Assessment Process for Programming Assignments

Kirsti Ala-Mutka, Hannu-Matti Järvinen
Tampere University of Technology
Institute of Software Systems
{kirsti.ala-mutka, hannu-matti.jarvinen}@tut.fi

Abstract

Assessment provides means for influencing students' learning results and strategies on university courses. However, supporting students to learn practical programming skills requires carefully designed assignments and their assessment. This paper studies challenges and approaches for assessing homework programming assignments. We present and discuss a semi-automatic assessment system for supporting programming assignments on large courses.

1. Introduction

With the rapid evolution of computers and information technology, computer science has gained a significant role in the technology education. The basics of computer science is needed in several curricula. Programming is an essential part of computer science and usually incorporated already to the introductory studies. However, the practical programming skills have proved to be difficult to learn and teach. For example, McCracken et al. [6] showed that 1st year computer science students were seriously lacking programming abilities in all the universities involved in the study.

Programming abilities are usually assessed by practical programming tasks. Typically, these are either homework assignments or laboratory examinations in controlled circumstances. Laboratory examinations provide means for ensuring the personal skills of the students, as opposed to the possibilities for cheating with homework assignments. On the other hand, laboratory programming tasks need to be very small because of limited amount of time. Homework assignments provide possibilities for more creative tasks that illustrate the complexities of programming better.

This paper gathers together viewpoints relating to the assessment of programming assignments. First, we introduce some typical problems faced by students and teachers and review existing assessment practices for programming. Then, we present our semi-automatic coursework assessment process for supporting both students and teachers on large courses. Finally, we discuss the process model and conclude the paper.

2. Challenges on programming courses

Robins et al. [7] gathered together problems of novice programmers in their survey on educational research on programming education. Students have often difficulties in building mental model of computer programs, since it differs from the structure of natural language. Even when the students have learned the programming concepts and languages, they may still lack the skills for using this knowledge to create computer programs. Thus, if the students are expected to learn to generate computer programs, it requires "hands-on" experience with practical programming tasks.

Previously, practical work was partly covered by voluntary exercises. However, we have noticed that nowadays more and more students try to minimize their workload on the courses. For this reason, practical programming cannot be left as a voluntary task. Practical programming needs to be required and assessed during the course, to ensure the gradual development of the practical skills of the students. A similar phenomenon has also been reported by Woit and Mason [10]. Their experiences showed that students did not work on voluntary assignments. Getting the students involved with the practical components of the course required frequent (online) evaluations. They proposed that the students should either frequently submit laboratory assignments or be required to answer weekly quizzes about the contents of the assignments.

A possible reason for the unmotivation is that students sometimes see programming assignments as separate tasks with unnecessarily complex assessment requirements. Real-world applications and software projects are so large that they cannot be covered on one or even several courses. Still, the students should learn skills for working in such situations. Therefore, the complexities and practices of professional work must be introduced partly in theory and partly by assignments that are simplified from real-world systems. For teachers, this means that they need to plan the assignments very carefully. For students, this means that they are required to learn and follow several basic rules, although the effects of their neglections cannot always be shown in practice.

The complex requirements of good and correct programming practices make the assignments hard to assess. Novice programmers are usually not very good at evaluating their work, as even incorrect programs can seem to work as desired. Also if the students have not yet learned the issues of “good programming”, they cannot assess them effectively either. For this reason, assessment and feedback by an expert is always needed. This places heavy workload for teachers. In addition to the work needed for giving good feedback, also the issues of assessment objectivity, consistency and speed are hard to take care of. These problems become emphasized on large courses, where several tutors are needed for the assessment work.

Another widely recognized problem is cheating. Since computer programs are in electronic form, they are easy to copy. Sheard et al. [9] had concerning results in their study of IT students’ attitudes to cheating at two universities. 34%/28% of the respondents admitted that they had copied a majority of an assignment from a friend. 53%/42% had collaborated on an assignment that was meant to be completed individually. This fact needs to be taken into account for ensuring students’ learning. Woit and Mason [10] used online quizzes and assignments to reduce cheating. However, on courses with emphasis on larger (homework) assignments their solution is difficult to apply.

3. Assessing programming assignments

With programming assignments, the performance of the students is generally assessed by the resulting programs. Thus, the first step to design the assessment is to plan the objectives of the assignment. Especially on the first programming courses, it is important to analytically assess all the essential requirements to ensure the learning of the basic issues. These are the elements that remain the same troughout the courses. On advanced courses, special attention is paid also to the specific goals of the course, for example, data structure design.

No common basic measurement criteria for programming assignments can be found from the literature. Typically, correct functionality, good design, and good programming style are required. However, the definitions and relative weights of these features in the assessment vary greatly. Different teachers emphasize different features, basing their criteria on their personal experience and sense of course objectives. This suggests that the assessment criteria should be clearly specified, especially, if there are several teachers assessing the coursework on a course. Becker [2] has proposed and given examples of using rubrics for assessing programming assignments. She used two rubrics, one for general style and design and another for assignment specific elements. When also students are provided with clearly stated objectives and assessment criteria, they are able to control their learning process and become more self-directed learners.

Lister and Leaney [5] classify assessment on a programming course according to the cognitive levels of the Taxonomy on Educational Objectives [3], i.e. knowledge, comprehension, application, analysis, synthesis and evaluation. Grades are assigned according to these levels to form the basis for assessment design. They strongly object the norm-referenced approach, where grades are decided according to the student score distribution. In their approach, the emphasis is on the assignment design that can emphasize, e.g., analysis (debugging) or synthesis (program design and construction) skills. The actual assessment of the resulting programs may be similar on different levels.

In addition to summative grading purposes, the assessment can be formative, to support both students’ learning and teachers’ work. Formative evaluation for programming assignments can be provided by offering students feedback and a possibility to improve their coursework without effects on the final grade. This also provides teachers a possibility to follow the students’ progress and learning, and to use this knowledge for improving the instruction.

4. Student-centered assessment process

Since programming assignments are executable, electronic products, their evaluation is possible via other computer programs. With some features, e.g., program functionality, computer usage is essential, since these are very difficult to analyze statically from the program source code. Several different automatic evaluation criteria have been introduced in systems like CourseMarker [4]. The obvious benefits of automatic assessment are the objectivity, consistency and speed of assessment, together with 24h service. Also the assignment descriptions and measurement criteria are carefully designed by necessity, since they have to be programmed to the automaton.

On large courses, providing feedback on several programming assignments requires automatic assistance. Otherwise the workload would simply be unmanageable. Unfortunately, computers have limited capabilities for providing feedback, i.e. they can evaluate only issues with measurable definitions and provide only pre-determined feedback messages. To provide students also with more descriptive feedback on more complex issues, the automatic feedback can be supplemented with instructor feedback.

Our approach uses this kind of a semiautomatic model for supporting students formatively during the programming process and for assisting teachers in the final summative assessment. This approach is primarily aimed at homework assignments with similar minimum requirements for all students.

4.1. Description of the process

The basic components of the coursework process are presented in Figure 1. The first elements of the coursework are the basic information sources offered to students. The assignment descriptions and assessment principles are specified in a detailed form (since these are prepared for automatic assessment). These specifications are published for students to guide their work with the assignment. E.g., these may include the assessment configuration files for the tools, commented descriptively to help students' understanding.

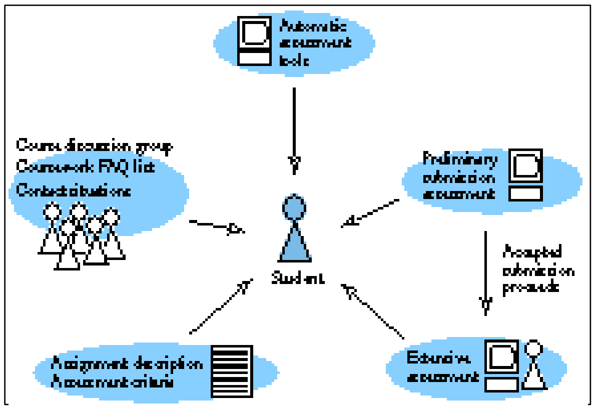

Figure 1. Student's view to the assessment

Students are not left alone to work with computers and definitions. There are also contact education situations, tutor appointment hours, and electronic discussion groups for discussing the assignments and assessment. In addition, the students are provided with a list of frequently asked questions (FAQ) on a web page maintained by tutors.

Brief descriptions of the developed automatic assessment tools have been presented in [1]. These tools assess, for example, program compilation, structural deficiencies, dynamic behavior (output, memory management, efficiency), and compatibility with the required coding guidelines. Students have unlimited access to the automatic tools to check whether they have met the automatically measured requirements. The tools provide both numerical and verbal feedback on the basic programming issues as well as problem specific issues. Teachers can tailor the assessment criteria and feedback messages according to the objectives of the assignment.

Students can submit their work as soon they think it is finished. The system uses the same automatic tools that have been available for students to check that all basic requirements are met. If this is the case, the program proceeds to further evaluation to the tutor. If the program fails in some of the requirements, the student receives immediately knowledge and automatic feedback about the failed issues.

After student's submission has passed the preliminary (automatic) assessment, tutors assess more advanced issues, such as advanced program design aspects and more specific dynamic behavior. If the program has deficiencies in these issues, the tutor asks for resubmission. Also the resubmitted assignments must pass the preliminary inspection.

4.2. Feedback and experiences

The presented process has been used on several courses during 2000-2004. Feedback from students has been collected with 7 different paper and web-based questionnaires (N=54–513, altogether 1115 responses) as well as in personal interviews. A web-based survey and personal interviews have also been carried out for altogether 20 tutors and teachers. In addition, trends and development needs have been recognized by following and classifying messages from course discussion groups. This section sums up the experiences and feedback of both students and teachers on using the presented process model.

Generally, the clear assignment and assessment specifications have received good feedback. However, students do not always understand the wordings used in the instructions and feedback. Some students use the FAQ lists effectively, but in teachers' opinion, all students do not utilize them as much as they could. Also the automatic verification of basic requirements causes sometimes complaints from the students. On the other hand, these bring more opportunities to discuss the assessment criteria and their relevance with the students. Course discussion groups are used actively for discussing coursework related issues.

According to the surveys, the students appreciate the quick feedback received from the automatic tools, but wish more detailed descriptions of the errors in the program. The tools list and approximately point the problems in the programs, but it is left for the students to analyze and correct the errors. This is an intended feature, since analysis and evaluation skills are an integral part of programming. The technical implementation of the coursework process has received mainly positive feedback but some usability issues have been recognized for improvement.

Some clear correlations have been recognized from the students' answers. The students that considered the coursework most educational were the ones that most carefully prepared and tested their assignments before invoking any automatic tools. The students that complained receiving too little or bad feedback, commonly began working on the assignment only few days before deadline, and tested their programs poorly before submission attempts. Surveys on different courses also show that the more the students have used the tools, the more they appreciate them as learning aids. These suggest that students with independent learning skills or already some experience in programming, benefit the most from this system.

Tutors and teachers consider the clarity and readability of the submitted programs as the most important benefits of this coursework process. These help teachers to understand the structure and design of the student's solution better. All tutors agreed that without the automated preliminary inspection phase it would not be possible to evaluate this amount of practical programming assignments. When the submitted assignments are known to fulfill all the basic requirements, the tutors can concentrate on giving feedback on more advanced and course-specific issues. They perceive that the total amount, variety and quality of feedback given to students have improved with this model.

4.3. Lessons learned

These lessons are collected mainly from the data structures and algorithms course that has been using the system for several years. The course is aimed at students, who already have the basic knowledge of programming concepts. The quality of the practical coursework is an important goal on the course.

One of the most obvious benefits of this system is the capability of checking the format and contents of the submission. When these issues are noticed immediately, students know to correct and resubmit their work without waiting for tutor's response. Previously, with email-based submission, approximately 20% of all submissions had format, content, or compilation problems. In these cases, the tutor could not assess the program and had to wait for resubmission.

When basic issues are tirelessly assessed by automatic tools, all students have to take them into account in their programming coursework. In fact, with automatic evaluation it is possible to prohibit submissions with certain typical errors. For example, dynamic memory management neglections have completely disappeared from the students' assignments. Rechecking some of the previously accepted assignments showed that many of them contained these errors, because at that time these could not be systematically assessed. Now all students are faced with these issues consistently and coursework quality has generally improved.

Logfiles from an assignment (year 2003) with 240 students show that 38% of the students received automatic resubmission requests in the preliminary inspection phase. 82% of all students passed the preliminary inspection without any or at most two resubmission requests, while 7% iterated their submission six or more times. In fact, this 7% fraction of the students caused 50% of all the resubmission requests in the preliminary inspection phase. Closer examination reveals that these students often did their resubmissions only few minutes after another and performed only little changes to their program between the attempts. Also other assignments show similar phenomenon.

5. Discussion

The basic novice programmer types, categorized by Robins et al. [7] as stoppers, movers and tinkerers, are recognized also in our process model. Movers learn from the feedback and utilize it in developing their program. Stoppers stop after meeting difficulties and cannot go forward without special assistance. In this case, instead of working on the assignments, these people often keep criticizing the tools and the feedback. Tinkerers use trial-and-error tactics, without really considering the problem and often not proceeding very much in their work. The learning of stoppers can be supported with personal assistance and on discussion groups, but the tinkerers do not even want to use or get advice. Their working strategies could be guided by, e.g., limiting the number of submission attempts or requiring minimum time between the submissions.

The "work avoidance" strategy of some students appears also in our system. These students try to leave all the evaluation work to the automatic tools without considering the quality of their work by themselves. Involving students more in program evaluation could be achieved by peer evaluation practices. Supplementing automatic and teacher feedback with peer evaluation would ensure students both with more descriptive feedback and personal evaluation experiences. Rust et al. [8] had good results in coursework process with an intervention, where students had to assess their own and model solutions according to given assessment criteria. Both the understanding of assessment principles and the self-evaluation skills of the students were improved. Hence, also coursework quality improved. Although this was not a programming subject, a similar approach could be applied to programming courses.

The presented process does not limit the settings of assignments. It can be used both for personal and group assignments as well as for homework assignments or laboratory examinations. The size of assignments as well as the assessment tools and criteria can be freely chosen. The variety of supported assignments types depends on the assessment tools available. E.g., graphical user interface evaluation is not possible at the moment, but by developing a tool for simple GUI assessment, also this kind of programs could be used.

This model could be used for programming assignment that emphasize different cognitive levels, following the design ideas of Lister and Leaney [5]. However, the teachers have mainly used only program construction tasks for ensuring practical program creation skills. A more versatile assignment design would support gradual progress towards highest cognitive levels. When students master well practical programming skills on, e.g., analysis level (debugging), they have less problems on synthesis level, i.e. in designing and implementing programs. Also the program evaluation skills could be more explicitly assessed, e.g., with an assignment to select code components for producing a high quality program with a complete test data set.

The presented model does not contain special means for preventing cheating. However, it provides information for inspecting suspected plagiarism. We have a comparison tool to check students' submissions for plagiarism after deadline. The saved submission histories and program versions provide a means to prove whether plagiarism has taken place. Students sometimes try to submit somebody else's final version as their own submission, which creates odd development paths to the version history.

6. Conclusion

On a programming course, getting students involved in actual programming requires compulsory practical assignments. Clear objective definitions and assessment criteria increase the impact and benefits of the assessment both for students and teachers. Versatility in the assessment design provides students better possibilities to develop their practical skills relating to different cognitive levels.

The possibility for improving the programming assignment after receiving feedback enhances learning results and supports constructive learning. On large courses, this requires automatic tools for giving the feedback. Automation of assessment can be used for consistently and quickly ensuring that all students achieve the specified minimum requirements. Especially self-directed learners can use automatic tools efficiently to support their learning. By combining both automatic and teacher-given assessment, students can be provided with versatile feedback. Logfiles of such automatic feedback systems can also provide useful information for improving the instruction.

7. References

[1] Ala-Mutka, K. Computer-assisted Software Engineering Courses. *Proc. CATE2002*, May 20–22, 2002, Cancun, Mexico, Acta Press, pp. 111–116.

[2] Becker, K. Grading Programming Assignments Using Rubrics. ACM SIGCSE Bulletin, 35(3), 2003, p. 253.

[3] Bloom, B.S. (ed.) *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I. Cognitive Domain*, McKey, New York, 1956.

[4] Higgins, C., Hergazy, T., Symeonidis, P., and Tsinsifas, A. The CourseMarker CBA System: Improvements over Ceilidh, *Education and Information Technologies*, 8(3), 2003, pp. 287–30.

[5] Lister, R. and Leaney, J. (2003). First Year Programming: Let All the Flowers Bloom. *Proc. ACE2003, Adelaide, Australia*, 2003, ACS, pp. 221–230.

[6] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students, ACM *SIGCSE Bulletin*, 33(4), 2003, pp. 125–180.

[7] Robins, A., Rountree, J., and Rountree, N. Learning and Teaching Programming: A Review and Discussion, *Computer Science Education*, 13(2), 2003, pp. 137–172.

[8] Rust, C., Price, M., and O'Donovan, B. Improving Students' Learning by Developing their Understanding of Assessment Criteria and Processes. *Assessment & Evaluation in Higher Education*, 28(2), 2003, pp. 147–164.

[9] Sheard, J., Dick, M., and Markham, S. Cheating and Plagiarism: Perceptions and Practices of First Year IT Students. *Proc. ITICSE'02,* June 24–26, Aarhus,Denmark,pp.183–187.

[10] Woit, D. and Mason, D. Effectiveness of Online Assessment. *Proc. SIGCSE'03*, February 19–23, 2003, Reno, Nevada, USA, ACM Press, pp. 137–141.