

Web-CAT: A Web-based Center for Automated Testing

Anuj Shah

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. James Arthur

Dr. Mary Beth Rosson

May 13th, 2003

Blacksburg, VA, USA

Keywords

Computer Science education, Software Engineering across the curriculum, Software Testing, Test-Driven Development, Test-first coding

Web-CAT: A Web-based Center for Automated Testing

by

Anuj Shah

Abstract

The fundamentals of software testing and related activities are often elusive in undergraduate curricula. A direct consequence of the lack of software testing efforts during education is the huge losses suffered by the software industry when applications are not sufficiently tested. Software practitioners have exhorted faculty members and institutions to teach more software testing in universities.

The purpose of this research is to provide answers to the needs of such practitioners and introduce software-testing activities throughout the curriculum. The most important goal is to introduce software-testing education without requiring a significant amount of extra effort on behalf of faculty members or teaching assistants.

The approach taken comprises the development of the Web-based Center for Automated Testing (Web-CAT) and the introduction of test-driven development (TDD) in courses. Web-CAT serves as a learning environment for software testing tasks and helps automatically assess student assignments.

A comparison of student programs developed using Web-CAT with historical records indicated a significant decrease in the number of bugs in submitted programs. Undergraduate students also received exposure to the principles of software testing and were able to write test cases that were on an average better than those generated by an automated test case generator designed specifically for the assignment.

Acknowledgements

The very foundation of this thesis is based on four strong pillars. I would like to express my heart-felt gratitude to each of them.

First and foremost, I would like to thank my parents and my family. It is because of their support and encouragement that I have realized a few of my goals and ambitions.

To my advisor, Dr. Stephen Edwards, words are inadequate in expressing my gratitude. It is because of your help, motivation, and of course deadlines, that today I am able to wrap up this thesis. I am sincerely indebted to you.

To my committee, Dr. James Arthur and Dr. Mary Beth Rosson, thank you for making this thesis a very valuable learning experience.

Lastly, to the Singhal sisters, Mudita and Vibha, thanks for hearing my never ending stories about my thesis work and also sitting through long sessions of debugging Pascal code.

To all the people who have directly or indirectly influenced my work thank you.

Table of Contents

Abstract	ii
Acknowledgements	iii
Table of Contents	iv
Table of Figures	vii
Table of Tables	viii
Chapter 1: Introduction	1
1.1 Introduction.....	1
1.2 Problem statement.....	2
1.3 What is test-driven development?.....	2
1.4 Web-CAT and TDD together.....	3
1.5 Organization of chapters	4
Chapter 2: Related Work.....	5
2.1 Importance of software testing.....	5
2.2 Role of information technology	5
2.2.1 Automated grading.....	5
2.2.2 Submission front end	7
2.3 Use of test driven development in courses	8
2.4 Shortcomings of previous efforts.....	8
2.5 A new approach to teaching software testing and automated grading.....	9
Chapter 3: Web-CAT Design Goals and Objectives .	11
3.1 Automated grading.....	11
3.2 Curator: Virginia Tech's online grading system.....	11
3.3 Web-CAT.....	11
3.3.1 Goals and objectives	12
3.3.2 An online course management system for instructors	12
3.3.2.1 Creating an assignment	12
3.3.2.2 Uploading a student roster	13
3.3.2.3 Viewing grades	13
3.3.3 An online submission system for students.....	14
3.3.3.1 Submitting an assignment	14
3.3.3.2 Viewing reports.....	14
3.4 Advantages of Web-CAT	15
Chapter 4: Web-CAT Architecture	16
4.1 User interface design.....	16
4.1.1 Student interface design	17
4.1.1.1 Submitting an assignment	17
4.1.1.2 Viewing reports for submitted assignments.....	21
4.1.2 Instructor interface design.....	22
4.1.2.1 Creating an assignment	22
4.1.2.2 Uploading a student roster	28

4.1.2.3 Viewing student scores	29
4.2 Server side design	30
4.2.1 Explaining the terms	31
4.2.2 Curator subsystem.....	32
4.2.2.1 Assessing student programs.....	32
4.2.2.2 The feedback report	34
4.2.2.3 Description of important classes.....	35
4.2.3 Core subsystem	37
4.2.3.1 The wizard-based interaction.....	38
4.2.3.2 Brief description of important classes.....	40
4.3 Benefits from this design	42
Chapter 5: Experimental Evaluation.....	43
5.1 Evaluation plan	43
5.2 Data collection	44
5.2.1 Collecting derived data	44
5.3 Experimental design.....	45
5.4 Results and discussion	46
5.4.1 Comparing raw scores.....	46
5.4.2 Comparing Web-CAT scores.....	46
5.4.3 Comparing Curator scores	47
5.4.4 Comparing scores without coverage information.....	48
5.4.5 Comparing coverage scores	49
5.4.6 Comparing submission times.....	49
5.4.7 Comparing number of submissions	50
5.4.8 Comparing bug densities.....	50
5.4.9 Inferences.....	54
Chapter 6: Survey Analysis.....	55
6.1 Student survey.....	55
6.2 Questionnaire for CS 3304.....	55
6.3 Survey evaluation.....	58
6.3.1 Discussion of responses	58
6.3.2 Discussion of responses to open-ended questions	62
6.4 Summary of student responses.....	63
Chapter 7: Conclusion and Future Work.....	64
7.1 Summary of results	64
7.2 Contribution	65
7.3 Conclusion	65
7.4 Future work.....	66
Appendix A.....	67
A.1 Sample test case file	67
Appendix B.....	68
B.1 Script used for phase I of assignment processing	68
B.2 Script used for phase II of assignment processing	72
B.3 Script used for phase III of assignment processing.....	82

Appendix C	87
C.1 Assignment description	87
References.....	93
Vita	96

Table of Figures

Figure 4.1 Architectural overview of Web-CAT	16
Figure 4.2 Login screen	17
Figure 4.3 Student task list screen	18
Figure 4.4 Course selection screen	18
Figure 4.5 Select assignment	19
Figure 4.6 Upload program file	19
Figure 4.7 Upload test case file.....	20
Figure 4.8 Confirm the submission.....	20
Figure 4.9 Final report screen	21
Figure 4.10 Task list screen as viewed by instructors/administrators	22
Figure 4.11 Select an assignment.....	22
Figure 4.12 Assignment details.....	23
Figure 4.13 Upload first script file.....	24
Figure 4.14 Upload second script file	24
Figure 4.15 Upload third script file.....	25
Figure 4.16 Upload fourth script file	25
Figure 4.17 Select grading profile	27
Figure 4.18 Grading options	28
Figure 4.19 Sample student Roster	29
Figure 4.20 Upload student file	29
Figure 4.21 Select students	30
Figure 4.22 Sample output file produced by Web-CAT	30
Figure 4.23 Final feedback report.....	35
Figure 4.24 Wizard navigation	39
Figure 4.25 Login validation.....	40
Figure 4.26 Web-CAT	40
Figure 5.1 Bugs v\s tests failed.....	52
Figure 6.1 Response of students on survey questions	59

Table of Tables

Table 5.1: Final scores of students on respective grading systems.....	46
Table 5.2: Final scores of all students on Web-CAT.....	47
Table 5.3: Significance of scores on Web-CAT	47
Table 5.4: Final scores of all students on Curator	47
Table 5.5: Significance of scores of students on Curator	48
Table 5.6: Scores excluding coverage information.....	48
Table 5.7: Significance of scores excluding coverage information.....	48
Table 5.8: Coverage score comparisons	49
Table 5.9: Time of initial submission (minutes before the deadline)	49
Table 5.10: Time of final submission (minutes before the deadline)	50
Table 5.11: Test cases failed.....	51
Table 5.12: Significance of test cases failed.....	52
Table 5.13: Model summary	53
Table 5.14: ANOVA results for bug density	53
Table 5.15: Linear regression analysis.....	53
Table 5.16: Significance of bug densities	53
Table 5.17: Significance of bug densities for all students	54

Chapter 1: Introduction

1.1 Introduction

Software testing is an important and integral part of any software development activity. It is not unusual for a software development organization to expend 30 to 40 percent of total project effort on testing [22].

Computer software testing can be performed at different levels starting from unit testing and moving on to integration testing, systems testing and acceptance testing. During the early stages of the testing cycle, the software developer himself/herself does most of the testing. Unfortunately, this activity is perceived as boring, tedious and uncreative work by practitioners, less than 15% of whom ever receive formal training in the subject [25].

Defective and bug-riddled software has been one of the major problems of the software industry and has accounted for huge losses, as well as the failures of large projects. According to Ricadela, “defective code remains the hobgoblin of the software industry, accounting for as much as 45% of computer-system downtime and costing U.S. companies about \$100 billion last year (2000) in lost productivity and repairs” [26]. A recent study by NIST says, “Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated \$59.5 billion annually, or about 0.6 percent of the gross domestic product” [28]. Much of these losses can be attributed to a lack of formal training in software testing for practitioners.

How can one provide this much-needed formal training in software testing? One approach to addressing this problem is to introduce concepts of software testing in undergraduate curricula at universities to produce better testers. Along with teaching the concepts of developing efficient and complex computer software in courses such as Data Structures, Computer Algorithms or Concepts of Programming Languages, it is also important that the students be taught to develop more robust, less-buggy software. Introducing testing principles and fundamentals at this stage will enable them to become better testers and produce more robust code. A recent article in *Communications of the ACM* exhorts faculty to teach more software testing: “Students today are not well

equipped to apply widely practiced techniques... They are graduating with a serious gap in the knowledge they need to be effective software developers” [29].

1.2 Problem statement

The objective of this research is to design and develop a web-based environment that helps students effectively learn the concepts of software testing without introducing the overhead of additional courses or significant extra learning effort on part of the students or the faculty members.

Such an environment shall include extensive support for test driven development, which shall help realize our ultimate goal of making software testing a norm with students.

This web-based learning environment, Web-CAT, shall serve to complement the classroom and help students carry out a set of software testing tasks and activities. These activities are discussed in more detail during the latter half of this thesis.

Web-CAT shall serve as a submission front-end for programming assignments with an easy-to-use web interface. Students shall login to Web-CAT, make a number of submissions for their assignments and view results of each submission. The main purpose for such an environment shall be to support the automatic evaluation of programming assignments and provide timely feedback to students such that it avoids any extra grading effort required by teaching assistants or instructors.

1.3 What is test-driven development?

As the name suggests, test-driven development (TDD) also called test first coding [4] is a new strategy that is centered on using tests to develop code. It is not a testing paradigm or activity but it focuses on developing code in a robust manner. Extreme programming [13, 27] incorporates this code development strategy and has played an important role in popularizing test-driven development.

In TDD, no code is written unless there is an accompanying test case. In fact, a test case is the first thing that is written when developing code; only afterwards code is written to pass the test. By constantly running all existing tests against a unit after each change, and always phrasing operational definitions of desired behavior in terms of new

test cases, TDD gives a programmer a great degree of confidence in the correctness of his/her code.

TDD [27] is a programming strategy where

- The programmer maintains an exhaustive suite of unit tests,
- No code goes into production unless it has associated tests,
- The programmer writes the tests first
- The tests determine what code the programmer needs to write or fix.

TDD helps students focus more on their testing efforts. The use of TDD, when tightly coupled with rewards for additional testing effort (points on programming assignments), can help create a culture where students feel naturally inclined to do more testing on their own.

1.4 Web-CAT and TDD together

For educating students in software testing, simply introducing the concepts would not prove of great help. It is important for the students to gain practical experience testing real software and repeatedly continuing such activities across a number of courses. Unfortunately, as students start out they are neither well equipped nor capable of doing a comprehensive job of testing. Web-CAT and TDD can help overcome this initial barrier.

A brief introduction to test driven development accompanied by development of a few on-the-fly code samples during classroom activities can give students an idea of how develop their own assignments. These assignments can be submitted to Web-CAT for feedback at any stage.

Once feedback on areas for improvement is received, students can add more test cases, add more code to correct anomalies, and re-submit for more feedback. A number of such “review and correct” cycles can help students test their solutions more as well as improve their grade on such assignments.

The use of Web-CAT and TDD consistently across a number of core courses will make it possible to introduce software testing activities and principles across the undergraduate curriculum. Students shall be more aware of testing standards and better equipped to produce more reliable and robust code.

1.5 Organization of chapters

CS educators and instructors have in the past made numerous efforts to introduce the concepts of software testing in undergraduate curricula. Chapter 2 discusses some of these efforts and identifies the novelties of the approach presented here. Chapter 3 lays out the goals and objectives of the online interactive learning environment, Web-CAT, while Chapter 4 discusses the design of Web-CAT in detail and the automated grading back-end support it provides. The latter half of this thesis provides information about the experimental setup and data collection mechanisms used to evaluate student performance. Chapter 5 and Chapter 6 identify the subjects, the experiment, and data collection while Chapter 7 presents the conclusions, summary and suggested extensions to the research.

Chapter 2: Related Work

2.1 Importance of software testing

What is the definition of software testing? Glen Myers [21] defines testing as “Testing is the process of executing a program or system with the intent of finding errors.” Other definitions of testing include, “Finding bugs in programs”, “Showing correct operation of a program”, “Testing is the process of establishing confidence that a program or system does what it is supposed to do” Hetzel [9].

These definitions of testing and many others depict testing as a follow-up activity and not something that needs to be done throughout the development process. Software developers have adhered to these definitions in the past. Procrastination and lack of the testing process has led to failures of a large number of projects and losses of billions of dollars to the software industry.

Software practitioners have time and again exhorted universities to teach more testing and software engineering in core courses across the undergraduate curriculum [15, 16, 17, 20, 30].

2.2 Role of information technology

A variety of authors have addressed the issue of teaching the fundamentals of software testing in introductory computer science courses [8, 16, 17, 21, 25, 30] heavily relying for support on information technology. Numerous frameworks as well as online communities have been developed to foster the use of software testing principles by students. Information technology has played a major role as a catalyst in this process.

In the following sections work that is most relevant to our research has been briefly addressed. We have tried to leverage previous work and take lessons from the shortcomings of previous research in this area.

2.2.1 Automated grading

Instructors and teaching assistants are already overburdened with work when conducting computer science courses. An automated grading system can reduce this overhead and allow instructors to concentrate on other important issues of designing

interesting as well as challenging program assignments. The use of an automated grading system provides additional benefits in terms of consistency, thoroughness and efficiency [11]. Every submitted program is checked with the same level of efficiency and shall be free from any instructor bias or side effects of lethargy. One of the major advantages of such systems is that timely feedback can be given to students on their performance.

In her paper [11], Isong discusses one possible approach to developing an automated program checking system. The primary responsibility of the program checker is to assess the correctness of student programs. The instructor is responsible for the assignment specification and creation of a concrete plan to grade and assess student assignments. The assignment specification is such that the instructor can develop an extensive suite of test cases from the assignment specification that shall be used to assess student submissions. Her approach eliminates the issues of style and documentation in programs and focuses only on program correctness.

The use of PGSE [17] in courses has reported results that are encouraging and motivate us to use an automated grading system. The focus of Jones' approach [17] is the incorporation of software testing activities in classrooms. The PGSE is a UNIX based automated grading system that operates in either a fully automated mode or a semi-automated mode. The fully automated mode is more relevant to our research and we leverage principles from his work. However there are a number of issues that need to be addressed when using Jones' approach to automated grading. The primary drawback is the lack of feedback presented to students on submission of an assignment. All grading is deferred until the due date, which would not allow students to correct bugs in their programs. There is an element of surprise that catches students off guard. An initial learning curve is introduced that needs students to adjust to the grading criteria.

The work of David Jackson and Michelle Usher is most relevant to our research. In using the ASSYST system [12] for automatically grading programming assignments, the researchers have done a very comprehensive job evaluating student performance. The assessment process used helps analyze most prominent issues of software programming including documentation, style and design. They also require students to submit a set of test cases along with their program implementations. Such a grading process focuses on rewarding students for performing testing on their own. The only shortcoming of their

approach is that they have not yet successfully automated the entire process; however use of ASSYST has been encouraging [12].

Another automated grading system is Virginia Tech's Curator system [4]. The Curator has been successful in undergraduate courses at Virginia Tech. It takes care of compilation and execution of student programs, based on shell scripts (could be .bat, .exe, .sh) provided by the instructor during assignment set up. The tool serves is helpful in providing quality assessment of student programs especially when there are a large number of submissions. As this process is automated, instructors and teaching assistants can spend more time and grading effort on other aspects such as assessing design, style and documentation. In most cases, instructors allow more than one submission for a given assignment. This helps students receive feedback from the Curator and gives them an opportunity to make corrections, and hence, obtain a higher grade. More number of submissions fosters a number of "review and correct" cycles on behalf of students as the deadline approach. Another advantage is that the students start on their assignments at an earlier date to get adequate feedback on areas of improvement. The use of Curator has affected the programming activities of students at Virginia Tech.

2.2.2 Submission front end

The Curator system at Virginia Tech is novel in the sense that it incorporates a user interface and gives the look and feel of a simple web-based application. However, a number of usability problems with this system have motivated us to devise a new automated system that focuses on concrete grading criteria as well as keeps the students unaware of the intricacies of such grading systems.

The automated grading systems are mainly UNIX-based. Very few systems concentrate on a user-friendly submission front end. It is not clear how submissions are made to ASSYST [12] as user interface issues are not discussed.

The TRY system developed by Reek [24], initiates the need of a submission front end for secure submission of programming assignment. However he relies on a UNIX-based tool once again to take care of submissions. Such command line systems would involve additional learning overhead for students when submitting programming assignments although it might make it easier in terms of keystrokes.

2.3 Use of test driven development in courses

Cartwright et al. [2] has demonstrated the resounding success of using TDD in courses. The use of TDD helps students focus more on testing activities. Through TDD, a student becomes more aware of the testing process and is forced to do testing on his own. Running their test cases against implementations of other students encourages students. Such an approach rewards students for performing testing on their own.

However Cartwright's approach introduces a significant amount of overhead for the course staff as well as the students and it warrants a new course.

2.4 Shortcomings of previous efforts

We have identified the following critical weaknesses in the above approaches:

1. Students generally discontinue the use of or forget the techniques applied in a particular class. These techniques will not become an inherent part of future activities unless they are systematically applied throughout the curriculum.

2. Student may view practices that are not consistently integrated as part of programming, as additional work.

3. The quality of feedback that students receive must be direct and directed towards continual improvement. They should receive concrete steps and pointers to develop better code.

Students must view any extra work as helpful in completing working programs, rather than a hindrance. This will help students follow the technique faithfully in any future programming activities that they indulge in.

The Curator has been successful in classrooms and has mainly served to relieve teaching assistants and instructors from grading activities. It also helps in providing timely feedback to students and thereby facilitates more "correct and submit" cycles. However, there are a few inherent problems with such an automated system. The most immediate feedback that the students receive is in terms of code correctness. As a result students only focus on one aspect of their program. Secondly the Curator does not reward students for performing testing on their own. These and some other usability and user interface problems have forced us to take a new approach to automated grading.

The idea of using test-first programming in classrooms in itself is not revolutionary [1], however the real issue is to address and overcome the above noted pitfalls. The approach should be systematically applied across the curriculum in a way that makes it an inherent part of all programming activities. Also students must receive sufficient feedback on their performance that provides them with clear benefits.

2.5 A new approach to teaching software testing and automated grading

Our approach leverages principles from the above-presented research. Assignments shall be developed using the test-driven development strategy and submissions shall be made using Web-CAT for automated grading and feedback.

Web-CAT serves not only as an online submission and automatic grading system but also as an environment where students can execute a number of software testing tasks.

Web-CAT supports unlimited student submissions for any programming assignment. Once the assignment is submitted to Web-CAT, it immediately grades the program and provides feedback to the students. Such an instantaneous feedback mechanism can help introduce a number of “correct and submit” cycles until the students reach a perfect score. Feedback on all aspects of the programming assignment is provided however the teaching assistant later provides the comments on style and documentation.

In order to carry out any activity on Web-CAT, the student is presented with a wizard-based user interface. Choices made by the student on each page drive the wizard and help complete trivial tasks of submitting assignments and viewing reports. From the view point of the students they shall be using just another web-based application for making submissions and are completely unaware of the back end processing done on each assignment for devising a suitable assessment. Student are only aware of the grading criteria that is used to assess their assignments are obtain timely feedback to improve their performance.

The onus of creating an assignment is on the instructor. He needs to devise testable specifications as well as a grading plan that shall help assess assignments in terms of completeness as well as correctness.

Web-CAT also has extensive support for assignments developed using test-driven programming. A test suite can accompany each implementation submission and the grading process is responsible for assessing the completeness, correctness and validity of the test suite as well as the student programs.

Web-CAT also maintains extensive logs of all activities of logged in users. This can help troubleshooting procedures and make life easier for instructors when setting up assignments.

The design objectives, goals and architecture of Web-CAT are discussed in the following chapters.

Chapter 3: Web-CAT Design Goals and Objectives

3.1 Automated grading

Automated grading is an important element in the realization of the idea of infusing software testing in undergraduate curriculum [11, 12, 17]. The need for automation arises from the fact that instructors and TA's are already overburdened with work while teaching computer courses. It would be difficult for them to provide extensive feedback on every student program especially if the class size is a large number. The lack of appropriate feedback and assessment of student programs could serve to be a major hindrance to including software testing in the classroom.

3.2 Curator: Virginia Tech's online grading system

The Curator system has a number of inherent difficulties and pitfalls:

- Most importantly, students focus on output correctness first and foremost; the students fail to concentrate on aspects of programming including style and documentation. This is mainly due to the type of feedback received by students on assignments.
- Students are not rewarded for performing testing of their own implementations. As a result, students perform less testing on their own. Instead, they rely on instructor provided sample data and ignore the possibility of varying scenarios.
- The Curator does not support assignments that incorporate test-driven development and require submission of a test case file along with the source file.
- When using the Curator to set up assignments, the instructor not only needs to provide a reference implementation but also a test case generator to test student implementations.

3.3 Web-CAT

The Web-based Center for Automated Testing, "Web-CAT" shall meet the needs of faculty and students alike and provides much functionality to carry out software testing tasks. Web-CAT is an environment for learning software testing tasks. It shall provide a unique active-learning experience for a wide spectrum of students.

3.3.1 Goals and objectives

The goals envisioned for Web-CAT are:

1. Create an environment where students can submit their programming assignments and receive appropriate feedback on areas for improvement.
2. Support the submission of test cases along with an implementation file fostering the use of test driven development in classrooms.
3. Create a wizard-based user interface for students to carry out tasks such as submitting an assignment and viewing reports for already submitted assignments.
4. Enhance the user interface as well as provide additional functionality for instructors and teaching assistants for conducting computer science courses. Instructors shall be able to set up assignments, download student score files and upload student rosters.
5. Support the automatic evaluation of programming assignments.
6. Evaluate the use of Web-CAT in a classroom environment using student grades, surveys of student and direct measurement of performance on authentic testing tasks.

The development of Web-CAT and its introduction in undergraduate courses shall help create a culture where students shall make use of software testing activities on their own.

3.3.2 An online course management system for instructors

Requirement: Web-CAT shall serve as a course management system for instructors in order to conduct computer science courses at universities.

Discussion: A number of services shall be provided by Web-CAT to instructors. These services help the instructors to effectively conduct computer science courses. Each of the services provided to the instructor are discussed in detail.

3.3.2.1 Creating an assignment

Requirement: Web-CAT shall provide a wizard-based interface to instructors for setting up programming assignments.

Discussion: One of the primary tasks that an instructor has to perform while using automated program checkers is setting up programming assignments. In order to create an assignment to accept student submissions, the instructor is expected to provide a number of defining attributes such as the assignment due date, the maximum number of submissions allowed and the description of the assignment. These and other choices are presented to the instructor through a wizard-based interaction for creating an assignment. However, the primary focus of the instructor is devising script files that handle the compilation, execution and grading of submitted assignments.

It is at the instructors' discretion to provide either a single script file or up to a maximum of four script files for processing submitted assignments. Each uploaded script needs to be carefully drafted. Communication amongst these scripts is done via XML.

More details on the design features of the scripts and the grading back end are provided in Chapter 4.

3.3.2.2 Uploading a student roster

Requirement: Web-CAT shall provide instructors with the ability to upload a list of students that are enrolled-in a particular course.

Discussion: The task of uploading student rosters shall follow a similar wizard-based interaction sequence. The instructor is expected to provide a comma-separated file that follows a pre-defined schema to successfully upload a student list. Web-CAT provides support for files generated by the Banner [3] system for class rosters.

3.3.2.3 Viewing grades

Requirement: Web-CAT shall also provide the instructor with the ability to view grades of students, either individually or as a class.

Discussion: The instructor shall be able to download a comma-separated file containing the scores obtained by students on their last submission. Additional programming can help instructors retrieve important information such as time of submission, number of submissions, score on any particular submission or time of any particular submission. The instructor shall also be able to view scores using the Virginia Tech "pid" of students.

3.3.3 An online submission system for students

Requirement: Web-CAT shall serve as an online submission system for students and allow them to view reports for already submitted assignments.

Discussion: Web-CAT shall allow a student to login to the system using his/her Virginia Tech “pid” and password. Once logged in to the system, the students shall be able to perform tasks to make submissions to the system and view reports for previously submitted assignments.

3.3.3.1 Submitting an assignment

Requirement: Web-CAT shall allow students to submit an assignment for automatic grading and feedback.

Discussion: Students using Web-CAT shall be able to make submissions to the system using a wizard-based interaction sequence. At every stage of the wizard, students shall make a number of choices to effectively complete the task.

A test case file shall accompany every student program, which forms an important part of the grade received on the submission. To view the format of the test case file refer to Appendix A. Students shall be allowed to make a number of submissions depending on the limit set by the instructor for that particular assignment.

Once a student submits an assignment, the scripts uploaded by the instructor shall process each submission and results are presented back to the student immediately. The immediate feedback shall help students detect defects in their program implementations.

3.3.3.2 Viewing reports

Requirement: Web-CAT shall allow students to view reports for already submitted assignments using a wizard-based interaction.

Discussion: Web-CAT shall also maintain a record of all the past submissions made by students for all assignment. Students shall be able to view reports for graded programming assignments once they are logged into the system. A wizard-based interaction sequence shall help them view reports for past submissions.

3.4 Advantages of Web-CAT

The use of Web-CAT in classrooms shall prove advantageous in a number of ways.

Students shall receive timely and instantaneous feedback for their submitted assignments when using Web-CAT. This shall reduce the additional overhead for instructors and teaching assistants for grading these assignments. The immediate effect shall be a number of “correct and submit” cycles that shall help students receive a better grade on the assignment. If Web-CAT is consistently used across the entire curriculum it hopefully will introduce a cultural shift in the way students create programs and help them become better testers as well as produce more robust code.

Web-CAT also maintains extensive records of past submissions, reports and activities. As a result, students always have a copy of their last submission on the file server. It also helps a great deal to view past reports as they can provide valuable help in detecting anomalous program behavior. Instructors and administrators can review logs and detect misbehavior of the system as well as any problems with the created assignments or uploaded student rosters.

As Web-CAT grades submitted assignments, teaching assistants can concentrate only style and other documentation issues. They are completely relieved from grading student submissions based on correctness or other such attributes that require them to expend a lot of effort taking demos and running student programs. Most importantly Web-CAT shall be extremely easy to use and require no additional work on the part of students while submitting programming assignments.

Chapter 4: Web-CAT Architecture

Web-CAT is based on a three-tiered architecture.

1. **Client Tier:** At the client tier, we have the web browser that is capable of sending requests and receiving a response from the server.
2. **Middle Tier:** This tier is the web server tier. All Java classes and other business logic are located at this tier. The web server serves HTML pages to users and responds to user requests.
3. **Database Tier:** The third tier is made up of the database and the file server. The file server holds the submitted files of the user, while the database stores information related to users, courses and wizard states.

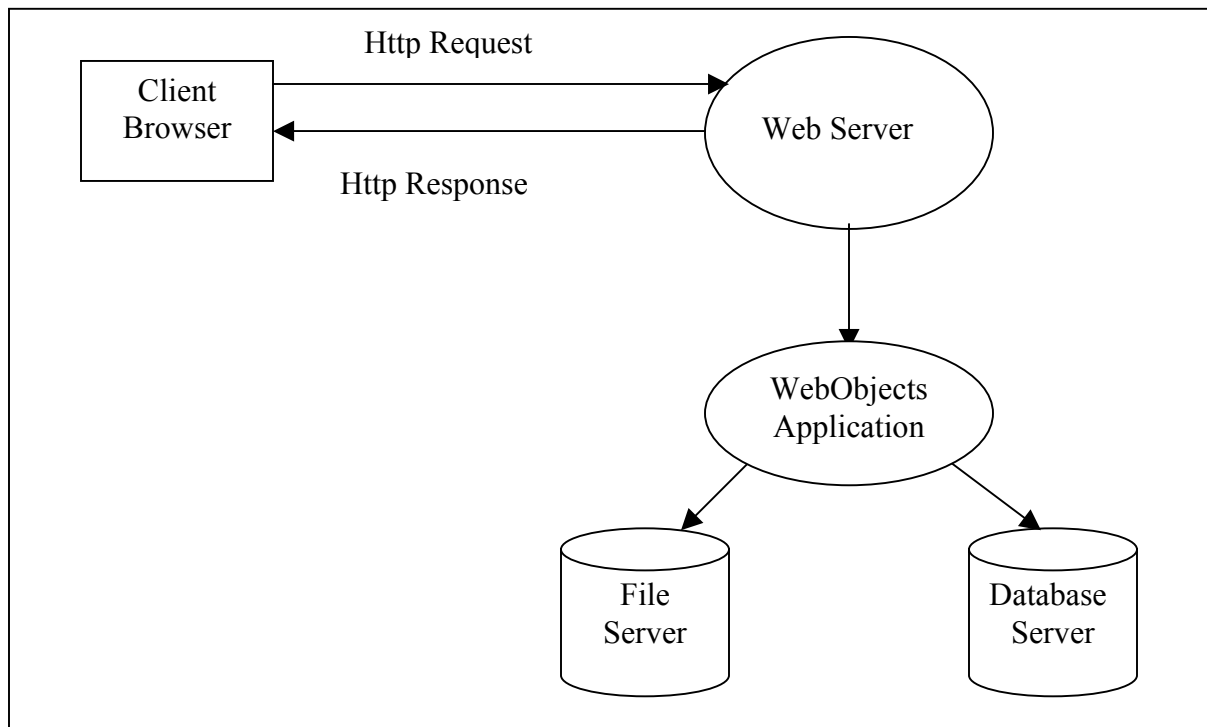


Figure 4.1 Architectural overview of Web-CAT

4.1 User interface design

One of the important goals of Web-CAT is that such a system shall be extremely easy to use and shall not introduce any additional overhead in terms of learning its particulars. The user interface design of Web-CAT is such that the use of the system shall

be highly intuitive to both students and instructors. To carry out tasks on Web-CAT, a simple wizard-based navigation technique is used.

A sequence of dynamic HTML pages, extensively supported by server-side Java classes, is presented to the user. These pages resemble wizard sequences similar to those used during software installation steps. At each stage of the wizard, the user makes a choice that carries the task closer to completion. These choices are recorded in a database so that the user does not have to repeatedly make the same choices again and again.

4.1.1 Student interface design

Web-CAT allows students to login to the system and submit programming assignments. Students can also view the grade and feedback results obtained on already submitted assignments.

4.1.1.1 Submitting an assignment

The following screenshots explain the sequence of interactions between the student and Web-CAT in order to submit an assignment.

Web - C A T

Web Center for Automated Testing (Web-CAT)

Welcome to Web-CAT. This site provides software testing services to help students learn how to test software more effectively. It automates many routine testing tasks for you.

Only registered users may use Web-CAT. Please login:

PID:

Password:

If you teach at another university and are interested in using Web-CAT, please contact the [Web-CAT administrator](#).

Figure 4.2 Login screen

The first step would be to login to Web-CAT.

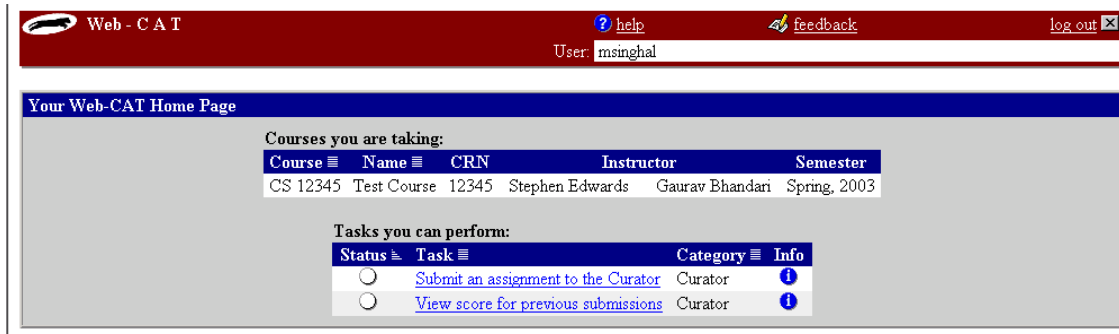


Figure 4.3 Student task list screen

The initial screen presents a list of tasks that can be carried out by the student. The student can select the appropriate link to begin submitting an assignment.

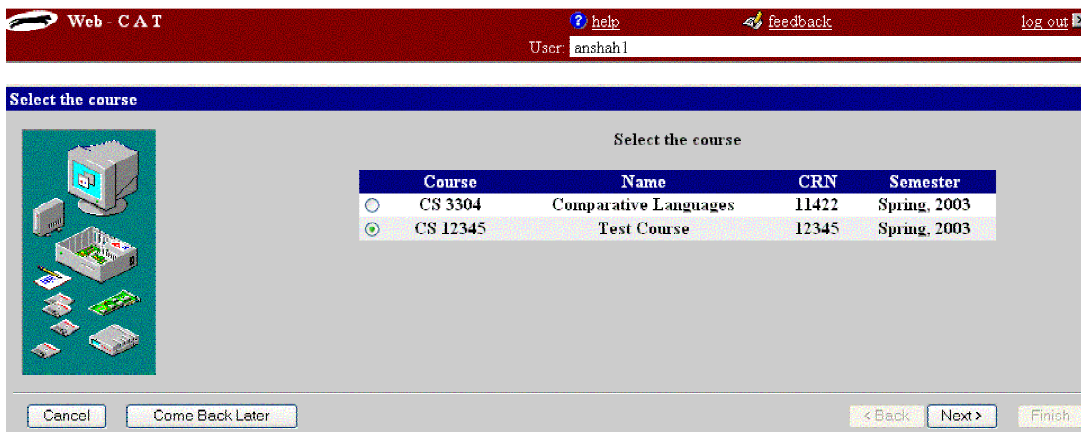


Figure 4.4 Course selection screen

The third step in the wizard sequence would be to select the particular course for which the student wishes to make a submission. A list of courses that the student is enrolled in is presented on this screen.

Web - C A T help feedback log out

User: anshah1

Submit an Assignment

Select the assignment for this submission:

Assignment	Description	Deadline
java1	Sample Java assignment	07/01/03 11:59PM
Pascal Clone	submit assignments for year 2001	04/03/03 11:59PM

Assignments not currently accepting submissions:

Assignment	Description	Deadline
Test Assignment	kggl	02/14/03 11:59PM

Cancel Come Back Later < Back Next > Finish

Figure 4.5 Select assignment

After selecting the course, the student is presented with a list of programming assignments for that course. Assignments that are currently accepting submissions are presented in a selectable list. On this screen the student is expected to select the particular assignment of interest and click “Next”.

Web - C A T help feedback log out

User: anshah1

Submit an Assignment

Your previous submissions for this assignment:

Total of 63, Showing 5 per page, page 1 of 13

No.	Time	Score
63	04/05/03 07:45PM	2.0
62	04/05/03 07:42PM	4.0
61	03/19/03 11:12PM	44.0
60	03/19/03 11:11PM	42.0
59	03/19/03 11:11PM	21.0

Choose the file to upload:

Browse...

Cancel Come Back Later < Back Next > Finish

Figure 4.6 Upload program file

The fifth screen on the submission wizard presents a list of past submissions with their respective scores. A “Browse” button is provided on this screen to help the student locate the program file that he wishes to submit for grading. After selecting the appropriate file the student can click on “Next” to proceed.

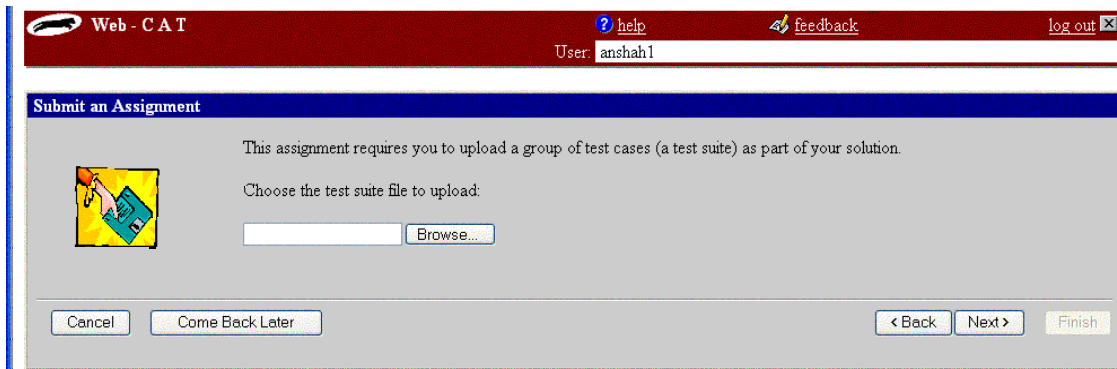


Figure 4.7 Upload test case file

After uploading the source code for the submission, the student is expected to submit a test suite. This file could be a simple text file. To learn the format for the test case file refer Appendix A. The file “Browse” button helps locate the test case file on the local file system. The student is expected to click “Next” in order to proceed to the next screen.

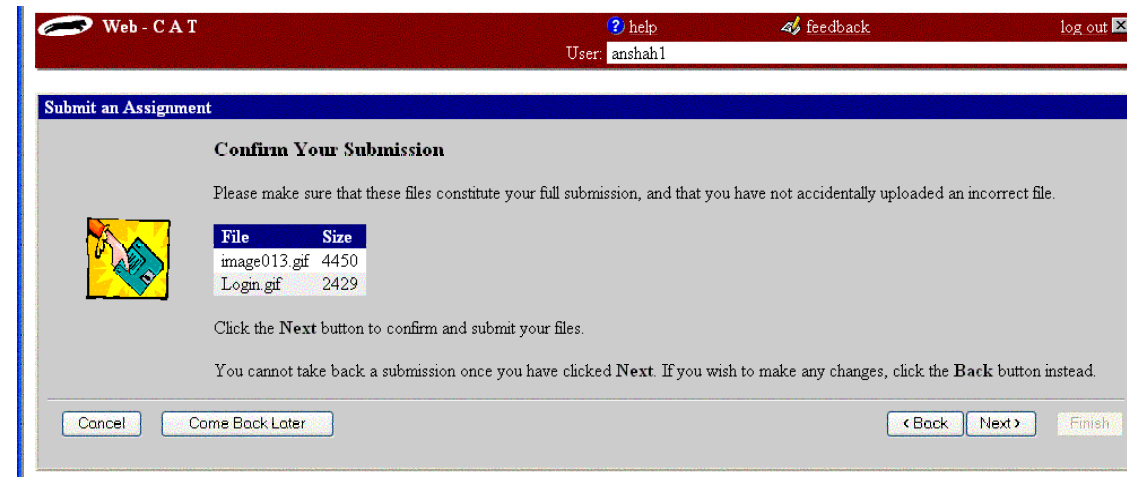


Figure 4.8 Confirm the submission

The last step in the wizard interaction for submitting an assignment for grading is to confirm the submission. Once the student clicks “Next” on this screen, the submission is processed by Web-CAT.

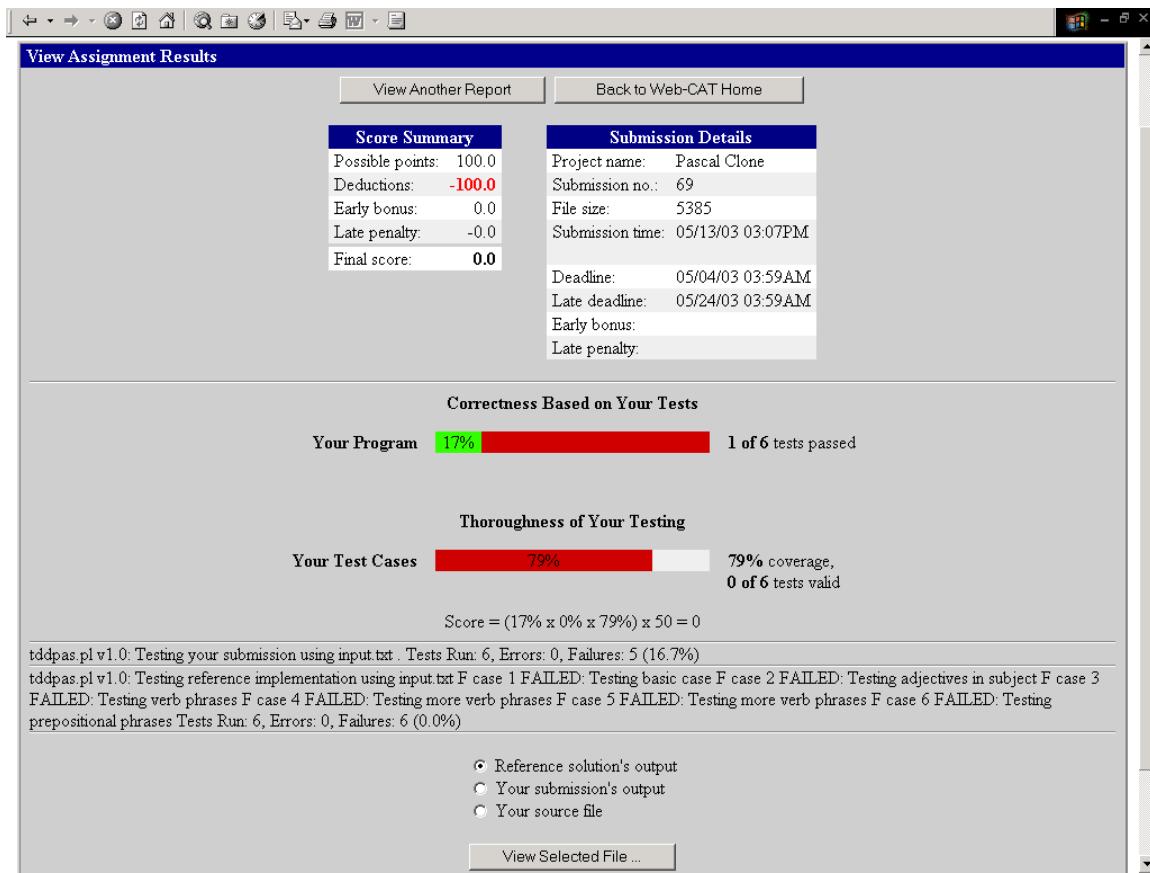


Figure 4.9 Final report screen

The last screen presents the results for the current submission to the student. The report generated for students serves to provide constructive feedback for various parts of the grading process. More details of the generated report are presented in section 4.2.2.2.

4.1.1.2 Viewing reports for submitted assignments

Web-CAT shall also maintain a record of each of the submissions made by the student along with their results. These results are available to the students whenever they are logged into Web-CAT.

The sequence of steps to view the reports follows the same basic wizard sequence as making a submission. On selecting a particular assignment, the student is presented with a list of submissions made for that particular assignment. The student can select any submission and a corresponding report page shall be displayed as shown in Figure 4.9.

4.1.2 Instructor interface design

Web-CAT provides support for instructors to maintain and conduct computer science courses. Wizard sequences exist for creating an assignment, uploading student class lists, and viewing grades of students both individually and for the entire class.

4.1.2.1 Creating an assignment

The following screen shots present a walk-through for the task of creating an assignment. To create an assignment the instructor needs to invoke the create assignment task.

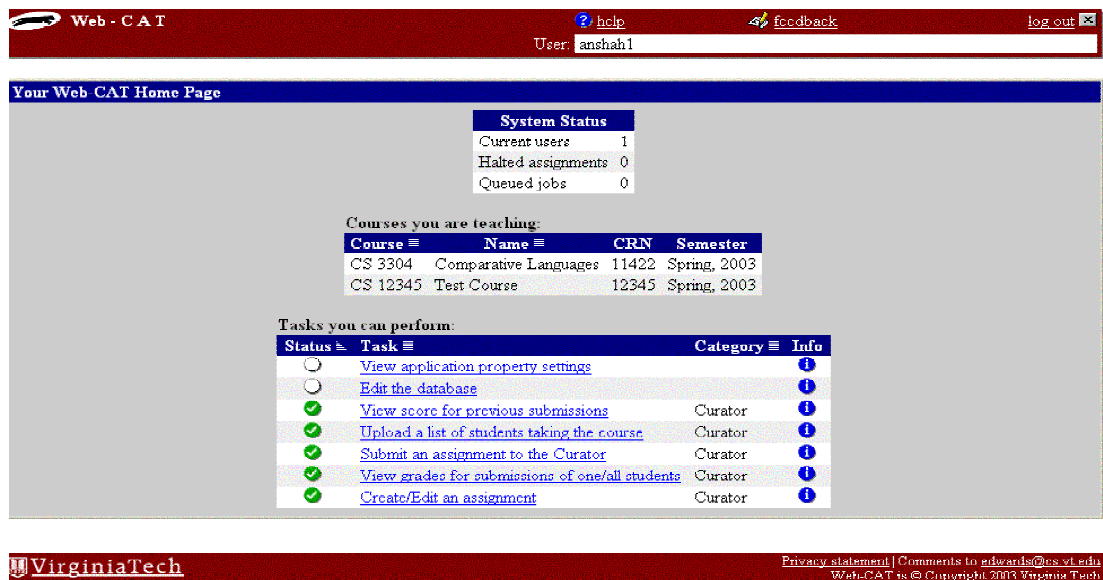


Figure 4.10 Task list screen as viewed by instructors/administrators

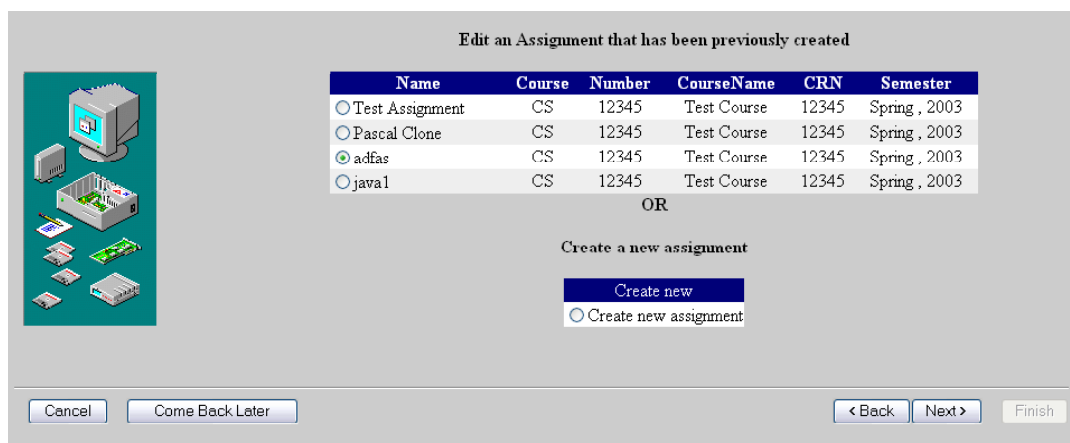


Figure 4.11 Select an assignment

After selecting to create an assignment, the instructor has the option of either editing a previously created assignment or creating a new assignment. If the instructor decides to create a new assignment, the next step in the sequence would be to select a course for the assignment. However when editing a previously created assignment, the course information has already been recorded. After making the appropriate selections the instructor presses “Next” in order to proceed with the task.

Field	Value	Notes
Assignment Title	Pascal Clone	
Assignment Description	submit assignments for year 2001	
Directory name to use	pascalClone	
Assignment URL	http://www.vt.edu	
Assignment due date	04/03/2003	(mm/dd/yyyy)
Assignment due time	23:59:59	(eg. 16:59:59)
Test Driven Programming Assignment	<input checked="" type="checkbox"/>	
Make Assignment visible to students	<input checked="" type="checkbox"/>	

Buttons: Cancel, Come Back Later, < Back, Next >, Finish

Figure 4.12 Assignment details

At this stage of the wizard, the instructor provides information about the assignment such as its title, description, the URL for further information, the assignment due date and the due time. The check box titled “Test Driven Programming Assignment” indicates whether the current assignment requires the submission of a test suite file along with the implementation while “Make Assignment visible to students” determines whether students can see and make submissions to the assignment. This option helps the instructor set up the assignment in the background and only publish the assignment once he gains sufficient confidence in its operation.

The next few steps in the sequence of creating an assignment are to upload a set of scripts that shall process each submission.

Compile Script Options

☒ This assignment requires compilation

Enter additional command line arguments here (names of main Java file)

Enter timeout to use for compilation (seconds)

Choices for Compile Script

☐ Built-in Compilation Script

☒ Select one of the uploaded compilation script

Uploaded File Name	Uploaded On
<input type="radio"/> compile.pl	04/17/03
<input type="radio"/> compileScript.pl	02/25/03

☐ Upload/Replace compilation script

Figure 4.13 Upload first script file

Web - C A T [help](#) [feedback](#) [log out](#)

User: anshah1

Instructor Script Options

☐ This assignment requires an instructors version


Enter additional command line arguments here

Enter timeout to use for instructor script (seconds)

Choices for Instructors Script

☐ Upload/Replace instructor script

Figure 4.14 Upload second script file


Web - C A T
[help](#)
[feedback](#)
[log out](#)

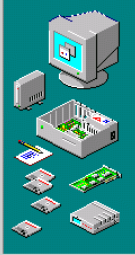
User: anshah1

Execution Script Options

☒ This assignment requires execution

Enter additional command line arguments

Enter timeout to use for execution script
 (seconds)



Choices for Execution Script


☐ Built-in Execution Script

☒ Recently uploaded execution script

Uploaded File Name	Uploaded On
<input checked="" type="radio"/> executeScripts.zip	05/13/03 04:00AM
<input type="radio"/> isrq.txt	02/08/03 05:00AM

☐ Upload/Replace execution script

Figure 4.15 Upload third script file


Web - C A T
[help](#)
[feedback](#)
[log out](#)

User: anshah1

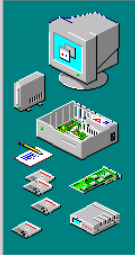
Grading Script Options

☒ This assignment requires grading

Enter additional command line arguments

Enter timeout to use for grading script
 (seconds)

Grading is suspended for the assignment
☐



Choices for Grading Script

☐ Built-in Grading Scripts

☒ Select one of the uploaded grading scripts

Uploaded File Name	Uploaded On
<input checked="" type="radio"/> gradePascaITDD.pl	05/13/03 04:00AM
<input type="radio"/> SElist.txt	02/08/03 05:00AM

☐ Upload/Replace grading script

Figure 4.16 Upload fourth script file

As explained in Section 3.3.3.1, the instructor can upload a maximum of four scripts to do the compilation, execution and grading of assignments. To process each submission with Web-CAT, we follow a four-step process.

Stage 1: Once a submission is made to Web-CAT, the source file is compiled using a PASCAL compiler. Appropriate diagnostic information is provided to the student if the compilation of the submitted source file fails. If the compilation of the submitted source file is complete, information related to the location of the executable and applicable deductions if any are conveyed to the next stage using XML. The script is responsible for generating this XML output.

Stage 2: Once an executable is generated, the second stage is responsible for assessing the correctness of the students program. In the second stage, the student-supplied test cases are run against the student executable and a correctness score is communicated to the next stage. A percentage score is assigned based on the total number of test cases supplied and the total number of test cases that pass the students implementation.

Stage 3: The third stage assesses the validity of the student provided test cases. To assess the validity of the student provided test cases, these test cases are run against an instructor-provided implementation. If any test case fails on the instructor's reference implementation, it is considered to be outside the problem domain and invalid. A limitation of this approach is that a student may be penalized for any valid extension to the problem domain.

Stage 4: To assess the completeness of the student's test cases, a coverage score based on a percentage scale is collected while the student's test cases are run against the instructor's source-instrumented implementation. The instructor's implementation is instrumented to account for branch as well as decision-condition coverage. This coverage score is based on a percentage scale from 0-100.

For the purpose of our research, three scripts take care of the above four stages. The first script is responsible for the compilation of each assignment. If the compilation fails, appropriate diagnostic information is displayed to the students. This information is communicated to other scripts using XML.

The second script is responsible for processing the student uploaded test case file and execution of these test cases against the instructor's reference implementation as well as the student's own implementation. The student uploaded test case file [Appendix A] is first separated in test cases and expected outputs. These test cases are then executed against the student's implementation as well as the instructor's implementation. The outputs are collected in text files and stored on the file system as temporary files. The location of these files is also communicated to the grading script via XML.

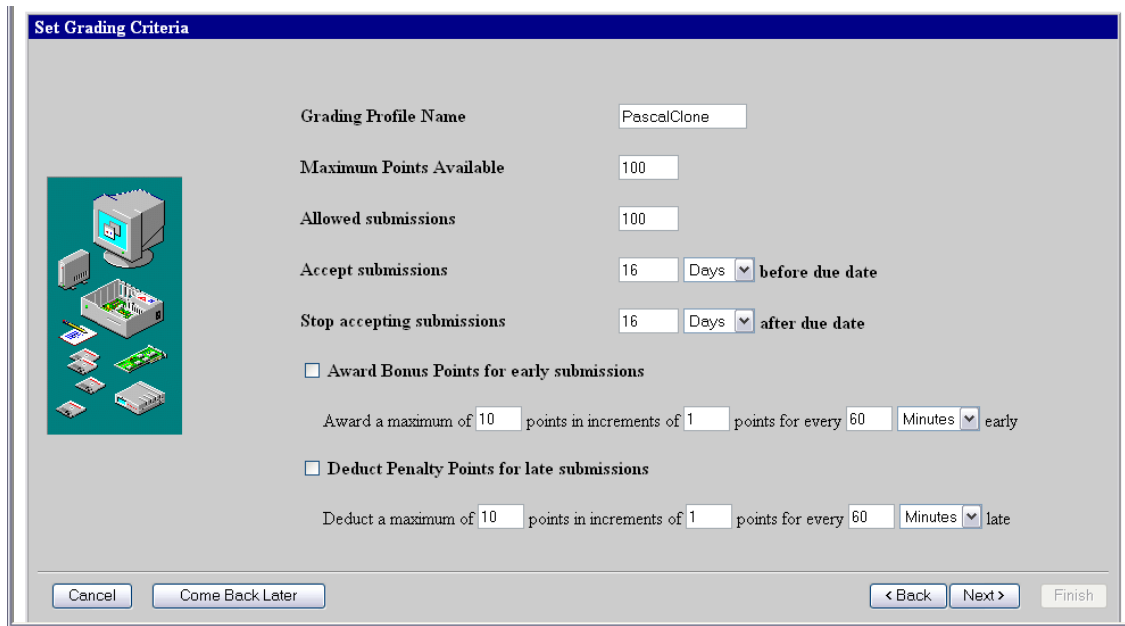
The last script collects score information from each of the previous scripts and is also responsible for comparing the outputs of the students programs (stored in temporary files) with the expected outputs present in the test suite. This script also is responsible for generating parts of the output that the students receive.

Each of the scripts used in the assignment processing is attached in Appendix B.

Select Grading Profile/Create a new Profile						
Use a grading profile that has been previously created						
	<table><thead><tr><th>Name</th></tr></thead><tbody><tr><td><input type="radio"/> Enter name here</td></tr><tr><td><input type="radio"/> Enter name here</td></tr><tr><td><input checked="" type="radio"/> PascalClone</td></tr><tr><td><input type="radio"/> Profile 2</td></tr></tbody></table>	Name	<input type="radio"/> Enter name here	<input type="radio"/> Enter name here	<input checked="" type="radio"/> PascalClone	<input type="radio"/> Profile 2
Name						
<input type="radio"/> Enter name here						
<input type="radio"/> Enter name here						
<input checked="" type="radio"/> PascalClone						
<input type="radio"/> Profile 2						
OR						
Create a new Grading Profile						
<input type="button" value="Create new"/>						
<input type="radio"/> Create new Grading Profile						
<input type="button" value="Cancel"/> <input type="button" value="Come Back Later"/> <input type="button" value="Back"/> <input type="button" value="Next"/> <input type="button" value="Finish"/>						

Figure 4.17 Select grading profile

After uploading scripts to process each submission, the instructor is expected to select a grading profile. The grading profile configures Web-CAT to start accepting and stop accepting assignments, decide late penalties, early bonus points and specify a limit to the number of submissions allowed for each assignment. On this page the instructor is presented with an option of creating a new grading profile or uses an already existing one.



Set Grading Criteria

Grading Profile Name: PascalClone

Maximum Points Available: 100

Allowed submissions: 100

Accept submissions: 16 Days before due date

Stop accepting submissions: 16 Days after due date

☐ Award Bonus Points for early submissions

Award a maximum of 10 points in increments of 1 points for every 60 Minutes early

☐ Deduct Penalty Points for late submissions

Deduct a maximum of 10 points in increments of 1 points for every 60 Minutes late

Cancel Come Back Later < Back Next > Finish

Figure 4.18 Grading options

This is the second last step in the setting up of an assignment. The instructor specifies information related to the maximum number of submissions, the maximum points for the assignment, the date when to start accepting submissions and the date to stop accepting submissions. Lastly this page also presents options to specify the policy to award early bonus points or deduct late penalty points on assignments. All choices made by the instructor at each wizard-page are presented on pressing “Next”.

The last step in setting up an assignment is to confirm these choices.

4.1.2.2 Uploading a student roster

In order to enroll students in a particular course, Web-CAT provides instructors with a wizard to upload a comma-separated file. The first step in uploading a list of students taking the course is to invoke the task from the task list presented on the main page. The screen presented to the instructor is the same as Figure 4.2. The next step in the wizard is to select the course in which the students are enrolled. The screen-shot is the same as in Figure 4.3.

"SSN"	"LASTNAME"	"FIRSTNAME"	"DEPARTMENT"	"YEAR"	"EMAIL"	"PHONE"
"XXXXXXXXXX"	"Andy"	"Anny"	"CS"	"Junior"	"g@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Beer"	"Lest"	"CS"	"Junior"	"a@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Beny"	"Mith"	"CS"	"Senior"	"b@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Beer"	"Gary"	"CS"	"Senior"	"c@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Sush"	"Mils"	"CS"	"Junior"	"d@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Bran"	"Will"	"CS"	"Senior"	"d@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Camy"	"Bran"	"CS"	"Senior"	"a@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Coop"	"Jatt"	"CS"	"Junior"	"f@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Dany"	"Raja"	"CS"	"Senior"	"r@vt.edu"	"540 1234567"
"XXXXXXXXXX"	"Dash"	"Pete"	"CS"	"Junior"	"x@vt.edu"	"540 1234567"

Figure 4.19 Sample student Roster

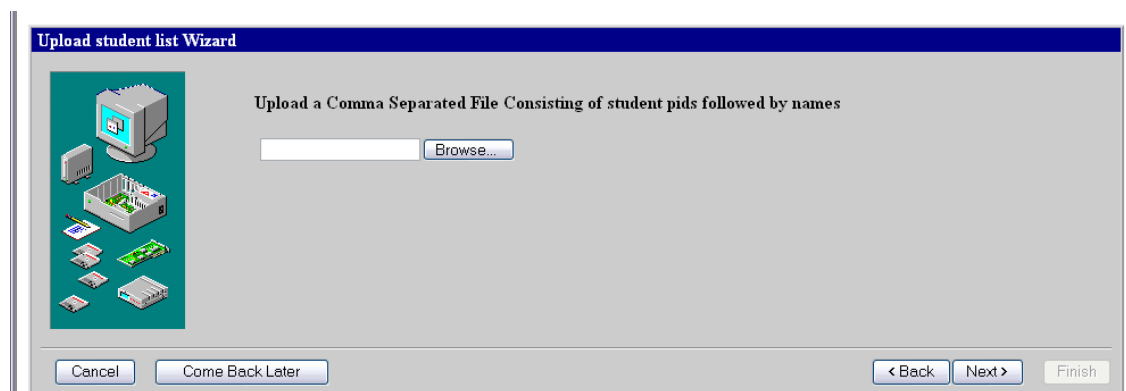


Figure 4.20 Upload student file

On this page the instructor selects a comma-separated file (with a defined interface) containing information about the students enrolled-in the already selected course. Web-CAT provides support for files generated by the Banner [3] system for class rosters.

4.1.2.3 Viewing student scores

Web-CAT provides the instructor with the option of downloading the scores of each of the students. The downloaded file is also a comma-separated file and the fields of this file are in a specified order.

The first step in downloading a score file is to invoke the task from the main page. The next step is to select a particular assignment for which the instructor wishes to view grades.

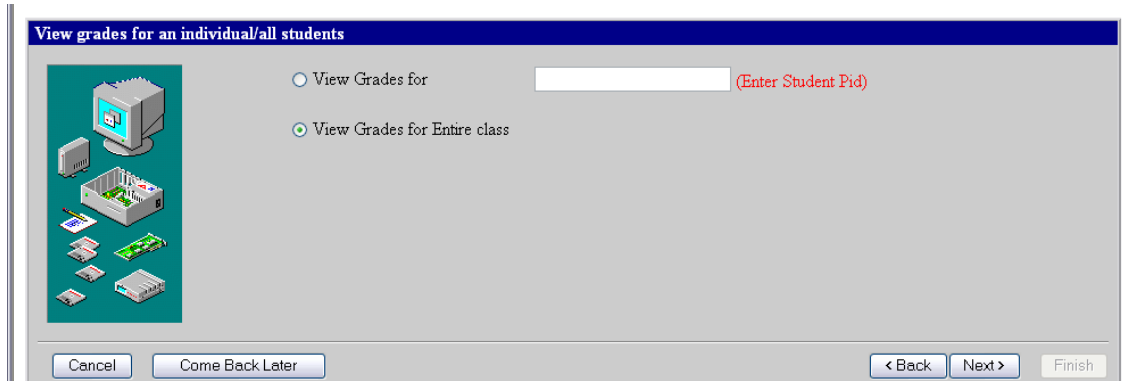


Figure 4.21 Select students

```

Number of header lines : 10
*****Start header *****
***   Score File
***   Assignment p4-Prolog

***   Course offering :CS 3304 Comparative Languages
***   Students : 64
File format
Pid   Score on   Time (min) of   Score on   Time (min) of   Raw   Max   Total
      last      last          first      first          Score score Submissions
      submission submission submission submission
*****End Header *****
A      44      1051500935000      00      1051151753000      44      50      34
B      47      1051285834000      02      1050960098000      47      50      11
C      18      1051498573000      00      1051043607000      18      50      36
D      39      1051471257000      00      1051378061000      39      50      28
E      45      1051494678000      00      1051390647000      45      50      14
F      47      1051207456000      42      1051066694000      47      50      09
G      33      1051404954000      24      1051368804000      33      50      32
H      48      1051229489000      36      1051149033000      48      50      15
I      19      1051510874000      00      1051255250000      39      50      30

```

Figure 4.22 Sample output file produced by Web-CAT

The last step in this task is to select the students to view grades. On this page the instructor has the option to either select the whole class or enter the “pid” of a single student to view individual grades. An email consisting of the final scores of all the students is sent to the instructor if he/she selects “View Grades for Entire class”. The score file is a comma-separated file with a pre-defined format.

4.2 Server side design

The implementation of Web-CAT is divided into a number of subsystems. Each subsystem is responsible for providing a number of services and maintains its own database. As a result each subsystem can provide testing tasks that are specific in nature

and related to each other. A central subsystem known as the core subsystem is responsible for initializing each of the other subsystems dynamically and reading information related to the tasks supported by them.

The two main subsystems of Web-CAT are the curator subsystem and the core subsystem.

4.2.1 Explaining the terms

Before presenting the details of each of the subsystem designs, it is important to understand a few frequently used terms.

Framework: A set of supporting classes that handles the basic operations of establishing a foundation upon which other classes build and function. The framework also does the work of supporting database operations for the classes using the framework.

Subsystem: A comprehensive set of “tasks”. Each subsystem can have a number of “tasks” that it supports. The subsystem uses one or more underlying frameworks that it needs to import. Each subsystem can also have its own set of database tables or use the database supported by the main framework also called as “Core”.

Task: A set of actions that a user can perform is modeled as a task. Each action consists of a series of wizard pages. This interaction will be uniform across a number of tasks and it might be that one or more wizard pages can be reused in multiple tasks.

Wizard: Similar to the convention of a wizard as seen in installation of software systems where a number of options are presented to the user. The user makes a number of decisions at each “page” in the interaction that affect the coming pages as well as the actions taken.

Page: A set of related choices are represented as a page. This constitutes one particular set of choices in a Wizard.

Status of a task: This helps distinguish tasks that are completed from those that are suspended at some stage and need to be resumed.

Wizard state: This indicates a set of past choices that the user has made in the wizard.

4.2.2 Curator subsystem

The automated grading features of Web-CAT are encapsulated in the curator subsystem. This subsystem meets the testing needs of the students and the course instructors alike.

This subsystem supports a number of tasks for both the faculty and the students. Each task for the faculty (create/edit an assignment, upload a list of students taking the course, view submission results for all students) and the students (submit an assignment to Web-CAT, view grades for previous submissions) is modeled as a subsystem task with wizard-based interaction.

The design of the curator subsystem is conversant with the design of the core subsystem and abides by the restrictions placed by the core subsystem. More details on the design of the core and its support for other subsystems are presented in later sections.

4.2.2.1 Assessing student programs

The primary tasks of the curator subsystem are to grade student programs and provide appropriate feedback to students. Extensive server side support is provided to the curator subsystem to facilitate the automatic evaluation and assessment of submitted programs.

In order to produce a cultural shift in the way students program and develop code, it is necessary to devise concrete grading criteria that not only assesses a student's program but also rewards him/her sufficiently for performing testing on his/her own. Our approach in grading student assignments focuses on the program's correctness and completeness as well as the students testing performance.

The assessment approach:

- Requires students to submit a set of test cases along with every implementation.
- Encourages students to write thorough tests and write tests as they code.
- Provides timely and appropriate feedback on submitted programs as well as test cases
- Rewards students for doing more testing.

- Evaluates the validity, correctness and the completeness of the test cases submitted as well as the program implementation.

Once a submission is received by Web-CAT, the system follows a definite grading process to assess the submission.

1. **Compilation:** Every submitted student program is compiled on the server side. If the compilation of the student program fails, appropriate diagnostic information is presented to the student in the feedback that is generated by Web-CAT.
2. **Execution of student program on student tests:** To assess the correctness of the student programs, each of the submitted test cases is executed on the student implementation. Part of the final score is derived from a percentage of the test cases that pass the students implementation. It is expected that all students will receive a 100% score on this phase since students shall submit only those test cases for which their implementation produces the right output.
3. **Execution of student tests on reference implementation:** In order to assess the validity of the tests submitted by a student, we used a reference implementation. The instructor of the course provided a solution implementation for the assignment. If any particular test case failed the instructor's solution it was judged to be an invalid test case. A percentage score is obtained from this phase as well. Once again students are expected to receive a 100% score for this phase since they are already aware of test cases that are valid and those that are invalid. Test cases defined by the grammar are considered valid by the reference implementation.
4. **Collection of coverage statistics:** The reference implementation also provides a score based on a percentage of the problem domain that the test suite executes. The reference implementation is instrumented with statement as well as branch coverage information to arrive at a concrete measure of the breadth of the executed test cases. This percentage score forms the third and the most important part of the final score.
5. **Provide feedback:** The final step is to provide students with adequate feedback on their testing performance. A report is presented in HTML format that

includes details of the student performance. A final score is also assigned to each student submission by multiplying each of the three scores described above. As a result, all three scores gain equal importance in the assessment scheme and students cannot afford to neglect any of these aspects.

The devised assessment approach serves as a concrete mechanism for assessing student programs and student testing performance together.

4.2.2.2 The feedback report

At the end of the compile/execute/grade cycle a comprehensive grading report of the student's performance is produced by Web-CAT. The final report displayed to the client as an HTML page consists of a number of different parts.

The feedback consists of up to five parts. Each of these five parts is explained in detail here:

1. **Score summary:** The first part of the report is presented in tabular format. These tables provide information on the total number of points that the assignment is worth, the submission deadline, the date and time of actual submission and other deductions if applicable. Other details related to the assignment are also present in this part.
2. **Correctness and completeness scores:** This half of the report aims to provide information related to the correctness and completeness of the program. The sliders present in the report are adapted from JUnit, the only difference being that they are not dynamic. The first slider gives the percentage of test cases that passed when run on the student's own implementation. The second slider gives an assessment of the validity of the tests as well as their completeness in terms of the branch coverage attained on the instructor's reference implementation.

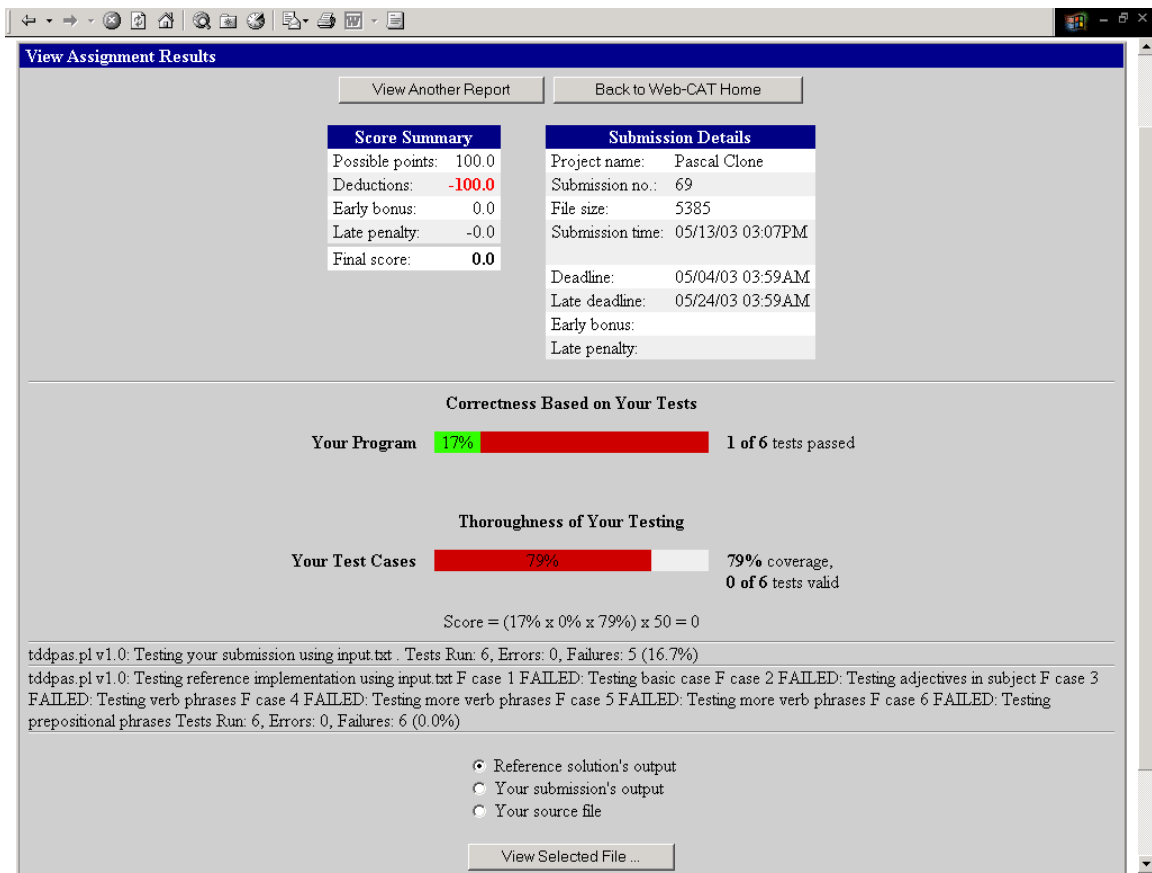


Figure 4.23 Final feedback report

3. **Program Correctness (Your Solution):** This part displays the labels of test cases that failed and also shows the number of test cases passed.
4. **Test Validity (Reference Solution):** provides similar information on the instructor's implementation.
5. **View Selected Files:** This part of the report provides a list of options to view submitted files, test cases and the expected outputs for each of the submitted test cases.

4.2.2.3 Description of important classes

A description of the server side classes used to implement the curator subsystem is presented below.

CuratorSubsystem: This is the main class of the subsystem and it extends the *Subsystem* class. It also defines the class file that represents the database table used to store the state of wizards. This class also instantiates each of the “SubsystemTask”s namely:

- *CreateNewAssignmentWizard*

- *ViewSubmissionGradesWizard*
- *SubmitAssignmentWizard*
- *ViewReportsWizard*

FacultyTaskWizardState: This class represents the database table that is used to store the state of the wizard sequence. The member variables represent the columns present in the database. This is where WebObjects uses enterprise objects and key-value pairs to store and retrieve values from database tables.

CreateNewAssignmentWizard: This class initializes the database state variable to store the selected options at each stage.

CreateNewAssignmentWizardState: This class extends the *FacultyTaskWizardState*. The *FacultyTaskWizardState* is sub-classed here so that only those values that are required for this interaction are used and modified. This class defines the number of pages that exist in the Wizard as well as provide the class names for those. As a result Web-CAT can call the appropriate classes at runtime and have display the right pages. The pages in this interaction are

- *SelectCourse*
- *AssignmentDetails*
- *SelectGradingProfile*
- *Confirmation*.

SelectCourse: This class is the user interface element for displaying a list of courses that the instructor is taking this semester. This component contains a reusable component to display the database values and also contains method to create a new assignment and store database values.

AssignmentDetails: This class contains methods to set the member variables for each of the assignment objects. This class also contains UI elements for the user to fill.

SelectGradingProfile: This class presents the user with a list of grading profiles to select from as well as four file upload boxes to upload custom scripts for compiling, executing instructors version, executing submitted programs and grading submitted programs.

Confirmation: This class is the UI element that shows the details provided by the faculty. All other classes for tasks are modeled in similar manner.

ViewGradesWizard: The sequence for this task is: *SelectCourse*, *SelectAssignment*, *SelectStudents* and *ViewGrades*.

SubmitAssignmentWizard: The sequence is: *SelectCourse*, *SelectAssignment*, *UploadProgramFile*, *UploadTestCaseFile*, *ConfirmSubmission*, *Results*.

ViewReportsWizard: The sequence is: *SelectCourse*, *SelectAssignment*, *Results*.

4.2.3 Core subsystem

The core subsystem forms the heart of Web-CAT. The core subsystem is responsible for the successful implementation of the curator and the other subsystems within Web-CAT. The core subsystem serves as a high-level framework for supporting multiple subsystems. Each of the subsystems in turn can consist of a number of tasks that the user can perform. The core subsystem itself supports a few administrative level tasks for editing the database, setting up course information and setting up access levels of users.

Other than these tasks that are visible to the user, the core exports a relatively small number of interfaces that a subsystem needs to implement in order to function as part of Web-CAT.

For the core to recognize the existence of a subsystem the right interfaces need to be implemented and the name of the subsystem needs to be placed in a “properties” file along with its main class information. To provide maximum flexibility core supports subsystems in the form of an executable JAR as well. However the developer of the subsystem shall have to place the JAR file in a local specific directory.

To perform the set up task for each of the subsystems that are listed in a properties file, the core maintains a database table that lists each of the subsystems as well its main class information. The main class file is used to invoke the subsystem.

Using the interfaces implemented by the subsystems the core subsystem recognizes the list of tasks that are supported by each subsystem and invokes respective tasks whenever a user clicks on the link. Once all the tasks from all subsystems are loaded, the client user interface presents them in a table, each with a respective hyperlink.

When the user clicks on this link, the entire interaction is then handled by the subsystem and Web-CAT simply supervises the proceedings.

4.2.3.1 The wizard-based interaction

The core subsystem is responsible for the management and execution of each and every wizard based task within Web-CAT. Each of the subsystem tasks is modeled as a series of wizard pages. A user makes a number of choices at each point of the interaction. Each choice made by the user is recorded in a database table. This helps us to remember the choices made by the user at various points of the interaction as well as resume the wizard from where the user left.

The core subsystem uses a number of classes to handle this interaction. The core also provides a wrapper for the wizard and has implementations for all the buttons of the wizard. Any additional functionality that is needed by the subsystem needs to overload the respective methods and then place a call to the parent method.

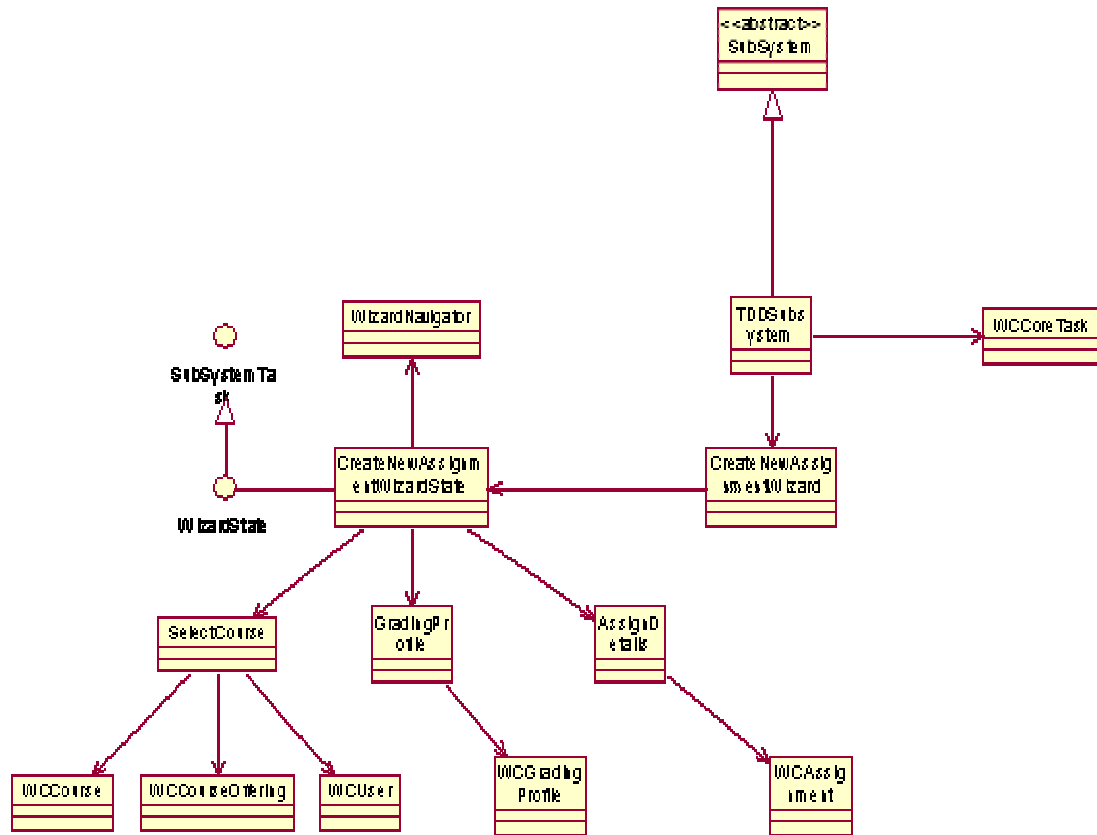


Figure 4.24 Wizard navigation

The database where the state of the wizard is stored belongs to the respective subsystems. This helps introduction of subsystem-specific logic as well as keeps the core subsystem aloof of the low-level subsystem specific details.

The status of each wizard sequence is also stored in respective databases. This enables to distinguish tasks that are completed from those that were suspended due to some reason and need to be resumed.

4.2.3.2 Brief description of important classes

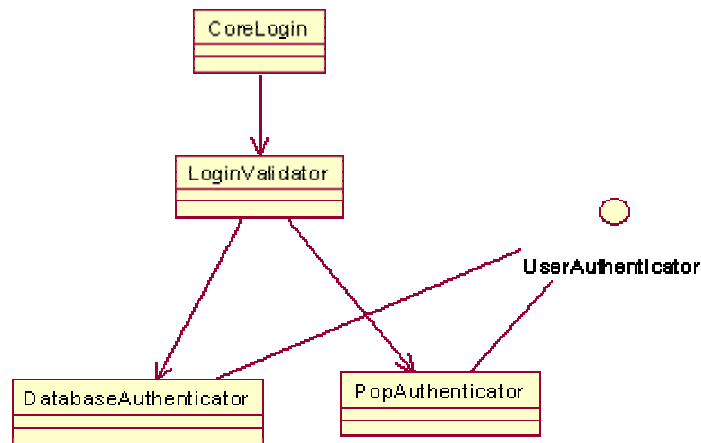


Figure 4.25 Login validation

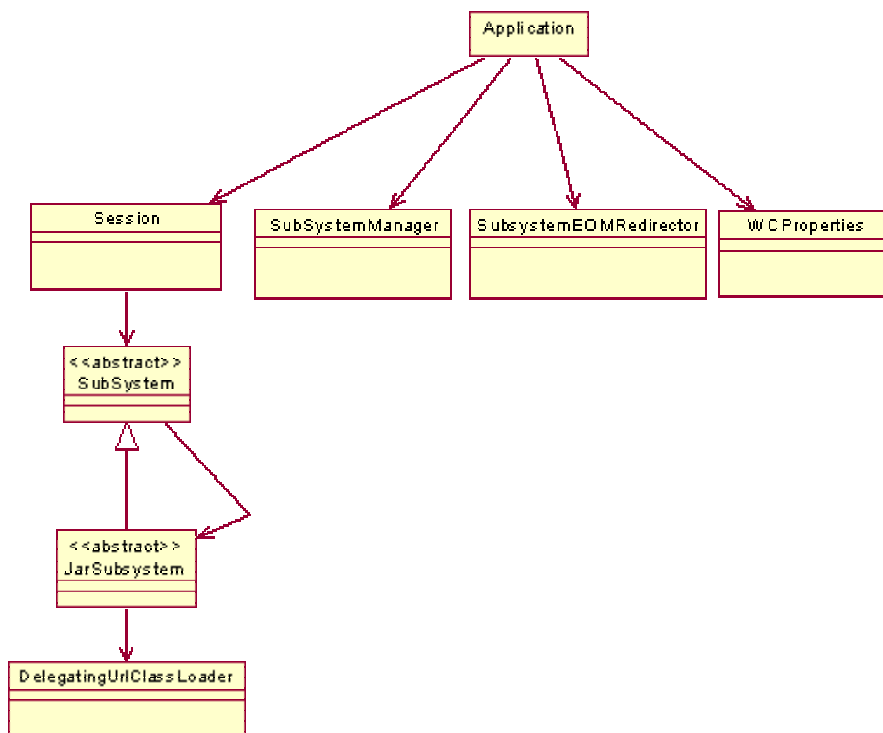


Figure 4.26 Web-CAT

Application: The application class creates an instance of the running application. There is one Application object per application instance. This class is responsible for setting up the database models (EOModels) that are present in the subsystem. It is also responsible

for deleting stale sessions. It calls the Subsystem manager abstract class to set up subsystems. All members of this class are shared across the entire application instance however not across instances.

Session: This class handles the session management for each user. It is also responsible for timing out sessions and setting up the access control levels for the current logged-in user.

SubsystemManager: The SubsystemManager is responsible for loading the subsystems. It first loads all the subsystems that exist in the specific jar directory as executable jars. After loading subsystems from the directory it loads all subsystems that are present in the properties for this instance's setting. It is an abstract class with implementations for a few non-abstract methods.

JarSubsystem: JarSubsystem is responsible for loading all subsystems that are stored as executable jar files. It is an abstract class with implementations for some non-abstract methods.

SubsystemTask: This is an interface that is used to maintain the state of the subsystem wizards. This is used to pass their state back and forth from the subsystem to Web-CAT.

Subsystem: This class defines the interface used by Web-CAT to communicate with the subsystems.

CoreLogin: CoreLogin implements the user interface login functionality for Web-CAT. It presents the user with a login component and instantiates the LoginValidator.

LoginValidator: This instantiates the selected authenticator for this application instance using UserAuthenticator. The authenticator can be either a database authenticator or a VT mail server authentication. The database authenticator is used for the purpose of development.

UserAuthenticator: An interface for all authentication techniques.

DelegatingUrlClassLoader: This class serves as the central loading point for each of the subsystem. Web-CAT needs a Class loader since it stores the main class information in a database for all subsystems and tasks.

WCPlainPage: This is a reusable component created by Web-CAT. All pages that are not wizard pages need to extend this class. It contains methods for page title, window title, logout buttons etc.

WCWizardPage: The *WCWizardPage* class is an extension of the *WCPlainPage* class and it contains additional buttons. All the buttons for the wizard interaction are placed in this class. The methods that implement the back, cancel, comeBackLater, etc buttons of a wizard are defined in this class.

WCWizardPageDescriptor: A class used to describe the individual wizard page properties. It contains a brief description of the step that this page corresponds to. Example: Select a course.

WizardNavigator: This is an interface covering the basic requirements for classes used by the subsystem wizards. This helps them pass and store their state.

WizardState: This interface returns a *WizardNavigator* and is used by subsystems to store and pass state of wizard.

4.3 Benefits from this design

Web-CAT takes the initial step in providing a framework for introducing the fundamentals of software testing in classrooms. Its high extensibility is mainly due to the architecture and design that has been presented. The use of multi-tier architecture is ideally suited for an application that must support many concurrent users at the same time. The runtime plug in support for additional subsystems provides the developers with sufficient flexibility to plug in modules that cover other aspects of testing.

Chapter 5: Experimental Evaluation

5.1 Evaluation plan

In order to assess the effectiveness of Web-CAT in fostering software testing education, students used this prototype in the CS 3304 Comparative Languages course. The evaluation plan [7] encompassed statistical analysis of data collected on the performance of students as well as traditional methods of questionnaires and feedback from students using the center.

The evaluation plan had the following parts:

1. **Student feedback opportunities:** An online feedback mechanism was provided to students using Web-CAT. Using this feature, students commented on the various aspects of the learning center. Feedback from students in this respect played an important part of formative evaluation, most importantly during the initial use in courses. Such a feedback mechanism allowed the students to express their opinions about the environment and report any anomalies they encounter during their experience with Web-CAT.

2. **Student surveys:** At the end of the course where Web-CAT was used, students were given an optional survey. The survey was directed at learning their overall experience with Web-CAT, views on its effectiveness, and suggestions for improvement.

3. **Student grades:** Student grades on programming assignments throughout the semester were analyzed relative to historical data for prior offerings of the course to look for any statistically significant difference in overall programming performance.

- 4 **Measurement and tracking of defect rates:** Web-CAT collected programming assignments and submissions. The grades on each of these submissions were compared with those on historical submissions that have been collected in courses using the Curator. This historical data provide a basis of comparison in terms of defect rates and bug densities. All submissions were subjected to powerful test oracles and variegated test cases to get an estimate of the number of bugs that each program had.

5.2 Data collection

To assess the effectiveness of Web-CAT and TDD, a meticulous data collection process was followed. Every student activity on Web-CAT was recorded and extensive logs were maintained. Every file submitted to Web-CAT was stored on the file server and the results of each submission were recorded in the database.

Student submissions were graded using the process explained in section 4.2.2.1. The final score was computed after deducting any applicable late penalties. The database maintained information about the final score, the late penalties, the actual score (final score + late penalty), the coverage score and the test validity score for each submission. At a later stage, all these scores were exported to excel worksheets in order to facilitate statistical analysis of such data.

Students from the current semester submitted a set of test cases along with their solutions for the assignment. Each of these test suites was run against a source instrumented reference implementation. A score was assigned to each set of test cases based on the coverage of the problem domain.

The other attributes of particular interest for each student were the time of first submission, the time of the last submission, the number of submissions as well as the difference from the due date on each of these values.

Similar metrics were retrieved from submissions during the spring 2001 offering of CS 3304. However in 2001, the students were only required to submit their implementation and not a set of test cases. Their solutions were tested against test cases generated by an automated test case generator written by the instructor. Each of these test case files were collected and run against the reference implementation to collect a coverage measure.

5.2.1 Collecting derived data

In order to make a number of comparisons between the two student populations as well as speculate on the effectiveness of the prototype and TDD we conducted some more data collection activities.

Student submissions from 2001 along with the test cases generated for each submission were collected and run through Web-CAT for grading. As a result we negated

the influence of the grading mechanism when comparing scores of students. These scores were recorded and later analyzed to check for differences. Similarly submissions from 2003 were graded once again using the Curator grading set up. A comparison of these scores helped us gain valuable insight on student programs and their corresponding scores.

Students from the current semester gained constant feedback on all aspects of their implementations including coverage scores. The coverage score is the determining factor in the grading mechanism. Thus the students from 2003 could always resubmit their solutions with additional test cases try to improve their coverage scores. However the students from 2001 did not have such an opportunity. As a result, it was necessary to compare the scores from both the populations neglecting the coverage scores. Since each part of the score was recorded separately on all submissions graded using Web-CAT, it was possible for us to collect scores excluding coverage scores and analyze them.

5.3 Experimental design

Hypothesis: The null hypothesis is that there would be no significant difference between the performances in terms of scores of students on the first programming assignment from 2001 and 2003.

Subjects: The subjects using Web-CAT were undergraduate students from Virginia Tech enrolled-in CS 3304, Comparative Languages, in spring 2003. These 56 students were typically at their junior or senior level and working towards a bachelor's degree in Computer Science.

A similar group of 59 undergraduate students from the spring 2001 offering of the Comparative Languages course served as subjects in this experiment. As explained above, similar metrics were collected on students from both populations.

Experimental setup: Care was taken to ensure that both student populations were operating under the same environment. The programming problem presented to students was the same during both the years.

The programming assignment required students to diagram English sentences according to a pre-defined grammar [23]. The details of the assignment are attached in Appendix C. The parser was created using the Pascal programming language and

appropriate guidelines were provided on the website. Students from the current year were encouraged to use the TDD methodology of code development for this particular assignment and required to submit the suite of test cases they used for grading through Web-CAT while students from 2001 used the Curator system at Virginia Tech for submissions. The sizes of the student groups were large enough to prevent differences between individuals from affecting the results.

5.4 Results and discussion

The collected student scores were analyzed using t-tests. The t-test formula used also accounts for the unequal variances between the two groups, if any. As the t-test formula is a ratio of the difference of means between the two populations to the standard error of the difference, it applies a correction for unequal variances. A complete discussion of how t-test is acceptable even in cases of unequal variances is discussed in [30]. A detailed analysis report of our findings and results is presented in the sections below categorized by different scores.

5.4.1 Comparing raw scores

Year	N	Mean	Std. Deviation	Std. Error
2003	56	47.0179	2.1783	0.2911
2001	59	46.9387	6.6385	0.8643

Table 5.1: Final scores of students on respective grading systems

A comparison of the “raw” scores (final scores excluding late penalty if any) was conducted between the two student populations of 2001 and 2003. The Curator assigned the scores of students from 2001 while those for students in 2003 were assigned by Web-CAT. The average score of students from 2003 was higher than the average score from 2001. However this difference was not significant. The students from 2001 scored on an average 46.9 points on a scale of 50 while those from 2003 did a little better with an average of 47 points.

5.4.2 Comparing Web-CAT scores

A comparison of the final scores had little significance as submissions during 2001 were graded using different criteria than those in 2003. In order to compare

differences between the two populations we ran each of the previous year's submissions through Web-CAT and each of the current year's submissions through the Curator. Students during 2001 did not submit a test suite along with their solutions. Implementations in 2001 were graded using a test case file generated by an automated test case generator written specifically for the assignment. These test case files were used for grading through Web-CAT assuming that students would themselves write those test cases.

Group Statistics

	YEAR	N	Mean	Std. Deviation	Std. Error Mean
WEBSC	2001	59	38.4237	13.04925	1.69887
	2003	56	47.0000	2.16585	.28942

Table 5.2: Final scores of all students on Web-CAT

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
WEBSC	Equal variances assumed	42.575	.000	-4.854	113	.000	-8.5763	1.76681	-12.07663	-5.07591
	Equal variances not assumed			-4.977	61.361	.000	-8.5763	1.72334	-12.02190	-5.13064

Table 5.3: Significance of scores on Web-CAT

Students in 2003 scored higher when graded using Web-CAT. The average score of students in 2003 on Web-CAT was approximately 8.5 points higher.

5.4.3 Comparing Curator scores

Group Statistics

	YEAR	N	Mean	Std. Deviation	Std. Error Mean
CURSC2	2001	59	46.9386	6.63846	.86425
	2003	56	48.1808	2.17579	.29075

Table 5.4: Final scores of all students on Curator

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
CURSC2	Equal variances assumed	16.952	.000	-1.334	113	.185	-1.2422	.93140	-3.08751	.60302
	Equal variances not assumed			-1.362	70.914	.177	-1.2422	.91185	-3.06046	.57597

Table 5.5: Significance of scores of students on Curator

The Curator scores of the two populations however were not significantly different. Students from the year 2003 scored 48.2 points while students from 2001 scored an average of 46.9. We speculate that these results are not significant as there is very little margin of improvement because of the higher class averages.

5.4.4 Comparing scores without coverage information

It appears that a significant difference between the two populations when graded using Web-CAT was due to the nature of the grading mechanism. The grading scheme used on Web-CAT gave importance to the coverage score, which was obtained based on the number of code branches excited by the submitted test suites.

Group Statistics

	YEAR	N	Mean	Std. Deviation	Std. Error Mean
NOCOVER	2003	56	49.9554	.2743	3.666E-02
	2001	59	42.8842	13.8915	1.8085

Table 5.6: Scores excluding coverage information

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
NOCOVER	Equal variances assumed	68.459	.000	3.808	113	.000	7.0711	1.8571	3.3919	10.7504
	Equal variances not assumed			3.909	58.048	.000	7.0711	1.8089	3.4503	10.6920

Table 5.7: Significance of scores excluding coverage information

As a result, a comparison of scores was done excluding the coverage measures from the Web-CAT generated scores. Our findings were no different. Students in 2003

scored significantly higher than those in 2001 when coverage measures were not considered.

The significance of the above results allows the null hypothesis to be rejected. Thus there was a significant difference between the two student populations and students from 2003 produced solutions that obtained better scores on Web-CAT.

5.4.5 Comparing coverage scores

All graded submissions from 2001 were originally tested using a test data generator. The test cases produced by the generator were extracted into files to assess their coverage of the instructor's reference implementation. The generator's coverage scores from 2001 were compared with the student coverage scores from 2003.

Coverage scores	2003 Students	2001 Students
Mean	0.935692	0.899942
Variance	0.002607	0.000252
Observations	56	59
P(T<=t) one-tail	5.73E-07	
T Critical one-tail	1.658095	

Table 5.8: Coverage score comparisons

Students from 2003 submitted test cases that were on an average much better than those generated automatically ($p < 0.05$). We believe that this is one of the major successes of our research. Students are more aware of a large number of execution sequences and as a result have structured their solutions to meet these requirements. This also helps us make some conclusions about the robustness of the submitted programs and their ability to handle variegated execution sequences.

5.4.6 Comparing submission times

All submissions to both Web-CAT and the Curator were time-stamped. Tables 5.9 and 5.10 summarize the differences between the two groups in terms of the average time of the first submission and the time of the final submission.

Group Statistics for "Time of initial submission"				
Year	N	Mean	Std. Deviation	Sig. (2-tailed)
2003	56	6036.4982	4875.2101	.001
2001	59	3101.5593	4792.8679	

Table 5.9: Time of initial submission (minutes before the deadline)

Group Statistics for “Time of final submission”				
Year	N	Mean	Std. Deviation	Sig. (2-tailed)
2003	56	1852.0446	1548.5362	.991
2001	59	1846.5763	3464.2816	

Table 5.10: Time of final submission (minutes before the deadline)

Students from 2003 started working on the assignment earlier. Thus Web-CAT and TDD created enough excitement amongst the students to start programming at an earlier date. ($p < 0.05$).

5.4.7 Comparing number of submissions

The students during the current semester were allowed to make an unlimited number of submissions for the assignment while those in 2001 were limited to only 5 submissions. Scores were collected for both groups for the first and the last submission.

We analyzed this information to make some predictions about the effect of the number of submissions on the final scores obtained as well as the learning curve of students. However unfortunately there were no noticeable trends in this data. The number of submissions did not significantly make any difference to the final scores obtained by any of the students.

5.4.8 Comparing bug densities

The estimation of the bug density (bugs/KLOC) of the respective student populations was a key component of our evaluation process. To arrive at measures of this attribute a meticulous and repetitive process was followed.

To determine the number of bugs present in each of the student implementations, we used a reference set of **unique** test cases that obtained a coverage score of 100% on the instructor’s reference implementation and ran it against the student’s solution. The total number of test cases that fail on this unique test suite could then be directly related to the number of bugs in the implementation.

The test cases submitted by every student were collected in one file and the final submissions of every student were run against these. This produced a set of 45,486 test cases. This test case file also attained a 100% coverage score on the instructor’s reference implementation. All these test cases were valid however there were a large number of duplicates present.

To remove duplicates from this huge test suite, the entire test suite was run against each of the submissions from both years. A simple two-dimensional matrix was created with student executables as the columns and every test case made up the row. The corresponding row, column entry consisted of a 0 (fail) or 1 (pass). The totals of each row were recorded as an additional column. Unique rows from these files were extracted using a number of UNIX utilities. Every unique row corresponded to a unique test case that would pass and fail the student solutions like no other test case. As the number of student programs was large we reduced the unique set of test cases to 1064. These 1064 test cases executed different aspects of the student programs and also attained a coverage score of 100% on the reference implementation. This test oracle formed the basis of our bug density estimation.

A random sample of 9 programs from each set, 2001 and 2003 students was taken to arrive at an estimate of bug density for the entire population of students. These 18 programs were modified so that they would pass each of the 1064 test cases. After successful modification of each of the student programs, we calculated the total number of lines modified/added/removed to get a full score on the test cases. These values were then normalized to a scale of 1000 lines. Finally ratios were obtained for the normalized number of lines to the number of test cases that each of these students failed. Results of these comparison as well as trends in these values are discussed in the following chapters.

After reducing the test suite to a set of unique test cases, we collected the number of test cases that failed on each submission. A number of submissions in both sets were too buggy to execute completely on this test suite. After removing these buggy implementations we were left with a reduced set of submissions from each year.

Group Statistics

	YEAR	N	Mean	Std. Deviation	Std. Error Mean
NRFAILED	2003	46	264.7609	197.0795	29.0578
	2001	50	390.0600	150.0414	21.2191

Table 5.11: Test cases failed

Independent Samples Test									
		Levene's Test for Equality of Variances		t-test for Equality of Means					
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference
NRFAILED	Equal variances assumed	5.481	.021	-3.522	94	.001	-125.2991	35.5796	-195.9432 -54.6551
	Equal variances not assumed			-3.482	83.883	.001	-125.2991	35.9806	-196.8520 -53.7463

Table 5.12: Significance of test cases failed

Out of the reduced set of 1064 test cases, a significantly large number of cases failed on the 2001 submissions. This gave us more confidence in the robustness of the 2003 submissions over those in 2001 ($p < 0.002$).

The above tables show the comparison of the data for only those programs that successfully completed execution of the entire suite of test cases. 10 programs from 2003 and 9 programs from 2001 were extremely buggy and are not considered in this comparison. The mean number of bugs detected in the solutions of 2003 is comparatively much less than that in 2001. The comparison of the number of test cases that failed including those buggy programs is presented below.

The number of bugs in both populations was extremely high compared to industry standards however this was what we had expected as the populations under consideration mostly comprised of junior and senior level students

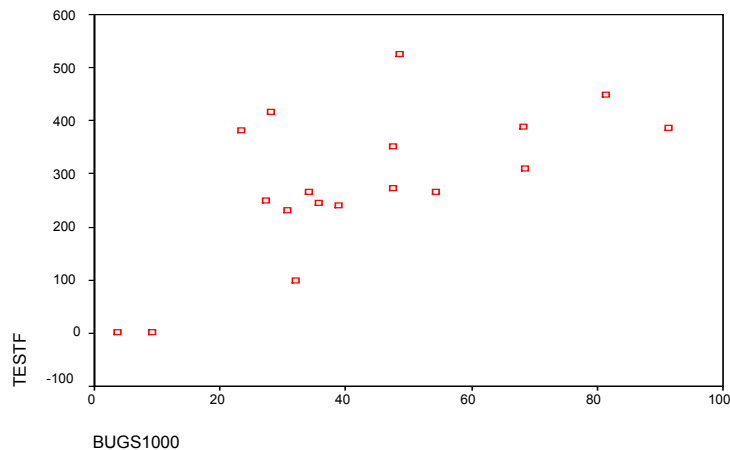


Figure 5.1 Bugs v\vs tests failed

As expected the number of bugs in each solution had a significant bearing on the number of test cases failed from the 1064 unique test cases. There was a statistically

significant linear relationship between the number of bugs and the number of test cases failed. However the year of the students did not make any difference to their bug density values.

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.651 ^a	.424	.388	110.87

a. Predictors: (Constant), BUGS1000

Table 5.13: Model summary

ANOVA^b

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	144734.1	1	144734.054	11.775	.003 ^a
	Residual	196673.9	16	12292.122		
	Total	341408.0	17			

a. Predictors: (Constant), BUGS1000

b. Dependent Variable: TESTF

Table 5.14: ANOVA results for bug density

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	113.522	55.792		2.035	.059
	BUGS1000	3.951	1.151	.651	3.431	.003

a. Dependent Variable: TESTF

Table 5.15: Linear regression analysis

This significant linear relationship was used to predict the bug density values for the remaining programs. Due to the linear relationship we could correctly predict the bug density values for the rest of the population using a simple slope-intercept equation.

Bugs per KLoc	2003	2001
Mean	38.29389	70.01519
Observations	46	50
P(T<=t) one-tail	0.000332	
t Critical one-tail	1.661226	

Table 5.16: Significance of bug densities

Once again the above tables ignore the buggy programs. The following tables present comparison of the entire student populations.

<i>Bugs per KLoc</i>	<i>2003</i>	<i>2001</i>
Mean	39.13477	70.72981
Observations	56	59
P(T<=t) one-tail	2.64 E-05	
t Critical one-tail	1.661226	

Table 5.17: Significance of bug densities for all students

After running a simple t-test on these obtained bug densities we obtained results that were very encouraging. The bug densities in the current population had been decreased by almost 45%. This was one of the major successes of our research.

5.4.9 Inferences

Our null hypothesis that there is no significant difference in the performance of the two groups of students is rejected after noting the above observations. Thus we conclude that there is a significant difference in the performance of the two groups on the first programming assignment. The factors that have influenced these differences are the introduction of TDD and Web-CAT during the current semester. As most of the other independent variables were controlled and efforts taken to minimize their effects, we could reasonably assume that use of our approach has helped students in 2003 develop more robust code than those from 2001.

Chapter 6: Survey Analysis

6.1 Student survey

The qualitative assessment of Web-CAT was conducted with a survey distributed to students taking the Comparative Languages course at the end of the first programming assignment. The main purpose of the survey was to elicit student responses giving us an indication of their perceptions about our approach. Questions on the survey were also directed at getting feedback on Web-CAT and its use with TDD. The survey qualified for exemption from the Institutional Review Board (IRB). The exemption was approved as the research was conducted in established or commonly accepted educational settings, involving normal educational practices.

6.2 Questionnaire for CS 3304

For this questionnaire, “**Web-CAT**” refers to the current online submission system used in *CS 3304* this semester, while “**Curator**” refers to the previous version you may have used in other classes.

Have you used the previous version of the Curator system at Virginia Tech to submit programming assignments in other classes?

Yes

No

Please circle your response to the following statements based on the scale below:

	Strongly Disagree	Disagree	Neutra l	Agree	Strongly Agree
1. Web-CAT is easier to use than the Curator system.	1	2	3	4	5
2. Results produced by Web-CAT are more helpful in detecting errors in my program than those produced by the Curator.	1	2	3	4	5

- | | | | | | |
|---|---|---|---|---|---|
| 3. Web-CAT provides better help features than the Curator. | 1 | 2 | 3 | 4 | 5 |
| 4. Using TDD increases my confidence in the correctness of my programs. | 1 | 2 | 3 | 4 | 5 |
| 5. Using TDD helps me complete my programming assignments earlier. | 1 | 2 | 3 | 4 | 5 |
| 6. Using TDD increases my confidence when making changes to my programs. | 1 | 2 | 3 | 4 | 5 |
| 7. TDD increases the amount of time I need to complete programming assignments. | 1 | 2 | 3 | 4 | 5 |
| 8. Using TDD makes me test my own solution more thoroughly. | 1 | 2 | 3 | 4 | 5 |
| 9. Using TDD makes me take a more systematic approach to devising tests. | 1 | 2 | 3 | 4 | 5 |
| 10. With TDD, I spend more time writing code. | 1 | 2 | 3 | 4 | 5 |
| 11. With TDD, I spend more time writing tests. | 1 | 2 | 3 | 4 | 5 |

- | | | | | | |
|---|---|---|---|---|---|
| 12. With TDD, I spend more time debugging code. | 1 | 2 | 3 | 4 | 5 |
| 13. In the future I am more likely to use TDD even if it is not required by the assignment. | 1 | 2 | 3 | 4 | 5 |
| 14. As a result of using TDD in this class, I am now able to write better test cases. | 1 | 2 | 3 | 4 | 5 |
| 15. Using TDD adversely affected my grade. | 1 | 2 | 3 | 4 | 5 |
| 16. Without using TDD, I would have scored higher on this assignment. | 1 | 2 | 3 | 4 | 5 |
| 17. The tddpas.pl script distributed for student use was hard to use. | 1 | 2 | 3 | 4 | 5 |
| 18. I preferred using Web-CAT instead of the provided tddpas.pl script I could run myself. | 1 | 2 | 3 | 4 | 5 |
| 19. Even if it were not required, I would like to use Web-CAT to test my programs for class before turning them in. | 1 | 2 | 3 | 4 | 5 |
| 20. The Web-CAT environment | 1 | 2 | 3 | 4 | 5 |

provides excellent support for programming using TDD.

21 What are the things, if any, you found most useful or valuable about using TDD or Web-CAT?

22 What are the things, if any, you found least useful or valuable about using TDD or Web-CAT?

23 Have you any suggestions for improving Web-CAT, TDD or software testing for your programming assignments?

6.3 Survey evaluation

Each of the surveys was collected from students and later evaluated using qualitative as well as quantitative evaluation procedures. Student responses on each of the questions were recorded in excel worksheets. These values were later analyzed for observing interesting responses. The following figure shows the responses on each of the questions on the survey.

All students enrolled-in the course had used the Curator system at Virginia Tech for making submissions previously. This was indeed of great help as it added more weight to their responses on questions that asked them to compare Web-CAT and Curator.

6.3.1 Discussion of responses

Statement 1: A majority of the students felt that Web-CAT was easier to use than the previous Curator system. The Curator system, used in a number of undergraduate courses for online submissions, has a number of usability problems. The student responses were testimony to this fact and they were overall happy with the user interface design of Web-CAT and its wizard-based interaction.

Statement 2: One of our design goals was to provide effective and appropriate feedback to students on their performance as well as pinpoint anomalies in behavior of

submitted programs. A number of students felt that we did successfully achieve this goal. Out of the 49 students that were present for the survey, 34 students felt that Web-CAT helped detect more errors than the Curator does and provided better feedback features.

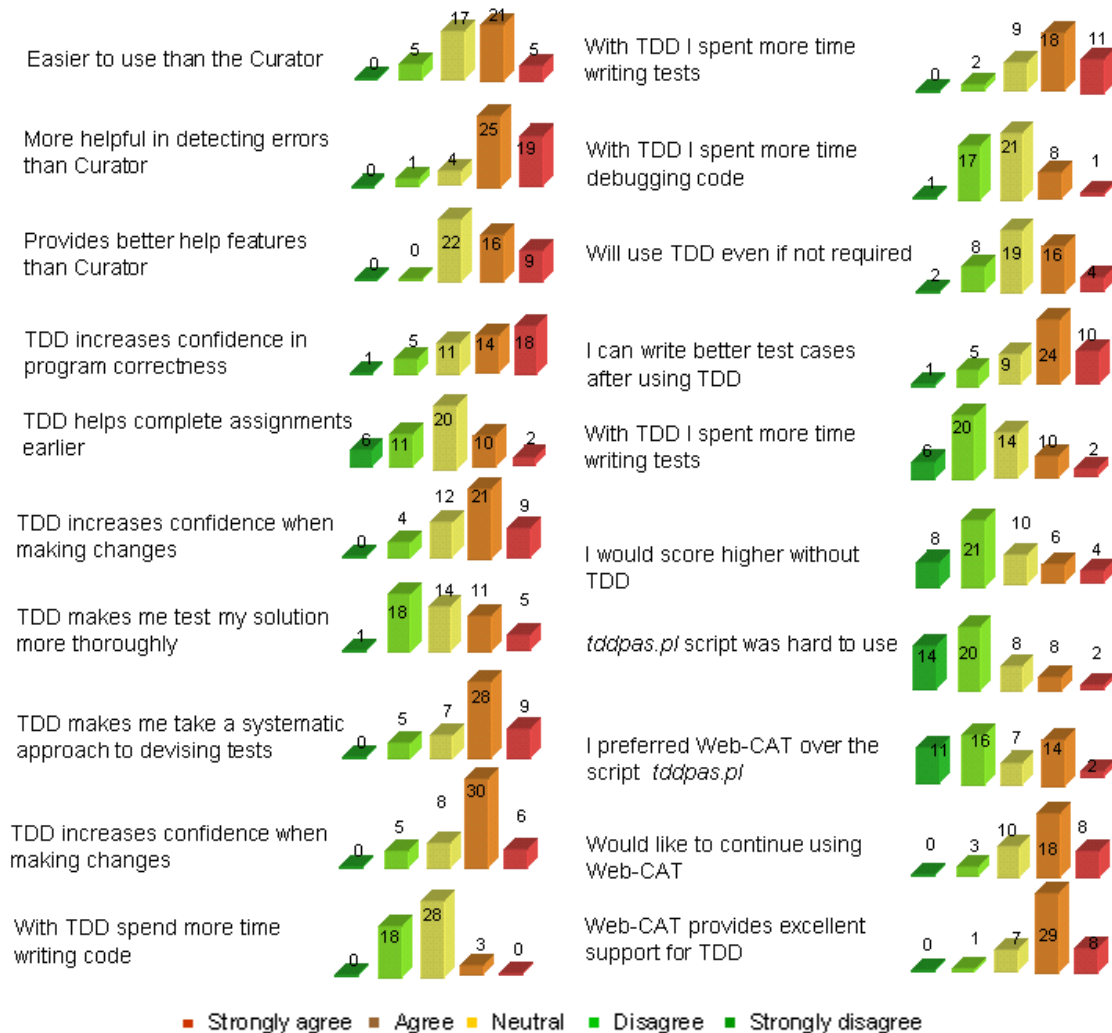


Figure 6.1 Response of students on survey questions

Statement 3: The student response to this question was binary. Students either were neutral to this statement (22) or they agreed to this statement (25). The help menu of Web-CAT provides information based on the current situation of the student and description of the page of the wizard that is currently displayed. None of the students disagreed with this statement.

Statement 4: The overall response of student to the TDD strategy was extremely encouraging during class activities. A similar trend was observed in their responses on the survey as well. A vast majority of the students, 32 in total, either partially or strongly

agreed to this statement. They believed that TDD helped them gain increased confidence in the correctness of their program. This is mainly due to the large number of test cases that are written and new code is added only when each of these test cases is successfully tackled.

Statement 5: The students however did not strongly feel that using TDD helped them complete their assignments at an earlier date. More number of students disagreed with this statement. The reason for such a response is mainly due to the fact that TDD was used for the first time in a course and students were least equipped to effectively practice such a strategy.

Statement 6: This statement once again was aimed at eliciting student perspectives on TDD. A majority of the students, 33 of them, agreed with this statement. They felt that they were more confident while making changes to already existing code. Thus TDD helped them gain sufficient confidence in the operation of already implemented code.

Statement 7: Responses to this statement established sufficient confidence that even though TDD did not help them complete assignments earlier it did not introduce extra overhead and delay the completion of assignments. In fact around 19 students disagreed with this statement.

Statement 8: Most of the students agreed that TDD helps them test their assignments more. Thus one of our goals was achieved and students were provided sufficient exposure to testing programs.

Statement 9: The responses on this statement helped us understand that students took a more systematic approach to devising test cases. The repeated use of TDD in courses shall help students' test more often and completely eradicate the practice of relying on instructor provided sample test data. Thus testing activities will come naturally to students in such an environment.

Statement 10: Almost all students responded in a neutral manner to this statement and the rest of them believed that they spent more time writing tests.

Statement 11: This statement simply complements the above and serves to verify the responses on statement 10 and 11. The students spend more time testing their code

than developing it. As a result students are now more aware of the testing fundamentals and are capable of approaching the task of testing in a systematic manner.

Statement 12: A large number of the students were neutral to this statement. The other half of the class disagreed that they spent more time debugging code.

Statement 13: A mixed response was elicited for this statement. A larger section of the student population was neutral to this idea however of the remaining students, many were keen on using it in future for programming activities.

Statement 14: The responses to this statement once again prove the effectiveness of TDD in educating students to write better test cases. We believe that with such activities students shall become better testers and developers.

Statement 15: Most of the students taking the course disagreed with this statement. The appreciation for TDD amongst students was once again noticed in these responses.

Statement 16: 29 students out of a class of 56 disagreed to this statement. The majority of the students felt TDD did not adversely affect their grade however around 10 students claimed that they would have scored higher without using TDD.

Statement 17: This statement and the following few were aimed at learning the perspectives of students on using Web-CAT and instructor provided tools for running the test cases against their own solutions. A PERL script was distributed to students that would help them run these test cases against their implementations at their workplace if they were not able to submit online to Web-CAT. Students were more or less happy with the operation of this script. Students found this script easy to use.

Statement 18: Responses to this statement confirm our above inference. The primary advantage of the tddpas.pl script file was that students could use it offline and would not have to connect to the Internet for receiving feedback on their test cases. The script was limited in its functionality as it did not produce the same graded output as Web-CAT but produced a JUNIT style output. It simply gave the number of test cases that were run and the number of test cases that failed along with their labels.

Statement 19: Most of the students were interested in using Web-CAT in future for turning in programming assignments and testing them. Thus we were successful in

introducing the students to testing activities and invoke sufficient interest to continue them in future.

Statement 20: Almost all students agreed and were convinced that Web-CAT provides excellent support for using TDD. Thus the use of Web-CAT and TDD in classroom activities helps introduce software-testing principles if accompanied by similar code development activities.

6.3.2 Discussion of responses to open-ended questions

A qualitative analysis of the student responses on the last three open-ended questions revealed a number of trends and perspectives.

A number of students once again specified that TDD gave a lot of assurance with respect to the correctness of their programs while they were still in the process of developing it. The use of TDD also helped them gain increased confidence in the programs while making changes. Thus it helped avoid the traditional big-bang approach that students indulge in. Responses on both the open and the closed-ended questions suggested this trend.

Students were also satisfied with the overall operation of Web-CAT however expressed concerns about its stability. One possible reason for this was the innumerable last minute changes that were made to the prototype, as a few race conditions were hard to reproduce. Overall students appreciated the fact that it produced results on submissions immediately and supported unlimited submissions. They therefore had a chance to review their defects and gain a higher score.

The feedback provided by Web-CAT on programming assignments was found to be the most useful thing by students. However many students expected more feedback especially on the coverage metrics and tips to improve their scores. The students heavily relied on the feedback provided by Web-CAT to make changes to their implementations and score higher points. Web-CAT failed to provide feedback on coverage scores and as a result students were not completely sure as to what aspects of the problem domain are not being tested.

Lastly students suggested that the wizard sequence be reduced once the first submission is made. They wished that after the first submission only a single page be

presented that allowed them to upload both, the test case file and the source implementation. They were tired of repeatedly providing the same details every time a submission was being made.

6.4 Summary of student responses

Students were excited about the use of Web-CAT and TDD during the course. Their responses on the survey helped us gain a better understanding of their learning abilities and their perspectives on the use of Web-CAT and TDD.

They preferred Web-CAT to the Curator system largely due to its better feedback and usability features. They were also pleased with the system supporting unlimited submissions and gave them a chance to correct their mistakes. However many students expressed concern over the stability of Web-CAT and were also interested in seeing a few changes to wizard sequences. A reduced interaction sequence was advocated by most of the students.

The introduction of TDD helped them gain increased confidence in their program's correctness; confidently make changes to programs, regression test each change and take a systematic approach to test case design. They felt confident of producing better test cases and aware of more testing terminology than before. Most of the students however felt that they spent more time writing test cases than code.

Chapter 7: Conclusion and Future Work

The purpose of this thesis was to develop an interactive learning environment for students to better equip them in performing software-testing activities. This thesis is part of a vision that aims at creating a culture for students where software testing is a religion. Students indulge in testing without any additional pressure or effort. The following sections present a concise overview of the results, contributions and suggest directions for future work.

7.1 Summary of results

The hypothesis we tested was that there was no difference between the performance of the two student's populations from 2001 and 2003.

The data analysis procedures however revealed a significant difference in their performance. Our results indicate that students from the current semester produced programs that were on an average 45% more bug proof ($p < 0.0004$) than those submitted during the spring semester of 2001. The average number of bugs in programs submitted during 2003 was 38, which is close to 45% less than those present during 2001 (70). This was a significant achievement for our research activities.

The second most significant result of our research was that students from 2003 produced test cases that were significantly better ($p < 0.0000$) than those produced by a carefully written automated test script generator. This test script generator was hand coded by the instructor as part of the assignment set up during spring 2001. The use of Web-CAT and TDD significantly improved student performance on programming assignments and helped reduce bugs in student programs by a large number.

The survey analysis revealed that students were very comfortable using TDD. They gained increased confidence in the correctness of their programs, were confident while making changes to the existing solution and were more aware of testing computer software. They believed that they would now be able to do a much better job testing software as well as developing robust programs.

7.2 Contribution

In addition to the conclusion we have the following contributions:

We successfully developed an environment where students can perform software-testing activities as well as gain appropriate feedback on their performance.

The use of Web-CAT is not restricted to Virginia Tech alone, it can be easily extended to serve other universities and we strongly encourage people to use Web-CAT and TDD in classrooms.

We introduced a new approach to automatically assessing student programs and thereby relieve the teaching assistants and instructors from the overhead of grading computer programs manually. This additional effort can be expended in providing feedback to students on design, coding styles and documentation issues as well as on conducting courses in a more effective manner.

Lastly the numbers presented in the results of our evaluation are obtained from data that was collected after the first programming assignment. Similar analysis on programs submitted later during the semester is bound to yield better results and establish extended confidence in this approach.

7.3 Conclusion

The introduction of Web-CAT and TDD in the CS 3304, Comparative Languages course helped achieve wonderful results. The average bug density of students in 2003 reduced significantly with our approach. More importantly students in 2003 also produced test cases that accounted for more coverage of the problem domain than an automated test script generator written by the instructor of the course. These results give us an indication of the success of our approach. If Web-CAT and TDD are consistently used across all undergraduate courses, testing activities shall become a norm with students and they shall henceforth program differently. They shall be better equipped to produce more robust software and shall reside in a culture that gives sufficient importance to software testing activities.

7.4 Future work

During the course of this research and experimentation, a number of interesting ideas or suggestions came our way.

Many students commented on the wizard-based interaction both during class activities as well as in responses on the survey. One area to touch upon would be to enhance this interaction so that once the first submission is made using a wizard sequence; the student does not have to repeat any of the steps unless he wishes to submit a different assignment. It would be very helpful to introduce navigation links to jump to different pages on the sequence. This would make the process of submitting assignments faster and less irritating for students. Similar navigation on sequences for creating assignments would help instructors simply replace a few things and set up assignments with minimal effort.

Documentation and style are important aspects of any software programming activity. Feedback on these issues is hardly touched upon in our approach. We suggest the use of automated tools such as Clover, for source code instrumentation that help detect undocumented code as well as other aspects of style and design.

Web-CAT can be easily used in courses requiring object-oriented programming as well. However to successfully extend our approach a revised grading criteria would be required. Our suggestion would be to use a reference test suite that captures the end-to-end behavior required by the assignment specification. We suggest the use of such a test suite to measure the completeness as well as correctness of student programs.

Appendix A

A.1 Sample test case file

```
//== Testing verb phrases case number 6
cow saw quickly alice
//--
(("cow") ("saw" "quickly") ("alice"))
//== Second error case number 13
alice alice alice
//--
Input is not a sentence.
//== Only one word case number 14
cow
//--
Input is not a sentence.
//== lean mean green alice saw book case number 19
lean mean green alice saw book
//--
(("lean" "mean" "green") "alice") ("saw") ("book")
//== cow with alice saw quickly book case number 28
cow with alice saw quickly book
//--
(("cow" ("with" ("alice")))) ("saw" "quickly") ("book"))
```

Appendix B

B.1 Script used for phase I of assignment processing

```
#!c:\perl\bin\perl.exe -w
#=====
#   @(#) $Id$
#-----
#   Web-CAT Curator: compile script for Free Pascal submissions
#
#   usage:
#       compile.pl <pid> <working-dir> <script-home-dir> <log-dir> \
#                   <XML-output-file-name> <timeout>
#=====

use strict;
use File::stat;
use Win32::Job;

#=====
# Bring command line args into local variables for easy reference
#=====

my $spid      = $ARGV[0]; # student's pid
my $working_dir = $ARGV[1]; # working dir for compilation
my $script_home = $ARGV[2]; # directory where script is located
my $log_dir    = $ARGV[3]; # directory where logs should be generated
my $XML_result = $ARGV[4]; # XML result output goes here
my $timeout    = $ARGV[5]; # Timeout for compile job

#-----
# In addition, some local definitions within this script
#-----

my $script_log    = "$log_dir/compile-script-log.txt";
my $compiler_log  = "$log_dir/compiler-log.txt";
my $exec_file     = "Executable.exe";
my $scan_proceed = 1;
```

```

my $pascal_file = "";
my $compiler     = "c:/freepascal/bin/win32/ppc386.exe";
my $status       = "success";

#=====
# Script Startup
#=====

# Change to specified working directory and set up log directory
chdir( $working_dir );
if( ! -e $log_dir )
{
    mkdir( $log_dir );
}
if ( -f $exec_file ) { unlink( $exec_file ); }

#=====
# Find the pascal file to compile
# Extensions supported *.pp, *.pas
#=====

my @sources = (<*.pp *.pas>);
if ( $#sources < 0 || ! -f $sources[0] )
{
    open( SCRIPT_LOG, ">$script_log" ) ||
        die "cannot open $script_log: $!";
    print SCRIPT_LOG "Cannot identify a Pascal source file.\n",
        "Did you use a .pas or .pp extension?\n";
    close( SCRIPT_LOG );
    $can_proceed = 0;
    $status = "error";
}
else
{
    $pascal_file = $sources[0];
    if ( $#sources > 0 )
    {

```

```

        open( SCRIPT_LOG, ">$script_log" ) ||
            die "cannot open $script_log: $!";
        print SCRIPT_LOG
            "Multiple source files present.  Using $pascal_file.\n",
            "Ignoring other .pas/.pp files.\n";
        close( SCRIPT_LOG );
    }
}

#=====
# Compilation Phase
#=====
# Create a separate thread for compilation, and run it with a timeout

if ( $scan_proceed )
{
    my $job = Win32::Job->new;
    $job->spawn( "ppc386.exe",
                "$compiler $pascal_file -o$exec_file -vi- -
Fe$compiler_log"
                );
    if ( ! $job->run( $timeout ) )
    {
        $status = "timeout";
    }
    # system( "$compiler $pascal_file -o$exec_file -vi- -
Fe$compiler_log" );
    if ( ! -f $exec_file ) { $status = "error"; }
}

#=====
# Generate XML Output Documenting Result
#=====

open( XML_RESULT, ">$XML_result" );
print XML_RESULT "<?xml version=\"1.0\" ?>\n<result
status=\"$status\">\n";
if ( -f $script_log )

```

```

{
    print XML_RESULT "      <report to=\"student\" format=\"text\" ",
        "name=\"$script_log\" />\n";
}
if ( -f $compiler_log )
{
    print XML_RESULT "      <report to=\"student\" format=\"text\" ",
        "name=\"$compiler_log\" />\n";
}
if ( $status eq "success" )
{
    print XML_RESULT "      <executable file=\"$exec_file\" />\n";
}
else
{
    print XML_RESULT "      <score raw=\"0\" />\n";
}
print XML_RESULT "</result>\n";
close( XML_RESULT );

```

```

#-----
exit( 0 );
#-----

```

B.2 Script used for phase II of assignment processing

```
#!c:\perl\bin\perl.exe -w
#=====
#   @(#) $Id$
#-----
#   Web-CAT Curator: execution script for Free Pascal submissions
#
#   usage:
#       executePascal.pl <pid> <working-dir> <script-home-dir> <log-
dir> \
#           <XML-output-file-name>
#=====
use strict;
use English;
use File::stat;
use Win32::Job;

#=====
# Bring command line args into local variables for easy reference
#=====

my $pid          = $ARGV[0];
my $working_dir  = $ARGV[1]; # working dir for compilation
my $script_home  = $ARGV[2]; # directory where script is located
my $log_dir      = $ARGV[3]; # directory where logs should be generated
my $XML_result   = $ARGV[4]; # XML result output goes here
my $timeout      = $ARGV[5]; # Timeout for compile job

#-----
# In addition, some local definitions within this script
#-----

my $script_log    = "$log_dir/execute-script-log.txt";
my $test_input    = "$log_dir/tests-in.txt";
my $expected_output = "$log_dir/expected-out.txt";
my $student_output = "$log_dir/output.txt";
```



```

my $instr_output      = "$log_dir/reference-out.txt";
my $student_rpt       = "$log_dir/student-tdd-report.txt";
my $instr_rpt         = "$log_dir/reference-tdd-report.txt";
my $assessment        = "$log_dir/TDD-assessment.txt";
my $student_exec       = "Executable.exe";
my $instr_exec        = "$script_home/Instructor.exe";
my $can_proceed       = 1;
my $student_tests     = "";      # test case file
my $status            = "success";
my @student_eval      = ();
my @instr_eval        = ();
my $coverage          = 0;

# From tddpas.pl
#-----
my $version           = "1.0";
my @labels            = ();      # User-provided test case names
my @test_cases        = ();      # test case input
my @expected_output   = ();      # corresponding expected output
my @case_nos          = ();      # test case number for each output line
my $temp_input        = $test_input;      # Name for temp test input
file
my $delete_temps      = 0;      # Change to 0 to preserve temp files

#=====
# Script Startup
#=====
# Change to specified working directory and set up log directory

chdir( $working_dir );
if( ! -e $log_dir )
{
    mkdir( $log_dir );
}

#=====

```

```

# Find the test suite file to use
# Extensions supported *.txt, *.text, *.tst, *.test
#=====

my @sources = (<*.txt *.text *.tst *.test>);
open( SCRIPT_LOG, ">$script_log" ) ||
    die "cannot open $script_log: $!";
if ( $#sources < 0 || ! -f $sources[0] )
{
    print SCRIPT_LOG
        "Cannot identify a test suite file.\n",
        "Did you use a .txt, .text, .tst, or .test extension?\n";
    $scan_proceed = 0;
    $status = "error";
}
else
{
    $student_tests = $sources[0];
    if ( $#sources > 0 )
    {
        print SCRIPT_LOG
            "Multiple test suite files present. Using
$student_tests.\n",
            "Ignoring other .txt/.text/.tst/.test files.\n";
    }
}
close( SCRIPT_LOG );

#=====
# A subroutine for normalizing output lines before comparing them
#=====

sub normalize
{
    my $line = shift;
    $line =~ s/^\s+//o;      # Trim leading space
    $line =~ s/\s+$//o;      # Trim trailing space

```

```

        $line =~ s/\s+/ /go;      # Convert multi-space sequences to one
space
        $line =~ tr/A-Z/a-z/;     # Convert to lower case
        return $line;
    }

#=====
# A subroutine for executing a program/collecting the output
#=====

sub run_test
{
    my $pgm      = shift;
    my $name     = shift;
    my $outfile  = shift;
    my $resultfile = shift;

    # Exec program and collect output
    my $job = Win32::Job->new;
    $job->spawn( "cmd.exe",
                "cmd /c $pgm < $temp_input > $outfile"
                );
    if ( ! $job->run( $timeout/2 ) )
    {
        $status = "timeout";
    }
    #   system( "$pgm < $temp_input > $outfile" );

#=====
# Compare the output to test case expectations
#=====

    my $line      = 0;      # next line in @expected_output to match
    my $last_failed = -1;   # index of last failed case
    my $failures  = 0;      # count of failures

```

```

my $errs          = 0;      # Number of runtime errors, which is 0 or 1
since
                        # such an error crashes the program

open( STUDENT, "$outfile" ) ||
    die "Cannot open file for input '$outfile': $!";
open( RESULT, ">$resultfile" ) ||
    die "Cannot open file for output '$resultfile': $!";

print RESULT
    "tddpas.pl v$version: Testing $name using $student_tests\n\n";
while ( <STUDENT> )
{
    if ( $line > $#expected_output )
    {
        # If the expected output has run out, just add up the
remaining
        # lines as errors or crashes.
        while ( defined $_ )
        {
            if ( m/^\runtime error/io )
            {
                $errs++;
                last;
            }
            $failures++;
            $_ = <STUDENT>;
        }
        last;
    }

    # If the line does not match the expected output
    if ( normalize( $_ ) ne $expected_output[$line] )
    {
        my $this_fail = $case_nos[$line];
        print RESULT "F";
        if ( $this_fail != $last_failed )
        {

```

```

        print RESULT "\ncase ", $this_fail + 1,
            " FAILED: $labels[$this_fail]\n";
        $failures++;
        # print "expected: '$expected_output[$line]'\n";
        # print "got:      '", normalize( $_ ), "'\n";
    }
    $last_failed = $this_fail;
}
else
{
    print RESULT ".";
}
$line++;
}

close( STUDENT );

if ( $line <= $#expected_output )
{
    $failures += $#expected_output - $line + 1;
}

my $num_cases = $#labels + 1;
my $succeeded = $num_cases - $failures - $errs;
my $eval_score = ( $num_cases > 0 )
    ? $succeeded/$num_cases
    : 0;
print RESULT
    "\n\nTests Run: $num_cases, Errors: $errs, Failures: $failures
(",
    sprintf( "%.1f", $eval_score*100 ),
    "%)\n";
close( RESULT );
return ( $eval_score, $succeeded, $num_cases );
}

if ( $can_proceed )

```

```

{

#=====
# Phase I: Parse and split the test case input file
#=====

open( SCRIPT_LOG, ">>$script_log" ) ||
    die "cannot open $script_log: $!";
open ( CASES, $student_tests ) ||
    die "Cannot open '$student_tests': $!";

my $scanning_input = 0;
my $case = -1;
while ( <CASES> )
{
    # skip comment lines
    next if ( m,^/(?!--|=),o );

    if ( m,^/|=,o )
    {
        if ( $scanning_input )
        {
            print SCRIPT_LOG
                "$student_tests: ", $INPUT_LINE_NUMBER - 1,
                ": improperly formatted test case.\n";
        }
        my $label = $_;
        chomp $label;
        $label =~ s,^/|=([-=\s]*,,o);
        $label =~ s,([-=\s]*$,,o);
        if ( $label eq "" ) { $label = "(no label)"; }
        push( @labels, $label );
        push( @test_cases, "" );
        $case++;
        $scanning_input = 1;
    }
    elsif ( m,^/--,o )
    {

```

```

if ( ! $scanning_input )
{
    print SCRIPT_LOG
        "$student_tests: ", $INPUT_LINE_NUMBER,
        ": improperly formatted test case; cannot proceed.\n";
}
$scanning_input = 0;
}
else
{
    if ( $scanning_input )
    {
        # Then this is an input line
        if ( $#test_cases < 0 )
        {
            print SCRIPT_LOG
                "$student_tests: ", $INPUT_LINE_NUMBER,
                ": improperly formatted test case.\n";
        }
        else
        {
            $test_cases[$#test_cases] .= $_;
        }
    }
    else
    {
        if ( $#labels < 0 )
        {
            print SCRIPT_LOG
                "$student_tests: ", $INPUT_LINE_NUMBER,
                ": improperly formatted test case.\n";
        }
        push( @expected_output, normalize( $_ ) );
        push( @case_nos, $case );
    }
}
}
close( SCRIPT_LOG );
}

```

```

close( CASES );

# Produce stdin input file for programs to use:
open( INFILE, "> $temp_input" ) ||
    die "Cannot open '$temp_input': $!";
print INFILE for @test_cases;
close( INFILE );

#=====
# Phases II and III: Execute the program and produce results
#=====

@student_eval = run_test( $student_exec,
    "your submission",
    $student_output,
    $student_rpt );
@instr_eval    = run_test( $instr_exec,
    "reference implementation",
    $instr_output,
    $instr_rpt );

open( COVERAGE, "coverage.txt" ) ||
    die "Cannot open 'coverage.txt': $!";
$coverage = <COVERAGE>;
chomp $coverage;
close( COVERAGE );

open( ASSESSMENT, ">$assessment" ) ||
    die "Cannot open 'coverage.txt': $!";
print ASSESSMENT
    join( " ", @student_eval ), "\n",
    join( " ", @instr_eval ), "\n",
    "$coverage\n";
close( ASSESSMENT );

}

#=====

```



```

# Generate XML Output Documenting Result
open( XML_RESULT, ">$XML_result" ) ||
    die "Cannot open '$XML_result': $!";
print XML_RESULT "<?xml version=\"1.0\" ?>\n<result
status=\"\$status\">\n";
if ( stat( $script_log )->size != 0 )
{
    print XML_RESULT "    <report to=\"student\" format=\"text\" ",
        "name=\"\$script_log\" />\n";
}
if ( $scan_proceed )
{
    print XML_RESULT<<EOF;
    <studentoutput>
        <file name=\"$student_output\" />
        <file name=\"$instr_output\" />
    </studentoutput>
EOF
}
else
{
    print XML_RESULT "    <score raw=\"0\" />\n";
}

close( XML_RESULT );

#-----
exit( 0 );

```

B.3 Script used for phase III of assignment processing

```
#!/c:\perl\bin\perl.exe -w
#=====
#   @(#) $Id$
#-----
#   Web-CAT Curator: grading script for all submissions
#
#   usage:
#       grading.pl <pid> <working-dir> <script-home-dir> <logDir> \
#                               <resultXMLfilename> <timeout>
#=====
use strict;
use File::stat;

#=====
# Bring command line args into local variables for easy reference
#=====

my $pid          = $ARGV[0];   # pid of student
my $working_dir  = $ARGV[1];   # working dir for grading
my $script_home  = $ARGV[2];   # directory where script is located
my $log_dir      = $ARGV[3];   # directory where logs should be generated
my $XML_result   = $ARGV[4];   # output xml file name
my $timeout      = $ARGV[5];   # timeout to use for script

#-----
# In addition, some local definitions within this script
#-----

my $report        = "$log_dir/grading.html";
my $assessment    = "$log_dir/TDD-assessment.txt";
my $status        = "success";

#=====
```

```

# Script Startup
#=====

# Change to specified working directory and set up log directory
chdir( $working_dir );
if( ! -e $log_dir )
{
    mkdir( $log_dir );
}

#=====
# Print out the HTML report fragment
#=====

open( ASSESSMENT, $assessment ) ||
    die "Cannot open '$assessment': $!";
$_ = <ASSESSMENT>; chomp;
my ( $student_percent,
    $student_passed,
    $student_total ) = split( /\s+/ );
my $student_percent_int = int( $student_percent * 100 + 0.5 );
$_ = <ASSESSMENT>; chomp;
my ( $instr_percent,
    $instr_passed,
    $instr_total ) = split( /\s+/ );
my $instr_percent_int = int( $instr_percent * 100 + 0.5 );
my $coverage = <ASSESSMENT>; chomp $coverage;
my $coverage_int = int( $coverage * 100 + 0.5 );
my $final_score =
    int( $student_percent * $instr_percent * $coverage * 50 + 0.5 );
close( ASSESSMENT );
my $score_equation = sprintf( "(%d%% x %d%% x %d%%) x 50 = %d",
    $student_percent_int,
    $instr_percent_int,
    $coverage_int,
    $final_score );

```

```

open( HTML, ">$report" ) ||
    die "Cannot open '$report': $!";

print HTML<<EOF;
<table border="0" cellpadding="8" cellspacing="0">
<tr><td></td><th>Correctness Based on Your Tests</th><td></td></tr>
<tr><td valign="top" align="right" width="25%"><b>Your Program</b></td>
<td valign="top" width="50%" align="center">
    <table border="0" cellpadding="0" cellspacing="0" width="100%">
    <tr>
    <td bgcolor="#33ff00" width="$student_percent_int%"
align="center">$student_percent_int%</td>
EOF
if ( $student_percent < 1 )
{
    print HTML "    <td bgcolor=\"\#cc0000\">&nbsp;</td>";
}
print HTML<<EOF;
</tr></table>
</td>
<td valign="top" width="25%"><b>$student_passed of $student_total</b>
tests passed</td>
</tr>
<tr><td>&nbsp;</td><td></td><td></td></tr>
<tr><td></td><th>Thoroughness of Your Testing</th><td></td></tr>
<tr><td valign="top" align="right" width="25%"><b>Your Test
Cases</b></td>
<td valign=top width="50%" align="center">
    <table border="0" cellpadding="0" cellspacing="0" width="100%">
    <tr>
EOF
if ( $instr_percent < 1 )
{
    print HTML "    <td bgcolor=\"\#cc0000\" ";
}
else
{
    print HTML "    <td bgcolor=\"\#33ff00\" ";

```

```

}
print HTML
    "width=\"${coverage_int%}\" align=\"center\">${coverage_int%}</td>\n";
if ( $coverage < 1 )
{
    print HTML "    <td bgcolor=\"#eeeeee\">&nbsp;</td>";
}
print HTML<<EOF;
</tr></table>
</td>
<td valign="top" width="25%"><b>${coverage_int%}</b>
coverage,<br><b>${instr_passed} of ${instr_total}</b> tests valid</td>
</tr>
<tr><td colspan="3" align="center">Score = ${score_equation}</td></tr>
</table>
EOF

```

```
close(HTML);
```

```

#=====
# Print out the final XML result file
#=====

```

```

open( XML_RESULT, ">${XML_result}" ) ||
    die "Cannot open '${XML_result}': $!";

```

```

print XML_RESULT<<EOF;
<?xml version="1.0" ?>
<result status="${status}">
    <report to="student" name="${report}" inline="true" />
    <report to="student" name="${log_dir}/student-tdd-report.txt"
inline="true" />
    <report to="student" name="${log_dir}/reference-tdd-report.txt"
inline="true" />
    <report to="student" name="${log_dir}/output.txt" inline="false"
        label="Download your output" />
    <report to="student" name="${log_dir}/reference-out.txt" inline="false"

```

```
        label="Download expected output" />
    <score raw="$final_score" />
</result>
EOF
```

```
close( XML_RESULT );
```

```
#=====
exit( 0 );
#=====
```

Appendix C

C.1 Assignment description

Program Assignment 1

100 Points

Due: 2/14 at the start of class

Problem

At some point in your schooling, you may have encountered the concept of *diagramming sentences*. A diagram of a sentence is a tree-like drawing representing the grammatical structure of the sentence, including parts such as the subject, verb, and object.

Here is an extended BNF grammar for some simple English sentences that we will use for the purposes of diagramming.

```
<sentence>    --> <subject> <verb_phrase> <object>
<subject>     --> <noun_phrase>
<verb_phrase> --> <verb> | <verb> <adv>
<object>      --> <noun_phrase>
<verb>        --> lifted | saw | found
<adv>         --> quickly | carefully | brilliantly
<noun_phrase> --> [<adj_phrase>] <noun> [<prep_phrase>]
<noun>        --> cow | alice | book
<adj_phrase>  --> <adj> | <adj> <adj_phrase>
<adj>         --> green | lean | mean
<prep_phrase> --> <prep> <noun_phrase>
<prep>        --> of | at | with
```

This grammar can generate an infinite number of sentences. One sample is:

```
mean cow saw carefully green alice with book
```

(For simplicity, we ignore articles, punctuation, and capitalization, including Alice's name or the first word of the sentence.)

Implementation Language

For this assignment, your program must be written in **Pascal**. Your solution may only use standard Pascal code, not extensions or special library routines provided with your compiler. If your Pascal compiler includes object-oriented features, you may not use them for this assignment. Your program must adhere to proper use of the Pascal language as well as good programming style, including embedded routines, scoping, parameter passing, identifier naming, routine length, etc.

This assignment will also give you more experience using BNF. Decide what the basic operations are that you need to implement, how those will be written as functions or procedures, and then how they can be used to build higher level functions/procedures.

An easy way to organize your solution is to construct one (parameterless) procedure for each nonterminal in the grammar. See Chapter 4 in Sebesta (in 4th ed., pp. 123-125 of Chapter 3) for details on recursive descent parsing. For example, the following procedure parses the non-terminal *<adj>*:

```
procedure adj;
  { The global variable "next_token" holds the next
    token in the input stream. See Sebesta for details }
begin
  write ('(');
  if ((next_token = 'green') or
      (next_token = 'lean' ) or
      (next_token = 'mean' )) then
    begin
      lexical;      { A function you write to advance to
                    the next token }

      write (')')
    end
  else
    error ('Input is not a sentence'); { A function you write
                                       to handle errors }
  end;
end;
```

Input Format

Your program should read "candidate sentences", one per line, from its standard input. For each line of input, your program should attempt to interpret the line's contents (a whitespace-separated list of words) as a sentence. You will need to read each entire line

in as a string. This string will have to be split into tokens by a lexical analyzer, which will then feed them to your parsing routine.

Each input line is guaranteed to fit into a `String` variable in FreePascal, and each input line is guaranteed to end in a newline. If you are developing your solution under Unix, be sure that your code works correctly whether the standard input stream uses Unix (a line feed) or Windows (a carriage return/line feed pair) line ending conventions. The best way to address this problem is simply to use the `readln()` operation to read input a line at a time. Alternatively, if you choose to read input one character at a time, you can use the `eol` function to detect line endings and then use `readln()` to advance over the line terminator, all in an OS-independent manner.

Output Format

The output of your program should be produced on standard output (in the Pascal sense of the term). Your program should produce a "diagrammed" version of the input string, which means a sentence in properly parenthesized form. "Properly parenthesized" means that each nonterminal appearing in the input string now has parentheses around it (omitting all redundant parentheses). For instance, the input string:

```
"alice found mean green book"
```

would be parenthesized as

```
((("alice")) ("found")) ((("mean" "green")) "book"))
```

A more complicated example is

```
"mean cow saw carefully green alice with book"
```

which returns

```
((("mean") "cow") ("saw" "carefully")) ((("green") "alice" ("with" ("book"))))
```

In addition, there are two distinct error conditions that your program must recognize. First, if a given string does not consist of valid tokens, then respond with this message:

```
"Input has invalid tokens."
```

Second, if the parameter is a string consisting of valid tokens but it is not a legitimate sentence according to the grammar, then respond with the message:

```
"Input is not a sentence."
```

Note that the "invalid tokens" message **takes priority** over the second message; the "not a sentence" message can only be issued if the input string consists entirely of valid tokens.

Test Driven Development

In addition to writing your program, you will also need to write a set of test cases that demonstrate that your program works correctly. The Curator-assessed portion of your program grade will depend on all three of these factors:

- The correctness of your test cases (each test case must have the right expected output)
- The thoroughness of your test cases (together, they must exercise all behaviors required by this assignment)
- The correctness of your program (it must pass all your test cases)

Review the [test case input guidelines](#) to see the format for your test cases. They should all be combined into a single ASCII file that will be submitted to the Curator along with your program. In addition, you can use the [tddpas.pl](#) tool to run test cases on your own machine.

Pascal Implementation Hints

Here are some random bits of Pascal information that you may or may not find useful as you implement this assignment:

- Free Pascal supports a string type called `String`. By default, this type supports strings up to 255 characters in length. Strings are stored as character arrays, with a "hidden" byte at the beginning to record the current length of the string. Unlike C, C++, or Java, in Pascal, the first character in a string is at **index position 1**. See [Section 2.13.2.4 of the FP Reference Guide](#) for details on the basic functions supported for strings.
- String comparisons are performed using `"="`:

- `s : String;`
- `...`
- `if s = "while" then`
- `...`
- You can use `readln(s);` to read all characters on the current input line into a string variable `s`, and then advance over the newline to the start of the next line. Each OSIL input line will be no longer than 255 characters.
- You can use `writeln(x);` to print out any value of any scalar type.
- You can use `ord(c)` to convert any character `c` into its corresponding ASCII code as an integer.
- You can use `chr(x)` to convert an integer value `x` into a character with the corresponding ASCII code.
- You can use `val(s, x, err)` to parse a string `s` into its corresponding integer value, which will be placed in `x`. The integer `err` will be set to the result code of the operation (0 for success, or the position where parsing failed if there was an error).
- You can use `copy(s, start, len)` to extract a substring from a string, or use `delete(s, start, len)` to remove characters from a string.
- You can use a **forward declaration** to introduce a subroutine's name and parameter profile (so it can be called by others) before you give its implementation. This is necessary for mutually recursive or mutually dependent operations.

Submitting Your Program

Your Pascal implementation is to be organized in a single file. All documentation and identifying information should be placed in comments within the single source file. The comments at the beginning of the file should identify the assignment and give your full name. Every Pascal procedure/function should be preceded by comments explaining its

purpose, the meaning of each parameter, and the general strategy of its implementation if applicable.

In addition to your Pascal source file, you will also need to submit a plain ASCII file containing your test cases. Your program and test case file will be both be submitted electronically through the Web-CAT Curator for automatic grading. See the program guidelines for information on submitting. The Web-CAT Curator will grade one half of your program score (50 points), as well as assess any deductions based on the late policy described in the syllabus. The TA will assess the remaining 50 points based on the program grading criteria. There is no need to print or turn in a hard copy version of your program; the TA will mark up a PDF version of your program and e-mail it back to you.

References

1. Allen, E., Cartwright, R., Stoler, B. DrJava: A Lightweight Pedagogic Environment for Java. In *Proc. 33rd SIGCSE Technical Symp. Computer Science Education*, ACM, 2002, pp 137-141.
2. Allen, E., Cartwright, R., and Reis, C. Production programming in the classroom. In *Proc. 34th SIGCSE Technical Symposium. Computer Science Education*, ACM, 2003, pp. 89-93.
3. Banner system at Virginia Tech. Web page last accessed May 18, 2003: <http://www.computing.vt.edu/administrative_systems/banner/index.html>
4. Beck, K. Aim, fire (test-first coding). *IEEE Software*, 18(5): 87-89, Sept./Oct. 2001.
5. Curator: an Electronic Submission Management Environment. Web page last accessed May 18, 2003: <<http://ei.cs.vt.edu/~eags/Curator.html>>
6. Decker, R. and Hirshfield, S. Top 10 reasons why object-oriented programming can't be taught in CS1. In *Proc. 25th Annual SIGCSE Symposium on Computer Science Education*, ACM, 1994, pp 51-55.
7. Goldwasser, M.H. A gimmick to integrate software testing throughout the curriculum. . In *Proc. 33rd SIGCSE Technical Symposium of Computer Science Education*, ACM, 2002, pp. 271-275.
8. Hetzel, Bill. Complete Guide to Software Testing, 2nd Ed., New York: John Wiley and Son, 1983
9. Hilburn, T. Software engineering – from the beginning. In *Proc. Ninth SEI conference on Software Engineering Education (Daytona Beach, FL, April 1996)*, 1996, pp 29-39
10. Hilburn, T and Towhidnejah, M. Software Quality: a curriculum postscript? In *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, 2000, pp 167-171
11. Isong, J. Developing an automated program checker. *Computing in Small Colleges*, 16(3): pp 218-224.

12. Jackson, D., and Usher, M. Grading student programs using ASSYST. In *Proc. 28th SIGCSE Technical Symposium Computer Science Education*, ACM, 1997, pp. 335-339.
13. Jeffries, R. <http://www.xprogramming.com/xpmag/whatisxp.htm>, last accessed on April 10, 2003.
14. Jones, E.L. Software testing in the computer science curriculum-a holistic approach. In *Proc. Australasian Computing Education Conf.*, ACM, 2000, pp. 153-157.
15. Jones, E.L. Integrating testing into the curriculum—arsenic in small doses. In *Proc. 32nd SIGCSE Technical Symposium of Computer Science Education*, ACM, 2001, pp. 337-341.
16. Jones, E.L. An experiential approach to incorporating software testing into the computer science curriculum. In *Proc. 2001 Frontiers in Education Conf. (FiE 2001)*, 2001, pp. F3D7-F3D11.
17. Jones, E.L. Grading student programs – a software testing approach. *J. Computing in Small Colleges*, 16(2): pp. 185-192.
18. Junit Home Page. Web page last accessed April 2, 2003: <<http://junit.org>>.
19. Krause, K.L. Computer science in the Air Force Academy core curriculum. In *Proc. 13th SIGCSE Technical Symposium of Computer Science Education*, ACM, 1982, pp. 144-146.
20. McCauley, R and Jackson, U. Teaching software engineering early-experiences and results. In *Proc. of the 1998 Frontiers in Education Conference*, 1998, pp. 800-804.
21. Myers, G.J. The Art of Software Testing, *John Wiley & Sons*, 1976.
22. Pressman, R. Software Engineering – A Practitioner's Approach, 5th edition, *McGraw Hill*,
23. CS 3304, Comparative languages, assignment description page. Web page last accessed April 22, 2003, <<http://courses.cs.vt.edu/~cs3304/Spring03/programs/p1.php>>

24. Reek, K.A. A software infrastructure to support introductory computer science courses. In *Proc. 27th SIGCSE Technical Symp. Computer Science Education*, ACM, 1996, pp. 125-129.
25. R.C. Wilson. UNIX Test Tools and Benchmarks. *Prentice Hall, Upper Saddle River*, 1995.
26. Ricadela. The state of software: Quality. *InformationWeek*, (838), 2001, pp 43.
27. Sane Aamod, Test Driven Programming. Web page last accessed April 22, 2003: <<http://xp.c2.com/TestDrivenProgramming.html>>.
28. Tassey, G. Software Bugs Take Bite Out of Nation's Economy. *NIST Study*, 2002, pp 23.
29. T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Communications of the ACM*, 44(6), 2001, pp 103-108.
30. The T-Test. Web page last accessed May 22, 2003: <http://trochim.human.cornell.edu/kb/stat_t.htm>

Vita

Anuj Ramesh Shah, the son of Ramesh. B. Shah and Mrs. Surekha Shah, was born on November 4th, 1979, in Mumbai, India. He graduated from Fr. Conceicao Rodrigues College of Engineering in May 2001 with a Bachelor's of Engineering Degree. He came to the United States to study at Virginia Tech in August 2001. This thesis completes his Master's degree in Computer Science from Virginia Tech.