

The Quiver System

Christopher C. Ellsworth, James B. Fenwick Jr., and Barry L. Kurtz

Appalachian State University

Boone, NC 28608

blk@cs.appstate.edu

ABSTRACT

The Quiver (QUIZ VERification) System is an Internet server for building, maintaining, and administering programming quizzes. It is similar to the online judges used for programming contests but differs in that it targets the classroom use of programming quizzes as a teaching aid and evaluation tool. It can provide very detailed feedback regarding quiz behavior so that the student can debug her program. This system is developed as part of the grant “Intra-Curriculum Software Engineering Education” funded by the National Science Foundation (DUE 0127439).

Categories and Subject Descriptors

Lab Environments, Pedagogy, Web-based Technologies

General Terms: Verification

Keywords: Automated Testing, Quizzing Systems

1. A Brief History of Quizzing Systems

The testing of individual skills in a lab environment is not a new idea. Ken Bowles [1] at the University of California San Diego (UCSD) introduced the use of drills and programming quizzes to support instruction in UCSD Pascal in the CS1 course. Kurtz has used this approach at two other major universities: the University of California Irvine and New Mexico State University [2]. At NMSU, Kurtz adapted the programming quizzes to teach abstract data types (ADT) in a data structures course [3]. Some CS1 or CS2 books, such as [4], provide supplemental software that will automatically test methods students have written for correct operation.

Automated testing systems are essential for programming contests. The programming contest environment and example problems are described in detail in the book “Programming Challenges” [5]. Unlike the systems used in programming contests that provide minimal feedback to students, the Quiver system provides case-by-case analysis giving the input, the expected output, and the actual output.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE '04, March 3–7, 2004, Norfolk, Virginia, USA.

Copyright 2004 ACM 1-58113-798-2/04/0003...\$5.00.

When teaching software engineering, it is expected that students specify input/output behavior of all methods before actually implementing these methods (in the spirit of Extreme Programming). Both Java and C++ provide unit testing, JUnit or CppUnit [6, 7], to insure that a complex software system continues to work correctly as it evolves in functionality. In our intra-curriculum approach to software engineering, the software engineering students specify the expected operation for classes using descriptive text and UML but leave implementation of these classes for students in other courses. Rather than use JUnit or CppUnit directly, the software engineering students specify the requirements as a set of instructions and test cases in the Quiver system. By using Quiver, students in the other courses are able to receive immediate feedback on their software implementation for a variety of test cases and able to debug their software before submitting it to students in the software engineering course.

The Quiver System differs from other systems in several significant ways:

- Our system works with both C++ and Java taking advantage of code reuse so that a quiz only has to be specified once and will work in both languages.
- The system also works with MIPS Assembly Language (MAL) using standard input/output.
- We supply the student with as much information as possible to help the student debug her program.
- We can test both input/output behavior using standard input and standard output and the correct operation of individual methods by testing for a return value for a specific set of parameter values.
- Our system provides elaborate administrative tools for an instructional environment.
- The entire system is run from a server environment and does not require local installation on the client machine.
- Quiz verification can be performed in two different ways: supply the correct answer in each test case (similar to JUnit) or supply a working method to generate the correct answer.

2. The Quiver System

The QUIVER system consists of a single web application developed using Java Servlets and using XML/XSLT. We decided to use the Apache Cocoon XML publishing

framework [8], which is designed around a very flexible pipelined SAX processing system.

2.1 Purpose of the System

The main goal of this system is to provide instructors with an automated tool to evaluate a student’s programming abilities. Unfortunately, some students are graduating with computer science degrees that do not have the fundamental programming skills needed in the real world to develop commercial-grade software. Students often believe that if their methods work for “normal” cases, the problem is solved. When they do discover cases that do not work properly, students often start making “random” changes, such as changing “count <= limit” to “count < limit”, in a desperate hope the desired results are attained. After several such attempts, the code is often beyond fixing.

By using our automated system an instructor can specify a wide variety of “limiting or abnormal cases” to train students to produce “bullet proof” software. By putting a student in a closed lab environment and observing the student solving different types of problems, we are able to better understand and address the individual needs of each student.

2.2 Built-in Features to Support Instruction

The system provides the ability to add students, release quizzes, create quizzes using a GUI Quiz-Builder, and verify quiz submissions.

The Quiz Server is very flexible in the type of quizzes that may be built. This allows the instructor to evaluate students whose programming skills range from the CS1 level through the graduate level.

Here are some examples of the types of problems that we have already tested in the quizzing environment:

- Single functions that return a unique value, such as an iterative Fibonacci method.
- Missing methods of a given abstract data type, such as multiplication and display methods for a complex number class.
- Using features in Java, including collections, event handling, GUI layouts, threads, and sockets.

Although Quiver works with MIPS Assembly Language, we have not yet tested this feature with students.

3. Using Quiver

3.1 Use as an Administrator

Students can be added to the system either individually or from a class roster. Implementation of upload from a class roster is dependent on the class roster format of the academic computing system used by a university and can be easily adapted for each particular installation.

Figure 1 shows a portion of the screen for quiz release. One or more quizzes can be selected from the “enable

quiz” pull down menu and placed in an enabled status. Similarly quizzes can be transferred from an enabled to a disabled status using the right hand menu. The instructor can also specify an automatic timeout, such as one hour, where the quiz will automatically become disabled.

Release Quiz By User

User Name	ID	Group	Enable a Quiz	Disable a Quiz
Christopher Ellsworth	coe	admin	<input type="checkbox"/> BoxQuiz	<input type="checkbox"/> FactorialQuiz
Barry Kurtz	blk	admin	<input type="checkbox"/> BoxQuiz	<input type="checkbox"/> TaylorSeriesQuiz
Jay Fenwick	jbf	admin	<input type="checkbox"/> BoxQuiz	

Figure 1: Quiz Release Subsystem

The Quiz-Builder generates a Java class for each quiz. A single quiz can verify both C++ and Java solutions for a given problem without having duplicate quizzes for each language supported. This is because the quiz subsystem is implemented using a blend of C++ and Java, providing a simple but robust interface for communicating between languages in the multi-threaded runtime environment.

When the quiz verification process begins, the submitted code is instantiated and passed sequentially to each test case, so a data type’s state persists across test cases. This allows for the testing of a data type whose state is dependent on the events that took place in the previous test cases. Optionally, the submitted code may be instantiated at the beginning of any test case to remove the dependency on previous cases. Each test case is evaluated individually as either succeeding or failing.

3.2 Designing a New Quiz

Each quiz consists of a description, source code files to be included, and test cases. The Quiz-Builder provides a way to import existing C++ or Java source code from the local file system. This code may be used to supply correct answers as a verification tool or to provide source to be modified by the student as part of a quiz. Multiple test cases may be added for each quiz, complete with support for exception and assertion handling in both languages. For example, a test case may fail if an exception is not thrown at the appropriate time for a particular set of data values.

Figure 2 shows the files associated with the Distinct Elements quiz discussed in detail in Section 4. There are three source code files shown:

- The source file “distinct_elements_good.C” is the class correctly implemented which is used for verification.
- The source file “distinct_elements.C” is the entire class but prototypes have been substituted for the methods the student is supposed to implement.
- The file “QuizProxy.C” provides communication between the Java test cases and the native C++ implementation. This is automatically generated based on the methods specified to be verified.

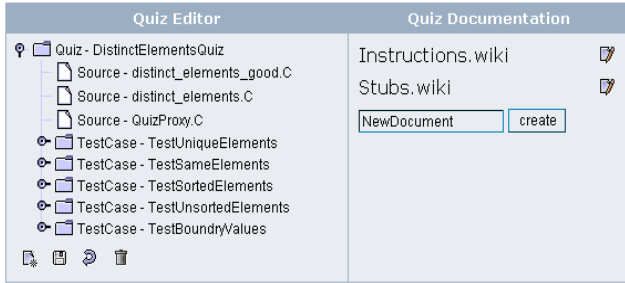


Figure 2: Building a Quiz

The test cases themselves are specified in Java using a simple API for testing each case on a pass/fail basis and communicating these results to the student. Figure 3 shows the test case for the fibonacci quiz that verifies all values from 0 through 20. In this example, verification is based on comparing the value returned from the student's method with the value returned by a correctly implemented method. Each case is reported individually.

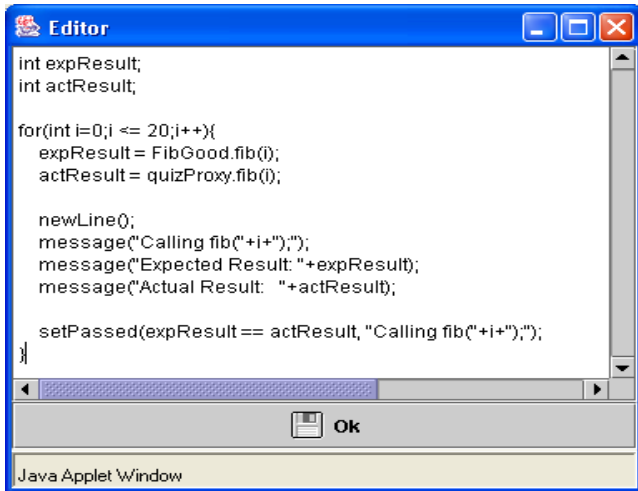


Figure 3: Specification of Test Cases

3.3 Use as a Student

When a student selects a quiz, she should first read the instructions that consist of an overview, a detailed description, and sample test cases. The instructor may supply stubs for the methods to be implemented. The student then cycles through a sequence of three operations: edit the source code, compile the program, and submit the solution for verification. Figure 4 shows a snapshot of the source code editor; this editor runs on the client using Java Web Start (JWS). This was one student's solution for the expansion of a Taylor series that generates e^x for a given value of x and n , a specified number of terms.

Compilation errors are reported back to the student in textual form. Once the compilation is successful, the student may submit her program to the automated testing system. This process continues until the student either passes all tests or the time limit for the quiz has expired.

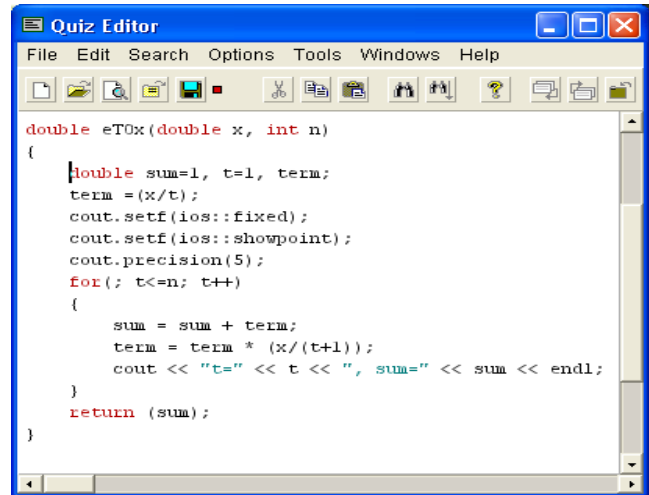


Figure 4: Source Code Editor

4. Classroom Use of Quiver

We used our automated testing system with students in three different courses. In our CS1 class using C++ we administered eight quizzes in a closed lab environment, as listed in Figure 5.

Lab Name	Topic
Fibonacci (iterative)	While and If statements
Combinations*	Midterm 1, while and if
Distance class	Classes
Print roman numerals	Switch statement
Taylor series expansion	For statement
Complex numbers*	Midterm 2, classes
Very long integer	Arrays
Distinct elements*	Final, arrays

Figure 5: CS1 Programming Quizzes

Three of the quizzes (*) were part of exams. Students were given an hour fifteen minutes for the written part of the exam (closed book) and an hour fifteen minutes for the program quiz (open book and notes). We now discuss the distinct elements quiz associated with the final exam.

The class *my_array* contains two private attributes, a one dimensional integer array called *data* and the integer *current_size* that specifies how many elements, starting from index 0, are to be processed by the methods.

The first method the students implemented is: *bool isSorted()* that returns *true* if the array *data[0]* through *data[current_size-1]* is sorted and *false* otherwise. The second method is *int countUniqueElements()* that returns the number of unique elements in the array provided it is sorted or returns 0 if it is not sorted. Figure 6 shows how these two methods should function.

```

here is the full array test1
1 1 1 2 2 2 3 3 4 4 4 7 7 9 9 9 9 9 9 9
array test1 with 20 elements has 6 unique elements
array test1 with 7 elements has 3 unique elements

```

```

here is the full array test2
1 1 1 2 2 2 3 3 4 4 4 3 3 3 3 2 2 2 1 1
array test2 with 20 elements has 0 unique elements
array test2 with 6 elements has 2 unique elements

```

Figure 6: Sample Functionality

A correct solution is shown in Figure 7. This proved to be a challenging problem for introductory students to get completely correct. Commonly failed tests were an array with one element or an array where every element had the same value.

```

bool my_array::isSorted()
{
    for(int count = 0; count < current_size-1; count++)
        if (data[count] > data[count+1]) return false;
    return true;
}

int my_array::countUniqueElements()
{
    int count, unique = 1;
    if (isSorted())
        for(count = 0; count < current_size-1; count++)
            { if (data[count] != data[count+1]) unique++;}
    else unique = 0;
    return unique;
}

```

Figure 7 : A Correct Solution

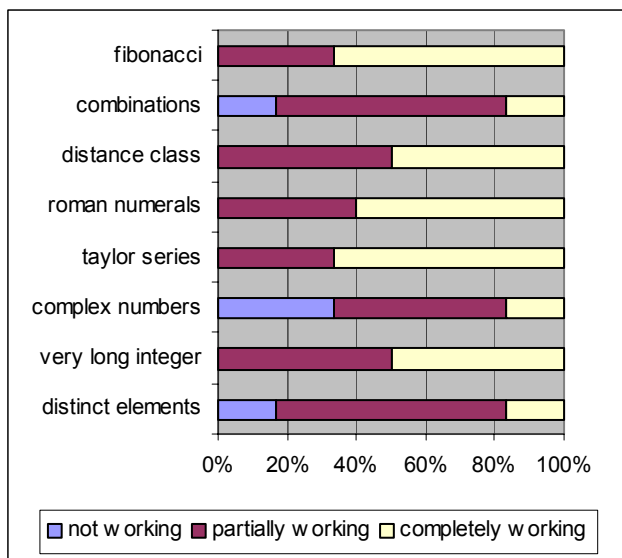


Figure 8: CS1 Quiz Performance

We found the anxiety of not getting a completely correct solution needs to be kept within reason by assigning partial

credit. It appears that the adage about “the thrill of victory and the agony of defeat” applies to students trying to take programming quizzes in a closed lab environment with a fixed amount of time.

The performance on all programming quizzes is shown in Figure 8 where solutions were classified as: not working, partially working, and completely working.

Quiver was also used in our graphical user interface (GUI) course, a junior level course where students experienced in C++ are introduced to Java for the first time. The six quizzes stressed features that tend to be unique to Java, as shown in Figure 9.

Lab Name	Topic
Distinct Strings	String Objects
Word Line Count	Collections
Binary Adder	Event Handling
Batmobile GUI	Layouts and Objects
Producer-Consumer	Threads
Instant Messages	Client-Server Programming

Figure 9: Java Programming Quizzes

The Distinct Strings quiz is similar to the distinct elements quiz already described for C++, except the objects were of type String instead of int. Not surprising, performance was better than our CS1 students. In “Word Line Count” students had to count the number of distinct lines on which every unique word appeared. A successful solution involved using a HashMap to map each word to its count and a HashSet for each line of text so that a word is only counted once per line.

In the Binary Adder quiz students were given a GUI with six buttons, 0, 1, +, =, C, CA, along with a TextField for display. Student had to write the event handlers for each of these buttons based on an English language description of the tasks to be performed. The Batmobile GUI reversed this situation, students were given the event handlers and had to build a GUI with the appropriate objects. They were told to use a grid layout of a particular size (based on a picture) and the objects included JLabel, JRadioButton, ButtonGroup, JCheckBox, JComboBox, JTextField, and JButton. Students also had to add the appropriate listeners.

The final two quizzes concentrated on distributed programming. In the Producer-Consumer quiz students had to write the put and get methods for a bounded buffer class that used a simple array to store objects. In addition to testing the students understanding of the wait and notify methods, students had to properly index the locations for input and output. The Instant Message quiz involved writing the run method in the client handler thread that received commands, such as “Send Message:”, and data from the client and carry out the appropriate operation.

The performance on the Java programming quizzes is shown in Figure 10 where solutions were classified as: not working, partially working, and completely working.

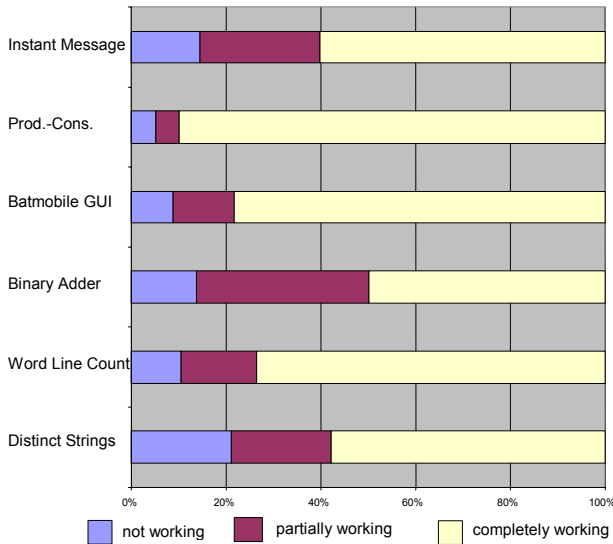


Figure 10: Java Programming Quiz Performance

In the CS1 course students had 90 minutes for each of the C++ quizzes. In the GUI course lab time was limited to 50 minutes. For some of the programming quizzes this was not long enough for an average student to complete the quiz. The beauty of the Quiver system is that we could allow additional time for a particular quiz in an open lab environment. We trained our lab monitors to check the student ID, release the quiz for an additional hour to work on, and then disable the quiz after the student passed the quiz or reached the time limit.

We also used Quiver in a senior-level required course as a programming "competency" check. There were 12 students (4 graduate and 8 undergraduate) who took a single quiz, which could be solved using either C++ or Java. Seventy-five percent passed the quiz within the expected time. The remaining students passed when they were given some additional time. The number of submissions before passing varied widely: from 3 to over 30. In a few cases it appeared the students succeeded through sheer perseverance rather than through logical debugging.

5. Future Work

We have plans for several enhancements for the Quiver System. We plan to add a timing mechanism that will detect code that runs slowly. This will be particularly useful for more advanced students where efficiency of code is an important consideration in addition to correctness of results. We plan to add more administrative tools to allow the instructor to view student progress dynamically during a closed lab session. We will also improve the quiz release system so that the quiz may only be taken on a machine with a particular IP address. Finally, we will create quiz groups where the quizzes deal with the same topic and are

judged to be of equivalent difficulty. Quizzes may be assigned randomly and, in the case a student is taking a second quiz from the group, an alternative will be randomly selected.

6. Conclusions

Our use of the Quiver System in a CS1 course confirmed what we suspected: some students were passing the course (C or better) by performing satisfactorily on exams and completing programs outside of class, yet these students were still lacking some of the basic programming skills we expect from students completing this course. Using Quiver has helped these students develop their programming skills.

In the Java course we used Quiver to introduce programming assignments in a closed lab environment. For example, in the Producer-Consumer quiz the students only had to write the synchronized methods for get and put. In the subsequent assignment they had to complete the three threads, Producer, BoundedBuffer, and Consumer, and then design a GUI that allowed a user to control the rates of production and consumption with sliders and display the contents of the buffer dynamically.

We have found the Quiver system to be a valuable pedagogical tool to help students develop essential programming skills. It has also proved to be a viable mechanism for introducing core concepts for a topic that could be expanded into a complete, out-of-class programming assignment.

References

- [1] K. Bowles, A CS1 course based on Stand-Alone Microcomputers, *SIGCSE Bulletin*, Vol. 9, No. 1, February 1978, pp. 125-127
- [2] J.M. Adams, B. Kurtz, A State-of-the-Art CS Undergraduate Lab, *Software Engineering Education - Lecture Notes in Computer Science #423*, Springer-Verlag, 1990, 85-94
- [3] B. Kurtz, H. Pfeiffer, "Developing Programming Quizzes to Support Instruction in Abstract Data Types", *SIGCSE Bulletin*, Vol. 21, No. 1, February 1989, pp. 66-70
- [4] Michael Main, Walter Savitch, *Data Structures and Other Objects Using C++*, 2/E, Addison-Wesley, 2001
- [5] Steven S. Skiena, Miguel Revilla, *Programming Challenges*, Springer Verlag, 2003
- [6] <http://www.junit.org/>
- [7] <http://cppunit.sourceforge.net/>
- [8] <http://cocoon.apache.org/>