Assessing Process and Product
– A Practical Lab Exam for an Introductory Programming Course

Jens Bennedsen[1] and Michael E. Caspersen[2]

Abstract - The final assessment of a course must reflect its goals, and contents. An important goal of our introductory programming course is that the students learn a systematic approach for the development of computer programs. Having the programming process as learning objective naturally raises the question how to include this in assessments. Traditional assessments (e.g. oral, written, or multiple choice) are unsuitable to test the programming process.

We describe and evaluate a practical lab examination that assesses the students' programming process as well as the developed programs. The evaluation is performed in two ways: By analyzing the results of two lab examinations (with more than 500 students) and by semi-structured individual interviews with representatives of the involved persons (students, TAs, lecturer, and examiner).

The result of the evaluation is encouraging and indicates the value of alignment and strong conformity between goal, content and assessment of the introductory programming course.

Index Terms – CS1, Examination, Evaluation, Programming Process, Objects-First, Pedagogy.

INTRODUCTION

The final assessment must reflect aims, goals, and contents of a course [1].

An important goal of our introductory programming course is that the students learn a systematic approach to the development of computer programs. Learning a systematic approach to programming implies that the students must gain a clear understanding of the programming process and the activities that are part of this process. They must also develop the ability to apply these to develop programs.

Recognizing the importance of programming techniques and the programming process when designing a programming course implies the need for adoption of a suitable assessment form. Traditional assessment forms (e.g. oral or written examinations, multiple choice questions) are unsuitable to test the programming process.

Another equally important argument for assessing the programming process is that "The spirit and style of student assessment defines de facto the curriculum" [2][p.1]. Ramsden makes a similar observation: "the type of grading influences the student's learning approach" [3].

The bottom line is that it is essential to apply an evaluation form where the students demonstrate their practical programming skills as well as their understanding of the fundamental concepts and theories from the curriculum of the course. Consequently, we need to develop a new type of assessment suitable to test the programming process as well as the product.

The lab examination described and evaluated in this paper has as characteristics that it

i. provides a valid and accurate evaluation of the student's programming capabilities,
ii. evaluates the process as well as the product,
iii. encourages the students to practice programming throughout the course, and
iv. can be used assess 120-140 students pr. day.

The rest of the paper is structured as follows: Section 2 describes the context of the lab examination. Section 3 gives a more thorough description of the final lab examination. Section 4 presents and discusses the findings from the evaluation of the lab examination. In section 5 we discuss related and future work. The conclusions are drawn in section 6.

GOALS, CONTENT AND ASSESMENT

To provide an understanding of the context, this section describes goal, form, and content of the introductory programming course.

General Information

Our programming course spans the first half of CS1 at University of Aarhus. The course runs for seven weeks, and after the course there is a lab examination with a binary pass/fail grading.

The grading is based solely upon the behaviour in and result of the final examination; acceptable performance during the course is a prerequisite for the final exam but does not count as part of the grading.

There are approximately 250 students per year from a variety of study programmes, e.g. computer science, mathematics, geology, nano science, economy, multimedia. 40% of the students are majors in computer science, and they are the only group of students that continue with the second half of CS1. The rest of the students proceed to other programming courses related to their fields (e.g. multimedia

[1] Jens Benendsen, IT University West, Fuglsangs Alle 20, DK- 8210 Aarhus V, Denmark, jbb@it-vest.dk
[2] Michael E. Caspersen, Department of Computer Science, University of Aarhus, DK-8200 Aarhus N, Denmark, mec@daimi.au.dk

1-4244-0257-3/06/$20.00 © 2006 IEEE **October 28 – 31, 2006, San Diego, CA**

programming, scientific computing) if they proceed with programming at all.

The students are grouped in teams of 18-20 students; typically there are 13-14 teams per year. Each team has its own teaching assistant (TA) – a PhD or MSc student.

Goals

The purpose of the course is that students learn the foundation of systematic construction of simple programs and through this obtain knowledge about the role of conceptual modelling in object-oriented programming. Furthermore, it is the goal that students become familiar with a modern programming language, fundamental programming language concepts, and selected class libraries.

After the course the students must be able to explain and use fundamental elements in a modern programming language, use conceptual modelling in relation to preparing simple object-oriented programs, implement simple object-oriented models in a modern programming language, and use selected class libraries.

Form

The course runs for seven weeks; every week there are four lecture hours and one lab hour plus three class hours with a TA. In addition to the scheduled hours, students work approximately seven hours per week in study groups or on their own.

The four lecture hours per week are used for presentation and discussion of general concepts and the programming process. The programming process is revealed through live programming in front of the students in the lecture theatre using computer and projector and through process recordings (narrated, screen-captured video recordings of program development sessions), see [4].

Every week (except for the first) there is a mandatory assignment that must be submitted to the TA. The TA examines the assignments and gives personal as well as collective feedback to the students. Approval of five out of six weekly assignments is a prerequisite for the final exam but does not count as part of the grading. The weekly assignments are primarily used to keep the students up to the mark on the practice of programming.

Content

The course content is fundamental programming language concepts, object-orientation, and techniques for systematic construction of simple programs.

- **Fundamental programming language concepts**: variable, value, type, expression, object, class, encapsulation, control structure, method/procedure, recursion, type hierarchies.
- **Object-orientation**: modelling; class structures (specialization, aggregation and association); use of selected class libraries (in particular collection libraries), interfaces and abstract classes.
- **Systematic development of small programs**: modularization, stepwise refinement/incremental development, test.

This is a logical listing of the course contents; it is *not* the order in which the content is covered. The content is covered using a spiral approach [5]; for further details of the structure and content of the course, see [6, 7].

ASSESSMENT THROUGH A LAB EXAMINATION

This section discusses the examination requirements, the organization of the lab examination and the actual lab examination.

Conformity between Goals, Content, and Assessment

As mentioned in section "Goals", the goals of the course are that the student must be able to explain and

- *use* fundamental elements in a modern programming language,
- *use* conceptual modelling in relation to preparing simple object-oriented programs,
- *implement* simple object-oriented models in a modern programming language, and
- *use* selected class libraries.

During the course, as in real life, programs are developed using a standard development environment running on a computer. An ordinary written exam with pen and paper is an artificial situation and therefore insufficient and inappropriate to test the student's ability to develop programs. For the same reasons an ordinary oral examination and a multiple choice test would be inappropriate.

To ensure alignment and maximum conformity between goals, content, and assessment we have designed a practical examination organized in a lab.

Organization of the Lab Examination

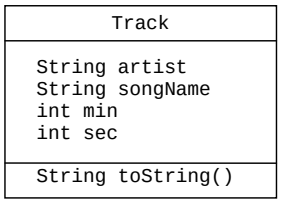
The examination resembles an ordinary lab session. 20 students are tested concurrently.

We schedule one hour per group of 20 students, but only 30 minutes for the actual lab examination. The rest of the time is used for administrative activities and as buffer.

Each group of students receives a different assignment consisting of nine small progressive programming tasks. In principle the assignments are identical (they are all instances of the same generic assignment), but the students does not know nor realize this. The similarity of the assignments is important for fairness as well as comparability of the students' results. The sample assignment in Figure 1 deals with tracks and play lists; other exercises concern luggage and flights, employees and departments, museums and paintings, etc. Although the concepts modelled by the classes vary, the assignments have similar structure.

Lab Exam Exercise (30-minute exam)

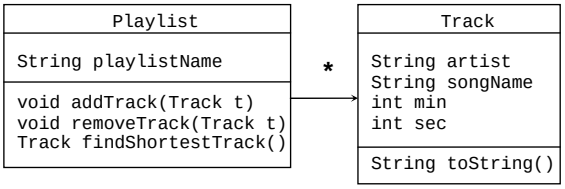
1. Create a class, *Track*, that represents a piece of music; the *Track* class is specified in the following UML diagram.



   The four field variables must be initialized in a constructor (through four parameters of suitable types). The method *toString* must return a string representation for a piece of music, e.g.

   `"Yesterday: The Beatles (2:05)"`

2. Create a test method named *exam* in class *Driver*. The method must be static, have return type void, and have no parameters.
3. Create two *Track* objects in the exam method using object references *t1* and *t2*; print the two *Track* objects using the *toString* method.
4. Create a new class, *Playlist*, representing a collection of *Track*s; the *Playlist* class and its relation to the *Track* class is specified in the following UML diagram:



5. Implement the method *addTrack* (and *removeTrack*) so that it adds (removes) the object *t* to (from) the *Playlist* object.
6. Create a *Playlist* object in the *exam* method in the *Driver* class; associate the two existing *Track* objects with the *Playlist* object.
7. Implement the method *findShortestTrack*. The method must return a shortest (measured in playing time) *Track* object from a *Playlist* object. You can assume a non-empty *Playlist* object. In other words, you need not worry about the playlist being empty.
8. Use methods *findShortestTrack* (from class *Playlist*) and *toString* (from class *Track*) to print the shorter of the two *Track* objects created in task 3.
9. Let the *Track* class implement the *Comparable* interface. The natural order of *Track* objects is defined by the length of the song.

Figure 1: *Sample Lab Exam Exercise*

Assessment of Product and Process

In test for completed apprenticeship of traditional crafts, the examiner inspects the apprentice while they construct their exam product; the quality of the apprentice's construction process as well as the quality of the final product counts in the final grading.

Because of similar goals regarding the assessment of process and product, we have adopted a similar examination form where the lecturer and the external examiner evaluate the programming process as well as the program produced by each student by inspecting the students during the examination.

To avoid practical problems during start-up and finalization of the lab examination (e.g. login problems, applying naming conventions, delivery of the exam products), and to ensure that minor unimportant programming errors, tool problems, etc. does not hinder the student's problem solving and programming, five TAs are present during the lab examination to support the students. If the TAs have doubts about their role (e.g. how much to interact with the students), they consult the lecturer or external examiner on-the-fly.

To let the students settle down and get started, they are not inspected until they have passed a checkpoint after the first three programming tasks. The students are instructed to call upon a TA or the lecturer when they reach the checkpoint to show and demonstrate their solution. When a student has passed the checkpoint, the lecturer and external examiner start inspecting the student's behaviour. The poorest students never reach the checkpoint i.e. the inspection time is focused on those students who have a chance of passing.

The examiner and lecturer note the time when the first three tasks are done. After five to seven minutes, they start inspecting the process of each student; around that time, and after a short inspection of the students programming process, it is usually possible to determine the pass grade. This is a very efficient way to know when and in what order to look at the students' solutions. This is also a method to ensure that the students have some silence and can concentrate during the exam.

To allow for efficient inspection, the students are instructed to keep all editor windows open and tiled on the screen.

The students' behaviour as well as the quality of the programs they produce count in the final grading but not on equal footing. An appropriate and systematic programming process can compensate for minor flaws and errors in the product and result in a pass mark for the student, and similarly a poor process can be the determining factor when the product is on the edge. Although we emphasize the programming process, it is not the case that a nice product will be turned down due to a poor process (which is unlikely anyway).

EVALUATION

In this section, we present and discuss an evaluation of the lab examination described above.

Evaluation Method

The evaluation of the lab exam was performed in two ways: By analyzing the results of three consecutive lab examinations (2003, 2004 and 2005) and by semi-structured

individual interviews with students, TAs, the examiner, and the lecturer.

Quantitative Evaluation

For each of the three years we have collected data about the students for four variables (and two derived). The description of the variables can be found in Table 1.

Variable	Description
students	students enrolled for the course
abort	students that aborted the course before the final exam
exam	students allowed to take the final exam
skip	students that did not show up for the final exam but was allowed to
fail	students who failed the final exam
pass	students who passed the final exam

Table 1: *Description of type of data*

The numbers in table 1 are related as follows:

students = *abort* + *exam*
exam = *skip* + *fail* + *pass*

From these numbers we calculate *exam rate*, *pass rate* and *retention rate* (*exam*/*students*, *pass*/*exam*, *pass*/*students*). The results are presented in Table 2.

	2003	2004	2005
students	276	220	295
abort	63	26	28
exam	213	194	267
exam rate	77.2 %	88.2 %	90.5 %
skip	13	5	3
fail	15	19	29
pass	185	170	235
pass rate	86.9 %	87.6 %	88.0 %
retention rate	67.0 %	77.3 %	79.7 %

Table 2: *Statistics from three years of practical lab exams*

The figures in Table 2 reveals two interesting aspects: the improved exam rate (and retention rate) from 2003 to the following years, and the high pass rate in general.

The curriculum was radically redesigned in 2003 going from a semester structure to a quarter structure; consequently the traditional CS1 course was split in two courses with an exam in between. The students of 2003 were the first to take the new course with the new examination form, and therefore there where no tradition for the students to lean on. In the following years (2004-2005) the students have had the old exam questions to use for practice, and older students to hear war stories from. In the following years the lecturer could be more explicit when describing the requirements for the exam and the exam form. We believe that this is the primary reason for the improved exam rate.

The pass rate is high compared to what others report [8, 9]. We believe that this primarily is due to the alignment and the strong conformity between goal, content and assessment of the course.

Qualitative Evaluation

The semi-structured interviews were conducted two to three weeks after the final exam. Ten students were selected to get a mixture of major and gender. One interviewer conducted each interview. The interviews were audio taped for later analysis. The interviews followed an interview guide focusing on three topics: The lab exam form in general, this specific exam, and the evaluation form compared to other evaluation forms. In the analysis that follows, quotations from the interviews are presented that describes the general attitude of the group. The interviews were done in Danish, and the quotations translated into English by the authors.

The Students

There was a very little difference in the way that the interviewed students had experienced the lab exam; their answers were largely similar. We find therefore that the students are representative of the general attitude towards the exam, although we cannot be sure.

All of the interviewed students found the evaluation form fair. They defined *fair* as "*if you have practiced during the course, you can expect to pass the exam*". They all found that the form and content of the exercise was very adequate with respect to the goals of the course. As one student noticed: "*Programming requires very abstract thinking, but it is also a craft ... the examination form perfectly suits this mixture.*"

One of the students did not like that a TA was looking over her shoulder. She felt insecure and nervous. However, she was the only one having this experience – no one else minded having the TAs around (some even found their presence to give more peace of mind).

The examination incited the students to practice programming. As an option for the students, exam exercises from the previous year were available for preparation for the exam. As one student replied when asked about his preparations, "*I solved all the [old] exam exercises*".

Students were instructed to call the TA after solving the first three tasks of the exercise (Figure 1) to demonstrate what they had achieved. None of the students found this to be problematic, but some of them pointed to the possible problem, that the slow students might feel this as an extra stress factor (knowing that many of the other students have finished). In conclusion, only one of the interviewed students felt the examination to be stressful.

All of the interviewed students felt that a more fine-grained marking could take place, but it would require more time and more tasks. Most thought that one hour would be sufficient for this.

The Teaching Assistants

The interviews with the teaching assistants in many ways supported the statements from the students. They also found the exam to be fair and had the impression that it evaluates the students programming skills.

In the beginning, the TAs had some difficulties knowing to what extent they could answer questions. During the exam,

the TAs developed a practice: they helped a student who had spent several minutes trying to figure out a simple problem, but did not help with problems that were more fundamental. If in doubt, the TAs asked the lecturer or examiner. Apart from this, they did not feel uncomfortable with they role.

The Lecturer and the External Examiner

Both the lecturer and the examiner found the exam form to be both fair and evaluating the learning objectives of the course. The external examiner found that the exam evaluated the student's understanding of the general concepts although it was impossible to evaluate that the student was "*able to explain [...] fundamental elements in a modern programming language*". They found that it was easy to assess an objective pass/fail criterion due to the generic exercises. The examiner thought that a little longer time would give an even better evaluation criterion.

The examination gave a good impression of the students programming skills including their programming process. As the examiner said: "*When you get an error message from the compiler you must be able to figure out what is wrong … that is a part of a practical programming skill*".

Concluding the Evaluation

The exam tests the process as well as the product. In some cases the process was the decisive factor. One special example of this was a student that was ill and therefore worked very slowly; however slow, her programming process was very good demonstrating a systematic approach to solving the problems.

The evaluation indicates that the lab examination supports the learning objective of the course. The students and the lecturer/examiner consider the lab examination fair. The assessment does not require many resources: 250 students can be handled using less than 90 person-hours.

Low retention is one of the main problems in CS1 courses. As noticed by [10][p.40] their retention "has been around 50%". In this course, the retention is around 75%. We have found that the examination form kept the students up to the mark; they did actually practice programming. We think this is one of the explanations of the relatively high retention rate.

For computer science students the examination form must be seen in conjunction with the examination form of the following course (the second part of CS1), which is an oral examination focusing more on the conceptual aspects of introductory programming. There is a progression from the first exam to the next, from testing practice to testing conceptual knowledge.

RELATED AND FUTURE WORK

Recently, a growing number of papers reporting on laboratory exams for introductory programming courses have been published [11-15]. All report good results using this apparently novel assessment form. However, a common characteristic of the assessment methods presented in these articles, and a deficiency compared to the method described herein, is that the evaluation and grading is based solely upon the end product, the students' final solutions.

In [12] the authors describe the grading in their lab final (their word for lab exam): "*Grading on the exam is focused on working programs*". Only the result of the process is evaluated, not the process. Barros [11][p.18] report on the use of lab exams during the course, but the final exam is a traditional written exam. The "*rationale behind maintaining code written in the final exam was to evaluate the students in an environment where trial and error is simply not possible*". Again, they do not include an evaluation of the programming process in their lab exam; the focus is on the final product only.

Focus on the programming process during the course is very important. We are currently investigating the idea of having the students supply information about their programming process (in the form of a screen capture of a programming session) and include this as part of their weekly, mandatory assignment. We expect this information to be valuable and useful for the TAs and the lecturer in order to provide feedback on the process as well as the product, and in general to improve the ability to address the actual needs of the students.

CONCLUSION

We have described and evaluated a lab exam which has a number of advantages. It is simple to evaluate the student's programming process as well as the product (the result of the student's efforts). It is a fair and effective exam. We use standardized exercises that each covers more than 80% of the curriculum. The environment for the exam is the normal daily work environment. It is a lightweight exam easy to prepare and carry out. It requires a couple of days to prepare the exercises for the exam, and we had a throughput of 100 students per day. Everyone involved, in particular the students, regard form as well as content of the exam to be very good and in excellent correspondence with the learning objectives of the course.

ACKNOWLEDGEMENT

It is a pleasure to thank Gudmund Frandsen for valuable comments during development and practice of the lab exam described and evaluated in this paper. We will also like to thank the students and TAs who participated in the interviews. A special thank to Michael Kölling for valuable comments on an earlier version of this article.

REFERENCES

[1] J. C. Prior and R. Lister, "The backwash effect on SQL skills grading," in *ITiCSE '04: Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2004, pp. 32-36.

[2] D. Rowntree, *Assessing Students. how Shall we Know them?* , vol. rev. ed., repr., London: Kogan Page, 1988,

[3] P. Ramsden, *Learning to Teach in Higher Education.* London: Routledge, 1992,

[4] J. Bennedsen and M. E. Caspersen, "Revealing the programming process," in *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 2005, pp. 186-190.

[5] J. Bergin. Fourteen pedagogical patterns. Available: http://csis.pace.edu/~bergin/PedPat1.3.html

[6] J. Bennedsen and M. E. Caspersen, "Programming in context: A model-first approach to CS1," in *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004, pp. 477-481.

[7] M. E. Caspersen and H. B. Christensen, "Here, there and everywhere - on the recurring use of turtle graphics in CS1," in *ACSE '00: Proceedings of the Australasian Conference on Computing Education*, 2000, pp. 34-40.

[8] R. Andersson and T. Roxå , "Encouraging students in large classes," in *SIGCSE '00: Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 176-179.

[9] J. Börstler, T. Johansson and M. Nordström, "Teaching OO concepts - a case study using CRC-cards and BlueJ," in *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*, 2002, pp. T2G-1-T2G-6.

[10] A. N. Kumar, "The effect of closed labs in computer science I: an assessment," *J. Comput. Small Coll.*, vol. 18, pp. 40-48, 2003.

[11] J. P. Barros, L. Estevens, R. Dias, R. Pais and E. Soeiro, "Using lab exams to ensure programming practice in an introductory programming course," in *ITiCSE '03: Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, 2003, pp. 16-20.

[12] M. E. Califf and M. Goodwin, "Testing skills and knowledge: Introducing a laboratory exam in CS1," in *SIGCSE '02: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, 2002, pp. 217-221.

[13] A. T. Chamillard and K. A. Braun, "Evaluating programming ability in an introductory computer science course," in *SIGCSE '00: Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 212-216.

[14] C. Daly and J. Waldron, "Assessing the assessment of programming ability," in *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004, pp. 210-213.

[15] N. Jacobson, "Using on-computer exams to ensure beginning students' programming competency," *SIGCSE Bull*, vol. 32, pp. 53-56, 2000.