# Improving the Quality of Programming Education by Online Assessment

Gregor Fischer
University of Würzburg
Am Hubland
97074 Würzburg
+49 / 931 / 888 – 66 11
fischer@informatik.uni-wuerzburg.de

Jürgen Wolff von Gudenberg
University of Würzburg
Am Hubland
97074 Würzburg
+49 / 931 / 888 – 66 02
wolff@informatik.uni-wuerzburg.de

## ABSTRACT

The paper presents an online Java course consisting of a tutorial that provides a high level of interaction and an assessment tool that analyses the code and enables the students to run a suite of predefined tests.

The hypertext tutorial contains a lot of interactive, editable examples and many exercises to check the student's progress.

In the assessment tool various electronic evaluators check for the conformance of uploaded student programs to coding conventions, proper documentation, compliance with the specification and, last but not least, the correct execution of supplied functional tests.

The tool not only provides a tremendous help for the correctors by reducing the manual assessment time by a factor of about 4, but also is appreciated by the students for its immediate reaction, because development times (especially during debugging) can be shortened considerably and students can gain a much higher confidence in the quality of their own program.

## Categories and Subject Descriptors

D.2.5 **[Software Engineering]**: Testing and Debugging – *Code inspections and walk-throughs, Testing tools.*

K.3.1 **[Computers and Education]**: Computer Uses in Education – *Computer-assisted instruction (CAI), Computer-managed instruction (CMI), Distance learning.*

K.3.2 **[Computers and Education]**: Computer and Information Science Education – *Computer science education.*

## General Terms

Measurement, Reliability.

## Keywords

Assessment, Programming Education, Quality of Programs, Teaching, Testing.


PPPJ 2006, August 30 – September 1, 2006, Mannheim, Germany.


## 1. OBJECTIVES

A sound and thorough education in programming is one of the key qualifications a student of computer science has to have and it is further a mandatory requirement for most jobs in IT, networking, etc. A clear and pleasing representation of the concepts of a programming paradigm and its realisation in a specific language as well as the opportunity for many practical exercises are crucial for the success of a course [3].

Since more than 25 years, we have made a lot of experience in running programming courses for different languages and paradigms. The lectures always included assignments for programming exercises, which were mostly handed in as handwritten programs via e-mail or WWW, in former times even as print-outs. These solutions were then manually inspected and evaluated.

It is our educational objective to advice the students to write good programs and not only to learn the syntax of the language. Programming is a creative process, it can only be learned by reading and, first of all, writing programs. In a more detailed study in [1] five steps of learning are distinguished: following, coding, understanding and integrating, problem solving, and participating. In this terminology we emphasize the first 3 layers.

For all these steps we provide an immediate response. All sample programs are executable and can be modified easily, hence, the reading or following task is supported. For the coding step we do not only rely on the error messages generated by a compiler, but also enforce several coding and documenting conventions. Our main focus, however is the provision of immediate, automatic checks for the written program. Hence the students learn to understand the meaning of the language constructs. The instantaneous feedback increases the motivation quite considerably.

Assignments of programming courses, hence, have to cover more than the syntax of the used language. We want to see and assess complete programs. Multiple choice questions are not appropriate, since they turn to focus on very particular or peculiar properties instead of supporting a good, solid programming style. That can be achieved by formal or structural tests (see below). The proper functionality of a given program may be assessed by checking its output (e.g. as a free form text). Again care must be taken in order to avoid problems with irrelevant details like whitespaces or formatting.

More sophisticated tests are necessary to judge the quality and functionality of uploaded programming exercises, such as the style of writing and formatting and the correctness of results. In

traditional courses these properties should be - and were actually - assessed by looking into the source code and documentation of the student's solution. Hence, the assessment of programming assignments usually took a lot of human corrector time.

The automation of these reviews and quality assurance tests by our tool is described in the following section.

## 2. AUTOMATIC ASSESSMENT OF JAVA PROGRAMS

In recent years we have developed an online Java course consisting of a tutorial that provides a high level of interaction and an assessment tool that analyses the code and enables the students to run a suite of predefined tests [2].

The hypertext tutorial can be navigated via a predefined learning path or by freely following the hyperlinks. It contains a lot of editable and executable examples and many exercises (from multiple choice and free text to programming exercises) to check the student's progress.

The solutions for the programming exercises, small or medium-sized programs, are uploaded to the assessment tool [4]. Thence, various electronic evaluators perform:

- *Formal tests*
  We check if the program compiles, conforms to coding and naming conventions, and if appropriate comments are present such as Javadoc comments.
- *Structural tests*
  The recommended usage of structured statements or data types is checked. Violation of these conventions may lead to hidden mistakes. It is e.g. possible to enforce that the statement following a loop or branch statement is a block or that case clauses are terminated by a break or a return statement.
- *Specification tests*
  The specification requires that classes with predefined names or methods with given signatures are expected. Classes have to extend other classes or implement given interfaces. In a teaching environment it may be interesting to prohibit the usage of packages or classes.
- *Functional tests*
  Comparison with a master solution or a set of JUnit tests are performed.

### 2.1 Functional Tests

The common practice for functional tests is black box testing, using e.g. the JUnit framework. These tests check the solution for well known test cases and their expected results. Therefore these results must be known in advance, what might be problematic, if the way to compute the results is rather complex.

Whereas the development of black box tests is or at least should be standard in a development team we also provide the opportunity to compare the results of the student solution with a master solution.

The comparator provides a set of classes and methods that mirrors the structure of both solutions. The implementation then calls the respective constructs from the student's and the master's implementation, and compares the results. The comparison is done on the real objects, taking into account the internal structure of arrays and collections. Special methods for comparison can be provided such as a possible equality relation for floating-point numbers that only checks, is the student's solution lies within a given interval.

This simplifies the writing of tests tremendously, as (seemingly) standard manipulation of objects can be used, and no explicit checking needs to be done. Of course explicit checking is also possible, in case the implicit is not sufficient.

### 2.2 Combining Tests

The tests can be configured and individually tuned for each specific exercise. The combination of different tests makes sense especially in a teaching environment where we are interested in more than results. We also want to know or prescribe the manner these results are produced. A method may have to be called recursively, another must not use a while loop and a third method has to advance an iterator of a data structure properly. The results have to be the same. Such conditions or requirements can easily be imposed by our tool by combining the appropriate (e.g. the structural and functional) tests.

All tests may be mandatory, optional or even secret. A program is only accepted, if it passes the required checks. Since the automatic assessment is carried out immediately, the students do not need to wait for feedback and can fix problems and upload corrected versions immediately. The number of turn in tries usually is not limited, there is only a general submission deadline for an exercise.

Having optional tests is especially important in courses, where programming is not the major objective. Here compliance to e.g. formal tests is often not enforced in order to allow students to concentrate on different matters. Often even some or all functional tests are configured as optional, so that partially incomplete solutions can be turned in, because rejecting those completely would be too strict. Yet, those tests are usually included, so that students remember, that their programs can still be improved.

## 3. EXAMPLE

Let us illustrate the work of our tool by a simple exercise:

Write a class StringReverse containing a method reverse(String) that reverses a string without using the method reverse() from the class java.lang.StringBuffer.

In the following we mimic the upload and assessment of several attempts to solve this exercise.

**1.Attempt:**

*Uploaded program* (shortened):

```
public static String reverse(String s) {
    return new StringBuffer(s).reverse().toString();
}
public static void main(String[] args) {
    for (int i=0; i<args.length; ++i) {
        System.out.println(reverse(args[i]));
    }
}
```

*Results of online checks* (shortened, translated from German):

Check 1: compilation: passed
Check 2: coding conventions: passed
Check 3: signature and hierarchy: failed
Invocation test checks whether prohibited classes or methods are used; call of method reverse from the prohibited class java.lang.StringBuffer

**2. Attempt:**

*Uploaded program* (shortened):

```
public static String reverse(String s) {
    if (s == null) return null;
    if (s.length() <= 1) return s;
    char first = s.charAt(1);
    String rest = s.substring(1);
    return reverse(rest) + first;
}
```

*Results of online checks* (shortened, translated from German):

Check 1: compilation : passed
Check 2: coding conventions: passed
Check 3: signature and hierarchy: passed
Check 4: dynamic grey box test: failed
StringReverse.reverse(java.lang.String) yields „llebe“ instead of „leben“

**3. Attempt:**

*Uploaded program* (shortened):

```
public static String reverse(String s) {
    if (s == null) return null;
    if (s.length() <= 1) return s;
    char first = s.charAt(0);
    String rest = s.substring(1);
    return reverse(rest) + first;
}
```

*Results of online checks* (shortened, translated from German):

Check 1: compilation: passed
Check 2: coding conventions: passed
Check 3: signature and hierarchy: passed
Check 4: dynamic grey box test: passed

## 4. TWO VERSIONS

Currently there are two versions (note that both are in German) of the tool available. A server based variant (at http://jop.informatik.uni-wuerzburg.de/), hosted by the University of Würzburg on behalf of the Virtual University of Bavaria (vhb). A demo version of the tutorial including assessment of exercises is available at http://jop.informatik.uni-wuerzburg.de/tutorial/demo/. The system is called Praktomat [6], it organizes the download of exercises and upload of solutions, it controls submission date and time, automatically runs the prepared checkers, manages the manual reviews, and supports e-mail communication with the students.

The second version is published as a book [5] and a CD, including the newly developed evaluation tool Java Exercise Evaluation Environment. JEEE allows editing, running and testing of examples and exercises in a secure and interactive environment, embedded in an easy to use graphical client. It integrates formal, structural, specification and functional tests, and can therefore be used as a replacement for the server based testing in offline use. It can also be used to further support human correctors, as it allows for an interactive evaluation of even modified student solutions. Nevertheless the server based system is still required at least for turn in and other management purposes.

Due to the high level of interactivity and automatic responses we expect for the new version that the support of the human correctors will be increased. Since it can be used without an internet connection, the flexibility of the students will be enlarged. This is particularly important in continuing education that is one of the main goals of the second version. We are convinced that the coaching time, i.e. the time of personal presence of the trainer can be reduced by a factor of at least 2. Individual schedules for each participant will be possible.

## 5. EVALUATION

The system has been used in various different lectures (“Algorithms and Datastructures”, “Software Engineering”, “Introduction to Computer Science for non CS Majors”, “Advanced Training for Teachers”, …) or practical courses (“Programming in Java” and “Software Development”) at the University of Würzburg or for the Virtual University of Bavaria (vhb). During these courses a total of more then 1000 subscribed students handed in more then 3000 programs (solutions) with an average size of about 1000 lines, which passed the required tests for over 80 different problems. More then one hundred thousand attempts were rejected because they did not fulfil all requirements. The high number of attempts can be explained by the fact that the students use the online assessment as convenient test tool during program development.

**Table 1. Application of Online Assessment**

| Category | Number of Courses | Total Number of Students | Total Number of Attempts | Total Number of Solutions |
|---|---|---|---|---|
| Programming Lab | 6 | 870 | ~121 000 | 2 933 |
| Online Course | 4 | 150 | ~13 000 | 442 |
| Lectures for CS Majors | 3 | ~350 | ~7 300 | - |
| Lectures for non CS Majors | 7 | ~500 | ~15 300 | - |

The tests and the immediate responses of the system helped the students very much. Indeed, they made it possible to cope with more advanced problems. Test configuration varied from only

enforcing the solutions to be compilable to requiring full compliance with coding standards and an extensive functional test suite.

The reliability of the tests is proven by the fact that less than 2% of the programs that passed all tests were not accepted by a human inspector. Despite the higher quality the rate of successful participants has not changed in comparison with former courses.

**Table 2. Programming lab breakdown**
(note that different assessment policies were used)

| | Spring 2006 (Programming lab) | Fall 2005/06 (Programming lab) | Spring 2005 (Programming lab) | Fall 2004/05 (Online course) |
|---|---|---|---|---|
| Enrolled Students | 145 | 32 | 195 | 74 |
| Participating Students | 106 (73%) | 24 (75%) | 140 (72%) | 25 (34%) |
| Successful Students | 55 (52%) | 17 (71%) | 57 (41%) | 13 (52%) |
| Exercises | 16 | 15 | 11 | 8 |
| Attempts | 15 970 | 11 511 | 33 185 | 3 261 |
| Solutions | 1015 | 301 | 990 | 105 |
| Excellent Solutions | 41.2% | 81.8% | 64.5% | 48.2% |
| Average Solutions | 53.4% | 15.9% | 32.8% | 49.5% |
| Bad Solutions | 4.9% | 0.5% | 1.5% | 1.4% |
| Manually Rejected Solutions | 0.5% | 1.8% | 1.3% | 0.9% |

The readability and the quality of the accepted programs have increased tremendously. The solution of more comprehensive and more complicated exercises can be required. This goes hand in hand with the reduction of the assessment time for a human corrector by a factor of 3 to 4. A more thorough code inspection has been made possible. Of course, this final inspection is necessary, because we cannot proof correctness of programs online.

# 6. FUTURE WORK & CONCLUSION

While other systems like Maven [7] or CruiseControl [8] can also perform automatic program checking, they do not integrate well in a teaching environment, where students usually do not work as a team. They also lack the necessary resource constraint checks along with review- and grading support.

We therefore consider our system as necessary for our purpose. Because of architectural limitations in the current implementation a complete rewrite of the system is underway that focuses on:

- Service Oriented Architecture
- Maintainability
- Internationalization
- Tight integration with IDE
- Faster feedback times
- Improved support for correctors and advisors
- Cooperative and competitive programming

Yet, even with the current system we have shown that automatic assessment of programming exercises is possible. The supplied structural and functional test can reach a level of confidence that is high enough to reject false solutions. For the acceptance of seemingly correct solutions the precision is very high (more than 98%). For two reasons, however, we did not stop the human supervision, first it is an administrative legal problem, and second many programs that passed the tests could still be optimized and corresponding advice was given by the program inspectors. Altogether for programming courses we have achieved a higher quality of programs hand in hand with a more reliable and faster assessment.

# 7. REFERENCES


[1] Bruce et al: *Ways of Experiencing the Act of Learning to Program: A Phenomenographic Study of Introductory Programming Students at University.* Journal of Information Technology Education Volume 3, 2004.

[2] H. Eichelberger et al: *Programmierausbildung Online.* DeLFI 2003, A.Bode,J.Desel,S.Rathmayer,M.Wessner (eds), Lecture Notes in Informatics, GI, p.134-143.

[3] G. Fischer, J. Wolff v. Gudenberg: *Java Online Pedagogy.* ECOOP'03 Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts.

[4] G. Fischer, J. Wolff v. Gudenberg: *Online Assessment of Programming Exercises.* ECOOP 2004 Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts.

[5] G. Fischer, J. Wolff von Gudenberg: *Programmieren in Java 1.5*. Springer-Verlag, Berlin, 2005.

[6] J. Krinke, M. Störzer, A. Zeller: *Web-basierte Programmierpraktika mit Praktomat.* Softwaretechnik-Trends 22:3, S.51-53, 2002.

[7] Maven - A software project management and comprehension tool. `http://maven.apache.org/`

[8] CruiseControl - A framework for a continuous build process. `http://cruisecontrol.sourceforge.net/`