IDE Support for Test-driven Development and Automated Grading in Both Java and C++

Anthony Allowatt and Stephen Edwards
Department of Computer Science
Virginia Polytechnic Institute and State University
660 McBryde Hall
Blacksburg, VA 24061
aallowat@vt.edu, edwards@cs.vt.edu

ABSTRACT
Students need to learn testing skills, and using test-driven development on assignments is one way to help students learn. We use a flexible automated grading system called Web-CAT to assess student assignments, including the validity and completeness of their own test cases. By building on existing educational plug-ins for Eclipse, and adding our own plug-ins for electronic submission and for unit testing support in C++, we are able to use Eclipse as a portal to all the services our students will need, allowing them to accomplish all their tasks entirely within the IDE, from their project's inception to its submission and evaluation. Further, we are able to carry students through the transition from Java programming to C++ programming within this same environment.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education; D.1.5 [**Programming Techniques**]: Object-oriented Programming; D.2.5 [**Software Engineering**]: Testing and Debugging—testing tools; D.3.2 [**Programming Languages**]: Language Classifications—C++.

General Terms
Languages, Verification.

Keywords
Test-driven development, test-first coding, electronic assignment submission, Eclipse IDE, extreme programming, electronic grading.

1. INTRODUCTION
Many educators believe that computer science students need to learn more software testing skills. In fact, writing a test case forces a student to articulate his or her understanding of how the code is expected to behave—a hypothesis they are then going to experimentally verify by executing the test. Encouraging students to reflect on, express, and then verify their understanding will help them learn more effectively.

At the 2003 OOPSLA Educator's Symposium [6], we described an approach to teaching software testing that will encourage students to practice testing skills in many classes and give them concrete feedback on their testing performance, without requiring a new course, any new faculty resources, or a significant number of new lecture hours. Our strategy centers on giving students basic exposure to test-driven development (TDD) [1], encouraging them to use it on all assignments, and then providing an automated service that assesses student submissions on-demand and provides feedback for improvement.

We have been using this approach in a number of our courses for two and a half years. We introduce software testing in the first lab assignment during the first week of classes in CS1, carry it through the first three semesters of required courses, and are now introducing it in some upper division courses. This experience has been positive, resulting in a 28% reduction on average in bugs per thousand lines of code for student work [7].

As our experience has grown, our efforts to provide more value to students through tool support have also increased. This paper describes the automated service we use to assess student code, and the Eclipse-based support we provide for students in both our Java and C++ courses to practice test-driven development and receive feedback. By combining prior educational plug-ins for Eclipse, including Gild [13] and DrJava [11], open-source plug-ins for Checkstyle [3] and PMD [10], and two new plug-ins of our own for electronic submission/feedback and for unit testing in C++, we are able to use a single IDE to transition students from one programming language to the other while seamlessly maintaining support for software testing and rapid feedback. Section 2 briefly describes Web-CAT, our automated assessment system. Section 3 describes our submission plug-in, and Section 4 describes how the pieces are brought together for our Java students. Section 5 describes our C++ unit testing plug-in, and how the same environment is used to help students bridge the two languages. Section 6 concludes with areas for future exploration.

2. WEB-CAT
From the very first programming activities in CS1, we give each student the responsibility of demonstrating how much of the

eclipse'05, October 16-17, 2005, San Diego, CA

required behavior he or she has implemented correctly. We expect and require students to submit test cases for this purpose along with their code, and assessing student performance includes a meaningful assessment of how correctly and thoroughly the tests conform to the problem. We rely on an automated assessment tool to provide rapid, concrete, and immediate feedback.

The assessment tool we use is called Web-CAT, the Web-based Center for Automated Testing [7,8]. Web-CAT is a web application with a plug-in architecture that can provide a variety of services for students. Its Grader plug-in provides a highly configurable and customizable automated grading and assessment service. The actions taken to process student submissions and the nature of the feedback produced are fully customizable without changing or restarting the web application, and can differ from one assignment to the next.

Because of our interest in supporting TDD, we typically configure our assignments on the Web-CAT Grader to emphasize testing. Instead of focusing on the *output* of a student's program, we focus our grading on what is most valuable: the student's testing performance. After all, the student has already provided their own evidence of how much behavior has been implemented correctly in the form of their test cases.

Instead of generating a simple correctness score, we use Web-CAT to assess the validity of the student's tests—how well they conform to the problem requirements—and give feedback about which tests are incorrect. In addition, Web-CAT also assesses the completeness of the student's tests, giving an indication of how to improve. Web-CAT also assesses the style and quality of the student's code, giving feedback about where improvements can be made. Finally, these results from these separate facets are combined into a single score for students.

We implement these assessments on the server end by relying on a number of open-source tools. We use JUnit for expressing and executing tests. We use Clover [4] to instrument Java programs and measure code coverage while executing student tests. We use Checkstyle [3] and PMD [10] for static analysis, both to assess style issues and to spot potential coding problems. Finally, we use ANT to manage the overall process.

3. ELECTRONIC PROJECT SUBMISSION

As a web application, Web-CAT allows students to log in and upload submissions using a web browser. Because we use this approach beginning in our CS1 course, streamlining the process of packaging up a multi-file project, logging in, uploading files, and confirming submissions was important. IDE integration was desirable to eliminate as many mechanical errors from this process as possible. It was also important to minimize the effort required of students, since we are trying to encourage them to seek feedback frequently and respond to the result they receive.

Our freshmen begin CS1 learning Java using BlueJ [2], which has a flexible and powerful submission extension. Appropriate configuration of the submission extension allows students to submit BlueJ projects to Web-CAT using a single menu command in their IDE, with results popping up in their web browser.

After CS1, however, students move to a more powerful IDE. Because we use Java in CS2, our department uses Eclipse. While a number of educationally oriented plug-ins for Eclipse exist [9,11,13], we could not find an electronic submission plug-in that was flexible enough to use in this context. As part of our efforts to reduce both user error and the time spenr outside Eclipse during development, we have developed a plug-in that minimizes the amount of user intervention required to submit a project.

Based on our experiences with other electronic submission systems, we had several goals in designing this plug-in:

1. The plug-in is for general use, and is not Web-CAT-specific.
2. The plug-in should support a wide (and easily extensible) variety of transport methods for submissions, including e-mail, FTP, HTTP, and HTTPS.
3. The plug-in should require minimal setup by students.
4. Submission targets (assignment definitions) should be configurable remotely, without student intervention. Ideally, multiple instructors in separate classes would not have to coordinate to set up their own independent assignments.

3.1 Configuring Submission Targets

Before making a submission, the plug-in must be able to find information about which submission targets are available, the files that a proper submission should contain, and the method by which the files will be delivered. To support goals the third and fourth goals above, there is a single user-settable URL in the plug-in's preferences tab. Instructors give students this URL at the beginning of the semester, and students can then "set and forget."

The plug-in slurps submission target information from this URL any time the student wishes to make a submission. This information is provided in the form of an XML configuration file written by the course instructor, who can post the file on the course website or elsewhere as deemed appropriate. The XML configuration file can indirectly include other XML configuration files by URL as well. This allows a single department to set up one URL for all students, with a department-level configuration file that remotely includes a number of separate per-course configuration files, each of which is under the control of a different instructor in their own web space.

Further, the URL provided to students need not point to an actual XML file. An alternative approach would be to point the submitter plug-in to a script that generates the XML dynamically. This can be extremely desirable because most automated grading systems maintain project information in their own databases; a script that translates this data into the form used by the plug-in would eliminate the duplication of effort by the instructor, reducing the likelihood of errors. This also allows for more advanced organization of the available projects presented to the user. For example, submission targets for which the due date has expired could be hidden or segregated into a "Late Projects" category, and this organization would occur automatically when the plug-in requests the submission targets from the server.

The structure of the XML file allows the instructor to create a tree of assignments and assignment groups. Settings at nodes higher in the tree are inherited by their descendants unless overridden deeper in the tree. An instructor for a Java course, for instance, could specify in the root element that all project submissions include files matching `*.java` and exclude those matching `*.class` unless otherwise specified in a particular target.

A simple XML configuration file would look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<submission-targets xmlns="http://web-
cat.cs.vt.edu/submissionTargets">
 <required pattern="*.java"/>
 <include pattern="*.java"/>
 <exclude pattern="*.class"/>

 <!-- submit by e-mail -->
 <assignment name="Project 1">
  <required pattern="*List.java"/>
  <transport uri="mailto:submissions@cs.vt.edu">
    <param name="subject"
           value="${assignment.name} by ${user}"/>
    <file-param name="submission"
                value="${user}.jar"/>
  </transport>
 </assignment>

 <!-- submit over the web -->
 <assignment name="Project 2">
  <exclude pattern="*.data"/>
  <transport uri="http://web-
cat.cs.vt.edu:9000/cgi-
bin/WebObjects.exe/Main.woa/wa/submit">
    <param name="u" value="${user}"/>
    <param name="p" value="${pw}"/>
    <param name="a" value="Project 2"/>
    <param name="d" value="VTEdAuth"/>
    <file-param name="file1" value="${user}.jar"/>
  </transport>
 </assignment>
</submission-targets>
```

3.2 Submitting Projects from Eclipse

Once the students have configured the plug-in, they can submit a project by using one of the "Submit…" actions available in the IDE, either from the "Project" menu or the Resource Navigator context menu, as shown in Figure 1. This opens a wizard that displays all the submission targets available as read from the remote XML configuration file, as shown in Figure 2. When a student selects a submission target and enters his or her authentication credentials (if required by the remote system), the project files will be packaged up and delivered to the target.

Many automated grading systems return a response when a project is submitted, perhaps as an HTML page displaying the results. Currently, the submitter plug-in supports response handling on a per-protocol basis. If a protocol is designed to handle a response, it returns a string that will be displayed in an embedded browser window in the Eclipse workspace.

3.3 Design Considerations

The primary design goal of the electronic submission plug-in was to create a framework that was robust but generic, and one that fit the same mold of the standard Eclipse plug-ins. While our hope is that most users will find the submitter useful as is, we also wish to provide a way to add functionality easily using separate extensions, rather than requiring deeper modifications of the submitter itself.

The submission wizard and related action sets are contained in a plug-in separate from the core submission engine, so users who find the existing user interface unsuitable for their purposes can remove the existing wizard plug-in and wrap a new interface around the core. The core API itself is simplified for client use; only three method calls are required to instantiate the engine, create a submission target tree from the XML configuration file, and finally to submit a project to one of those targets.

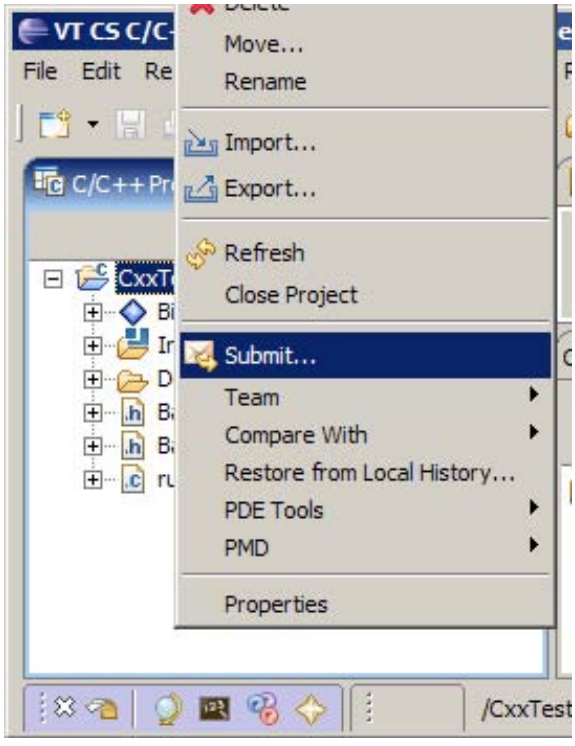

Figure 1: Submit projects directly from the IDE using a menu.

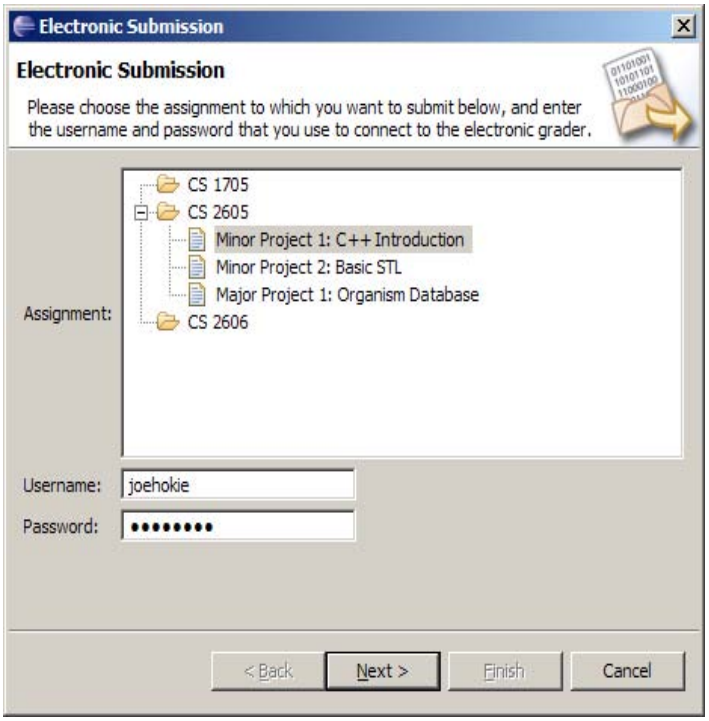

Figure 2: Choosing an assignment within Eclipse.

Likewise, extension points are provided to allow users to write packagers for other archive file formats, and to specify delivery methods for additional URL scheme types, whether they be actual protocols like `http` or "pseudo-protocols" to interoperate with systems that communicate in some proprietary fashion. To use any additional packagers and protocol handlers, the instructor need only reference them by their Eclipse-style fully-qualified IDs in the XML configuration file and ensure that the plug-ins implementing the extensions are installed in their students' copies of Eclipse. The plug-in currently implements extensions for packaging ZIP and JAR archives and delivering them via `http/https` (POST), `ftp`, `mailto`, and `file` protocols.

4. TYING IT TOGETHER FOR JAVA

Our experiences with the submission plug-in and with Web-CAT have been very positive. Eclipse also provides excellent support for JUnit, which is central to our testing approach. To provide the best educational support for our CS2 students using Eclipse, however, we were also interested in integrating in best practices regarding educational use of Eclipse at the introductory level. After examining a number of available Eclipse projects for undergraduate education, we adopted Gild [13] to bring a simplified user interface to our students, including the DrJava [11] plug-in for supporting interactive object manipulation.

In our experiences with students using Web-CAT, however, we noticed a trend. While we encourage students to submit their work often and to respond to the feedback they receive, two patterns emerged. First, students do not submit code that they know does not work—if any of their own test cases fail, they struggle along fixing things on their own *before* submitting work for assessment. In some cases, coding problems (improperly declared constructors, misused variables, name hiding, etc.) that would be directly flagged by Web-CAT's static analysis tools were the sources of errors, but students never received feedback or a push in the right direction because they did not submit their work. Second, we found that students often wrote their code without proper formatting or documentation initially, and once they submitted, the volume of diagnostic messages was occasionally overwhelming.

The root cause in both cases was a significant delay between when students introduced problems in their code and when they received feedback about how to correct those problems. As a result, in addition to using Gild and our submitter plug-in, we also equipped students with the Checkclipse plug-in for Checkstyle, and the PMD plug-in. Both plug-ins were configured with the same checks used on Web-CAT, and set up to run each time a project was compiled. Students now get all of their style and formatting feedback each time they compile, and are better equipped to incrementally fix these problems as they arise.

5. UNIT TESTING IN C++

While Eclipse has worked out well for our Java students in CS2, as students move further into our curriculum they switch languages to C++. Further, our undergraduate program involves the use of Unix-style OSes beginning in the sophomore year, so our students use g++ rather than Microsoft's products. When students begin learning C++ after CS2, we wish to continue the practice of unit testing and maintain the support we have built up for our introductory sequence.

Fortunately, Eclipse's C/C++ Development Tools (CDT) allows us to transition students from Java programming to C++ programming using the same IDE. Further, the portability of Eclipse means that students can develop g++-targeted code on a variety of platforms, even under Microsoft Windows. Our submitter plug-in works for any Eclipse project, including CDT projects. One missing capability, however, is the support that Eclipse provides for JUnit—there is no equivalent support appropriate for student use in C++.

We have chosen to use the CxxTest unit testing framework [5] for our courses, based on its relative simplicity and ease of use. At the core of CxxTest is a Perl (or Python) script that takes as input a user-specified set of test suite header files. The script parses each file, performing regular expression matches to determine if a class derives from the test suite base class and to determine which methods in the class are to be executed as tests. From this, a C++ source file is generated that contains the code to run the tests and report any failures that may occur.

5.1 Integration with Eclipse

To facilitate the transition from Java to C++, we aimed to make the CxxTest experience in Eclipse as similar as possible to that of JUnit. The processing performed by CxxTest should occur seamlessly from the viewpoint of the user, and the results should be displayed in a view similar to that used by JUnit.

The initial versions of our CxxTest plug-in divided the build process into two parts. First, an incremental builder collected the test suites in the project and passed them to the CxxTest Perl script. The generated C++ source file was then built by the Managed Make process with the rest of the project. A second incremental builder then executed the tests automatically after the build and displayed the results to the student. By default, CxxTest prints its results to standard output. The test runner parsed this output and places markers in the source files to indicate which tests, if any, failed.

This approach, while successful, came with some drawbacks. First, it required students to have Perl properly installed on their systems. While this was not a serious issue, it did introduce an additional point at which students could experience problems. Second, we only wished to pass files that contained test suites to the CxxTest script. Several schemes were considered to address this, the most feasible of which were either requiring a specific file naming convention for test suites or isolating all test suites in a predetermined subdirectory of the project. The latter choice was deemed less intrusive.

5.2 Interfacing with CDT

The latest version of the CxxTest plug-in was designed to eliminate the Perl dependency and more tightly integrate the C++ testing framework into the IDE. The incremental builders were rewritten in pure Java to mimic the behavior of the standard CxxTest Perl script.

Rather than requiring test suites to be contained in a specific subdirectory, the builder navigates the project DOM exposed by the CDT to determine which classes represent test suites, regardless of their locations within the project. Nodes in the tree are visited recursively until a compilation unit (representing a header or source file) is found. For each class, if any, in a

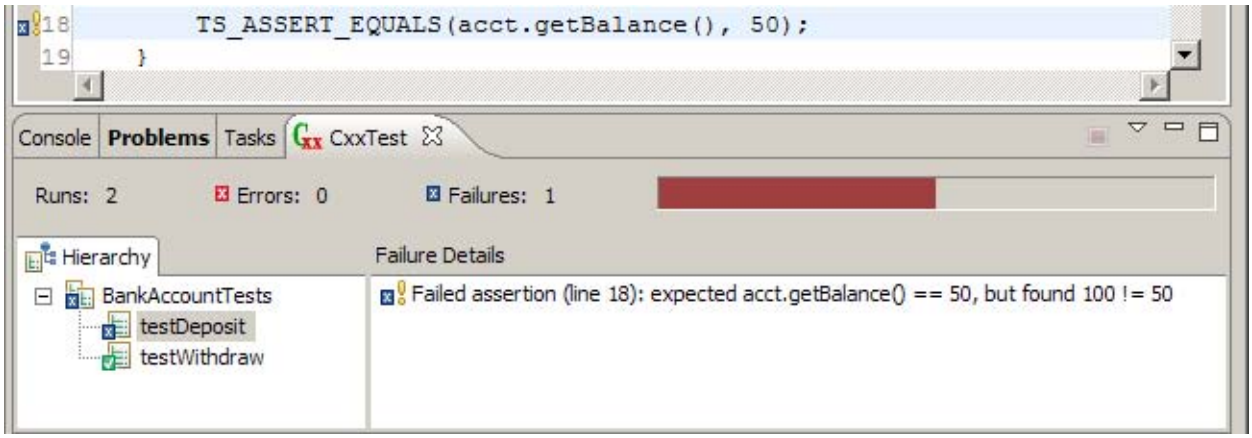

Figure 3: The dedicated CxxTest view mimics the one for JUnit.

compilation unit, the base class list is searched for an instance of the CxxTest test suite class. If found, we descend further in the DOM to collect any methods that match the signature of a test method: void return value, no parameters, and a name that begins with "test". With an internal tree of test classes and methods constructed, the builder generates a source file to execute the tests in the same manner as the original CxxTest script.

The main CxxTest libraries were supplemented to add a result printer that used an XML-based model that could easily be parsed by the second incremental builder that runs the tests. When the tests are executed, the results are displayed in a dedicated view designed to resemble the JUnit results view, as shown in Figure 3. The end result is simple and effective support for test-driven development in C++, where unit tests are simple for students to write, tests are re-run every time a project is compiled, and results are as easy to see and interpret as with JUnit in Eclipse.

6. FUTURE WORK

A major extension that is planned for the electronic submission framework would allow instructors to create specialized handlers for responses returned by grading systems. Currently, only HTML-based responses are supported. The new framework would create a new extension point that users could implement. Response handlers would be specified by ID in the XML target configuration file. Responses from the remote system would then be passed to a custom handler that could interact with the Eclipse IDE as it sees fit. Possible uses for this would be to annotate students' source code based on evaluations made by the grading system or to display students' scores and other evaluation information in a custom view.

Enhancements planned for the CxxTest plug-in include providing a means for students to specify which tests should or should not be executed, perhaps via a launch extension. Implementing a launch extension would also help to decouple the test builder from the test runner, for those instances when it may not be desirable to automatically execute the tests after every build.

Our plug-ins are open-source, available from our Sourceforge site at http://web-cat.sourceforge.net/.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Beck, K. *Test-Driven Development: By Example*. Addison-Wesley, Boston, MA. 2003.

[2] BlueJ home page. http://www.bluej.org/

[3] Checkstyle home page. http://checkstyle.sourceforge.net/

[4] Clover home page. http://www.thecortex.net/clover/

[5] CxxTest home page. http://cxxtest.sourceforge.net/

[6] Edwards, S.H. Rethinking computer science education from a test-first perspective. In *Addendum to the 2003 Proc. Conf. Object-oriented Programming, Systems, Languages, and Applications*, ACM, 2003, pp. 148-155.

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

[8] Edwards, S.H. Using software testing to move students from trial-and-error to reflection-in-action. In *Proc. 35th SIGCSE Tech. Symp. Computer Science Education*, ACM, 2004, pp. 26-30.

[9] Mueller, F. and Hosking, A.L. Penumbra: An Eclipse plugin for introductory programming. In *Proc. 2003 OOPSLA Workshop on Eclipse Technology eXchange*, ACM, 2003, pp. 65-68.

[10] PMD home page. http://pmd.sourceforge.net/

[11] Reis, C. and Cartwright, R. Taming a professional IDE for the classroom. In *Proc. 35th SIGCSE Tech. Symp. Computer Science Education*, ACM, 2004, pp. 156-160.

[12] Spacco, J., Hovemeyer, D., and Pugh, W. An Eclipse-based course project snapshot and submission system. In *Proc. 2004 OOPSLA Workshop on Eclipse Technology eXchange*, ACM, 2004, pp. 52-56.

[13] Storey, M.-A., Damian, D., Michaud, J., Myers, D., Mindel, M., German , D., Sanseverino, M., and Hargreaves, E. Improving the usability of Eclipse for novice programmers. In *Proc. 2003 OOPSLA Workshop on Eclipse Technology eXchange*, ACM, 2003, pp. 35-39.