Using Software Testing Techniques for Efficient Handling of Programming Exercises in an e-Learning Platform

Joachim Schwieren, Gottfried Vossen and Peter Westerkamp
University of Muenster, Germany
schwieren@uni-muenster.de
vossen@uni-muenster.de
westerkamp@uni-muenster.de

Abstract: e-Learning has become a major field of interest in recent years, and multiple approaches and solutions have been developed. A typical form of e-learning application comprises exercise submission and assessment systems that allow students to work on assignments whenever and where they want (i.e., dislocated, asynchronous work). In basic computer science courses, programming exercises are widely used and courses usually have a very large number of participants. However, there is still no efficient way for supporting tutors to correct these exercises, as experience has shown that correction (and, beyond that, automatic grading) are difficult and time consuming.

In this paper we present an enhancement of the xLx platform developed at the University of Muenster to efficiently support tutors in handling Java programming exercises electronically. The new component is based on concepts of automatic static and dynamic testing approaches, well known from software engineering, and provides an automatic pre-correction of submitted solutions. In addition, a tutor is able to annotate solutions manually, by adding comments that are associated with the source code of the solution in an intelligent way. Static tests are based on a compilation of the sources to find syntactical errors, while dynamic tests use test cases defined by tutors during the creation of the exercises and have to be executed correctly on the solutions in order to receive credits for the exercises.

Keywords: Programming exercises, Automatic pre-correction, e-Learning, Blended learning

1. Introduction

e-Learning has become a major field of interest in recent years, and multiple approaches and solutions have been developed so far. A typical form of e-learning applications are exercise submission and assessment systems that allow students to work on their assignments whenever and where they want (i.e., dislocated, asynchronous work). Since the demand for new types of exercises arises based on the teaching environment, it is important that e-learning systems provide a flexible way to add new types of exercises and that they are scalable concerning the demands of different courses and lectures. This paper describes an enhancement of an e-learning platform by an automatic program testing facility.

In undergraduate and basic computer science courses, programming exercises and assessments are widely used. These courses usually have a very large number of participants which leads to several problems when the submitted exercises have to be corrected and graded by tutors. The main reason for this is that programming exercises, no matter which programming language is used, tend to have a large degree of freedom for learners. Thus, simply comparing the provided solutions with a sample solution does not produce a reasonable result that can be used for grading, since different, yet still correct solutions to one and the same exercise exist. It is possible that the submitted solution still fulfils the required aspects of the exercise, but follows a completely different way to solve the problem than the given sample solution. Only a manual correction by an experienced tutor and a semantic comparison with a sample solution can lead to an acceptable form of correction. The tutor must be aware that there are a lot of ways to solve one specific programming problem. Some programming languages offer a larger degree of freedom than others, but generally this characteristic is typical for a high-level programming language.

As a consequence, correcting and grading of programming exercises is commonly done completely manually. Only centralized submission systems are sometimes used to standardize the submission process, but this does not solve the actual problem that arises with this special exercise type. Thus, tutors have to install a submitted solution on a test system to execute and run tests. Running tests on a submitted source code is essential, since it has a high complexity and even an experienced tutor might overlook faults by only reviewing the source code without executing it. In case of several hundred submitted exercises, which is a regular figure in courses at large universities, this can be an exhausting task for a tutor. Annotating source code is also not very comfortable since the student will have to browse through the full source code in order to find the annotations made by a

Reference this paper as:
Schwieren J et al (2006) "Using Software Testing Techniques for Efficient Handling of Programming Exercises in an e-Learning Platform" *The Electronic Journal of e-Learning* Volume 4 Issue 1, pp 87-94, available online at www.ejel.org

tutor. This is why annotations are often made manually on printouts of the source code.
A couple of approaches for automatically correcting and grading programming exercises exist in the literature, but recent solutions mostly offer an automated testing by interacting via the command line with provided solutions. These approaches are of course very limited since there are a lot of applications or software components that do not have a command line interface such as most Java classes, GUI based applications or stand-alone algorithms that are not embedded in a specific program. The most sophisticated system so far is "Praktomat" (Zeller 2000) which has been developed at the University of Passau to support programming learning classes. Other approaches such as BOSS (Joy and Luck 1995), TRY (Reek 1989), Online Programming Assessment Tool (Roberts and Verbyla 2003) or ELP (Truong et. al. 2003) mostly focus either on offering a solution to submit exercises using a Web interface or offer concrete testing functionalities but not both.

In this paper we provide a closer look at the integration of programming exercises in the xLx e-learning platform that has been developed at the University of Muenster (Hüsemann et al. 2002). The new component is Web based and builds on foundations well known from software engineering. The exercises are a typical part of undergraduate and basic computer science lectures and normally several hundreds of students assign to those courses. The paper starts with an explanation of the existing platform in Section 2, and then explains the enhanced architecture of the xLx system that now integrates JUnit and Apache ANT to automatically compile sources and execute test cases on a submitted solution in Section 3. In particular, we explain in detail how tutors can define exercises and assign (Java) test cases to them. We also show the learner's view and indicate how annotations are provided for learners to learn from mistakes. In Section 4 we provide a short explanation of security reflections that had to be done since submitted code is executed on the xLx server that might be maleficent, and we conclude with a short outlook.

2. xLx – a scalable e-learning platform

xLx is the abbreviation for "eXtreme e-Learning eXperience" (Hüsemann et. al. 2002, Vossen and Westerkamp 2004). It is a Web based online learning platform developed at the University of Muenster that can either be used in university or commercial contexts. The main objective of xLx is to support the exercise portion of technically oriented university courses (e.g., database systems, database implementation, computer networks, workflow management). xLx is part of a "blended learning strategy" that combines classroom teaching with electronic exercise work. This strategy is based on the observation that classroom teaching in the courses mentioned is necessary and leads to better learning results than a complete shift of teaching solely to Web courses. The original motivation for the development of xLx was based on the following observations: Current university classes (and embedded exercises) typically take place in strictly periodic meetings, are bound to certain teaching environments, mostly ignoring the progress, needs, and time constraints of individual learners. Students spend less and less time and effort to work on courses and exercises continuously. Reasons for this trend are manifold and shall not be discussed here. The target courses of our system, in particular database and information systems courses, offer lots of potentials for computer-based, interactive, often visualized or animated training and testing. Learners need to practice and train their skills with full-scale software systems (e.g., database management systems) that are reasonably administered at the university only.

xLx addresses these observations as follows. Students can work on assigned exercises anytime and anyplace if Internet access and a standard Web browser are available. Students may determine their own pace when solving exercises; however, a didactically meaningful sequencing of exercises is still enforced by the system (as is a time limit per assignment). Moreover, students may ask for additional exercises either if they have difficulties with the presented material or if they would like to work on more challenging problems. Finally, learning modules based on realistic problems and transparent access to underlying commercial systems raise hopes in more fun and better learning success while solving the exercises accompanying a course. Correcting and grading assessments can be quite time consuming (depending on the exercise types used in the assessment). The use of xLx to make the grading process more efficient, particularly for complex exercise types such as programming exercises is also one of its primary goals.

xLx embodies a personalized learning platform that offers hands-on experience in terms of practical exercises, covering a wide range of conceptual, language specific or algorithmic aspects of a particular field. xLx gives transparent access to underlying (commercial) systems (e.g., database or workflow management systems), which are centrally administrated. The xLx platform organizes exercise solving in terms of

closed user groups, where every member has his or her own password-protected account. Each account provides access to a course portal that offers traditional material such as slides, lecture notes, learning objects (Downes 2001), and further links as well as an email list, a discussion forum, and a personalized training section. This training section is divided into two parts: Test section: In this section students are able to train their skills concerning course relevant techniques (e.g., SQL queries, object-relational features of SQL: 1999, transformation of XML documents with XSLT or XQuery), and they can deepen their understanding of covered algorithmic techniques (e.g., database system algorithms such as algebraic query optimization, the two-phase-locking protocol for transaction synchronization, or the redo-winners protocols for restarts after system crashes, see Weikum and Vossen 2002).

Submit section: This section contains the exercises that have to be solved during the term and according to predefined deadlines. New exercises show up in this section as the necessary background has been covered in class. Solutions can be prepared and tested in the test section mentioned above. Once submitted, solutions cannot be changed any more, and they appear on a work list of a teaching assistant by whom they are corrected and annotated.

So far, xLx knows five types of exercises: free-text, multiple choice, SQL queries, XSLT and XQuery transformations. While the first two of these exercise types are standard ingredients of an e-learning system, the latter are unique to our system, as they are coupled with transparently integrated underlying systems, in our case a relational database for SQL (IBM DB2 Universal Database) and XSLT and XQuery processors. The integration of different systems avoids technological and administrative barriers, as students do not have to install these systems at home; instead, they are accessed via standard Web browsers. Finally, exercises for the last four of the above types are stored along with solutions inside the xLx platform, which allows for an automatic pre-checking of solutions and makes life of teaching assistants easier.

Technically speaking, xLx is a Web based application and implemented in typical three-tier client-server architecture. To access xLx only a standard Web browser is needed; special plug-ins or additional client-side applications such as Java Runtime Environment (JRE) or Flash™ are not required. The xLx platform is implemented on top of an Apache Web server and a mySQL database running on a Linux platform, i.e., the entire xLx system is based on open source software. The mySQL database contains student data, exercises and solutions. Communication between clients and the xLx platform is secured by SSL (HTTPS), which provides basic security of confidential student data (passwords, solutions, and student's grades). All Web pages are generated dynamically by PHP4 scripts (ordinary pages) and Java Servlets (database connections via JDBC). The database server IBM DB2 Universal Database is used for database related exercises (SQL: 1999, object-relational features, DB2 extenders). Thanks to the IBM DB2 scholar's program, there are no costs involved in using DB2 at universities. Finally, PHP is used for calls to the XQuery and XSLT command-line processors.

3. Programming exercises and assessments in xLx

Since xLx is mainly used in technically oriented computer science courses at university, one very specific type of exercise was so far missing to support the all base courses in an efficient way: xLx was not able to handle programming exercises; we will now explain how we have remedied this situation.

An analysis of this very specific type of exercise has come to the conclusion that only certain aspects of the solutions are relevant for grading. This is on the one hand the question whether the submitted solution fulfils the specifications stated in the exercise and on the other hand the way how certain problems have been solved (e.g., implementation of a specific sorting algorithm that is required in the exercise). Other aspects such as naming of internal variables, methods or classes are (usually) not relevant for grading, but prevent an automated code review based on a comparative approach. Our approach to the verification of programming exercise solutions is based on methods and techniques well known from the field of software testing. Software testing has become quite a large field of knowledge in recent years and many different techniques and methods exist. Owing to the fact that (automated) software testing represents an important aspect in the quality assurance process of commercial software development, our approach adopts these techniques and methods to the context of e-learning.

Two main types of software tests can be distinguished (apart from many other possible classifications that exist). On the one hand, *static* tests analyse or probe a test object (in the e-learning case this is the submitted solution) without executing it. A syntax-check of source code is an example for a static testing technique. In addition, all kinds of reviews such as technical

walkthroughs or even informal reviews can be classified as static tests. On the other hand, tests of functionality are known as *dynamic* tests. The first step in a dynamic test is to specify *test cases* that invoke a certain reaction or output on the test object. In addition, the expected outcome of the tests needs to be defined in advance. Comparing the expected behaviour with the actual behaviour builds the foundation to classify a test as failed or passed.

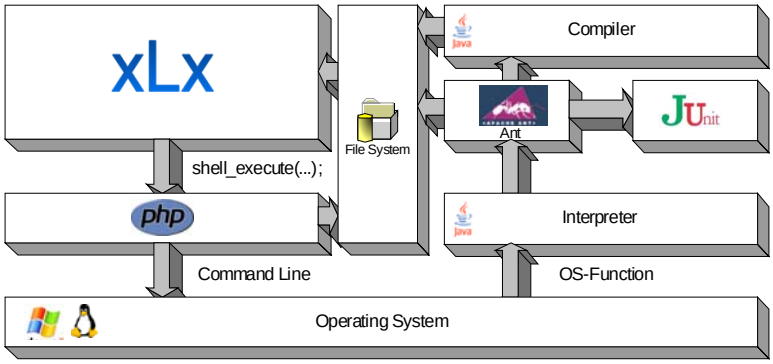

Figure 1: Integrative approach of the xLx-java-testing-module.

The new exercise type currently only supports Java programming exercises. This decision was made since Java is widely used both in educational and business contexts. In addition, sophisticated build tools such as Apache ANT and test tools such as JUnit (Massol 2003, Hatcher 2002) are available for Java. As Apache ANT and JUnit have already proved to work well in the testing framework on which xLx relies, they have been integrated into the platform. The enhanced architecture of the xLx system is shown in Figure 1. The basic xLx system uses a HTTP based upload mechanism to store the solutions of the learners in the file system of the server. The calls of the underlying test mechanisms are done by the PHP command-line functionality and are described next.

For pre-correction of programming exercises xLx uses both static and dynamic tests. Like for each exercise type supported by xLx the platform provides a framework to create, solve, grade, annotate and view the results of exercises. For the programming exercise type the user interface is quite simple, as the solution is developed on the student's machine by using an IDE (Integrated Development Environment) or editor program of his or her choice. xLx only has to provide a simple browser-based upload functionality in order to submit the file(s) of a learner's solution to the xLx server. After a learner has submitted a solution, the first step of the automatic correction facility is a static test by compiling the source code of the learner. This is done by xLx on the server. The compilation results are stored as an XML file that is later parsed by PHP to get the results back into the xLx system. It represents the static test results since only a syntactically correct Java file can be compiled. If the compilation fails (e.g., due to syntax errors or due to irresolvable dependencies to other java classes) the dynamic testing step is not executed and an error message is presented to the learner who has tried to submit the exercise. If the compilation was successful, the dynamic test cases are applied next.

JUnit is used as testing framework that executes the test cases and collects the results. Special JUnit test cases are defined in Java (see Listing 1) for every exercise. To better handle the compilation and test process, Apache ANT is used to both compile the submitted solutions, i.e., the Java files, and execute the JUnit test cases on these solutions. JUnit and Apache ANT can be integrated quite simple because of a so-called ANT Task (a plug-in for ANT) for JUnit that is already available. This enables a high flexibility that comparable approaches, which focus on a simple, command line-oriented, text-based input/output concept, cannot offer. In contrast to the new xLx module these command line-oriented systems cannot use, for example, the Java Reflection API in test cases to allow a very detailed way of analysing the submitted solutions. Other testing frameworks such as DejaGNU[1] do not offer a wide range of functionalities that can be used so easily because typically those frameworks use a very restricted and proprietary scripting language to specify test cases. Since Java is a full-fledged programming language there are actually no limits for the creativity of test case designers.

The test case shown in Listing 1 sketches the basic design of a JUnit test class. One or more methods beginning with the keyword "test" indicate the test methods that will be executed by the framework. After setting up the required

[1] http://www.gnu.org/software/dejagnu/

objects for the actual test (this setup is called fixture in the JUnit terminology), the specific tests are defined by using so-called *assertions* that are fully maintained by JUnit. An assertion compares a specified result value with its expected outcome. A textual description of the assertion can be added optionally. JUnit monitors the results of the executed assertions automatically. Finally, a test protocol is being created as a result.

```
import junit.framework.*;
public class TestMyTest extends TestCase {
  // Constructor to provide the class name
  public TestMyTest(String name) {
    super(name);
  }
  // actual test case
  public void testSampleTestmethod() {
    // Test fixture
    MyDate aMyDate = new MyDate();
    aMyDate.setJahr(2010);
    // Assertions
    Assert.assertEquals("Testing getter and setter
methods for year",
                        2010,
aMyDate.getJahr());
    // ...
```

Listing 1: A basic test case.

The information that a test case has passed or has failed is, in contrast to the compilation results, not necessarily presented to the learner. This decision depends on whether the test case has been declared as *public* or *hidden* by the tutor. For a public test case, the result of the test is presented to the student. These test cases are not used for grading and therefore no credits can be achieved for public test cases. So the basic idea of public test cases is to define a certain level of quality and/or functionality that the submitted solutions will have to fulfil in order to be accepted by the system. Hidden test cases, on the other hand, are used for grading the submissions of learners. When designing a programming exercise in xLx, the tutor can define for every hidden test case the credit points that can be achieved if a particular test case is executed successfully. Figure 2 shows the xLx front-end to define programming exercises. In the upper portion of the screen a tutor can assign an exercise to an already existing section that comprises several exercises to be solved by learners. The level classifies the difficulty of the exercise, for which in this case a maximum of 10 points can be achieved. The type of the exercise is “Java” which points to a (Java language) programming exercise. The text of the exercise to be solved is defined in the middle portion of the screen and will be displayed to the learner. The lower portion of the screen (with screen texts still in German) defines the test cases (here: Test1MyDate.java, Test2MyDate.java, and MyDateTestHilfsmethoden.java) that will be executed on the solutions of the learners. The first test case is marked as essential for this exercise and is defined as a public one without any credits. The second one is a hidden test case for which the learners can earn up to 10 points. In the lower right part of the screen tutors can upload sample solutions that will also be displayed for a corrector of the exercise.

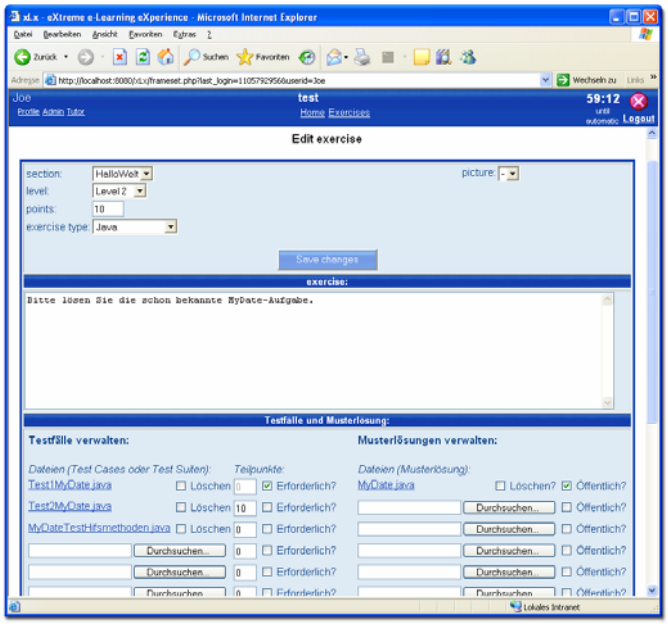

Figure 2: xLx GUI for a tutor to configure new exercises.

As mentioned before, test cases are specified using the JUnit test case class. All test cases that are stored in a separate class can be used for grading. If more than one test case is provided in a single class (a so-called *test suite*) the assigned credits for this suite can only be achieved if all tests inside the suite passed. This also gives the tutor a lot of flexibility when designing test cases and the associated grading scheme.

Clearly, test cases may fail for different reasons. One is that the expected outcome does not match the actual outcome of the submitted solution. This is the most typical reason why a test case fails and is just called “failure”. Another reason might be an unhandled exception during the execution of a test case. This could be the case when the test case calls a test object with values that are not allowed and that are not correctly rejected by the test object. To this end, xLx can also track exceptions. If a test case fails due to an exception this is called “error”. The test protocol does not only show the exact figure of passed and failed test cases; in case of an exception, detailed information is given about the latter and thus learners get a clue of what went wrong and tutors can get a better view inside the provided solution.

It should be obvious that an automated test can only provide limited feedback information to a student. To give learners more information on their solution, xLx also offers a possibility to review and annotate the source code of the provided solutions in a very comfortable way by a human tutor in addition to the automated features that do not require any interaction with the tutor. As shown in Figure 3, xLx displays all source files within the browser window and applies a special Java syntax-highlighting scheme to make the reading of the source code more comfortable. To annotate a certain line of code, the tutor simply writes notes into a special input box and xLx associates the comment with the source file and the specified line of code without changing it.

When a student takes a look on the corrected and graded solution, all initially submitted source files can be viewed within the browser window. Figure 3 pictures this screen that is comparable to the screen of the tutor. Every line that contains an annotation made by the tutor is marked with a special glyph that indicates the presence of a comment (see source code lines 9 and 10 in Figure 3). By clicking on the glyph the annotation made by the tutor is shown in the lower part of the window. The mixture of automated testing and grading in combination with a source code review done by a tutor provides a maximum learning experience for the student.

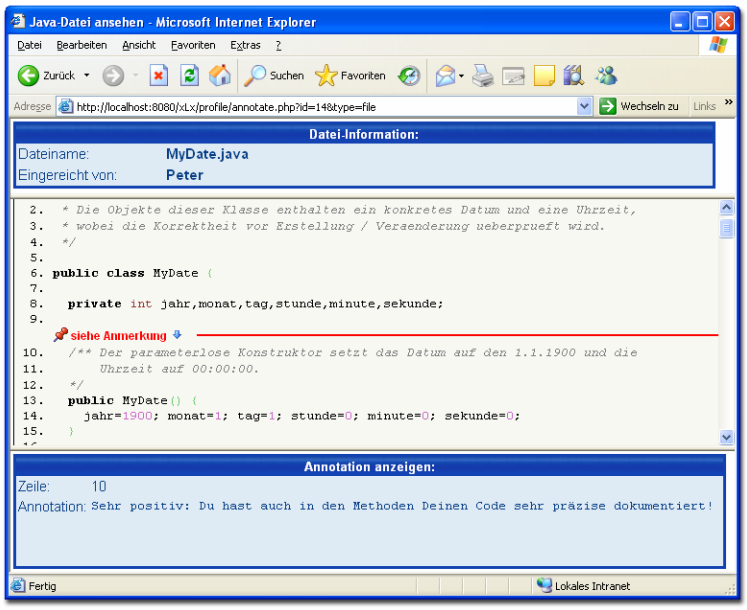

Figure 3: Annotation window from the student’s point of view.

When developing the Java testing module for xLx, a major aspect has been security. Since unknown code provided by students is compiled and executed on the same server on which xLx itself is running on, there must be a guarantee that malicious code cannot affect the system. Since Java applications are not executed directly on the physical machine but inside a virtual machine (VM), there are some integrated security mechanisms in the Java VM that can be used to secure the system. The Java interpreter can be configured using so-called Java *policies*. Policies are simple text files that specify detailed rules describing which classes of learners’ solutions are

allowed to execute which functionalities on the system. This also includes a precise way to control IO access to the hard disk and to the network. Using a very restricted policy for the programming exercise module of xLx, we can ensure that even malicious code cannot harm the system if it gets executed.

4. Summary and conclusions

xLx has been used successfully for several years in different courses and in different universities (VAWi 2002). The spectrum of these courses covered databases, XML and computer networks. Each student had to solve an average of 40 exercises throughout a term. One of the main intentions of the xLx platform is the natural integration of third-party modules to allow hands on experience with real-world enterprise application systems. The newly integrated Java exercise type has so far been tested in small courses only. A "real world" course scenario with a many participants will be introduced soon, in order to verify its design goals such as reduction of work and time effort on correcting and grading programming exercises.

It will be interesting to observe server performance since it has to handle a load of hundreds of users executing Java applications on it in huge courses. The Java compiler has not been designed to be used in a multi-user environment, and it will be interesting to see how performance will scale. If problems concerning system performance and stability should arise, a modification will have to be applied to the existing architecture: by implementing a ticket-based scheduler-driven compilation and testing process it will be possible to better balance the high load that is created by hundreds of submissions at the same time. Though students won't get an instant feedback any more after submitting their exercises since they will have to wait for the compilation/testing job to be completed, this should not be such a big problem because the time between submitting a solution and getting a feedback from the system should only take same seconds.

For courses done in the last couple of years we have recognized a high acceptance of the system and particularly of the test section. There has been a frequent usage of the attached third-party systems; for example, more than 100 students of a database course generated a total of 50.000 SQL statements against the underlying DB2 database. Students have also accessed the xLx platform from all over the world and all around the clock. Some of them, who stayed abroad, for example in Finland and Australia, have used the system to work on the exercises to manage their examination after their return.

xLx can be found on the Web at https://dbms.uni-muenster.de/xLx. Its front end is entirely designed in English so that even foreign learners can use it. A demo access can also be obtained over the Web.

References

Downes, S. (2001) "Learning Objects: Resources for Distances Education Worldwide." International Review of Research in Open and Distance Learning 2 (1).

Hatcher, E. (2002) "Java Development with ANT". Hanning, Book News, Inc., Portland, OR

Hüsemann, B., J. Lechtenbörger, G. Vossen, P. Westerkamp (2002) "XLX - A Platform for Graduate-Level Exercises." In Proc. International Conference on Computers in Education (ICCE) 2002, Auckland, New Zealand, pp. 1262-1266.

Joy, M., Luck, M. (1995) "On-line submission and testing of programming assignments." In J. Hart, editor, Innovations in Computing Teaching. SEDA, London, (1995).

Massol, V. (2003) "JUnit in Action". Hanning, Book News, Inc., Portland, OR

Reek, K. A. (1989) "The TRY system -or- how to avoid testing student programs." ACM SIGCSE Bulletin, vol. 21, no. 1, pp. 112–116, 1989.

Roberts, G.H.B. and Verbyla, J.L.M. (2003) "An Online Programming Assessment Tool." In Proc. Fifth Australasian Computing Education Conference (ACE2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, 20. Greening, T. and Lister, R., Eds., ACS. 69-75.

Truong, N., Bancroft, P. and Roe, P. (2003) "A Web Based Environment for Learning to Program." In Proc. Twenty-Sixth Australasian Computer Science Conference (ACSC2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, 16. Oudshoorn, M. J., Ed. ACS. 255-264.

VAWi (2002) "Virtuelle Aus- und Weiterbildung Wirtschaftsinformatik", [online], Universität Essen, Prof. Dr. H. H. Adelsberger, http://www.vawi.de/, 2002

Vossen, G., P. Westerkamp (2004) "XLX and L2P — Platforms for Blended Learning". EMISA Forum 2/2004, 18–20.

Zeller, A. (2000) "Making students read and review code." In Proc. Fifth ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), pages 89–92, Helsinki, Finland, July 2000.

Weikum, G., G. Vossen (2002) "Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery;" Morgan-Kaufmann Publishers, San Francisco, CA

