Making Students Read and Review Code

Andreas Zeller
Universität Passau
Lehrstuhl Software-Systeme
Innstraße 33
94032 Passau, Germany
zeller@acm.org

Abstract

The *Praktomat* system allows students to *read, review, and assess each other's programs* in order to improve quality and style. After a successful submission, the student can retrieve and review a program of some fellow student selected by Praktomat. After the review is complete, the student may obtain reviews and re-submit improved versions of his program. The reviewing process is independent of grading; the risk of plagiarism is narrowed by *personalized assignments* and *automatic testing* of submitted programs. In a survey, more than two thirds of the students affirmed that reading each other's programs improved their program quality; this is also confirmed by statistical data.

1 Introduction

How do we teach people how to write programs? By letting them write programs, of course. But how do we teach people how to write *good* programs? As instructors, we can easily assess the functionality by testing programs. We can also assess the readability by scrutinizing the code. But wouldn't it be much more beneficial if the students learned how to improve program quality *before* submitting their programs?

In industry, there are two *best practices* known to improve quality. The first is *testing*, which helps to improve the functionality. The second is to let someone else *read and review* the program. This is still the most effective way to obtain understandable, maintainable code. Not only does the quality of the programs increase this way. In the long term, authors and reviewers can foster a culture of egoless programming—no matter how smart a programmer is, reviews will be beneficial.

In a "traditional" programming course, students only experience one side of reviewing—the result of the instructor's assessment. What we'd like is that the student also finds herself in a *reviewer's situation*—struggling with code written by fellow students, and experiencing what it means if a program cannot be read or understood by other people.

For this purpose, we implemented *Praktomat*, an automated system for managing the submission, test, and mutual reviewing of student's programs. Praktomat streamlines program management by *on-line submission*, uses *automated testing* to assess program functionality, and provides voluntary *mutual reviewing* to improve readability and maintainability. The danger of *plagiarism* is countered by a couple of constructive measures, among them *personalized assignments*. An evaluation shows that program readability improved significantly for students that had written or received reviews.

2 Background: The Praktomat system

Our freshman programming course was attended by 118 students, with little or no programming knowledge from school. In a period of three months, they had to implement four programs realizing fundamental algorithms and data structures such as quicksort, graphs, or splay trees. The programs were to be written by each student alone; cooperative work would be taught in a subsequent large-scale software engineering project.

Our local legal situation allows teaching assistants only to give help and assistance for fellow students. The assessment of programs itself had to be carried out by the four instructors alone. Fearing to be overwhelmed with work, we built an automated system to manage the submission and assessment of student programs, named Praktomat (from "practice" and "automation").

In Praktomat, this is the basic process for solving a task:

1. The student logs into Praktomat and gets her assignment.
2. The student submits her program to Praktomat.
3. Programs that cannot be compiled or fail the test are rejected.
4. After successful submission, the student can retrieve programs of her fellow students in order to read and review them.

[0]To appear in *Proceedings of the 5th ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education*, Helsinki, Finland, July 11-13, 2000.

5. The student may obtain reviews and re-submit improved versions of her program.

6. Eventually, one of the instructors assesses and grades the program for functionality and understandability.

The system was implemented as a Python [7] CGI script, accessible via the WWW. Today, Praktomat consists of about 10,000 lines of code; it has been designed to be easily reused in other contexts.

3 Automatic Testing

In order to achieve basic functionality, Praktomat compiles and tests each submitted program, using the widespread DEJAGNU [6] and EXPECT [4] regression testing packages. The program itself is run in a *sandbox* where it could not access or manipulate any resources; time and memory usage were limited as well.

In Praktomat, programs can communicate with the user (and the testing package) only by means of standard input and output. Consequently, the assignment itself has to specify every single bit of interaction; and it has to be realized *exactly as specified*. Otherwise, the testing package rejects the program. As told in [2] or [5], our students found the demanded precision nit-picking at first, but got quickly used to read and realized the assignments verbatim.

The public test cases are disclosed to the students (Figure 1).

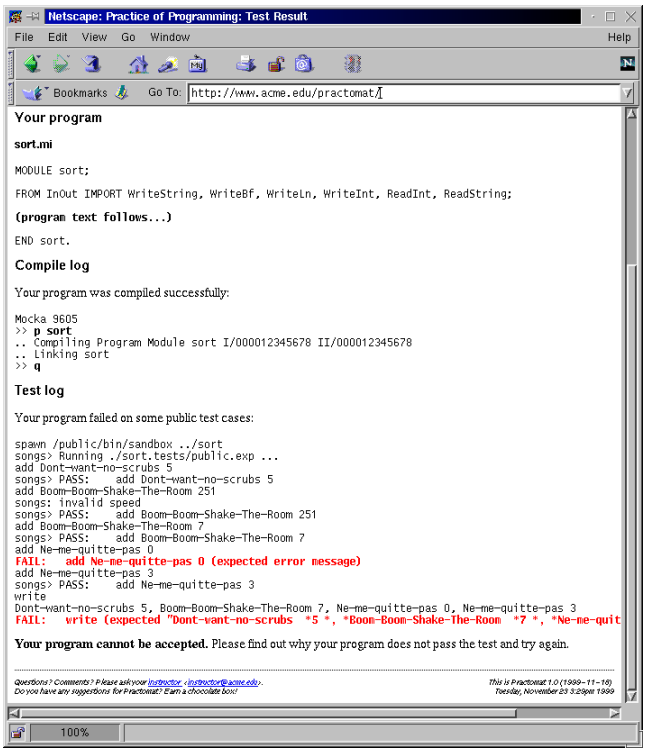

Figure 1: Testing student solutions in Praktomat

But Praktomat also runs a number of *secret test cases* whose results are disclosed to the instructors only when finally assessing the program. The students knew about the existence of these secret test cases, and they felt encouraged to realize correct and robust programs.

4 Personalized Assignments

As laid out in Chapter 1, we wanted to allow the students to read and review each other's programs. The first means to reduce the risk of plagiarism was to ensure that a student was allowed to review a program *only if she already had submitted her own solution*—that is, there would be no incentive to steal one other's work.

The second means was to ensure that in any case, the program author and the reviewer would have *differing assignments*. In fact, each assignment was *parametrized* by a number of boolean variables:

```
Use the ifdef(V1, Quicksort, Merge-
sort) algorithm to sort the entries
from the ifdef(V2, lowest, greatest)
to the ifdef(V2, greatest, lowest)
value.
```

This text was run through the macro processor M4, such that depending on the settings of `V1` and `V2`, a totally different assignment would be produced. Altogether, we had up to 256 different configurations, such that the chance of two students having exactly the same assignment was very small—and when matching author and reviewer, Praktomat would ensure a *maximum difference* between the two assignments.

The tests were personalized along with the assignments. In the example above, the test would expect a particular sorting order, depending on the value of `V2`.

5 Mutual Reviews

After a student has submitted her program successfully, she can retrieve a program in order to review it. Reviewing means that the program code could be read and annotated as well as assessed for various style considerations, such as program indentation, consistent usage of identifiers, structure, data flow, etc. (Figure 2 on the next page)—very much as in the BOSS system [3]. These style criteria were explained in the course; this same form would later be used by the instructors for the final assessment. Praktomat uses a *blind review process*: the author's identity is disclosed only after the assessment is done.

An important thing to note is that *the final assessment is totally independent from any earlier reviewing.* The instructors never see any review, nor do they learn anything about the review process, nor does the grade depend on sent reviews. Praktomat hides all these details from the instructors, making it a "students only" business.

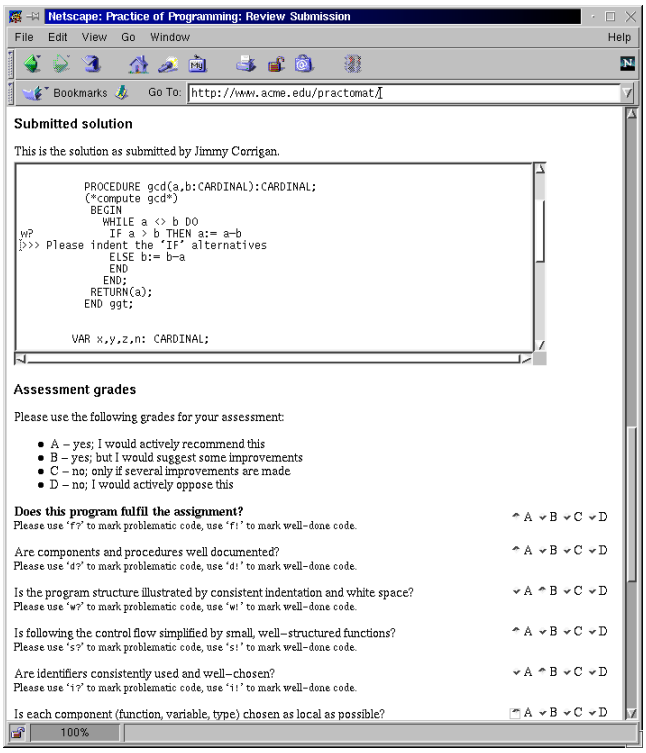

Figure 2: Mutual reviewing in Praktomat

To ensure a maximum of fairness among student reviewers, we implemented a number of *rules* that assign a solution to a reviewer. Whenever a student asks Praktomat for a program to be reviewed, Praktomat follows these simple rules:

Everyone gets his share. Praktomat prefers programs which have received a minimum of reviews so far. This ensures that eventually, every solution is reviewed.

Give, and it shall be given to you. Praktomat prefers programs whose authors have composed a maximum of reviews. This encourages early and multiple reviewing.

Tit for tat. Praktomat prefers programs whose authors have reviewed a program of the requesting student. This means that students review each other *in pairs:* If student A gives an unsubstantiated review to student B, she must fear that B applies the same low standards when sending him her review of A's program.

Opposites attract. Among all remaining solutions, Praktomat chooses the one whose assignment has a *maximum distance* from the reviewer's assignment—and it never chooses a solution below a minimum distance. This discourages plagiarism.

In the beginning of our course, each student who would send in a review was allowed to delay the submission of his final program for another week. It turned out that this incentive was not required: people were so eager to obtain reviews that they competed with each other in submitting a large number of early reviews.

The students took their job seriously: Much to our shame, some students complained that they got more feedback from their reviewers than from the instructors' assessments. During the course, 84 out of 119 students (70.5%) wrote at least one review. These 84 students wrote a total number of 275 reviews—on the average, 3.27 reviews per student.

6 Evaluation and Conclusion

From the student's view, Praktomat usage was a tremendous success. In the official evaluation, students were asked to judge whether some Praktomat usage had improved the quality of their programs:

- 57.7% confirmed the effectiveness of *automatic testing* (and an additional 28.8% confirmed this partially)
- 61.5% confirmed the effectiveness of *reading and reviewing* programs of fellow students (and an additional 19.2% confirmed this partially)
- 63.5% confirmed the effectiveness of *having their programs read and reviewed* by fellow students (and an additional 13.5% confirmed this partially)

The effectiveness of reading and reviewing is also backed by statistical data. Figure 3 on the following page shows the final grades obtained for *program readability*, partitioned by the number of reviews written. (Grades range from 0 "poor" to 3 "very good".) On the average, students who wrote no review at all, had a grade of 1.93; students who wrote one or more reviews reached a grade of 2.18. Figure 3 shows that the grade increases with the number of reviews written.

Receiving reviews improves readability even more than writing reviews. Figure 3 on the next page shows readability grades, partitioned by the number of reviews received. On the average, students who received no review had a grade of 1.82; students who received one or more reviews reached a grade of 2.16. Again, on the average, the grade increases with the number of reviews received.

These results alone would have sufficed to keep up the good work; but what is the instructors' story?

- The initial effort was high. *Creating the assignments and test cases* demanded much more effort than in non-automated settings. This was due to the high precision required for automated testing, but also to the high number of possible configurations. (*Writing Praktomat* also took a great deal of work—but this was a one-time effort.)
- Students and instructors perceived a much higher *fairness* in assessing the students' work. All programs were subjected to the same tests; detailed criteria unified the assessment of programming style.

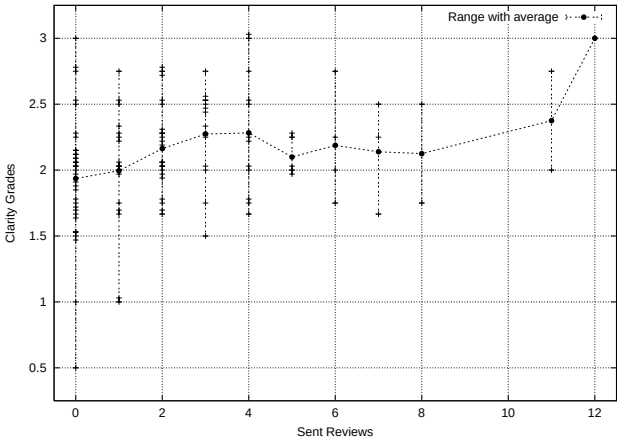

Figure 3: Grades versus *sent* reviews

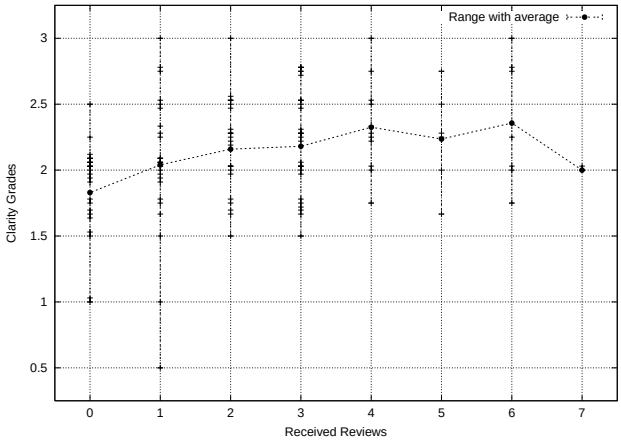

Figure 4: Grades versus *received* reviews

- We found that *automatic testing* considerably simplified assessing the functionality of the program. We also found that the *programming style* improved much faster than in earlier courses—which we assume is a result of mutual reviewing.

Although we found that personalized assignments helped a lot to inhibit plagiarism from the start, we want to enhance Praktomat with means to *detect remaining plagiarism automatically* [1]. Also, we want to introduce a certain *randomness* in test cases to exclude programs tailored to pass the public tests. Finally, we want to plug in a number of *static and dynamic analysis techniques* to ease assessment (and the students' self-improvement) even further.

Altogether, we find that Praktomat killed two birds with one stone: the course management is streamlined, and the students get an initial exposure to industry's best practices. We are convinced that automatic testing and mutual reviewing will help students to improve in all their future programming tasks.

Further information on Praktomat, including the Praktomat source code[1], is available at

http://www.fmi.uni-passau.de/st/praktomat/ .

References

[1] Gitchell, D., and Tran, N. Sim: A utility for detecting similarity in computer programs. In *Proceedings of the 4th Annual Conference on Innovation and Technology in Computer Science Education—ITiCSE* (Cracow, Poland, 1999), pp. 266–269.

[2] Isaacson, P. C., and Scott, T. A. Automating the execution of student programs. *ACM SIGCSE Bulletin 21*, 2 (1989), 15–22.

[3] Joy, M., and Luck, M. Effective electronic marking for online assessment. In *Proceedings of the 3rd Annual Conference on Integrating Technology into Computer Science Education—ITiCSE* (Dublin, Ireland, 1998), pp. 134–138.

[4] Libes, D. Regression testing and conformance testing interactive programs. In *Proceedings of the Summer 1992 USENIX Conference* (San Antonio, Texas, June 1992). Distributed with DejaGnu.

[5] Reek, K. A. The *try* system—or how to avoid testing student programs. *ACM SIGCSE Bulletin 21*, 1 (1989), 112–116.

[6] Savoye, R. *The DejaGnu Testing Framework*. Free Software Foundation, Inc., Jan. 1996. Distributed with DejaGnu.

[7] van Rossum, G. *Python Tutorial*, 1.5.2 ed. www.python.org, Apr. 1999.

[1] Right now, Praktomat is available in German language only. Please contact us for internationalized versions.