Computer Tutoring for Programming Education

Susan M. Eitelman
University of Central Florida
seitelman@gmail.com

ABSTRACT

Software is increasingly pervasive in the products we use. Consequently, more programmers are needed to develop the software, and consequently there is unmet demand on programming instructors. One possible solution to the increased demand is to complement human teaching with automated computer tutoring. Several examples of such computer tutors for programming already exist, however they have not found widespread success. In the operational world, there are several job-aids that support programmers in the field. Some of these job-aids reflect similar principles used in training tools, particularly scaffolding. Finally, several researchers in the realm of programming instruction indicate the importance of using a problem-based learning approach, or integrating learning and performance for learners. Thus, the paper concludes with questions revolving around how computer tutoring for programming may be enhanced, and lead to greater success, by developing an approach that similarly integrates performance and learning.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer science education

General Terms

Human Factors.

Keywords

Intelligent tutoring,

1. INTRODUCTION

In today's computer age, software is increasingly present in many products. Humans use software for a wide range of tasks. Applications range from entertainment support such as programmable digital video recorders, to computer based training and education tools, such as a reading tutor that listens (Mostow, Aist, Burkhead, Corbett, Cuneo, Eitelman, Huang, Junker, Sklar, & Tobin, 2003), to real-time critical systems such as in a nuclear power plant (e.g., Nguyen, & Ourghanlian, 2003). As the use of software-based products rises, so does the demand of software developers. Furthermore, there is an increased demand on programming instructors, which may be higher than the current supply of instructors available (Shaffer, 2005). How might the instruction of programming be better supported?

Computer tutoring systems are one way teachers are supported in the classroom. There are a variety of computer-based tutors available and several that focus on programming. There are also several job-aids aimed to support programmers, some of which reflect principles similar to those used in teaching. In particular, scaffolding mechanisms are present in programming job-aids that may be useful both in training and on the job. The present paper presents a review of some of the research involved in computer tutors for programming, and provides an example of a job-aid that reflects characteristics of a training principle of scaffolding. Finally, the conclusion contains questions to guide research for developing a learning and performance support system.

2. BRIEF REVIEW OF THE LITERATURE

There are several computer tutors that focus on the instruction of computer programming. An early programming computer tutor focused on the Lisp language (Anderson, Boyle & Reiser, 1985). Developed more recently, another computer tutor teaches the C language (Song, Hahn, Tak, & Kim, 1997). Ludwig, utilizes networking and online collaboration of people to support the acquisition of programming skills (Shaffer, 2005). Finally, another computer tutor system, that teaches programming in the Java language, involves both a particular curriculum design and specially-selected tools identified for their instructional value (Ragonis & Ben-Ari, 2005). There are many other programming computer tutors, which vary by the programming language they use and instructional approach they reflect; the ones listed are presented to provide a flavor for the variety.

Although many computer tutors exist that aim to develop programmers, they have not proven themselves largely successful (e.g., Deek & McHugh, 1998; Shaffer, 2005). Many researchers are asking why this is the case, but no clear answers seem to have manifested themselves, yet. It has been shown that computer tutors may improve learning programming or other skills such as geometry proofs when they are used to complement the teacher (e.g., Chalk, 2001; Schofield, Eurich-Fulcer, and Britt, 1994). Schofield and her team indicate that computer tutors may support learning in a traditional classroom, not replace it, and also change the learning atmosphere in a traditional classroom. The classroom becomes less "teacher-centered" and the role of the teacher becomes more facilitative (Schofield, et. al., 1994, p. 581). Thus, the success of computer tutors, particularly for teaching programming, may not be a question simply of instructional strategy and mechanism, but include several variables and their relationships including the environment, the actors, and the approach.

Problem solving is the foundation of programming skill (Deek & McHugh, 1998; Ragonis & Ben-Ari, 2005). Shaffer (2005) points out that "most attempts at using technology to train programmers have been unsuccessful because the designers have misconstrued programming as a knowledge-based rather than as a task-based discipline" (p. 56). Similarly, Pirolli and Recker (1994) explain that there is an important difference between teaching declarative and procedural knowledge. Furthermore, Shaffer, citing McBreen (2002), poses that the apprenticeship model may be more effective than that of the college setting because the experience produces useful work under the apprenticeship model. Pirolli and Recker also indicate that procedural learning occurs through problem-solving of new situations pointing out that learning occurs by doing. However, Shaffer also reminds us that some declarative knowledge is required; as he puts it: “learning to program is like learning a foreign language; you can never be truly fluent if you have to look stuff up all the time” (p. 59). Shaffer’s comment reflects the importance of declarative knowledge when acquiring a skill. Thus, the computer tutor should not replace the teacher, nor should it focus solely on declarative or procedural memory.

3. INTEGRATING LEARNING & PERFORMANCE

Although changing the collegiate system institution to incorporate the apprentice-model may at first seem somewhat unrealistic, the point resonates with a principle called the Janus principle. Hoffman (2004) poses the Janus principle: "Human-centered systems do not force a separation between learning and performance. They integrate them" (p. 79). The Janus principle underlies some of the tools currently available. For example, in their review of the literature, Deek and McHugh (1998) indicate four major categories of programming tools, both instructional and operational, one of which is the intelligent programming environment category. They define an intelligent programming environment as a programming environment that incorporates aspects of both a classic programming environment, which provides the basic tools to develop a program (e.g., text editor, compiler), and of an intelligent tutoring system (e.g., automated feedback and coaching). Such a system provides job support and training to the user, thus integrating learning and performance. In concert with the Janus principle, Deek and McHugh (1998) also purport that “environments for the facilitating the study of programming *must* be consistent with the activities of the actual learning situation and support the entire problem-solving and program development process” (p. 172). Furthermore, Dunlap (2005) investigates the effects of student self-efficacy on performance in a capstone software development course. She explains that to appropriately prepare software developers for working in their field, “educators need to create learning environments that engage students in ways that help them develop content expertise and problem-solving, collaboration, and lifelong learning skills” (p. 65). In other words, learning and performance should be integrated. Dunlap recognizes the Problem-Based Learning (PBL) approach as one method of integrating learning and performance in a learning environment, which reflects a basic application of the Janus principle in the programming domain. Additionally, the PBL approach seems to provide a mechanism for shifting the traditional classroom approach towards a more apprentice-like model.

To integrate learning and performance, one learning concept in particular sticks out as particularly useful. Scaffolding is an instructional strategy that dates back to the mid-1970s (e.g., Hobsbaum & Peters, 1996). Ragonis and Ben-Ari (2005) discuss lessons learned from their investigation into the understanding of programming concepts, particularly object-oriented programming concepts, by novices. One of their main findings is sequencing of topics. They also provide other guidelines, for example they indicate that when classes and objects are introduced, which should occur early in the curriculum, they should be introduced with the support of diagrams. However, Ragonis and Ben-Ari also point out that pictures involved in the problem may distract the learner. The study by Ragonis and Ben-Ari seems to be the first of its kind, a long-term systematic study of the teaching of object-oriented programming concepts. Thus, their results indicate an important example of the value of scaffolding. They point out that some concepts must be taught before others, indicating an instance and the importance of scaffolding.

Chalk (2001) explains the specific mechanisms required for a scaffolding approach. Briefly they are: recruit attention, reduce degrees of freedom, maintain direction, mark critical features, control frustration, and demonstrate solutions when learner can recognize them. First, a scaffolding approach will help the learner maintain focus on the problem at hand by recruiting attention. Second, the approach reduces complexity of the problem such that the learner is not overwhelmed or distracted by extraneous characteristics. For example, the guideline recommended by Ragonis and Ben-Ari about avoiding pictures in problem statements is an example of reducing the complexity of the problem by avoiding the presentation of extraneous details, which might occur in a picture. Third, the learner maintains direction to achieve the goal when supported by a scaffolding approach. Again using the examples provided by Ragonis and Ben-Ari, their diagram for introducing classes and objects may be considered a way to reduce the complexity of understanding a class and an object by depicting it in a simple and functional manner. Fourth, using an appropriate scaffolding approach should involve the marking of critical feature to support the learner in recognize what is important and avoid distraction by insignificant features and details of the problem. Fifth, the approach will control the learner’s frustration, facilitating the maintenance of direction to achieve the goal and maintenance of keeping the learners attention on the task. Finally, a scaffolding approach should include demonstration of solutions in a manner that facilitates the learner in recognizing how the problem was solved such that the learner may develop an accurate construct for solving similar problems.

Some of the scaffolding features introduced by Chalk are present in job-aids for programming. For example, Pike, Weide, and Hollingsworth describe work they did to develop a job-aid for facilitating programmers, not students of programming, with avoiding dynamic memory errors, such as memory leaks that are hard to debug and a common problem even for seasoned programmers (2000). Their tool is simply a layer, implemented as a header, or .h, file that supports the programmer in identifying dynamic memory allocation errors that may lead to memory leaks. The tool helps the programmer identify these early, during code implementation, as well as throughout the compilation and execution phases. After the programmer is satisfied that the program code is error-free, at least for dynamic memory allocation, then the header file may be removed without affecting the execution of the file when there are no dynamic memory allocation errors, which improves the efficiency of the program execution. In the present example, marking critical features is the predominant scaffolding mechanism, and frustration control is also present. The tool marks where the memory allocation error occurs, marking the critical feature in the code; then the tool arguably also controls the level of the programmer’s frustration by facilitating the debugging of the error.

Another example reflects a job-aid for a programming teacher. Warms (2005) describes a tool he uses in explaining concepts to his students. In particular, the tool incorporates visualization mechanisms that primarily reduce the complexity of using pointers. The tool helps the learner visualize the trace of memory allocation to addresses, this time static memory allocation, which is differs from the dynamic memory allocation errors described earlier. While Warms proposes his tool as a job-aid for teaching the concepts of pointers, the tool is clearly a training aid. Furthermore, it seems natural to extend such a tool to job-aid applications of programming itself – not just teaching programming. In particular, it may be useful for newer programmers who may not have yet become experts on the use of pointers even if they understand the concept at a basic level.

Another aspect to scaffolding is the removal of "scaffolds" as the learner progresses in proficiency, reflecting the Zone of Proximal Development (ZPD). With ZPD, the learner is able to achieve more with support, such as from a tutor, than they would otherwise be able to achieve alone (e.g., Fernández, Wegerif, Mercer, & Rojas-Drummond, 2001). So, a learning aid may be applied until it is no longer needed, as the learner is now able to achieve on his own what was previously achievable only with the aid. Such an approach is already demonstrated by the dynamic allocation memory error checking header file (Pike, et. al., 2000). Similarly, the pointer visualization tool, described by Warms (2005), might be used in a programmer's early stages but later be discarded once the programmer fully masters manipulating pointers. Thus, the Janus principle, namely the integration of learning and performance support, is already appearing in the development of computer programming tutors.

4. DISCUSSION

Although significant time and expense have been poured into the development of various computer tutors, they still suffer from not having demonstrating widespread success, particularly in light of the increasingly overwhelming demand on programming teachers indicated by Shaffer (2005). One conclusive question is: why have computer programming tutors not found widespread success? More specifically, the question arises: how can we employ computer programming tutors for training productive programmers more successfully?

In an effort to address the last question, integrating learning and performance support may lead to an answer. First, the development of programming skills must be defined. Such work is already documented in the literature, for example the work by Ragonis and Ben-Ari (2005) reflects a particular ordering of concepts in object-oriented program. A more complete model of the process of programming skill development, including advanced concepts, would support the development of a learning and performance support system for a programming environment. Second, a comprehensive description of language-independent concepts and tabulation of language-dependent implementation issues, at least for a select set of popular languages to start, would guide the development of such a system. In particular, designers would derive the declarative and procedural knowledge stores directly from such records.

The C-Tutor introduced by Song, Hanh, Tak, and Kim (1997) begins to reflect such an approach. Their tutor involves an intention-based diagnosis tool to intelligently identify the learner's intentions, and also it involves a debugging tool that analyzes how the learner is implementing the intentions, or goals. In this manner, the tutor allows the learner to actively develop a program, while it provides feedback. However, there are some opportunity gaps with the C-Tutor. One example is that it relies on example problems with solutions entered by the teacher. A facility that frees it from such a constraint, or at least from relying solely on examples, would enable the tool to grow into a job-aid with the learner. Potentially, the tool could *learn* new example-types and support the user in solving similar problems in the future, especially in a collaborative, online environment, which is addressed next.

Finally, various other questions manifest themselves in the search for better applications of computer programming tutors. For example, what is the relationship of the computer tutor and the human teacher, as introduced by Dunlap (2005)? Further, how can we foster the relationship to optimize learning? Also, what is the relationship of multiple learners in a group? Shaffer's (2005) online learning environment introduces the potential of learner-learner coaching and instructional support. How might the learner-learner relationship be fostered to optimize learning? In conclusion, it seems that although computer programming tutors have not yet shown widespread success, they still hold potential. A variety of research questions arise surrounding how this may be approached and include exploration of integrating learning and performance support tools, which complement the Problem-Based Learning approach.

5. ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

6. REFERENCES

[1] Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science, 228*, 456-462.

[2] Chalk, P. (2001). Scaffolding learning in virtual environments. *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, 85-88. Canterbury, UK: ACM Press.

[3] Deek, F. P., & McHugh, J. A. (1998). A survey and critical analysis of tools for learning programming. *Computer Science Education (8)*2, 130-178.

[4] Dunlap, J. C. (2005). Problem-based learning and self-efficacy: How a capstone course prepares students for a profession. *Educational Technology Research and Development, 53*(1), 65-85.

[5] Fernández, M., Wegerif, R., Mercer, N., & Rojas-Drummond, S. (2001). Reconceptualizing "scaffolding" and the zone of proximal development in the context of symmetrical collaborative learning. *Journal of Classroom Interaction, 36*(2), 40-54.

[6] Hoffman, R., Lintern, G., Eitelman, S. (2004). The Janus Principle. *IEEE Intelligent Systems*, 19 (2), March/April 2004, p. 78-80.

[7] Mostow, J., Aist, G., Burkhead, P., Corbett, A., Cuneo, A., Eitelman, S., Huang, C., Junker, B., Sklar, M. B., & Tobin, B. (2003). Evaluation of an automated Reading Tutor that listens: Comparison to human tutoring and classroom instruction. *Journal of Educational Computing Research, 29*(1), 61-117.

[8] Nguyen, T., & Ourghanlian, A. (2003). Dependability assessment of safety-critical system softare by static analysis methods. *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN '03)*. IEEE.

[9] Pike, S. M., Weide, B. W., & Hollingsworth, J. E. (2000). Checkmate: Cornering C++ dynamic memory errors with checked pointers. *ACM SIGCSE Bulletin, Proceedings of the thirty-fist SIGCSE technical symposium on computer science education SIGCSE, 32*(1), 352-356. New York, NY: ACM Press.

[10] Pirolli, P., & Recker, M. (1994). Learning strategies and transfer in the domain of computer programming. *Cognition and Instruction (12)*3, 235-275.

[11] Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education, 15*(3), 203-221.

[12] Schofield, J. W., Eurich-Fulcer, R., & Britt, C. L. (1994). Teachers, computer tutors, and teaching: The artificially intelligent tutor as an agent for classroom change. *American Educational Research Journal 31*(3), 579-607.

[13] Shaffer, S. C. (2005). Ludwig: An online programming tutoring and assessment system. *Inroads – The SIGCSE Bulletin, 37*(2), 56-60.

[14] Song, J. S., Hahn, S. H., Tak, K. Y., & Kim, J. H. (1997). An intelligent tutoring system for introductory C language course. *Computers & Education, 28*(2), 93-102.

[15] Warms, T. M. (2005). The power of notation: Modeling pointer operations. *Inroads – The SIGCSE Bulletin, 37*(2), 41-45.