joe's assignment and has given him a score of 41 points but hasn't finished grading the entire assignment yet. He now has an option of going back to grading Hokie, Joe's assignment by clicking on the score (41) underneath the TA column.

4.3.2 Pick a File to Grade

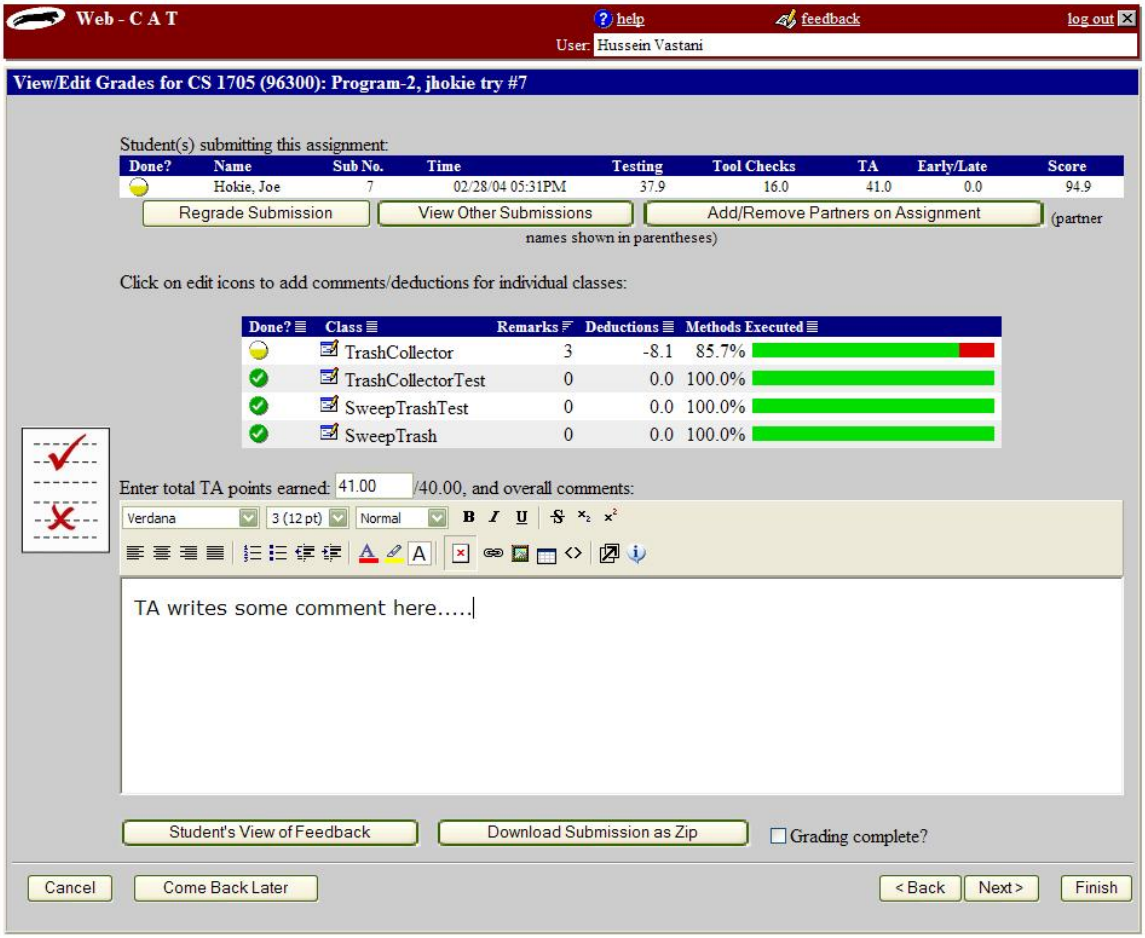

Figure 4.3 Screenshot of the page with the assignment information

This is the second step in the grading process after Vastani chooses to return to Hokie, Joe's assignment. On the top is the summary of the students score as seen earlier. This is followed by a table with links to individual class files in the assignment. The check marks next to the last three class files indicate that Vastani has already finished grading those files. The table also shows the number of remarks made by the tool checks and the TA, the deductions and the percentage of methods executed during testing of student code. At the bottom of the page is a text box where Vastani writes any overall comments he has for the assignment and edits the final score if necessary. Vastani can choose to view the previous submissions made by Hokie, Joe by clicking on the 'View Other Submissions' button. He clicks on the edit icon next to the first file (TrashCollector) to make comments for that file.

4.3.3 Make Comments and Point Deductions

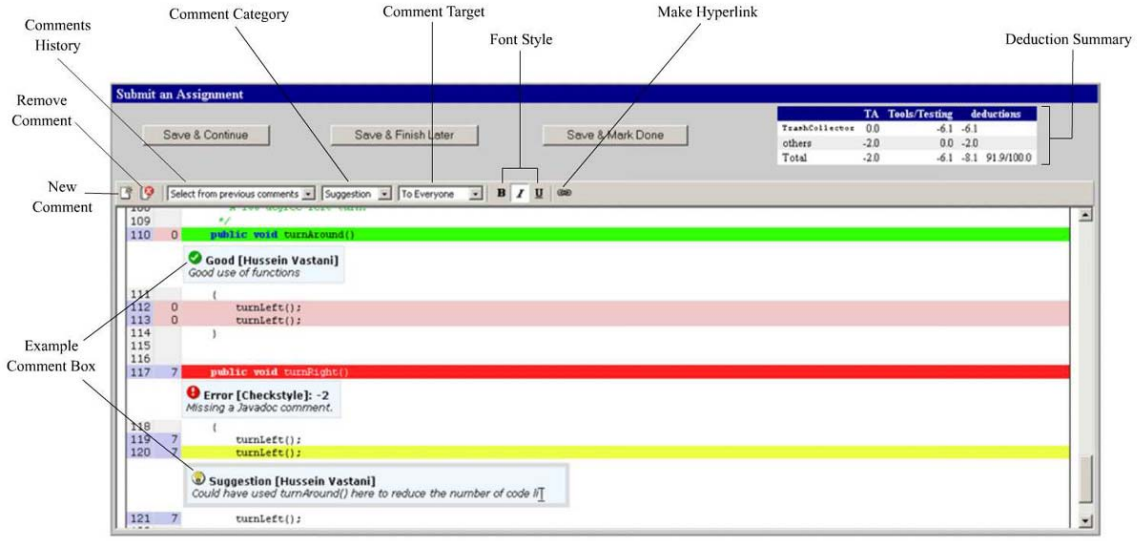

Figure 4.4 WYSIWYG editor (our new grading interface)

This is the WYSIWYG interface of our grading tool. The important features of the tool are marked with labels and their description and usage is explained below.

Description and usage of the interface controls:

New Comment: Click on any line or select any source code line and then click on this button to insert a comment box for that line.

Remove Comment: Select a comment box and then click on this button to remove the comment box from the display. A box pop ups for confirmation.

Select from previous comments **History List:** Use this drop down list to select from the twenty most recently used comments.

Suggestion **Comment Category:** Use this drop down list to select the category of the comment. The existing categories that can be selected are Error, Warning, Question, Answer, Good, Suggestion, Extra Credit ordered by severity.

To Everyone **Comment Target:** Use this drop down list to determine who can view the comment. The access or view rights of the comment can be set using this. Either Everyone, or Facultys/TAs or only Faculty can view the comments entered.

B *I* U **Font Style:** Use this button to switch font styles of the text in the comment box. The different styles are Bold, Italics and Underline.

Make Hyperlink: Use this button to convert the selected text to a hyperlink. First select the desired text from the message body of the comment box and then click on this button. In the dialog box that pops up, select the type and enter the URL. On pressing OK, the hyperlink is saved for the selected text.

Example Comment Box: Comment box can either be user generated or tool generated. The difference is the editable feature of the comment box. Automated tool generated comment boxes are never editable. The box mentions the kind of tool that generated the comment (PMD or Checkstyle). The editable feature of the user generated comment box depends on the access or view rights of the user making the comments. For example, a student performing a review of another student may not be able to edit the comments that were generated by the TA or Faculty. In most cases, the author of the comment can only edit (message and points) his/her own comments.

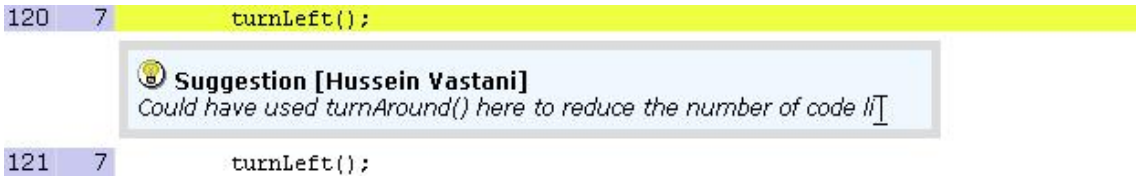

Figure 4.5 Example TA editable comment

Figure 3.5 is an example of a user generated comment box. It has been inserted for line 120 which has been executed 7 times and has the category 'Suggestion'. Each category has a color, an icon and points (not always) associated with it. The icon here is a small light bulb and line 120 has been highlighted with yellow color. If there is more than one comment for a particular line, the line is highlighted with the color associated with the category of the highest severity. A suggestion comment has no point deductions. The message here is editable. Notice the comment box has a thick gray border to show that it is currently selected. A comment box can be selected simply by clicking on it. Any changes can only be made when the box is selected. One can select another comment box or unselect this one by clicking outside the comment box. The name enclosed in angle brackets is the name of the person who inserted the comment, also called the author of the comment.

Deduction Summary: This table provides the user with information about the deductions made for the current file as well as other files in the project/assignment. The deductions made by the TA and the Tool & Testing along with the total points received are displayed.

	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 4.6 Example score summary box

Figure 3.6 is an example of the deduction summary table. There are no deductions made for the current TrashCollector java file. There is a total of 2 points deducted by the TA from the other files in the project. Thus the total of TA is -2.0. Tool/Testing has deducted 6.1 points from the current file and 0 points from the other files. Thus the total of the Tools/Testing is -6.1. Hence, the combined deductions made by TA and Tool/Testing for the current file is thus -6.1 and the combined deductions made by TA and Tool/Testing for the other files in the project is -2.0. The total deduction for the project is thus -8.1 (-2.0 + -6.1) and the overall score is a 91.9 out of a 100 possible.

Saving the comments: The comments can be saved in the database using the following buttons. One can choose to, either continue making more comments, come back later to add more comments or finalize the comments for a file. The categories that had no points associated with them (For example, Warning, Good, Question etc.) display a 0.0 point as default after the save operation has been performed.

On clicking this button, the comments made are saved and the user can continue editing comments on the same file**.**

On clicking this button, the comments made are saved and the user navigates back to the previous page. The status for this file is now unfinished.

On clicking this button, the comments made are saved and the user navigates back to the previous page. The status for this file is now marked done.

Vastani reads the source code of the student and enters any comments for any source line using the features described above. He then clicks on ‘Save and Mark Done’ to return to the previous page and chooses the next file. He does this until all the files have been graded.

4.3.4 Review Assignment/Double Check

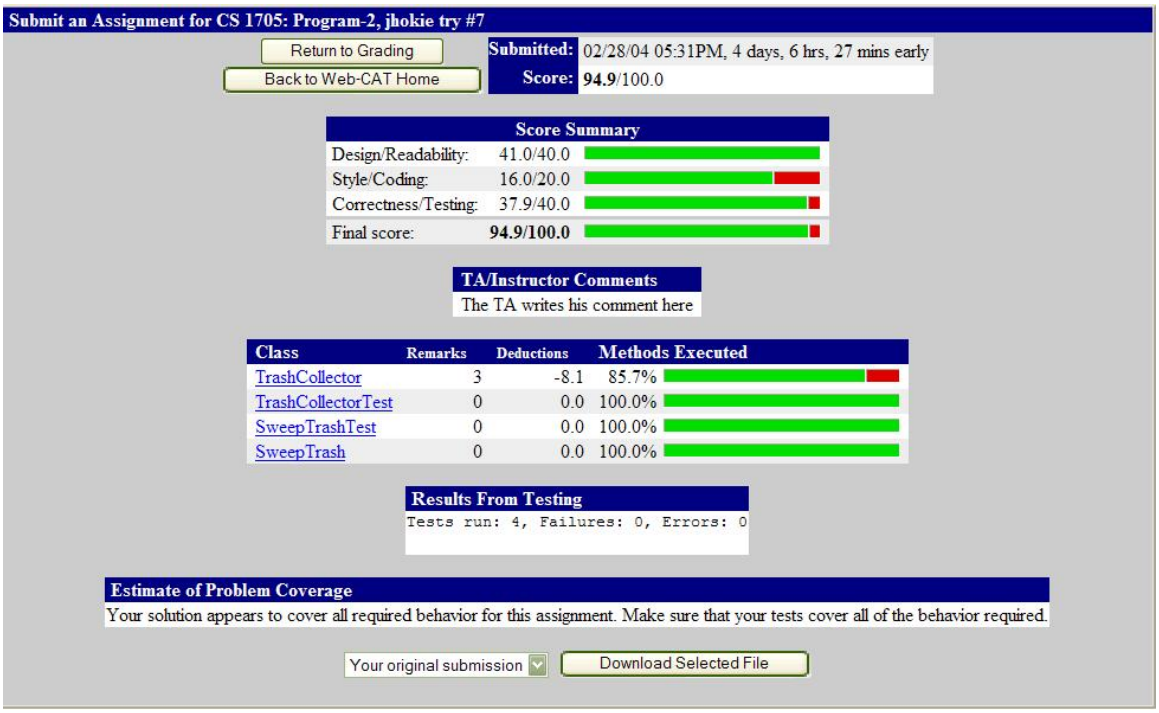

Figure 4.7 Screenshot of the final report page

Upon clicking View grade report, Vastani is directed to this page. This is the view of the report that the student also sees after the grading is completed. The page displays the score summary with respect to the Design/Readability, Style/Coding and Correctness/Testing. It has a table to show the comments that are written in the overall comment box. It also shows the list of classes which can be clicked to view the inline comments made by Vastani and the automated tool checks. It shows information on testing that was done on student code and a short description of problem coverage. Vastani can return to the grading page by clicking on the 'Return to Grading' button

4.3.5 Returns Assignment Scores Back to the Students

The advantageous feature of the Web-CAT system is that Vastani does not have to save and email the grade report back to the students or update any scores in a different spreadsheet. Once he is done with grading, all he does is check the finish grading check box and click on Finish. A notification email is automatically sent to the students informing them that their assignment has been graded completely and is available to view online. The student logs into the system using his username and password and navigates to view the grading report as shown above.

Thus we have seen how our tool streamlines 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 grading tool also supports 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 with Web-CAT soon.

Chapter 5: System Architecture

Our new grading interface is a part of the Grader/Curator subsystem of Web-CAT [Shah, 2003]. This subsystem is made of a three tiered architecture which includes the client tier, server tier and the database tier. The client tier is the web browser that displays the wizard-like pages to the user and accepts user inputs via HTML form elements. The server tier is where the logic resides. It has the java classes that support the generation of the dynamic web pages that are seen by the user. The database tier has the important tables to save the information related to courses, students and assignments.

In this chapter we will discuss briefly the Web-CAT system and how it supports the grading process. We will then discuss each tier of the grading subsystem in detail by answering questions such as, what it is that the grader sees, the logic of what happens and how the information is saved and retrieved.

5.1 How Grading Markup Fits Into the Web-CAT Application

Web-CAT is implemented on the WebObjects framework. WebObjects is a rapid application development environment with data access and page generation capabilities [WebObjects, 2004]. It is a powerful tool that is used to develop database applications with HTML interfaces. Applications in WebObjects are developed using the Java language. Use of WebObjects framework greatly helped in the development and deployment of our web application system on the server.

The Grader subsystem is responsible for grading student programs. It has a wizard based interaction that helps the user to navigate to different pages very easily. It has support for faculty tasks and student tasks and each successive step taken by the faculty or student brings them closer to completion of the tasks. When the student submits his/her assignment as a zip file, the system extracts the contents and runs the program based on the grading script provided by the instructor. Web-CAT system was designed with an intention to manage the test driven development approach taken by the students [Shah A., 2003] [Edwards, 2003]. Here students are responsible to write their own code and write test cases to test their code. The students' grade depends on the correctness, test coverage and other aspects like style, layout, naming, design and readability assessed by the teaching assistants.

5.2 Design Decisions in Formulating the Implementation

Following are the decisions made about the functionality of the grading process and the implementation of the interface:

1. Storing of TA comments in the database
2. Dynamically generating source code view
3. Implementing client side interface for editing of comments

4. Using WYSIWYG instead of HTML forms

5. Differential comment visibility for different users

TAs often tend to make changes and even erase the comments they have entered while grading student assignments. On paper, they may erase or cancel the comments or the point deductions they may have made. On an electronic copy, they may delete the text or the postit they have inserted. In order to allow TAs to flexibly enter and remove comments, we decided to save TA comments information in the database. Table 5.1 describes the structure of the database table that stores TA comments information.

Since TA comments were being stored in the database, the dynamic generation of the source code view was necessary. Whenever a TA attempts to view a source code file, his/her pre-entered comments for that file is retrieved from the database and embedded in the source code. The database is updated once the TA finishes making any changes to his/her comments on the source code.

However, the dynamic generation of source code view on the server tier would not be instantaneous. This created a need to minimize the request/response transactions as much as possible. We therefore decided to allow editing of comments on the client side in the users browser. The following options were considered while deciding the client side interface:

- **HTML forms:** We considered the option to have an HTML form and reload the page every time the user performed an action. However, this option would be very expensive since many request response will be made frequently accompanied by huge file transfer of the report file back and forth between the server and the client's browser. We rejected this idea because of its expensive operation, inefficiency and bad usability.

- **Java Applet:** The second option we considered was to write our own Java Applet to do the source code markup. We decided against it because of the possible modifications to the existing report file that may have been necessary to make to suit our purpose. This would reduce the reusability of these files. Also Java applets take too much time to load on clients' browser with slow internet connections. With respect to the implementation issue, we realized that even though applets are more usable than HTML forms, the code to do the markup using java applet would be very complex since our original report was fully HTML.

- **WYSIWYG:** We researched many open source projects on the internet [Google, 2004] [Sourceforge, 2004] and found a WYSIWYG editor [Interactivetools, 2004] replacement of the HTML textarea fields. This editor called 'HTMLArea' [HTMLArea, 2004] lets the user perform many functions that are usually included with any html editor. It uses pure Javascript to do this but works only on latest versions of Internet Explorer. Javascript is a client side scripting language and is fairly lightweight, fast loading and easily extendable [Interactivetools, 2004]. The

user can perform his/her actions where the functions to execute them are called locally. Since this was an open source project, we could easily learn and use existing code and make changes to it to suit our purpose. The code for generating and using the HTMLArea was long and a bit complicated to understand at first. We thus decided to use this WYSIWYG feature to allow direct editing of comments in the clients browser.

Our new grading interface uses the concept of HTMLArea [HTMLArea, 2004] with several additional functions added for insertions, deletions and editions of comment boxes. The interface embeds the generated HTML report into the HTMLArea inside a form with different elements like buttons and drop down list that interact with the actions of the grader. Upon every mouse click, several items in the form (history list, comment category, comment target, font styles) are updated keeping the user in context.

Request response transactions between the server and client browser are minimized since most of the editing is performed on the client side and the saving of comments is done on the server side. Even though the interface uses few form objects, it is comparatively more usable for the end user.

Finally, we decided to consider different classes of users that will use our grading interface. Since professors have more authority than the TAs, and the TAs have more authority than the students during grading, we decided to allow the users to hide their comments or details from other users depending on their role and purpose. The next section discusses the architecture in detail that supports the TA markups.

5.3 The Resulting Architecture for Markup Support

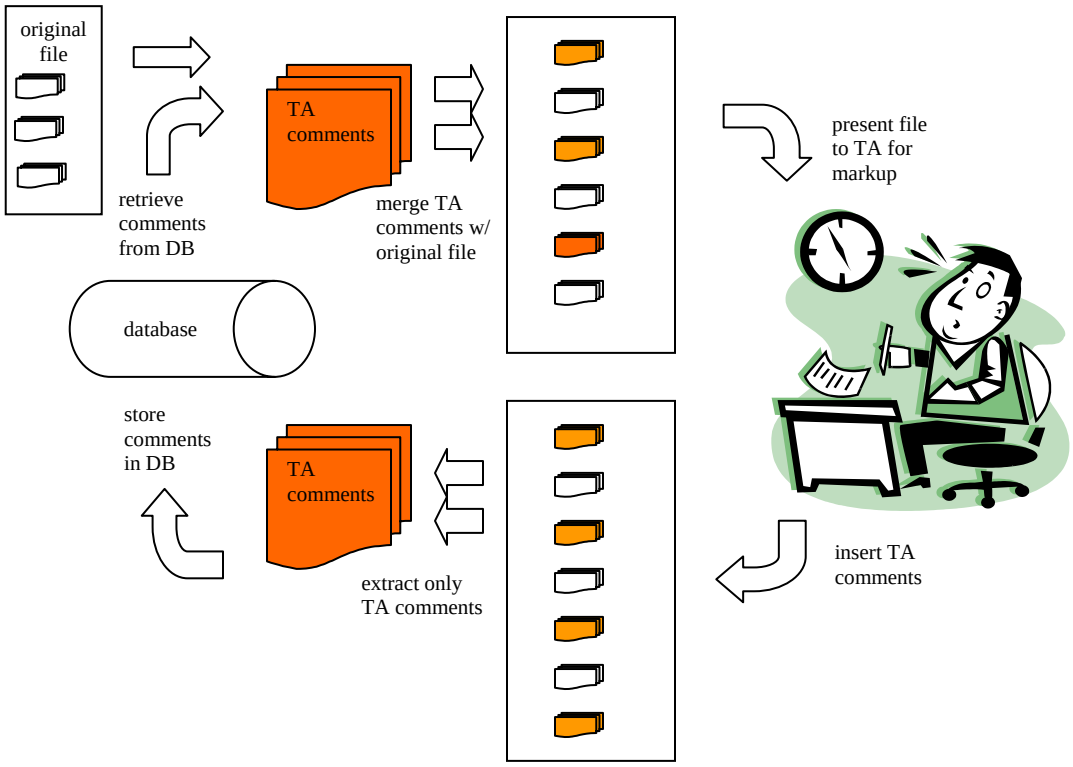

Figure 5.1 Architecture for markup support

Figure 5.1 gives an overall picture of the architecture that supports the TA markup. Following is the description of the architecture that supports the important steps involved in performing the TA markup.

5.3.1 Retrieve Comments from the Database

The database is a MySQL database and is a part of the third tier of the grading subsystem. WebObjects treats the database tables as java objects and the table rows are represented as arrays. TA comments are stored in the database. Each comment is stored as a row in the table. The following table shows the important attributes in the table that stores the TA comment information.

Attribute	Description
LineNo	Line number in the source code file to which the TA comment applies
Category	Category associated with the comment. *Error, Warning, Question, Suggestion, Answer, Good, Extra Credit*
To	To whom the comment is targeted. *To Everyone, To Faculty/TAs, To Faculty only*
Message	Actual text message body of the comment that the TA has written
Author	Name of the user who created the comment
Deduction	Point deduction made by the TA for that particular comment. This value is a floating point value

Table 5.1 Attributes and their description

Figure 5.1 shows the attributes of the most relevant table of the grading process. Other fields like the history list and the current user are stored in different tables. The actual retrieval of comments from the table is done at the server tier.

5.3.2 Merge TA Comments with the Original Marked Up File

The original files are stored on the server. Students submit a zipped archive of the java files via Web-CAT. Automatic assessment of the files is done using various automated and static analysis tools like Clover, PMD and Checkstyle. These tools mark up the source files by transforming them to HTML format and embedding their comments at the necessary locations in the HTML file. The server tier is responsible for merging the TA comments into these files. It has the required methods to perform the preprocessing of the files before the TA views it. We used JDOM [Hunter & McLaughlin, 2004], which is an open source library for XML manipulation, to traverse the HTML file. The comments that are read from the database are inserted one by one in the DOM tree for the appropriate line numbers. The comments entered are formatted to look like the comment boxes inserted by Checkstyle and PMD. After all the comments have been inserted, the document is then rendered as an HTML file using the appropriate JDOM functions. This document now has the TA comments in addition to any tool generated comments that were already present in the original file. To maintain consistency, all comment boxes have the same structure and properties (see Figure 4.4). Before a comment is merged into the HTML file, a check is performed to see if the user has the required privileges to view and edit the comment. The deductions made by the automated check tools (Clover, PMD and Checkstyle) and the TAs are retrieved from the database and recalculated to get the total of these deductions. The score summary that the user sees in the grading interface is then updated using these scores.

5.3.3 TA's View of the File

After logging in and navigating through the pages, the grader eventually is presented with the list of classes to markup in the student's assignment. Each java file is marked up by

Clover that generates a pretty printed version of the source code with color highlighted markup and embedded comments regarding students' test coverage. Lines highlighted in pink are the lines that have not been executed during testing. The comments generated by the static analysis tools such as Checkstyle [Checkstyle, 2004] and PMD [PMD, 2004] are also integrated in this report. These tools mainly check for stylistic or coding issues and highlight the lines in red for errors and yellow for warnings. These tools insert a comment box with the error description and zero or more point deduction for each violation. The first challenge was to merge the output from PMD and Checkstyle into the Clover report to have a unified report of the assessment done by these automated tools. This report is an HTML markup of the original source code and is generated for programs written using the Java language. Since our new grading interface is intimately dependent on the output generated by these tools, it currently works only for Java source code. The grader can then do his/her markup on this report using our WYSIWYG interface.

5.3.4 TA's Markup of the File

The grader or TA can use the features described in Chapter 4 and follow the same procedure to markup the java source code files. The comment boxes inserted by the author are editable by the author. He/She can type the message in the box and make point deductions (if needed) by typing the score. However, the comments generated by automated tools and the original source code itself are NOT editable by any user.

Each comment has a category associated with it and each category highlights the source code line with its respective color. For example, red for error, yellow for warning, etc. The categories are ranked in order of severance (Error, Warning, Question, Suggestion, Answer, Good, Extra Credit) and the source code is highlighted with the color of the most severe category. If the user deletes this comment, the color of the next severe category takes precedence.

The history list maintains the twenty most recently used comments. Every new comment inserted by the user is also added to the history list with the most recent one on top of the drop down list. If the user wishes to reuse a comment from the history list, the program treats it like a copy of the original comment and does not re-add it to the history list unless the copy is also modified. If an entry in the history list represents a single copy of a comment box, any modifications made to the comment box will simply update the values of that entry in the list and move it on top of the drop down history list.

The status icon associated with each file is updated based on TA's request to mark it done or leave it incomplete. Figure 4.5 shows an example comment box that a TA edits. This editable feature of the interface currently works only with Internet Explorer since other browsers do not support that feature yet.

5.3.5 Extract TA Comments from the Marked Up File

After a TA finishes marking up the source file, he/she can request the system to save the comments and return to the previous page. The entire marked up HTML file is sent back

to the server along with the current history list. The server tier is responsible for extracting the TA comments out of the marked up file and inserting them in the database. The comments in the database which are authored by the TA are first deleted before the new comments are inserted. Once again, the HTML file is traversed using JDOM functions and the TA comments extracted out. These comments are then transformed to be row objects in the table by setting the value of each attribute individually. As the records are created, they are inserted into the database one by one.

5.3.6 Store Comments in the Database

The important attributes of the table as we have seen in Table 5.1 are re populated with the new values that are extracted from the comment boxes. The server tier does the job of inserting the already extracted comments into the database. The system state is maintained and the appropriate page is displayed to the user.

5.4 Other Architectural Decisions

The merge-extract mechanism is repeated whenever any user with the right privileges requests to view the source code files for markup. After submission, when the student logs on to see the final feedback report, the dynamic generation of the report and the inserting of TA comments from the database into the source files are performed. Students can navigate the list of the source files from the report and view any TA comments embedded in those files. They however do not have the privileges to edit the comments made by TAs or instructors.

During peer review the complete merge-extract cycle takes place. The difference here is that during merging of the comments with the original marked up file, the comment boxes, not authored by the user, are made uneditable. Any point deductions associated with the TA created comments are hidden from the student who is not allowed to make point deductions for the new comment he/she inserts. Even the score summary is hidden in the student’s view in our grading interface. Peer review however, is still under development but our established architecture can easily be used to support peer markup.

Chapter 6: TA Opinions of the New Grading Interface

This is the first step of the evaluation process. After our new grading interface was developed and added to the Web-CAT grading system, the next step was to interview TAs that got an opportunity to use this new interface. In addition to the grading services provided by the Web-CAT system, our new grading interface helps TAs provide inline comments directly in the source code files. Inline comments can be made by simply clicking on or selecting the line to which a comment applies and clicking the insert comment button. TAs no longer have to remember and total the point deductions in their head. This new WYSIWYG interface also has a summary table of the point deductions made by the TAs and the automated tools like Clover, PMD and Checkstyle.

Four TAs were interviewed and were asked their opinions about our new grading interface. Two TAs were clear victims of the new software syndrome as they were reluctant to use the new system and continued using the old way of using the textbox to write comments. 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. The other two TAs used the new interface to provide few inline comments. They made few suggestions but were very happy overall with this new way of grading.

6.1 Features TAs Liked About the New Interface

Following are the features they liked or felt were advantageous and helpful about 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 (in Notepad and Acrobat way) 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. The header also displays the total of all the point deductions and the final score for the assignment. The header is placed in all the source files in the 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. Line highlighting with the insertion of a comment box also brought out the severity of the error and caught attention easily.

- **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.

- **Categorizing of comments**

Though some TAs didn't really care about categorizing the comments in the earlier methods of grading, few still felt that it makes it easier for the student to understand what the TA is trying to do or say. The TA could either write a comment as a suggestion or give a student a warning. For example, if a student happens to repeat an error that was clearly mentioned in class, the TA would give him a warning first and not take any points off. In the Textpad way of grading, the TA is unlikely to categorize the comment since it would just mean writing more text for him. In the Adobe way, some TAs highlighted the line with yellow color and assumed that students would interpret that as a warning. However, in our new grading interface, the TA can just select the comment category from the drop down list, which highlights the source code line with the appropriate color and displays the category name in the selected comment box.

- **Option to write comments for faculty only**

Often TAs need to discuss unethical behavior such as plagiarism in student assignments with the professor or have questions related to any changes the professor may have made in the specifications. In most circumstances, the TA would email the professor to clarify these, but with the access control feature in our new grading interface, the TA can directly write comments in the students' code and make it visible to the professor only. Once the professor is notified, he/she knows exactly what the TA is trying to say. In response to the TAs inquiries, the professor enters his/her comments and makes them visible to the TAs only. This two way textual conversation between the TA and the professor via the student's source code file is completely hidden from the student. The TAs who were interviewed did not get an opportunity to use this feature so far but are looking forward to use it in the future.

- **No hassle of emailing the students**

The biggest advantage of the Web-CAT system is that once the TAs have finished grading, the students can log on and check their grade report. This eludes the process of manually emailing students their grade report. Notifications to students have always been an important part of the grading process and our new grading interface has greatly reduced TAs grading time with respect to the same.

6.2 Suggestions Made For the New Interface

TAs had the following suggestions regarding the new grading interface but on the whole they really liked the interface and felt it will be really helpful in future grading.

- **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 postits 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.

- **Provide comments for a chunk 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.

- **Line highlighting overwrites clover report line highlight**

When a TA inserts a comment box, the source code line is highlighted with the related color to show the category of the comment that is selected from the drop down list of categories. This line highlight overwrites the color (pink) inserted by the Clover report generator tool to show code coverage and only a small part of it is displayed in the columns on the left. A TA felt that the color display in the columns on the left may not be that clear for students to understand and that they may miss out on this important information.

- **Cancel button next to the Save button**

The interface currently has options to 'Save and Continue', 'Save and Finish later' and 'Save and Mark done'. TAs suggested having a Cancel button as well. If they want to just view the file and quit without saving, the Cancel button would be appropriate to use. By clicking on Cancel, the status symbol for the file would not get updated. Currently this can only be done by logging out directly and not clicking any Save buttons.

- **Show information other than text in the history list**

Currently the history list displays the first twenty five characters of the message body of

the comments written by the TA. If the message text exceeds twenty five characters, an ellipsis is appended to the text message and displayed in the history list. Some TAs have suggested showing more information than just the message body to distinguish between comments whose first twenty five characters may be the same.

Thus, we have seen that even though the TAs who were interviewed did not get a chance to use the system thoroughly for a long time, they feel it has most of the features they have always desired while grading. It is expected that their suggestions (i.e., commenting chunk of code, line highlight issue, etc.) will be added as future work.

Chapter 7: Experimental Evaluation

The purpose of this experiment was to compare traditional paper based method of grading against the Web-CAT method of grading. Four TAs from introductory programming level courses (that were interviewed earlier) were asked to perform these experiments. Due to the lack of a large number of subjects to use for evaluation, the results obtained from the data do not have not much statistical significance yet. These experiments just lay the foundation for any future experiments that can be performed for better evaluation of our method. However, we have considered all combinations of inputs for our experiments using a factorial design.

A particular lab assignment from a previous semester's introductory programming level course was used for this experiment. This assignment had already been graded using the Web-CAT grading system and reviewed by TAs in that semester (writing comments on text block). We selected solutions from two different students who neither had a very high nor a very low score. They had a little less than the average score and had few comments that were made by the automated tools and the TAs. The same lab assignment was used this semester and our current TAs, had already finished grading these assignments by the time the experiments were conducted. They were thus familiar with the specification and the grading scheme.

The assignment involved writing a Java program to model 'Karel'- the robots actions [Karel et al, 2004]. For this assignment the students were required to write the java classes to model the robots actions and the test classes to test those actions.

7.1 Experimental Setup

It was necessary to plan out the experiments since we had two different assignments, two different methods of grading and four TAs to do the grading. Both assignments were printed on paper using a color printer and uploaded on Web-CAT system. In order to maintain anonymity, the two students' names were renamed to student1 and student2 for both paper and Web-CAT grading.

7.1.1 Paper Setup

Each student's assignment was printed twice using a color printer. Each page had the student's name and the java file name on the top. The first student had 11 java files with a total of 12 printed pages, and the second one had 8 java files with a total of 10 printed pages. Thus we had four total assignment solutions neatly stapled together.

7.1.2 Web-CAT Setup

Four courses were setup on the test version of Web-CAT using nonexistent Course Registration Numbers (CRN) but with the same course title and professor. Each of these courses was assigned a TA and had a single student's submission. The first two courses had student1's submission saved and the remaining two had student2's submission saved. TA accounts on Web-CAT were created with login names TA1, TA2….TA4 for the four

TAs. These TAs were assigned to the four courses already setup. TA1 was assigned to the first course, TA2 to the second course and so on.

The experiments were performed in the Computer Science graduate lab where the Web-CAT server resides with the test version of Web-CAT installed. There was a desk for the TAs to use for paper grading. The students present in the lab were informed about the experiments and were requested to maintain silence for a few minutes. The experiments were performed on separate days for the four TAs. The experiments did not have a time limit but the TAs grading was timed.

After the experiment, the TAs were asked which method of grading they preferred and why. Two TAs were asked to do grading on Web-CAT first and the other two on paper first. TAs were given the assignment specification and the grading criteria before the experiment started, to recollect what the assignment was about and the criteria to assign grades.

7.2 Experimental Design

A factorial design was used for evaluation. Four TAs were asked to perform the experiments. Each TA used one method on one assignment followed by second method on remaining assignment. This approach helped us counter balance the order. Our claim is that the Web-CAT method of grading helps the TAs to provide quality feedback in less amount of time as compared to the paper based method.

7.2.1 Controlled Factors

Following are the factors that remained constant throughout all the experiments.

- Location
- Computer and desk used
- Assignment specification and grading criteria
- Test cases provided by the students

7.2.2 Independent Variables

Following are the independent variables that changed through the course of the experiment but did not change over the course of all the experiments.

- Order of paper based assignment
- Order of Web-CAT assignment
- The assignment number

7.2.3 Dependent Variables

These are what we tried to measure for each assignment. They were expected to change for all the experiments

- Time it takes to grade an assignment
- Number of comments per assignment
- Number of words in the comment
- Categories the comments fall into
- Point deductions made by the grader

Following is the data that was collected based on the aforementioned criteria. Table 7.1 shows the three dimensions that were used during the experiment; the order of grading method, the type of the grading method and the type of assignment.

		Paper	Web-CAT
Assignment 1	Paper 1st	TA 4 *# TA comments*: 1 *time to complete*: 24min 35secs *# words in TA comments:* 5 words => 1 others => 5 checks, ,2 circles, 1 arrow, 2 underlines, 1 qt mark *categories*: naming, design, flow *points deducted*: 0	TA 1 *# TA comments:* 10 (8 in file, 2 overall) *# tool comments*: 11 *time to complete:* 15min 00secs *# words in TA comments:* 2-4 words => 5 5-7 words => 1 10-12 words => 1 > 20 words => 2 *categories:* naming, good, testing, warning, error *points deducted:* 1
	Web-CAT 1st	TA 3 *# TA comments*: 11 *time to complete*: 8min 0secs *# words in TA comments:* 2-4 words => 6, 5-7 words => 2, 9-13 words => 3 others => 32 circles, 7 arrows, 4 cross outs, 1 star, 1 check *categories*: naming, testing, design, efficiency, implementation *points deducted:* 3	TA 2 *# TA comments:* 10 (9 in file, 1 overall) *# tool comments*: 11 *time to complete:* 11min 25secs *# words in TA comments:* 2-4 words => 1 8-10 words => 1 10-14 words => 2 18-22 words => 3 > 22 words => 3 *categories:* design, testing, question, implementation, warning, suggestion, error *points deducted:* 1
Assignment 2	Web-CAT 1st	TA 2 *# TA comments*: 13 *time to complete*: 9min 55secs *# words in TA comments:* 1-3 words => 5, 4-6 words => 5, 9-10 words => 2 >20 words => 1 others => 6 circles, 2 arrows *categories*: naming, testing, design, commenting, bugs *points deducted*: 0.5	TA 3 *# TA comments:* 5 (2 in file, 3 overall) *# tool comments*: 4 *time to complete:* 11min 19secs *# words in TA comments:* 8-10 words => 1 14-17 words => 2 > 25 words => 2 *categories:* design, testing, naming, implementation, warning *points deducted:* 0, extra pts given 2.1
	Paper 1st	TA 1 *# TA comments*: 9 *time to complete*: 33min 10secs *# words in TA comments:* 1-2 words => 8, 14-16 words => 1 others => 35 checks, 22 labels, 2 circles *categories*: naming, testing, good *points deducted*: 0	TA 4 *# TA comments:* 0 *# tool comments*: 4 *time to complete:* 9min 00secs *# words in TA comments:* n/a *categories:* n/a *points deducted:* 0

Table 7.1 Comparison data of Web-CAT grading vs. Paper grading

7.3 Results and Discussions

Table 7.1 shows the comparison data of the two grading methods used by the TAs to grade two different student assignments. Despite the fact that the data may not be substantial enough, it did show interesting results. The most crucial factors that we tried to measure were, the time it takes for the grading process and the quality of feedback provided by the TAs. The grading time for the TAs was recorded from the point where the TA was given the student's source code printout till the time he/she finished writing comments. The grading time for Web-CAT was recorded from the time the TA entered his username and password to log on till the time he/she finished grading and logged off.

In the paper grading method, the time it takes the TA to return the document back to the student and update the scores for their own records was not measurable for the purpose of this experiment. Nonetheless, in the Web-CAT grading method, as soon as the TA finishes grading, the student is automatically notified by email that his/her assignment has been completely graded and is available to view online. The Web-CAT system, in this aspect, relieves the TA from the process of returning the students assignment and uploading/updating the scores.

From the data collected, we observed that TAs also took longer to grade the first assignment irrespective of the method used. TAs however, took much more time to grade the assignments on paper first as compared to Web-CAT first.

TAs wrote longer comments with Web-CAT than on paper. The total number of words in the comments combined for all the TAs was **377 on Web-CAT and 164 on paper**. TAs used more arrows and lines (approx 122) to point out mistakes on paper. In our new grading interface on Web-CAT, comments are placed underneath the source line to which they apply. The line is also highlighted with the category color of the comment. The TAs expressed that they preferred typing comments on a computer rather than handwriting them on paper.

In both methods of grading, the TAs made comments on similar categories like design, naming, implementation, testing, etc. TAs however felt that they could concentrate more on fewer and more important issues of grading in the Web-CAT method since the system already checks for functionality, testing and other instructor provided checks such as commenting, naming, etc using the automated and analysis tools. But they still had to double check if these automated tools took off a fair number of points for the mistakes. On the contrary, in the paper grading method, the TAs had to check for errors in all the areas since they were the only ones grading the assignment completely.

The total number of comments provided to a student using Web-CAT (TA comments + automated tool comments) was more than the ones provided by TAs on paper. The comments made by the TAs using our new grading interface not only pointed out the mistakes of the student but also suggested corrective solutions. TAs felt that they had more time to explain to the student why a particular mistake occurred.

The following figures show the data collected from the paper grading as well as the Web-CAT grading experiments. Figure 7.2 can be compared to Figure 7.3 to see the difference in the number of words written or typed in the comments for different categories by TAs only (for paper) and TAs + Tool checks (for Web-CAT).

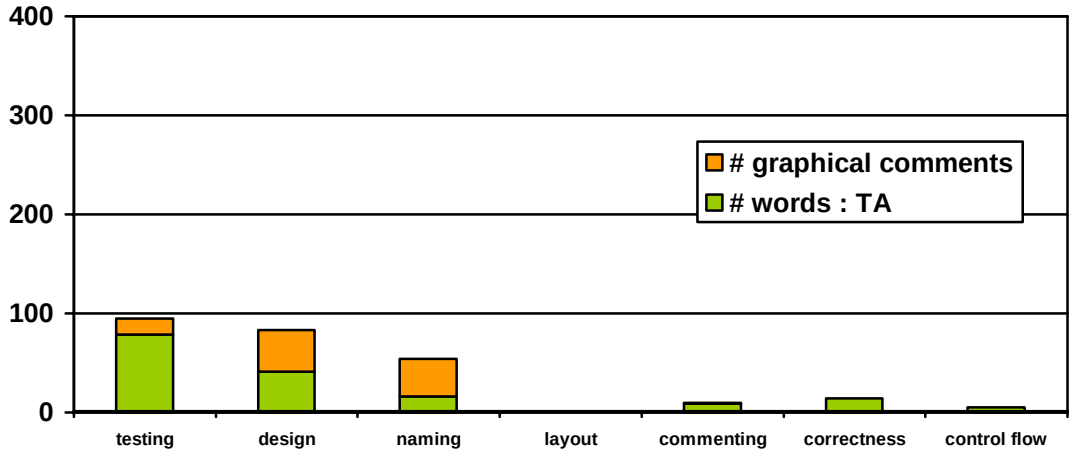

Figure 7.2 Data collected from TAs grading on paper

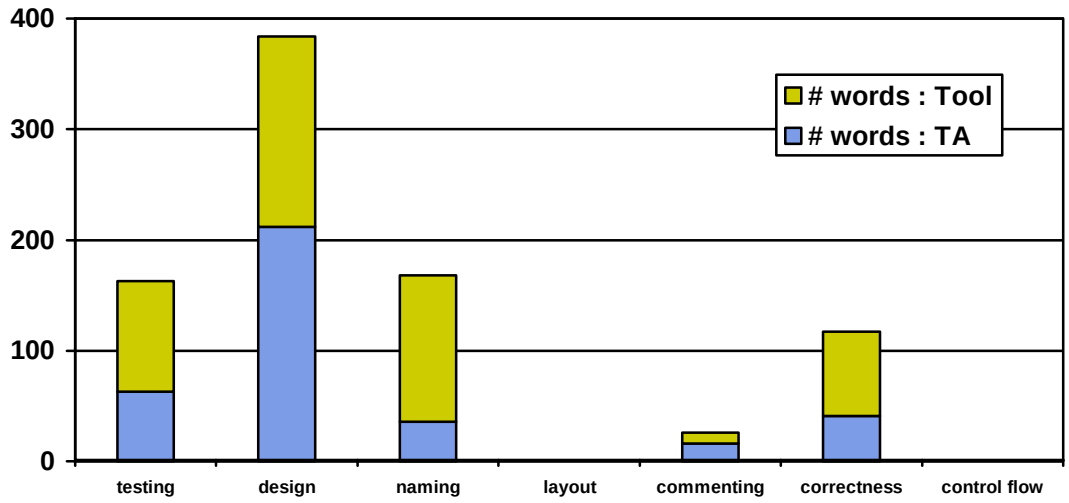

Figure 7.3 Data collected from TAs grading on Web-CAT

Figure 7.2 and 7.3 show the results of the data collected from the experiments performed by TAs grading on paper and Web-CAT respectively. They also show the number of words in the comments provided by TAs on paper and Web-CAT for the various categories. Figure 7.2 also shows the count of all the graphical comments (arrows, circles, lines, etc.) made by TAs on paper. In addition to the comments typed by TAs, Figure 7.3 shows the number of words provided by the automated tool checks in Web-CAT.

TAs provided more wordy comments for categories such as design, testing and naming. The number of words in comments provided by TAs related to design using Web-CAT is more than thrice than the ones made on paper. The number of words in the comments

provided by the automated tool checks is as much as the ones provided by TAs in Web-CAT grading.

The following figure shows the number of comments provided by the TAs per minute using Web-CAT and paper.

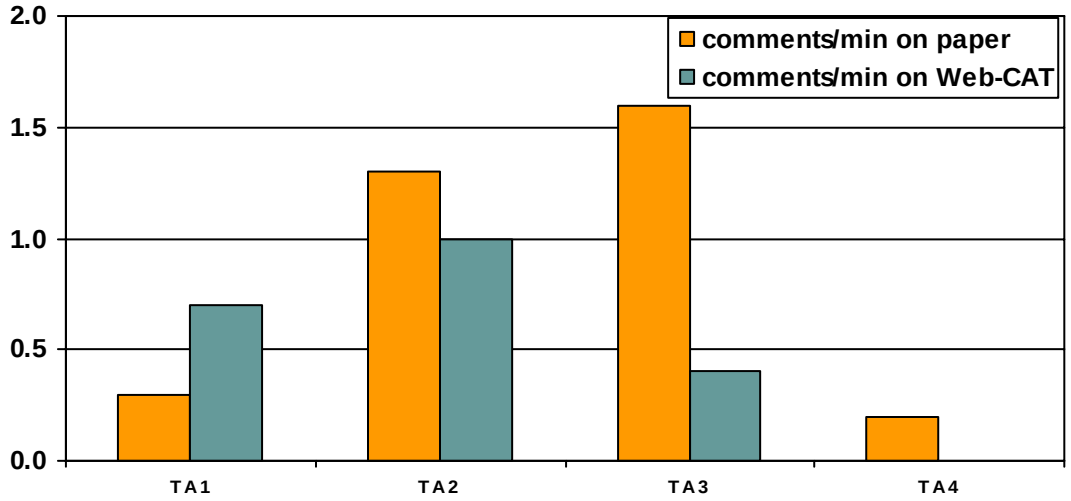

Figure 7.4 Average number of comments per minute made by four TAs on paper and Web-CAT

Figure 7.4 compares the average number of comments per minute made by four TAs using the paper grading method and Web-CAT grading method. For the Web-CAT grading method, only the TA entered comments and not the automated tool generated comments, are counted to compute the average. For the paper grading method, the graphical comments (i.e. circles, lines, arrows, etc.) that are not a part of the textual comments, are counted as separate comments. For example, a circled letter in a variable name, an underlined variable, a question mark next to a line of code, etc., that do not have textual explanations are counted as separate comments.

7.4 Conclusion

This chapter discussed the factorial design and the results of the experiments performed using teaching assistants. The purpose of the experiments was to compare traditional paper grading method and Web-CAT grading method. Four TAs from introductory programming level courses were requested to grade an anonymous student's lab assignment using one method and then another student's lab assignment using another method. The dependent variables measured were the time it took the TAs to grade the assignment, the number of comments made, the length of the comments and the categories which the different comments fell into. We observed that the total number of comments provided to the student using Web-CAT (inclusive of any automated tool comments) was more than the ones provided on paper. The comments provided using both methods were mainly on design, testing and naming but had more words typed by

the TAs in Web-CAT than the ones written by them on paper. Irrespective of the grading method used, TAs took longer to grade the first assignment.

Chapter 8: Survey of Instructor Opinions on Grading Practices

In order to gain an understanding of what professors do in other universities, an online survey was prepared and sent to the Special Interest Group on Computer Science Education (SIGCSE) mailing list. The main purpose of this survey was to elicit responses from the professors giving us an indication of their perception about the grading practices in their programming courses. The survey also aimed at gathering information about the expectations they have with respect to TA grading activities for programming assignments and the desired learning outcomes for students.

Questions on the survey were directed at getting feedback on the grading process followed by the grader of the programming assignments and comparison of different grading methods used by the professors. The data collected from the survey was used to perform a qualitative and a quantitative analysis of the same. The survey qualified from exemption from the Institutional Review Board (IRB).

In this chapter we will see the survey responses organized in tables for the ease of understanding and discussing responses along with actual sample responses provided by the survey takers.

8.1 Survey Evaluation

The survey was prepared using an online tool provided by Virginia Tech. [VT Survey, 2002]. It shows a formatted view of the survey's results which can be downloaded as an excel sheet as well. Sixty two people answered the survey questions and the values of the responses were analyzed for observing interesting results. The survey covered various aspects of grading. All the SIGCSE members and the computer science professors at Virginia Tech who answered the survey, had prior experience in grading student programming assignments. The following tables summarize the responses to the questions asked in the survey.

On average, how many minutes does the grader spend on each student's assignment?		On average, how many comments does the grader make on each student's assignment?	
5<	7 (11%)	5<	19 (31%)
5-10	16 (26%)	5-10	27 (44%)
10-15	21 (34%)	10-15	13 (21%)
15-30	12 (19%)	>15	3 (5%)
>30	6 (10%)		

Table 8.1 Survey Responses

The above table (Table 8.1) summarizes the responses to the questions regarding the average time spent and the average number of comments made while grading student assignments.

Section 1. Question	Strongly Disagree	Disagree	Agree	Strongly Agree	Average
I am satisfied with the quality of feedback students receive	2	16	27	16	2.44
I am satisfied with the quantity of feedback students receive	2	14	33	13	2.92
The feedback students receive is concrete and directed enough for them to know how to change or improve their performance	2	10	34	16	3.03
The grader regularly uses a written rubric as a grading method	4	14	24	20	2.97
The grader believes that the written rubric is useful to assess student work and provide necessary feedback	3	7	34	16	3.05

Section 2. Question	0 - 5	5 - 10	10 - 15	15 - 20	>20	Too much	Just right	Not enough
% of time on commenting	18	19	6	3	5	4	45	9
% of time on layout/indentation	35	13	2	0	1	3	45	10
% of time on naming	33	14	3	0	0	3	42	13
% of time on design	7	7	9	12	16	2	35	20
% of time on control flow	14	11	9	10	6	1	49	7
% of time on correctness/bugs	1	4	7	7	32	8	42	8
% of time on student testing	19	11	10	5	5	3	26	29

Table 8.2 Survey Responses (section 1 and section 2)

Section 1 in Table 8.2 summarizes the responses to the questions pertaining to feedback provided to students in their programming assignments by the grader(s). A four point Likert scale (Strong disagree, disagree, Agree and Strongly agree) was used to elicit opinions.

Section 2 in Table 8.2 summarizes the responses to the questions pertaining to the relative proportion (percentage) of time graders spend providing feedback on various categories. A “Too much”, “Just right”, “Not enough” scale was used. It also shows the number of responses for each percentage range.

Section 3. Question	Not at All	Very Little	Some-what	To a Great Extent	Average
Too many assignments to assess	5	10	22	25	3.10
Not enough time or resources to do a thorough job	1	9	29	23	3.19
Technical knowledge, capabilities, or experience of the grader(s)	27	15	17	3	1.94
Lack of a consistent rubric for grader(s) to follow	25	22	11	4	1.90
Poor code readability of student code	2	21	29	10	2.76
Poor layout and indentation of student code	6	27	24	5	2.45
Little or no commenting within the student code	5	24	25	7	2.56
The density of defects (bugs) in the student code	1	21	22	15	2.86
Poor testing of work by the student before submission	2	9	28	22	3.15
The logistics of executing code against instructor-provided tests to see if it works	12	17	20	12	2.85
Managing the submission of assignments and the return of the results	17	18	21	6	2.26
Section 4. Question	**Strongly Disagree**	**Disagree**	**Agree**	**Strongly Agree**	**Average**
An archive is valuable for resolving grade disputes	3	13	30	13	2.90
An archive is valuable for providing a backup of student work	4	9	34	12	2.92
An archive is valuable for detecting plagiarism/cheating	1	8	32	19	3.15
An archive is valuable for learning outcomes assessment for my course	4	14	32	9	3.28
An archive is valuable for longitudinal curricular assessment over multiple courses	2	12	36	10	2.9
An archive is valuable for educational research	2	12	36	9	2.88

Table 8.3 Survey Responses (section 3 and section 4)

Section three in Table 8.3 summarizes the responses to the questions pertaining to the impediments grades face while providing feedback on student programming assignments. A “Not at all”, “Very Little”, “Somewhat”, “To a Great Extent” scale was used for rating the options.

Section four in Table 8.3 summarizes the responses to the questions pertaining to archiving electronic copies of student submissions and the value of such an archive. A four point Likert scale (Strong disagree, disagree, Agree and Strongly agree) was used to elicit opinions.

Table 8.4 (below) shows the number of responses and the percentages for the multiple choice questions asked. These questions focus on learning the grading process (submission, assessment, feedback) from the students' and instructors' perspective.

How do students submit their programming assignments for assessment? (check all that apply)

They turn in a source code printout	32	(52%)
They give an electronic copy directly to the instructor or grader (on disk, by e-mail, using a shared directory, by ftp, etc.)	31	(50%)
They use a course management system (Blackboard, WebCT, etc.) to upload electronic copies	20	(32%)
They use an electronic submission and/or grading system specifically for programming assignments (describe it briefly below)	25	(40%)
Other (please elaborate)	5	(8%)

How does the grader read student submissions while grading? (check all that apply)

Using a paper printout	36	(58%)
Directly accessing a source code file at a computer	39	(63%)
Using an interface provided by an electronic submission and/or grading system	17	(27%)
Other (please elaborate)	3	(5%)

How is the student work assessed for correctness? (check all that apply)

By reading the source code only	27	(44%)
The student provides a printout of test results	21	(34%)
The student provides a live demonstration for the grader	12	(19%)
The grader hand-executes the student program against instructor-provided data	35	(56%)
An automated tool compiles and executes the student code against instructor-provided data	23	(37%)
Other (please elaborate)	8	(13%)

How does the grader provide comments to the student? (check all that apply)

By writing on a paper printout	40	(65%)
By typing comments in a separate feedback document that is sent back to the student	26	(42%)
By marking up an electronic representation of the printed work (for example, a PDF version of a source code printout)	3	(5%)
By marking up an electronic version of the source file (embedding textual comments directly in the code)	13	(21%)
Other (please elaborate)	11	(18%)

Besides using plain text, which of the following markup techniques are used by the grader to provide more understandable feedback in your course? (check all that apply)

Just written comments in plain text	49	(79%)
Using arrows, lines, or circles to point out source code issues	34	(55%)
Highlighting source lines with color or underlining	18	(29%)
Using icons or stamps	2	(3%)
Color-coding written comments (by category, severity, or author, for example)	3	(5%)
Other (please elaborate)	6	(10%)

Do you maintain an archive of electronic copies of student program submissions for your course?

No	18	(29%)
Yes, for all assignments in this course this semester/term	20	(32%)
Yes, for all assignments in this course over multiple semesters/terms	5	(8%)
Yes, for all assignments in multiple courses over multiple semesters/terms	19	(31%)

Table 8.4 Survey Responses (multiple choice)

8.2 Discussion of Responses

More than seventy percent of graders use a written rubric to grade student assignments since they feel that the rubric is a useful way to assess students' performance and provide feedback. Graders feel that there are too many student assignments to assess and less time and resources to help them provide necessary feedback. Poor testing done by the student and the bugs in the student code also create impediments for the graders. About seventy percent of the graders agree that managing of student assignments and return of results is an impediment in their grading process. It is evident from the results that most of the graders feel that archiving student assignments is valuable even though almost thirty percent of the responses say that they do not currently archive students' submitted assignments.

The following charts shows the percentage of graders that spend too much time and not enough time grading on categories such as student testing, design, naming, layout, commenting, correctness and control flow.

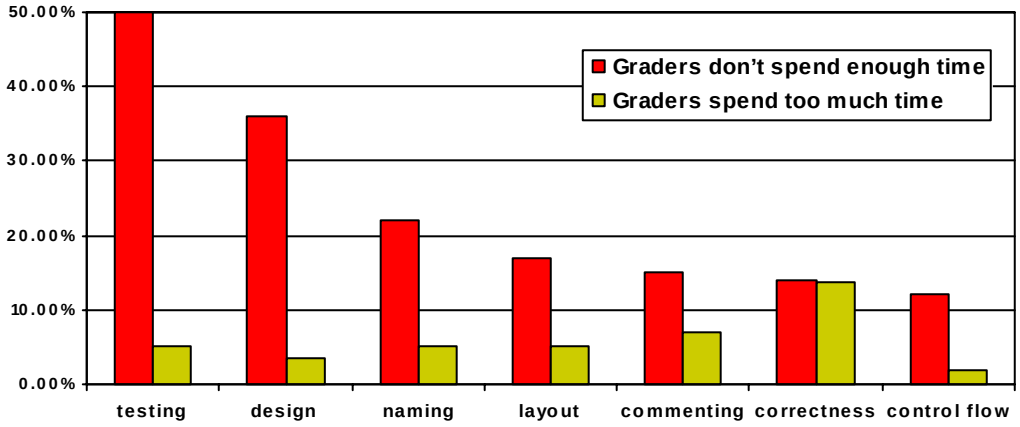

Figure 8.5 Percentage of graders that spend too much and not enough time on the listed categories

Figure 8.5 covers some interesting results. Almost fifty percent of the graders feel they do not spend enough time on student assignment testing and 36 percent feel they do not spend enough time grading on design. On the other hand, around fourteen percent of graders spend too much time on grading for correctness/bugs of the program and seven percent of them spend time checking if the student has done adequate commenting of the code.

Thirty two out of sixty two responses claim 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 electronic submission system. These systems are either locally designed web-based systems or one of the popular ones such as WebCT [WebCT, 2001] or Blackboard [Blackboard, 2004]. The professors write their own UNIX based perl/shell scripts that students use to submit their assignment directly to the professors allocated folders online. Similar observations were made from the following responses:

"We use the 'submit' command, which sends files from the student to a predefined directory (with permissions set accordingly) where the grader can evaluate the assignment"

"Shell script that runs on Unix server. Effectively equivalent to what CMS turnins provide except it's much more flexible, extensible, and easier to use for the instructor than turning on a CMS (though I've only compared with Blackboard, not others)."

About sixty five percent of graders write their comments on a paper printout and primarily use plain text and sometimes arrows, circles and lines to point out source code issues. The following sample responses show how the graders provide comments to the students and what other techniques they use.

"By providing a separate hand written set of comments. Sorry it is not typed simply as a time factor issue. If we had a marking system that allowed for standard comments then that would be used but we don't."

"Some embedded text comments include a link to a Web document that contains the instructor's solution at the relevant line in the instructor's solution source code. Students may access their graded assignment document via a web page from my course home page that brings up their graded work in a window with 2 frames. The link to the instructor's solution displays in the 2nd frame so that it appears side-by-side with the student's solution."

"The grading rubrik outlines all the essential elements to be graded. The students receive a copy of the rubrik when the program is assigned. The students received a marked up copy of the grading rubrik when the assignment is returned. Specific items are circled."

"I often copy and paste portions of the student's code into the feedback document to elaborate on my comments. This is particularly true for indenting comments.."

"Short comments (one to three lines) are posted to the student's score file. Longer comments, if necessary, are entered in a copy of the program and e-mailed to the student.."

"Some feedback is given verbally, during the demo."

To assess student assignments for correctness, many graders either hand-execute the assignments against instructor provided data or have a software to execute the file(s) and compare the results with the professors' output. Other interesting responses include

"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."

"Grader test cases and (visually/mentally) reasoning about the source code (to verify the meeting of the pre and post conditions of functions and class invariants)."

8.3 Discussion of Open-Ended Questions

A qualitative analysis of the survey takers responses in the open-ended questions revealed a number of trends and perspectives.

When asked about the significant impediments graders face while providing satisfactory feedback on students programs, thirty percent of them blamed it on the size of the class and the assignment. The following sample responses give an indication of other typical impediments faced by the graders during grading.

> "*Graders that do not have the experience or expertise to be able to adequately assess the student's program is our biggest challenge.*"

> "*In CS1, the students need to do several programs. Therefore, there is a lot to grade. They don't read the directions very well, and their code often has the missing parts. Coming with the rubric can be tough. How much does it count to include or omit some feature? We want to encourage but also be helpful. I spend little time on the good programs but a lot of time on the poor ones even though those students won't be continuing in the program.*"

> "*I think that providing students with test data is a very poor idea, do they do this in the 'real-world'. Students do not learn how to test thoroughly.*"

> "*For the programming project, however I don't want the students to focus only on right answers. I want them to consider design, readability, etc. These are hard things to automate.*"

The survey also asked which part of the grading process they would like to automate and why. Many preferred to automate the testing process.

> "*Would automate the making of a common feedback template for all students for each assignment.*"

> "*Direct testing – Doing so would allow more time to be spent evaluating the other parts of the assignment (documentation, style, organization).*"

> "*Running test cases only. Other parts should not be automated or you will loose touch with what the students are having trouble with. There's also more chance for catching cheating if the same person grades code manually.*"

> "*Submission and verification of code.*"

> "*I'd like truly convenient electronic markup and assignment return, and I'd really like a highly configurable automated program tester (for multiple languages).*"

Another important open-ended question aimed at learning the biggest disadvantage of the grading method used by the grader and why. Almost half of the responses were that their grading process was time consuming. Other responses were:

> "It *is slow enough that inconsistencies can creep in even though a rubric is used*"
>
> "*Time consuming and subjective ~ high variability*"
>
> "*I accept via email, so there are several steps to get it saved, headers trimmed, compiled, tested. An automated tool would save time for me.*"
>
> "*It is a pain to print out all their programs…*"
>
> "*With no e-copy of the source code, if there is a question that arises during grading we cant just run the program again (unless we ask the students for a second demo, which I have done on occasion). Another small problem is the possibility of a student forging initials on the cover sheet to indicate that a program has been demo'd.*"
>
> "*Feedback to the students is poor: don't know why the testing failed or why they lost points due to design/style. The grader also can't make comments on how to improve these design/style aspects.*"
>
> "*Sometimes, when there isn't time for student demos or TA evaluations, the student's grades come from the Curator. This does not happen often, but when it does, no one examines the code or executes the program, and the student gets a number back from a computer. Not a personal or thorough evaluation of the project in my mind.*"

Finally, the survey attempted to get the graders opinions on the most time consuming and difficult part of the grading process. One third of them felt that reading the code to decipher the errors was the most time consuming process. The graders felt it was difficult to find errors and defects in the code that sometimes tend to work. Graders want to know the students' thought process by trying to understand the logic of their source code, but find it difficult since the code is not always adequately documented. Testing students' code is another challenge faced by the graders. Other sample responses with respect to the same are:

> "*Hand checking student output and making comments on style, bugs, etc. I sometimes spend much time trying to find the source of logic errors. Also, grading programs whose output is graphical (eg Java, Tcl/Tk) takes more time as does grading interpreted programs (eg Lisp, Prolog).*"
>
> "*Blackboards upload and download is cumbersome, particularly for huge classes like mine..*"
>
> "*Assessing design and making suggestions..*"

"*Reading code. Students do not seem to put a lot of thought into their programs. Thus, the resulting code is often difficult to follow.*"

"*Grading documentation and style..*"

"*Having TAs read the code and write down comments on paper copy..*"

"*Checking design (because of inherent difficulty of applying clear and consistent grading criteria to multiple design choices possible).*"

"*Writing effective comments. Because they're lengthy..*"

8.4 Summary of Responses

Thus we have seen that despite the numerous developments in systems that automate the grading process, many instructors still prefer to use the paper grading method. The reason could be that most of the automated systems are developed in-house and are shell or UNIX scripts that are not portable to different machines. Graders still find it difficult to grade assignments in big classes and prefer to automate the testing process if possible. Most of the graders do the code markup on paper and many feel that they do not get enough time to grade for code design and student testing. Web-CAT encourages test driven development and has an automated feature to thoroughly test student code for coverage and functionality. Our new grading interface, which is a part of the Web-CAT grading system, allows TAs to markup student code directly by providing inline comments. It also automates the return of the results back to the students.

Chapter 9: Conclusion and Future Work

The purpose of this thesis was to research a way to streamline the grading process of the instructors and teaching assistants by providing them with a grading interface to enter comments directly in the students' source code file using a WYSIWYG editor. We interviewed teaching assistants of introductory computer science courses and asked them to compare this approach with the earlier methods of grading. We performed experiments using TAs to compare the traditional paper based grading method with the Web-CAT grading method. Finally we prepared and sent out a survey to the SIGCSE mailing list to get the perceptions of different professors in various universities about the grading process followed by them or their TAs. The following sections present an overview of the results of the analysis that was performed, as well as contributions and suggested directions for future work.

9.1 Summary of Results

TA interviews were very helpful in the sense that experienced TAs gave their perception of the different grading methods they had used and the drawbacks of those methods. The suggestions made regarding what they thought would improve their grading, are all incorporated in our new grading interface in Web-CAT. TAs who got the opportunity to use our new grading interface felt that the features it has will eventually benefit them or other TAs when they have large number of assignments to grade.

The experiments to compare paper grading and Web-CAT grading performed by the TAs, though not substantial enough, revealed interesting results. The total number and the length of the comments made by the TAs using our new grading interface were longer than the ones they had written on paper. It was evident from the comments made using our new grading interface that the TAs not only pointed out students' errors, but also suggested appropriate corrections. The grading report on Web-CAT had feedback comments provided by static analysis tools as well. Regardless of the grading method used, it was noticed that the first assignment took more time to grade than the others. However, the grading on Web-CAT also includes the return of the results back to the student instantly which is not the case in paper grading.

The analysis of the survey questions that were relevant to different areas in the grading process revealed interesting results. The size of the class and the lack of resources and time are the major impediments to providing feedback faced by the graders. Many graders felt that archiving students' assignments is very valuable. Forty seven percent of the graders felt that they do not spend enough time on students testing and thirty two percent felt that they do not spend enough time grading on design. Submitting source code as printed copies prevails despite the emergence of automated electronic submission systems. A few disadvantages of the current methods employed by graders include, but are not limited to student code readability and the length of grading time.

9.2 Contribution

We provided a solution to the problem many busy instructors face while grading programming assignments in huge classes. Our new grading interface will help instructors and teaching assistants mark up student code directly by entering comments using a WYSIWYG interface. This interface is incorporated in the Web-CAT grading system that already provides various services such as submission and archiving of students' electronic assignments, assessing the code for correctness and completeness and integration of static tools to check for other grading issues like style, naming and commenting. With all these features at hand, the instructors and TAs can easily focus on the deeper and more important issues of grading such as checking for design readability and efficiency. This novel approach will help them provide quality feedback to students in a timely manner. Therefore, we have suggested a mechanism to help instructors provide quality feedback to the students and help them learn better while still in the process of implementing their programs.

9.3 Future Work

This work can be extended to incorporate other features as suggested by the TAs during the interviews. One of the major suggestions made was to add a feature, which allows comments made by automated tools such as PMD and Checkstyle to be collapsed or hidden while the TA is grading. Another feature that can be easily added to our current interface is the further additions of categories such as design, documentation, style, naming, structure, etc. The TAs can specify what category their comment falls under.

Our new grading interface currently works only on assignments implemented in Java. One of the biggest extensions to this research would be to support markup for other programming languages as well. Our interface depends on a certain format of the report file, which is currently created by the Clover report and a few self written perl scripts. If other programming languages were to be embedded in HTML in a similar fashion, our interface would work without major modifications.

Lastly, further experimental evaluations using TAs can give us substantial amount of results to confirm our claim that Web-CAT grading is faster and provides better quality feedback than paper grading. We have already completed work in making the experimental design. Any future work can utilize this design and perform more tests to get convincing results.

.

Appendix A

Grading students programming assignments

This survey should take you only a few minutes to fill out. Please answer from the point of view of the course you are teaching this semester/term that involves programming assignments. If your current course does not involve any programming assignments, then please answer from the point of view of the programming course you last taught.

In the following questions the term "grader" refers to any instructional staff responsible for marking up or making comments on students programming assignments for the purpose of grading or reviewing.

If you are the grader, please answer the questions for yourself. If you have appointed someone to do part or all of the grading (e.g., a TA), then answer the questions with respect to the expectations you have with your grader.

On average, how many minutes does the grader spend on each student's assignment?

☐ 5< ☐ 5-10 ☐ 10-15 ☐ 15-30 ☐ >30

On average, how many comments does the grader make on each student's assignment?

☐ 5< ☐ 5-10 ☐ 10-15 ☐ >15

Please rate how strongly you agree or disagree with the following statements about feedback or comments provided to students on their programming assignments in your course (use Strongly Disagree, Disagree, Agree, Strongly Agree scale):

I am satisfied with the quality of feedback students receive

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

I am satisfied with the quantity of feedback students receive

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

The feedback students receive is concrete and directed enough for them to know how to change or improve their performance

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

The grader regularly uses a written rubric as a grading method

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

The grader believes that the written rubric is useful to assess student work and provide necessary feedback

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

Rate the relative proportion (percentage) of time graders spend providing feedback on each of the following items, and classify each as too much, just right, not enough:

% of time on commenting		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on layout/indentation		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on naming		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on design		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on control flow		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on correctness/bugs		☐ Too Much	☐ Just Right	☐ Not Enough
% of time on student testing		☐ Too Much	☐ Just Right	☐ Not Enough

---

Please rate the degree to which you believe that each of the following is a **significant impediment to providing effective feedback on student programming assignments:**

Too many assignments to assess

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Not enough time or resources to do a thorough job

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Technical knowledge, capabilities, or experience of the grader(s)

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Lack of a consistent rubric for grader(s) to follow

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Poor code readability of student code

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Poor layout and indentation of student code

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Little or no commenting within the student code

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

The density of defects (bugs) in the student code

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Poor testing of work by the student before submission

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

The logistics of executing code against instructor-provided tests to see if it works

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

Managing the submission of assignments and the return of results

☐ To a Great Extent ☐ Somewhat ☐ Very Little ☐ Not at All

In the space below, describe any other factors you believe are significant impediments to providing satisfactory feedback on student programming work:

How do students submit their programming assignments for assessment? (check all that apply)

☐ They turn in a source code printout

☐ They give an electronic copy directly to the instructor or grader (on disk, by e-mail, using a shared directory, by ftp, etc.)

☐ They use a course management system (Blackboard, WebCT, etc.) to upload electronic copies

☐ They use an electronic submission and/or grading system specifically for

programming assignments (describe it briefly below)

☐ Other (please elaborate)

Use this space to elaborate or clarify your answer, if necessary:

How does the grader read student submissions while grading? (check all that apply)

☐ Using a paper printout

☐ Directly accessing a source code file at a computer

☐ Using an interface provided by an electronic submission and/or grading system

☐ Other (please elaborate)

Use this space to elaborate or clarify your answer, if necessary:

How is the student work assessed for correctness? (check all that apply)

☐ By reading the source code only

☐ The student provides a printout of test results

☐ The student provides a live demonstration for the grader

☐ The grader hand-executes the student program against instructor-provided data

☐ An automated tool compiles and executes the student code against instructor-provided data

☐ Other (please elaborate)

Use this space to elaborate or clarify your answer, if necessary:

How does the grader provide comments to the student? (check all that

apply)

- ☐ By writing on a paper printout
- ☐ By typing comments in a separate feedback document that is sent back to the student
- ☐ By marking up an electronic representation of the printed work (for example, a PDF version of a source code printout)
- ☐ By marking up an electronic version of the source file (embedding textual comments directly in the code)
- ☐ Other (please elaborate)

Use this space to elaborate or clarify your answer, if necessary:

Besides using plain text, which of the following markup techniques are used by the grader to provide more understandable feedback in your course? (check all that apply)

- ☐ Just written comments in plain text
- ☐ Using arrows, lines, or circles to point out source code issues
- ☐ Highlighting source lines with color or underlining
- ☐ Using icons or stamps
- ☐ Color-coding written comments (by category, severity, or author, for example)
- ☐ Other (please elaborate)

Use this space to elaborate or clarify your answer, if necessary:

Do you maintain an archive of electronic copies of student program submissions for your course?

- ○ No
- ○ Yes, for all assignments in this course this semester/term

☐ Yes, for all assignments in this course over multiple semesters/terms

☐ Yes, for all assignments in multiple courses over multiple semesters/terms

Presume that you had an archive of electronic copies of prior student submissions for all assignments in your course. Consider each of the following purposes that might be served by this archive, and rate how strongly you agree with the **value of using such an archive this way.**

An archive is valuable for resolving grade disputes

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

An archive is valuable for providing a backup of student work

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

An archive is valuable for detecting plagiarism / cheating

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

An archive is valuable for learning outcomes assessment for my course

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

An archive is valuable for longitudinal curricular assessment over multiple courses

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

An archive is valuable for educational research

☐ Strongly Disagree ☐ Disagree ☐ Agree ☐ Strongly Agree

Given a choice, what part of the grading process would you automate and why?

What is the most time consuming or difficult part of the grading process and why?

What is the biggest disadvantage of the grading method used by you or your

grader and why?

Submit Query

References

[AAHE, 1993] AAHE Bulletin, "Deep Learning, Surface Learning", 45(8), 10-13, 1993

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

[Aiken et al, 2003] Aiken, A., Schleimer S., Wilkerson, D., Winnowing: Local Algorithms For Document Fingerprinting, SIGMOD 2003, San Diego, CA 2003

[Karel et al, 2004] Bergin, J., Stehlik, M., Roberts, J., Pattis, R. Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java, website last accessed on June 30, 2004, http://csis.pace.edu/~bergin/KarelJava2ed/

[Bhalerao & Ward, 2001] Bhalerao, A. and Ward, A., "Towards electronically assisted peer assessment: a case study", Association for LearningTechnology journal (ALT-J), 9(1), 26-37, 2001

[Blackboard, 2004] Blackboard, website last accessed on June 30, 2004, http://www.blackboard.com

[Brindley & Scoffield, 1998] Brindley, C., Scoffield, S., "Peer Assessment in Undergraduate Programmes", Teaching in Higher Education, 3(1), 79-89, 1998

[Brown & Bull, 1997] Brown, G., Bull, J., and Pendlebury, M., Assessing student learning in higher education, Routledge, London, 170-184, 1997

[Bull, 1999] Bull, J., "Computer-Assisted Assessment: Impact on Higher Education Institutions", Educational Technology & Society 2(3), 1999

[Cera et al, 2002] Cera C.D., Lass R.N., Char B., Popyack, J.L., Herrmann, N., Zoski, P., "Labrador: A Tool for Automated Grading Support in Multi-section Courses, Drexel University Programming Learning Experience, WebCT 2002, 4th Annual Users Conference, Boston, Massachusetts, July 24-26, 2002

[Checkstyle, 2004] Checkstyle home page, website last accessed on June 30th 2004, http://checkstyle.sourceforge.net/

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

[Dochy & McDowell, 1997] Dochy, F., McDowell, L., "Assessments as a tool for learning", Studies in Educational Evaluation, 23(4), 279-298, 1997

[Latham, 1995] 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

[Edwards, 2003] Edwards, S., "Rethinking Computer Science Education from a Test-first Perspective", OOPSLA '03, October 26-30, 2003, Anaheim, California, USA

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

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

[English & Siviter, 2000] English, J and Siviter, P., "Experience with an Automatically Assessed Course", Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, ACM Press, 2000, pp. 168-186

[Gehringer, 2001] Gehringer, E.F., "Electronic Peer Review and Peer Grading in Computer Science Courses", Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC, ACM Press, 2001, pp. 139-143

[Google, 2004] Google, website last accessed on June 30th, 2004, http://www.google.com

[HTMLArea, 2004] HTMLArea, website last accessed on June 30th, 2004, http://www.interactivetools.com/products/htmlarea/

[Hunter & McLaughlin, 2004] Hunter J. and McLaughlin, B., JDOM, web page last accessed on June 4th, 2004, http://www.jdom.org

[Interactivetools, 2004] Interactive tools site, website last accessed on June 30th 2004, http://www.interactivetools.com

[Isong, 2001] 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

[Jackson & Usher, 1997] Jackson, D. and Usher, M., "Grading Student Programs Using ASSYST", Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education, San Jose, CA, ACM Press, 1997, pp. 335-339

[Jackson, 2000] Jackson, D., "A Semi-Automated Approach to Online Assessment", iTiCSE 2000 7/00 Helsinki, Finland, ACM Press, 2000

[Jones, 2000] 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

[Luck & Joy, 1998] Joy, M. and Luck M., "Effective Electronic Marking for On-Line Assessment", Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education, Dublin, Ireland, 1998, pp. 134-138

[Luck & Joy, 1998] 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

[Lass et al, 2003] Lass R., Cera C., Bomberger N., Char B., Popyack J., Herrmann N., Zoski, P., "Tools and Techniques for Large Scale Grading Using Web-based Commercial Off-The Shelf Software, Drexel University Programming Learning Experience, ITiCSE, June 30-July 2, Thessaloniki, Greece, 2003, http://duplex.mcs.Drexel.edu

[MacPhereson, 1997] 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

[Mason & Woit, 1999] 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

[McQuain, 2004] McQuain, W., "Curator: an Electronic Submission Management Environment". Web page last accessed June 04, 2004: http://ei.cs.vt.edu/~eags/Curator.html

[Merriam-Webster, 1982] Merriam-Webster Online, http://www.m-w.com

[Pardo, 2002] Pardo, A., "A Multi-Agent Platform for Automatic Assignment Management", ITiCSE'02, June 24-26, 2002, Aarhus, Denmark

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

[Popyack et al, 2002] 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

[Pressman, 2000] Pressman, R. Software Engineering – A Practitioner's Approach, 5th edition, McGraw Hill

[Preston & Shackelford] 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

[Price & Petre] 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

[Reek, 1996] 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

[Reek, 1989] 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

[Schorsch, 1995] Schorsch T., "CAP: An Automated Self Assessment Tool To Check Pascal Programs For Syntax, Logic and Style Errors, SIGCSE'95, 3/95, Nashville, TN, ACM, 1995

[Shah, 2003] Shah Anuj, Web-CAT: A Web-based Center for Automated Testing, Master's thesis, Dept. Computer Science, Virginia Tech, Blacksburg, 2003

[Sitthiworachart & Joy, 2003] Sitthiworachart, J. and Joy, M., "Deepening Computer Programming Skills by Using Web-based Peer Assessment", Proceedings of the 4th Annual Conference of the LSTN Centre for Information and Computer Sciences, LSTN-ICS, 2003, pp. 152-157

[Somervell, 1993] Somervell, H., "Issues in assessment, enterprise and higher education: the case for self-, peer and collaborative assessment", Assessment and Evaluation in Higher Education, 18, 221-233, 1993

[Sourceforge, 2004] Sourceforge, website last accessed on June 30th, 2004, http://sourceforge.net/

[Topping, 1998] Topping, K., "Peer assessment between students in colleges and universities", Review of Educational Research, 68, 249-276, 1998

[Trivedi et al, 2003] Trivedi, A., Kar, D.C. and Patterson-McNeil, H., "Automatic Assignment Management and Peer Evaluation", The Journal of Computing in Small Colleges, Volume 18, Issue 4, The Consortium for Computing in Small Colleges USA, 2003, pp 30-37

[VT Survey, 2002] VT Survey, website last accessed on June 30th, 2004, http://www.survey.vt.edu

[WebCT, 2001] WebCT Inc, http://www.webct.com, 2001

[WebObjects, 2004] WebObjects Development, Student Guide

[WebObjects, 2004] WebObjects 5.2, web page last accessed on June 30th, 2004, http://www.apple.com/webobjects

[Zeller, 2000] Zeller, A., "Electronic Peer Review and Peer Grading in Computer-Science Courses", Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, ACM Press, 2000, pp. 89-92

Vitae

Hussein Vastani, son of Kamaluddin E. Vastani was born on 30th November 1979 in Mumbai, India. He graduated from Virginia Tech in May 2002 with a bachelor's degree in Computer Science. He is currently pursuing his master's degree in Computer Science and Applications at Virginia Tech. Hussein will work as a Software Engineer at Advanced Simulation Technology Inc located in Herndon, Virginia.