Exposing the Programming Process*

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

[1] IT University West, Denmark
jbb@it-vest.dk
[2] Department of Computer Science, University of Aarhus, Denmark
mec@daimi.au.dk

Abstract. One of the most important goals of an introductory programming course is that the students learn a systematic approach to the development of computer programs. Revealing the programming process is an important part of this. However, textbooks do not address the issue —probably because the textbook medium is static and, therefore, ill-suited to expose the process of programming. We have found that process recordings in the form of captured, narrated programming sessions are a simple, cheap, and efficient way of providing the revelation. We identify seven different elements of the programming process for which process recordings are a valuable communication media in order to enhance the learning process. Student feedback indicates both high learning outcome and superior learning potential compared to traditional classroom teaching.

1 Introduction

An important goal of an introductory programming course is that the students learn a systematic approach to the development of computer programs. Revealing the programming process is an important part of this, and we have found that process recordings in the form of screen-captured, narrated programming sessions are a simple, cheap, and efficient way to provide the revelation. We hereby expand the applied apprenticeship approach as advocated in the works of [Astrachan and Reed, 1995] and [Linn and Clancy, 1992].

Revealing the programming process to beginning students is important, but traditional *static* teaching materials such as textbooks, lecture notes, blackboards or slide presentations, etc., are insufficient for that purpose. They are useful for the presentation of a product (e.g., a finished program), but not for the presentation of the *dynamic* process used to create that product. Besides being insufficient for the presentation of a development process, the use of traditional materials has another drawback. Typically, they are used for the presentation of an *ideal* solution that is the result of a non-linear development process. Like

* This chapter is based on Bennedsen, J. and Caspersen, M. E. 2005. Revealing the programming process. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, USA, February 23-27, 2005, pp. 186-190.

J. Bennedsen et al. (Eds.): Teaching of Programming, LNCS 4821, pp. 6–16, 2008.

others [Soloway, 1986; Spohrer and Soloway, 1986a,b], we consider this to be problematic. The presentation of the product independently of the development process will inevitably leave the students with the false impression that there *is* a linear and direct "royal road" from problem to solution. This is very far from the truth, but the problem for novices is that when they see their teacher present clean and simple solutions, they think they themselves should be able to develop solutions in a similar way. When they realize they cannot do so, they blame themselves and feel incompetent. Consequently, they will lose self-confidence and, in the worst case, their motivation for learning to program.

Several tools for visualizing programs exist such as Jeliot [Levy et al., 2003] and Alice [Cooper et al., 2000]. These tools focus on visualizing the execution of programs, not the development of programs.

Besides teaching the students about tools and techniques for the development of programs (e.g., a programming language, an integrated development environment also known as IDE or programming techniques), we must also teach them about the development process. This can include the task of using these tools and techniques to develop the solution in a systematic, incremental and typically non-linear way. An important part of this is to expound and to demonstrate that

- many small steps are better than a few large ones
- the result of every little step should be tested
- prior decisions may need to be undone and code refactored
- making errors is common also for experienced programmers
- compiler errors can be misleading/erroneous
- online documentation for class libraries provide valuable information, and
- there is a systematic, however non-linear, way of developing a solution for the problem at hand.

We cannot rely on the students to learn all of this by themselves, but by using an apprenticeship approach we can show them how to do it. For this purpose, we use process recordings.

The chapter is structured as follows: section 2 is a brief introduction to the notion of process recordings. In section 3, we discuss the need for exposition of the programming process (e.g., through process recordings) and why textbooks are ill-suited for this purpose. Section 4 is a more detailed description of process recordings where we identify seven different categories of recordings. In section 5, we discuss the use of process recordings in a course context. Section 6 is a brief discussion of related work, and the conclusions are drawn in section 7.

2 Process Recordings — A Brief Introduction

Written material in general and textbooks in particular are not a suitable medium through which to convey processes. We have used process recordings and captured and narrated programming sessions to do that. The creation of a process recording is easy, fast, and cheap, and does not require special equipment besides a standard computer.

The term *process recording* refers to a screen capture of an expert programmer (e.g. the teacher) solving a concrete programming problem and thinking aloud as she moves along. A process recording can be produced by using a standard computer —there is no need for a special studio or other expensive equipment. The software for capturing is free, and depending on how advanced one needs the post production facilities to be, that software is either free or very cheap. We have used Windows Media Encoder and Windows Media File Editor which are both freeware programs.

We have found that 15 to 20 minutes is an appropriate duration of a process recording, although for some problems, the duration can be longer. For convenience, we offer an index (a topic → time mapping) to help retrieve sections of special interest. The index of each recording is stored in a database allowing the students to search for specific material at a later stage. Figure 1 shows a snapshot of a playback of a process recording.

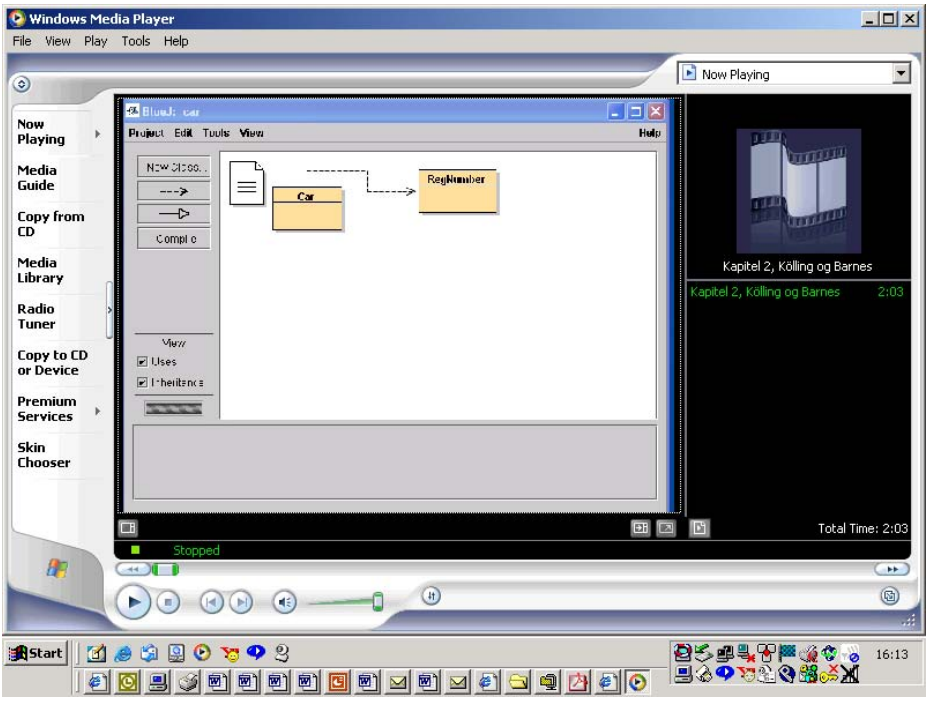

Fig. 1. Playback of process recording

2.1 The Production Process

Most process recordings can be produced without too much preparation. It is our experience that a detailed manuscript is superfluous. Too detailed a manuscript tends to make the process recording less authentic, and in the worst case, plain

boring. For a distance education introductory programming courses we have created approximately 60 process recordings. It is our experience that we use one hour to prepare a 30-minute recording and another 20 minutes for post-production.

The technical setup is rather simple. The lecturer sits in front of his or her computer with a microphone. She starts by introducing the problem and after that, talks aloud about the problems encountered and the possible solutions to these problems. Hereby, she makes the programming process explicit —what are the problems, what are the solutions, what are the alternatives...

To increase the motivation for the students, we have used some of their solutions as a starting point. One needs to be careful not to belittle the solution but to show how different techniques can improve an already working solution in order to make it more readable, shorter, and more reusable or whatever the focus is.

To increase usability we make it possible for students to navigate within the process recording. The addition of the topic → time mapping has added a new usage of the material because the students can search the material afterwards and use it as yet another part of their learning material repository. In this way, the value of the lectures has expanded from something that is only useful if you are present, to a material that can be used repeatedly over time.

3 Teaching the Process of Programming

The concern for teaching process and problem solving is not new. For example, [Gries, 1974] wrote:

> *Let me make an analogy to make my point clear. Suppose you attend a course in cabinet making. The instructor briefly shows you a saw, a plane, a hammer, and a few other tools, letting you use each one for a few minutes. He next shows you a beautifully-finished cabinet. Finally, he tells you to design and build your own cabinet and bring him the finished product in a few weeks. You would think he was crazy!*

Clearly, cabinet making cannot be taught simply by teaching the tools of the trade and by demonstrating finished products. The same applies to teaching programming! Nevertheless, this seems to be what was being attempted thirty years ago when Gries wrote the above analogy, and it seems largely to be the case today.

[du Boulay, 1989] identifies *pragmatics* —the skills of planning, developing, testing, debugging and so on— as an important domain to master. The latter is concerned with skills related to the programming process. However, only few of these are addressed in traditional textbooks on introductory programming.

[Caspersen and Kölling, 2006] describe in detail how a programming process for novices could be described. They focus on five steps:

1. Create the class (with method stubs)
2. Create tests
3. Alternative representations
4. Instance fields
5. Method implementation
6. Method implementation rules (pp. 893-894)

Their focus is on how newcomers can come from a specification of a class (expressed as a UML class diagram) to an implementation fulfilling the specification. In chapter 9, (*Model-driven Programming*) Bennedsen and Caspersen describe a process for implementing a UML class diagram with more classes; focus is particularly on the systematic implementation of relations between classes (e.g. associations).

3.1 Textbooks Neglect the Issue

In 2003, Kölling presented a survey of 39 major-selling textbooks on introductory programming [Kölling, 2003]. The overall conclusion of the survey was that all books are structured according to the language constructs of the programming language, and not by the programming techniques that we (should) teach our students. This is consistent with the findings by [Robins et al., 2003] which state *Typical introductory programming textbooks devote most of their content to presenting knowledge about a particular language* (p. 141). The prevailing textbook approach will help the students to understand the programming language and the structure of programs, but it does not show the student how to program. In short, it does not reveal the programming process.

We know what is needed, so why has the topic not found its way into textbooks on introductory programming? The best answer is that the static textbook medium is unsuitable for this kind of dynamic descriptions.

3.2 New Technology Allows for Changes

Earlier it has been difficult to present actual programming to students. When programs, in the form of finished solutions, were presented to students it was in the form of writings on the blackboard or copies of finished programs (or program fragments) on transparencies for projection.

Programming on a blackboard has the advantage that it is possible to create programs in dialogue with the students at a pace that the students can follow. Also, the teacher and the students can interact during the development of the program. The obvious drawback is that only small programs can be presented, and neither are we able to run and to modify the programs nor be able to demonstrate professional use of the development tool(s) and programming techniques.

Finished programs on transparencies provide a way of presenting larger and more complex programs to the students that we would never consider writing on a blackboard. This approach has the drawback that teachers tend to progress too fast and to exclude the students from taking part in the development.

Thus, the emergence of new technology has made it possible in a simple and straightforward manner to present live programming to students. Live programming can be presented in two different ways: live programming by using a computer and a projector and process recordings showing an expert at work.

Live programming in the lecture theatre by using computer and projector is like a combination of using blackboard and slides. But, it also has the important, additional ability to run and to test the program and to use the programming tools (e.g., IDE, online documentation, diagramming tools). This is much closer to the actual programming process than the first two approaches. However, there are still drawbacks that include limited time in the class room, which restricts the complexity of the examples that are presented. Also, the presentation vanishes as it takes place and nothing is saved afterwards.

Process recordings showing the programming process of an expert are similar to live programming, but without its limitations. In process recordings, you can take the time needed to present as complex an example as you wish, and the presentation can be reviewed over and over as many times as a student needs it.

The first three approaches have in common that they are synchronous, one time events. There is no possibility for the student to go back and review (a step in) the development process if there were something that he or she did not understand. This opportunity is exactly what is added by using process recordings.

4 A Categorization of Process Elements

In this section, we present a more detailed description of the process elements that are exposed through process recordings, and we identify seven different categories that we have found useful in CS1.

A typical programming process encompasses the following process elements:

- Use of an IDE
- Incremental development
- Testing
- Refactoring
- Error handling
- Use of online documentation
- Model-based programming

All are unsuitable for textual descriptions, but important for the student to master. For each process element, we will discuss how to address it in an introductory programming course and how process recordings can be used to reveal its core aspects.

Use of an IDE. We use a simple IDE [Kölling, 1999]. However, a short recording demonstrating the use of special facilities in the IDE makes it still easier for the students to start using it.

Incremental development. Students often try to create a complete solution to a problem before testing it. However, this is not the behaviour that we want the students to exhibit. Instead, we want them to create the solution in an incremental way by taking small steps where they alternate between implementing and testing. Following this advice makes it much easier to find and to correct errors and it simplifies the whole programming activity. This topic is very difficult to communicate in a book. With a process recording, it is simple and straightforward to demonstrate how to behave.

Testing. We promote two simple techniques for testing: interactive testing through the IDE (BlueJ) or the creation of a special class with test methods. The process aspect of the former technique is covered under "Use of the IDE" above (see also [Rosenberg and Kölling, 1997]). A textbook is useful for describing principles and techniques for testing, but the method of integrating testing in the development process is best demonstrated by showing a live programming process.

Refactoring. When the students read a textbook they easily get the impression that programmers never make mistakes, that programmers always create perfect, working solutions in take one, and that programmers, therefore, never have to correct and to improve their programs. [Fowler and Beck, 1999] state that an experienced programmer should expect to use approximately 50% of his or her time refactoring code. If this is the case for an experienced programmer, a novice programmer should expect to use significantly more time refactoring and correcting code. Clearly, students cannot expect to create perfect solutions in their first attempt; it is therefore important not to give them the impression that they should.

We have found it difficult to motivate the need for refactoring to students. The goal of refactoring is to create better programs in the sense of exhibiting lower coupling and higher cohesion. The students, however, do not know when it is advantageous to refactor a program. Instead, they consider the job done when the program can compile and run. However, showing them the refactoring techniques "live" gives them a much better understanding of the techniques and an appreciation of the necessity for refactoring. In order to optimise motivation, we often start out with a students program, showing how refactoring can make that program more readable, and how lower coupling and higher cohesion can be obtained through successive applications of simple standard techniques.

Error handling. In order to make the students feel more comfortable, it is important to show them that every programmer makes errors and that error

handling is a part of the process. It is important to show the students how errors are handled. In particular, it is important to demonstrate to the students that the output from the compiler does not always indicate the real error and that there are different types of errors. The process recordings help by being explicit and by dealing systematically with each kind of error.

Online documentation. Modern programming languages are accompanied by large class libraries, which the students need to use. The documentation for Java is available online, and the students have to be acquainted with the documentation and how to use it in order to write programs. When the students write code, we force them to write Javadoc too. In order to teach them how to write and to generate the documentation, we show them how to do this as an integrated part of the development process by using live programming and process recordings.

Model-based programming. We teach a model-driven, objects-first approach as described in *Model-Driven Programming* by Bennedsen and Caspersen. In order to do so, the students need to use more than the traditional programming tools. They also need to use a tool for describing the class models. The students also need to understand the interaction between the IDE and the modelling tool as well as the relation between model and code. To reinforce the importance of modelling as an integrated part of program development, it is vital to show the students the relevant tools.

5 Process Recordings in a Course Context

In this section, we will describe how the process recording materials are used in an introductory object-oriented programming course.

5.1 Categories of Process Recordings

We have created the following five different types of process recordings: introduction to assignments, solutions to the assignments, documentation of synchronous activities (lectures and online meetings), alternative teaching materials, and tool support.

Introduction to assignments. Many students struggle with getting started with an assignment and have questions like the following: what is the problem, how shall I start, what exactly is it that I have to do? Many such questions can efficiently be addressed in a process recording where fragments or the structure of a solution can be presented.

Solutions to assignments. This includes the presentation of a solution to a programming assignment along with aspects of the development process.

Documentation of synchronous activities. By capturing live programming as it takes place, the students will also have the opportunity to review (parts of) the process at a later stage.

Alternative teaching materials. For the core topics in the text, we create small programming problems to illustrate the use and the applicability of the topic. This provides diversity in the course material by supporting different styles of learning.

Tool support. We have created different kinds of process recordings for tool support. Like [Alford, 2003], we have found that, instead of creating written descriptions and manuals for these tasks, it is much easier for us as well as the students if we create a process recording that shows students how to *do* things. It is more effective to explain what one is doing on the screen while capturing it.

5.2 Student Feedback

Recently, we taught two introductory programming courses based on distance education with respectively 35 and 20 students (a detailed description of the design of this course can be found in [Bennedsen and Caspersen, 2003]. For these courses, we made extensive use of process recordings. All of these materials are stored on a web-server and the students can access them whenever they want and from where they want.

We have evaluated the use of process recordings in our introductory programming course. The evaluation was done quantitatively by using a questionnaire as well as qualitatively by interviewing a number of students about their attitude towards the material. From the questionnaire we can see that more than 2/3 of the students have seen more than 50% of the process recordings.

The distribution of hits for the different types of process recordings is as follows: introduction to assignments: 28%, solutions to assignments: 19%, documentation of synchronous activities: 9%, alternative teaching materials: 21%, and tool support: 23%. The interesting thing is that the possibility of reviewing the synchronous activities has by far the smallest hit rate. This indicates the web casting of lectures, which is a widespread use of process recordings [Berkeley, 2007,MIT, 2006], is viewed by the students as the least useful of the five categories.

The students have self-evaluated the learning outcome of the process recordings. The result of the evaluation is as follows: None: 21%, Small: 0%, Ordinary: 21%, High: 14%, and Very high: 44%. 58% has indicated a high or very high learning outcome which is very encouraging. In post-course interviews, the students generally confirmed these findings.

6 Related and Future Work

Streaming video has become more and more popular and common [Ma et al., 1996; Smith et al., 1999]. Compression techniques have been standardized and improved. Also, bandwidth is increasing (also in private homes) making it realistic to use videos in an educational setting.

Web casts of lectures is used by many universities including prominent ones like Berkeley and MIT [Berkeley, 2007; MIT, 2006] While such videos may be valuable

to students who are not able to attend the lecture or would like to have (parts of) it repeated, they do not significantly add new value to the teaching material.

The use of process recordings in teaching is not new [Smith et al., 1999]. Process recordings are used extensively in [Gries et al., 2002], but their use is somewhat different from ours. All process recordings are very short and focused on explaining a single aspect of the programming language or programming. However, the process recordings are "perfect", and they do not show that it is common to make errors (and how to correct them). Also, the process recordings do not show the integrated use of the different tools like IDE or online documentation, etc. The process recordings in [Gries et al., 2002] can be characterized as alternative teaching materials according to our categorization in the previous section.

Others use a much richer form of multimedia than plain video. One example is the learning objects discussed in [Boyle, 2003]. The same differences as described above apply here as well along with the fact that the production cost for creating these learning objects is extremely high.

Much more needs to be done in this area. The overall, long-term objective of programming education is that students learn strategies, principles, and techniques to support the process of inventing suitable solution structures for a given programming problem. One possible approach to advance our knowledge is to identify, to analyze, and to categorize existing methodological and systematic approaches to the practice of programming and programming education. This includes the classical programming methodology of the Dijkstra-Gries school, design by contract, elementary patterns, the existing but scarce literature on the practice of programming, and a study of the practice of masters. Furthermore, it is necessary to identify, to categorize, and to operationalize strategies, principles, and techniques for object-oriented programming. Similarly, it would be worth indicating how these can be made available to novices as well as more experienced programmers through education. Finally, the insight from this work should form the basis of a formulation of requirements for programming environments and languages for programming education.

7 Conclusions

The idea of revealing the programming process is not new as can be seen in the following quote:

> *Anyone with a reasonable intelligence and some grasp of basic logical and mathematical concepts can learn to program; what is required is a way to demystify the programming process and help students to understand it, analyse their work, and most importantly, gain the confidence in themselves that will allow them to learn the skills they need to become proficient.*

This quotation is almost twenty years old [Gantenbein, 1989], but nevertheless, the issue still has not found its way into programming textbooks.

Revealing the programming process is an important part of an introductory programming course that is not covered by traditional teaching materials such

as textbooks, lecture notes, blackboards, slide presentations, etc. This is just as good since these materials are insufficient and ill-suited for the purpose.

We suggest that process recordings in the form of screen-captured, narrated programming sessions is a simple, cheap, and efficient way of providing a revelation of the programming process. Furthermore, we have identified seven elements included in the programming process. For each of these, we have discussed how to address it in an introductory programming course and how process recordings can be used to reveal its core aspects.

From our evaluation of the approach we know that the students use and appreciate the process recordings, and that some students even find the material superior to traditional face-to-face teaching. The creation of video-mediated materials has proven to be easy and cheap as opposed to other approaches to create learning objects.

The advance of new technology in the form of digital media has made it possible to easily create learning material to reveal process elements that in the past only have been addressed implicitly. The students welcome the new material, which has great impact on the students' understanding of the programming process and their performance in practical programming. With new technology, in this case computers and video capturing tools, it becomes possible to store information that represents dynamic behaviour. This is something that is virtually impossible to describe and to represent by using traditional tools and materials such as blackboards and books. We are looking forward to further pursuing this new opportunity.