Revealing the Programming Process

Jens Bennedsen
IT University West
Fuglesangs Allé 20
DK-8210 Aarhus V
Denmark
jbb@it-vest.dk

Michael E. Caspersen
Department of Computer Science
University of Aarhus
Aabogade 34, DK-8200 Aarhus N
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.

Categories and Subject Descriptors
K3.1 [**Computers & Education**]: Computer Uses in Education – *computer-assisted instruction, distance learning*.

K3.2 [**Computers & Education**]: Computer and Information Science Education – *computer science education, information systems education*.

General Terms
Design, Documentation, Experimentation, Human Factors, Languages.

Keywords
CS1, Programming Process, Process Recording, Model-Based Programming, Objects-First, Design, Incremental Development, Testing, Refactoring, Programming Education, UML, Conceptual Modelling, Systematic Programming, Pedagogy.

SIGCSE'05, February 23–27, 2005, St. Louis, Missouri, USA.

1. INTRODUCTION
We believe that 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, and we have found that process recordings in the form of screen captured narrated programming sessions is a simple, cheap, and efficient way to provide the revelation. We hereby expand the applied apprenticeship approach as advocated in [2, 15].

Revealing the programming process to beginning students is important, but traditional *static* teaching materials such as textbooks, lecture notes, blackboards, slide presentations, etc. are insufficient for that purpose. They are useful for the presentation of a product – 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 which is the result of a non-linear development process. Like others [20, 21, 22], 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 when they see their teacher present clean and simple solutions, they think they themselves should be able in a straightforward fashion to develop solutions in a similar way. When they realize they cannot, they blame themselves and feel incompetent. Consequently they will lose self-confidence and in the worst case their motivation for learning to program.

Besides teaching the students about tools and techniques for the development of programs, i.e. a programming language, an integrated development enviroment (IDE), programming techniques, etc., we must also teach them about the development process, i.e. the task of using these tools and techniques to develop, in a systematic, incremental and typically non-linear way, a "good" solution for the problem at hand. An important part of this is to expound and demonstrate that many small steps are better than few large ones, that the result of every little step should be tested, that prior decisions may need to be undone and code refactored, that making errors is common also for experienced programmers, that compiler errors can be misleading/erroneous, that online documentation for class libraries provide valuable information, and that 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 using an apprenticeship ap-

proach we can show them how to do it; for this purpose we use process recordings.

The paper 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 and we identify seven different categories. In section 5 we discuss the use of process recordings in a course context. Section 6 is a brief discussion of related work. The conclusions are drawn in section 7, which also points to future work.

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, 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, thinking aloud as he moves along. A process recording can be produced 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 post production one needs, that software is either free or very cheap. We have used Windows Media Encoder and Windows Media File Editor, both freeware programs.

We have found that 15-20 minutes is an appropriate duration of a process recording; 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.

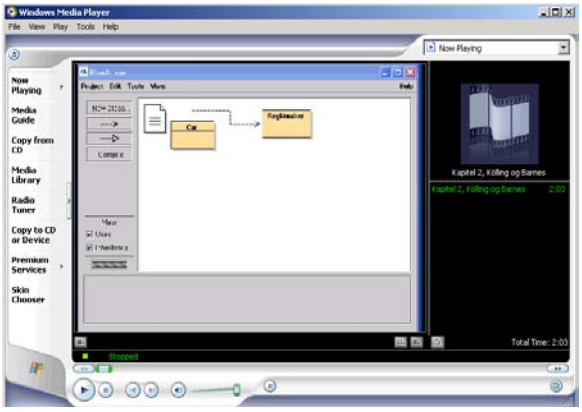

Figure 1: Playback of a process recording

3. TEACHING THE PROCESS OF PROGRAMMING

The concern for teaching process and problem solving is not new; in [10] David Gries 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 demonstrating finished products; but neither can programming. Nevertheless, this seems to be what was being attempted thirty years ago when Gries wrote the above analogy, and to a large extent it seems to be the case today.

du Boulay [6] 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.

3.1 Textbooks Neglect the Issue

At a recent workshop [14], a survey of 39 major selling textbooks on introductory programming was presented. The overall conclusion of the survey was that all books are structured according to the language constructs of the programming language, not by the programming techniques that we (should) teach our students. This is consistent with the findings in [18]: *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 – 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 dialog with the students at a pace 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 modify the programs nor 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, programs that we would never consider writing on a blackboard. This approach has the drawback that teachers tend to progress too fast and exclude the students from taking part in the development.

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 using computer and projector, and process recordings showing how the expert at work.

Live programming in the lecture theatre using computer and projector is like a combination of using blackboard and slides, but with the important additional ability to run and test the program and to use the programming tools (IDE, online documentation, diagramming tools). This is much closer to the actual programming process than the first two approaches. However, there are still drawbacks: time in the class room is limited and this restricts the complexity of the examples that are presented; also, the presentation vanishes as it takes place; 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 to.

The first three approaches have in common that they are synchronous, one shot events. There is no possibility for the student to go back and review (a step in) the development process if there were something he 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 we expose through process recordings, and we identify seven different categories that we have found useful in CS1.

4.1 Elements of Structures and Pragmatics

A typical programming process encompasses the following process elements: use of an IDE, incremental development, testing, refactoring, error handling, use of online documentation, and systematic construction of code from a model/specification. 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 the IDE: We use a simple IDE [13]. 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. This is not the behaviour we want the students to exhibit; instead we want them to create the solution in an incremental way taking very small steps alternating between implementing and testing. Following this advice makes it much easier to find and correct errors and it simplifies the whole activity. This is a topic that 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 [12]). A textbook is useful for describing principles and techniques for testing but how to integrate testing in the development process is best demonstrated showing a live programming/testing 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 improve their programs. In [8] it is stated that an experienced programmer should expect to use approximately 50% of his time refactoring his code. If this is the case for an experienced programmer, a novice programmer should expect to use significantly more time refactoring/correcting; clearly, students cannot expect to create perfect solutions in take one. But the students get the impression that they ought to be capable of this.

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 do not know when it is advantageous to refactor a program; they consider the job done when the program can compile and run. But 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 student's 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 how to write and generate the documentation, we show how to do this as an integrated part of the development process using live programming/process recordings.

Model-based programming: We teach a model-driven, objects-first approach as described in [3]. In order to do so the students need to use more than the traditional programming tools; they need to use a tool for describing the class models. The students also need to understand the interaction between the IDE and the modeling tool as well as the relation between model and code. To reinforce the importance of modeling as an integrated part of program development it is vital to show the students the 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 five different types of process recordings: introduction to assignments, solutions to the assignments, documenta-

tion 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: 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 also fragments/structure of a solution can be presented.

Solutions to assignments: Presentation of a solution to a programming assignment; besides presenting the solution, we also present aspects of the development process.

Documentation of synchronous activities: By capturing live programming as it takes place, the students get 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 applicability of the topic. This provides diversity in the course material supporting different styles of learning.

Tool support: We have created different kinds of process recordings for tool support. Like [1] 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 showing how to *do* things: just tell what you are doing on the screen while capturing it.

5.2 Production Details

Most process recordings can be captured without too much preparation. It is our experience that a detailed manuscript is superfluous; too detailed a manuscript tend to make the process recording less authentic and in the worst case plain boring. 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.

To increase usability we make it possible for students to navigate in the process recording. The addition of the topic→time index has added a new usage of the material: 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.

5.3 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 [4]). 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 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 hitrate; this indicates that web casting of lectures, which is a widespread use of process recordings [5, 17], is 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;: None 21%, Small 0%, Ordinary: 21%, High: 14%, 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 this. One student characterized the use of process recordings as follows: *I claim that the learning potential is better with this teaching form than for traditional class room teaching; in the virtual class room I can eliminate all kind of noise and interruptions. Combined with the opportunity to review (parts of) the session, the return on investment becomes optimal.*

6. RELATED WORK

Streaming video has become more and more popular and common [16, 19]. Compression techniques have been standardized and improved; 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 [5, 17]. 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 [19]. Process recordings are used extensively in [11], but the 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; the process recordings are "perfect", they do not show that it is common to make errors (and how to correct them); and the process recordings do not show the integrated use of the different tools like IDE, online documentation, etc. The process recordings in [11] can be characterized as alternative teaching materials according to our categorization in section 5.1.

Others use a much richer form of multimedia than plain video. One example is the learning objects discussed in [7]. The same differences as described above apply, and on top of that the production cost for creating these learning objects is extremely high.

7. CONCLUSIONS

The idea of revealing the programming process is not new:

> *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 fifteen years old [9]; 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 which is not covered by traditional teaching materials such as textbooks, lecture notes, black-

boards, 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; 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 has 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 represent dynamic behaviour, something which is virtually impossible to describe and represent using traditional tools and materials such as blackboards and books. We are looking forward to further pursue this new opportunity.

8. ACKNOWLEDGEMENT

It is a pleasure to thank Carl Alphonce, David Barnes and Michael Kölling for valuable comments and suggestions.

The production of process recordings was initiated in the Flexnet project under IT University West; we will like to thank IT University West for financial support for the project.

9. REFERENCES

[1] Alford, K., "Video FAQs – Instruction-On-Demand", *Proceedings of Frontiers in Education*, Boulder Colorado, 2003.

[2] Astrachan, O. & Reed, D., "AAA and CS1: The Applied Apprenticeship Approach to CS1", *Proceedings of the twenty-sixth SIGCSE Technical Symposium on Computer Science Education*, Nashville, Tennessee, 1995, pp. 1-5.

[3] Bennedsen, J. & Caspersen, M. E., "Programming in Context – A Model-First Approach to CS1", *Proceedings of the thirty-fifth SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia, 2004, pp. .

[4] Bennedsen, J. & Caspersen, M. E., "Rationale for the Design of a Web-based Programming Course for Adults", Proceedings of ICOOL 2003, *International Conference on Open and Online Learning*, Mauritius, 2003.

[5] Berkeley, http://webcast.berkeley.edu/courses/

[6] du Boulay, J.B.H., "Some difficulties of learning to program", in Spohrer, J.C. and Soloway, E. (Eds.), *Studying the Novice Programmer*, Hilldale, NJ, Lawrence Erlbaum Associates, Hillsdale, 1989., pp. 283-299.

[7] Boyle, T., "Design principles for authoring dynamic, reusable learning objects", *Australian Journal of Educational Technology*, 19 (1), 2003, pp. 46-58.

[8] Fowler, M., *Refactoring – Improving the Design of Existing Code*, Addison-Wesley, 1999. ISBN 0-201-48567-2

[9] Gantenbein, R. E., "Programming as Process: A 'Novell' Approach to Teaching Programming", *Proceedings of the twentieth SIGCSE Technical Symposium on Computer Science Education*, Louisville, Kentucky, 1989, pp. 22-26.

[10] Gries, D., "What Should We Teach in an Introductory Programming Course", *Proceedings of the fourth SIGCSE Technical Symposium on Computer Science Education*, 1974, pp. 81-89.

[11] Gries, D. & Gries, P., *ProgramLive*, John Wiley & Sons, 2001.

[12] Kölling, M. & Rosenberg, J., "Testing Object-Oriented Programs: Making it Simple", *Proceedings of the twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, San José, California, 1997, pp. 77-81.

[13] Kölling, M., "Teaching Object Orientation with the Blue Environment", Journal of Object-Oriented Programming, Vol. 12 (2), 1999, pp. 14-23.

[14] Kölling. M, "The Curse of Hello World", *Invited lecture at Workshop on Learning and Teaching Object-orientation – Scandinavian Perspectives*, Oslo, October 2003.

[15] Linn, M. C. & Clancy, M. J., "The Case for Case Studies of Programming Problems", *Communications of the ACM*, 35 (3), 1992, pp. 121-132.

[16] Ma, W., Lee, Y., Du, D.H.C. & McCahill, M. P., "Video-based Hypermedia for Education-On-Demand", *Proceedings of the fourth ACM International Conference on Multimedia*, 1996, pp. 449-450.

[17] MIT, www.swiss.ai.mit.edu/classes/6.001/abelson-sussman-lectures/

[18] Ronins, A., Rountree, J. & Rountree, N., "Learning and Teaching Programming: A Review and Discussion" *Journal of Computer Science Education*, Vol. 13 (2), 2003, pp. 137-172.

[19] Smidth, T, Ruocco, A & Jansen, B., "Digital Video in Education", *Proceedings of the thirtieth SIGCSE Technical Symposium on Computer Science Education*, New Orleans, Louisiana, 1999, pp. 122-126.

[20] Soloway, E., "Learning to Program = Learning to Construct Mechanisms and Explanations", *Communications of the ACM*, 29 (9), 1986, pp. 850-858.

[21] Spohrer, J. & Soloway, E., "Novice Mistakes: Are the Folk Wisdoms Correct?", *Communications of the ACM*, 29 (7), 1986, pp. 624-632.

[22] Spohrer, J. & Soloway, E., "Analyzing the High-Frequenzy Bugs in Novice Programs", In Iyengar, S. & Soloway, E. (Eds.), *Empirical Studies of Programmers*, Ablex, New York, 1986.