Song Debugging: Merging Content and Pedagogy in Computer Science Education

Tami Lapidot and **Orit Hazzan**
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa, Israel 32000
{lapidot, oritha}@tx.technion.ac.il

Abstract

This article suggests a song debugging activity that brings together two key ideas – the *analogy* between learning and debugging and the *pedagogical potential* of music in Computer Science Education (CSE). The paper can be viewed as the fourth in a series of papers published in *inroads* about the course Methods of Teaching Computer Science in the High School, but it can also stand on its own merit, since it discusses issues that are relevant to CSE in general.

Keywords: Computer science education, music, debugging, teacher education

1. Introduction

This article brings together two key ideas – the *analogy* between learning and debugging and the *pedagogical potential* of music in Computer Science Education (CSE) – using the active-learning based teaching model, presented by Hazzan and Lapidot (2004). The paper can be viewed as the fourth in a series of papers previously published in *inroads* about the course Methods of Teaching Computer Science in the High School (henceforth abbreviated MTCS), but it can also stand on its own merit since it discusses general relevant issues to CSE.

In Section 2, we elaborate on the importance of teaching debugging in CSE in general, and in the MTCS course in particular, and examine the analogy between learning and debugging. Section 3 discusses the use of music in CSE and Section 4 presents a musical debugging task that connects the two key concepts mentioned above. In Section 5 we discuss the main ideas of the paper.

2. The Learning-debugging Analogy

Mistakes and errors are part of our lives, as well as of our learning processes. Unlike other disciplines, however, CS students are forced to face and cope with errors from very early stages of their studies. Perkins and Martin (1986) address this issue by arguing that programming requires a kind of perfection that does not exist in other areas. Specifically, if one get 10% of the answers on a regular test wrong, one can still make the well-appreciated grade of 90; in contrast, in software development, if one writes a 90% correct program, the program will not work and the erroneous 10% might lead to a complex debugging process. This complexity can be explained by Dijkstra's statement from 1989, according to which the programmer "has to think in terms of conceptual hierarchies that are much deeper than a single mind ever needed to face before."

Within the CSE community, there is consensus that debugging is an important topic that should be addressed within the framework of CSE (cf., for example, Lieberman 1997; Spohrer & Soloway, 1986). Indeed, debugging has an exceptional educational value. We claim, however, that even if an automated debugger were to be created, it would still be beneficial to pay special attention to debugging in CSE in general and in the MTCS course in particular. In what follows, we explain this approach. We start by addressing the general importance of debugging in CSE.

First, like programming, debugging involves *problem-solving skills* that must be acquired. Clearly, debugging experience gained by students might help them with other (programming) tasks as well.

Second, unlike other areas in which errors are treated as a negative phenomenon, in CS, errors and debugging are a fact of life and, accordingly, deserve special attention. Like Papert (1980), we believe that learning to debug can *change students' negative attitudes towards errors* and increase their awareness with respect to the importance of mistakes in learning processes. In general, modern society tends to view success as a primary goal and, accordingly, a mistake is conceived of as being a disturbing element. Not surprisingly, learners are afraid to make mistakes and tend to adopt a "black and white" perspective of their achievements. This view is even strengthened among students that are accustomed to being evaluated by the

accepted grading system. The hidden educational message conveyed to them over the years is that mistakes or errors lower their grade and that their grade will rise if they avoid errors as much as possible. According to this perspective, a "smart" student is one who does not make mistakes and a "stupid" student is one who does. Clearly, once errors become a legitimate discussion topic, as proposed in this paper, it becomes easier to address the connections between learning processes and misconceptions.

Third, learning about debugging might improve students' *understanding of their own thinking processes*. According to the constructivist theory, learning consists of a debugging process of one's own knowledge. Specifically, learning (of any topic) starts by building a preliminary (sometimes erroneous) model and then proceeds with shaping it, through a successive refinement process, into another (more correct) model. In order to start the refinement process, one must acknowledge the fact that one's initial mental model might be erroneous. With respect to software development, each program we write reflects the way in which we conceive the problem we face (Papert, 1980); debugging is a means that enables us to come closer to the target. Awareness of this role of debugging can improve one's understanding of learning processes.

In what follows, we further explain why debugging deserves special attention in the MTCS course.

First, future CS teachers must be aware of the fact that *debugging strategies are important from the early steps of novice programmers' work* and that, accordingly, CS teachers should deal with debugging-related issues in their daily work. Further, Carver & McCoy (1988) indicate that "children typically respond ineffectively to debugging situations. Many of them panic and call the teacher insistently, making no attempts to correct the situation until help arrives. Others quickly erase their code and begin again without ever understanding their error." Consequently, CS teachers should be aware of different ways by which to approach such debugging-related situations.

Second, *CS teachers must themselves be good debuggers* in order to be able to understand students' buggy programs. Good debugging skills will enable them to guide students in their debugging processes. Although student-teachers are familiar with debugging strategies from their own CS studies, the attention given to debugging in the MTCS course might further improve their debugging skills.

Third, CS teachers should realize the *opportunity* they have, through the discussion on debugging, *to change negative attitudes* towards errors that they (or their future pupils) may have.

Fourth, debugging is an *excellent example of the connection* that exists *between content and pedagogy*. In a previous publication (Lapidot & Hazzan, 2003), we suggested the use of this special amalgam of CS and pedagogy as one possible organizing idea for the MTCS course and offered one possible implementation of the concept. Specifically, we demonstrated the theme of teaching soft ideas in CS using five activities that deal with the concept of abstraction. Debugging is another excellent example for such an amalgam. In other words, since *programming* is based on the debugging of algorithms and *learning* is based on the debugging of mental models, their integration in the MTCS course can help us convey several pedagogical messages to prospective CS teachers. Such messages may include the following: Debugging is a fact of life, both in learning and in programming, and therefore is inevitable; Mistakes are not a negative phenomenon, as long as one knows how to learn from them and how to fix them; Tools and strategies that support debugging exist and knowing how to use them can improve one's debugging skills; and finally, teachers should address debugging-related issues and teach their students to value the importance of debugging in software development processes. In Section 4, we present a specific example that illustrates this amalgam of CS and pedagogy and which can, accordingly, promote the discussion of such topics.

3. The Pedagogical Potential of Music in CSE

"Music is uniquely universal, touching equally across language, gender, cultural and age divides." (Hamer, 2004, p. 160)

"Although sound is not visible we are still able to construct mental images when presented with particular sounds or pieces of music." (Vickers, 1999, p. 15)

It is a common belief that one picture is worth a thousand words. Similarly, we propose that one musical note is worth at least a hundred words. Just like visualization, music is powerful in activating people's senses, changing people's moods, conveying messages very quickly (in horror movies for example), and providing information that sometimes is very difficult, or even impossible, to convey by other means. In a paraphrase of Allen, Davis & Johnson (1984) with respect to visualization, we suggest that musical outputs of computer programs might have substantial educational advantages over the more common textual output of programming assignments, since "music comes alive when it is performed." (Hamer, 2004).

Our literature review revealed that there have been only a handful of previous attempts to use music in CSE. Here are some examples.

Hamer (2004) suggests teaching design patterns by means of a music composition project. Among other objectives, Hamer uses this teaching approach to deepen students' understanding of programming by exploiting the analogy between a computer-program structure and a musical structure.

Franklin (2001) suggests that "computer generated music can be useful in introductory programming courses" and explains that "it is not a replacement for other teaching

techniques, especially visual ones, but it can augment these and perhaps create a different kind of appreciation of the algorithms".

Rubinstein (1995) describes his experience with music-related projects and claims, "They are an effective means to apply Computer Science to another area while gaining experience in a large scale design and development project, producing a useful, original, high-quality product. The students tend to be extremely enthusiastic about it, have fun doing it, and end up with useful products that they take pride in. And in the process, they learn a great deal."

Foss (1989) describes how "the inclusion of musical device control has served to highlight certain important concepts and programming techniques in a way that has proved to be both fun and insightful." We agree with his statement that "invariably, in computer science, one learns best by doing. I have found that the most fruitful way to teach a concept is to provide a goal that can only be reached with an understanding of that concept".

Additional examples of the use of music in CSE can be found in Francioni *et al*. (1991), Hotchkiss & Wampler (1991), Brown & Hershberger (1992), Levy (1995), Lapidot (1996-a), Lapidot (1996-b), Guzdial & Soloway (2002).

One work that is of particular relevance to our article is that of Vickers (1999) since he also connects *music with debugging*. Vickers argues that "the research community has been slow to recognize sound as a useful carrier of information in the world of software development". His thesis "investigates the use of music as a communication medium in the task of computer programming and debugging". To deliver his ideas, Vickers developed a tool, CAITLIN, which is "a non-invasive pre-processor that allows a novice programmer to auralise [executable] programs written in Turbo Pascal."

Like Vickers, we believe in the advantages of music and its relevance to debugging. However, the activity presented in the next section is different from CAITLIN in several respects. First, while CAITLIN was designed as a general tool for the debugging of *any executable* program, our activity is aimed at focusing students' attention on ways to bridge differences between *one* desired (known) behavior (of a familiar song) and a (intentionally) *bugged* program presented to them (which plays the song). Second, since the target of our task is to promote *reflection* on the debugging process (rather than to promote the debugging process itself), a *specific familiar* song is used. Finally, since CAITLIN is designed to assist with the debugging of executable programs (thus, accepting only programs that are free of syntax errors), it addresses only *logic bugs*. At the same time, the activity presented in the next section is based on *all types of bugs*.

Before we continue, it should be noteed that there are also disadvantages to the use of music in CSE. First, not everyone likes music. This, however, is also true with respect to other areas (for example, not everyone likes mathematical assignments). Second, if music is used in a computer lab, a great deal of noise is produced in the lab once students start playing their programs. This problem can be solved by using headphones (Franklin, 2001).

4. The Musical Debugging Activity

"Activities should connect to important domains of knowledge and more significantly, encourage new ways of thinking (and even new ways of thinking about thinking)." (Resnick *et al.,* 1996).

This section presents the musical debugging activity mentioned above. We propose that the teaching approach represented by such activity might help students become aware of their own debugging processes, of the importance of debugging and of its relationship to learning. Furthermore, it illustrates a non-conventional use of the computer laboratory. Since "debugging is often a struggle against complexity", we join in Lieberman's (1977) suggestion to let the computer "take an active role in helping the programmer deal with complexity" through visualization, or, in our case, through music.

The activity requires the students to debug a program that is supposed to play a specific song. The program has several syntax errors, such as wrong parameter names or missing identifiers, and an erroneous order of procedure calls which generate logic bugs. In our case, we use the Hebrew song "Twelve Months" (written by Naomi Shemer, the late well-known Israeli songwriter); but naturally other songs can also be used for this purpose, as long as the students are familiar with them. In what follows, we present the activity according to the 4-stage active-learning based teaching model introduced by Hazzan and Lapidot (2004).

Trigger: Students are given[1] the song lyrics, the computer program, the program listing, and the melody (both in musical notation and in Hebrew). They are asked to debug the program and to document their actions while working.

Activity: Students debug the program, trying to bring it to a working version, i.e. to play the correct melody. They start with syntax errors, which are easier to detect and obtain a program devoid of syntax errors. At this stage, they realize that the program does not play the correct melody and start debugging the semantic (or logic) errors of the program. During their work (or as soon as they complete working on it) they document their actions and the strategies they used during the debugging process.

Discussion: The discussion that follows the activity can be viewed on two levels. On the first level, content is addressed. Students present the errors they found in the program and describe their debugging strategies. This

[1] The original materials are presented on the Web at http://edu.technion.ac.il/Faculty/OritH/HomePage/MTCS_Website/index.htm.

presents an opportunity to address programming errors and their classification according to several categories, (e.g., syntax errors are the easiest to find whereas logic errors are the most difficult to locate). When students are asked to generalize their own strategies, the discussion focuses on general debugging strategies. The second level of the discussion focuses on topics such as the ones discussed in Section 2: the importance of debugging in CS and in CSE, the teaching of debugging and the place of debugging in the CS curriculum.

Summary: Students can be asked to prepare another debugging activity or to read and comment on articles that address debugging (e.g., Spohrer & Soloway, 1986; Perkins & Martin, 1986).

5. Discussion

The discussion examines the task presented in Section 4 from a pedagogical perspective by addressing two issues. Section 5.1 examines how the task expresses a merge between CS contents and pedagogy, while Section 5.2 discusses how the task, designed according to the active-learning based teaching model, provides students in the MTCS course with an opportunity to perform a variety of roles, which, as discussed previously by Hazzan and Lapidot (2004), might support the construction of the prospective CS teacher's professional perception.

5.1 Merging contents with pedagogy

Debugging, like other programming methodologies (stepwise refinement, for example), naturally merges content and pedagogy aspects. Debugging is, however, exceptionally unique in the way these aspects are merged together.

As was previously discussed, *learning is analogous to debugging*. Consequently, an integrated discussion about these concepts can be fruitful in two respects. First, it may enhance students' reflection skills of their own work (Clements & Guilo, 1984). Second, it may help prospective CS teachers in their future teaching since they become more familiar with debugging difficulties and this experience might help them relate to (general) learning difficulties their future pupils may face.

5.2 Using the active-learning based teaching model

In a previous article (Hazzan and Lapidot 2004), we suggest that it is important that students in the MTCS course experience wearing different hats – the student's, the teacher's and the high school pupil's hats. As is illustrated below, the task presented in Section 4 enables performing these different roles quite naturally.

When in the role of the student, based on the actual act of debugging, students might simply learn about debugging (e.g., strategies that enhance debugging processes). When wearing the teacher's hat, the analogy between learning processes and debugging processes is addressed. Such an examination might improve the prospective teachers' understanding, as well as their awareness, of the nature of learning processes, and consequently might influence their pedagogy in general and their teaching of debugging in particular (e.g., how they will help their future pupils debug their own programs). It might also help them adopt specific teaching techniques for the teaching of debugging. Finally, wearing the pupil's hat might help students in the MTCS course understand the resistance their future pupils might feel towards debugging. Clearly, such a multi-faceted perspective of debugging is important for prospective CS teachers.

References

Allen, J.R., Davis, R.E. and Johnson, J.F. (1984). *Thinking about TLC Logo – A graphic look at computing with ideas*. Holt, Rinehart and Winston.

Brown, M. H. and Hershberger, J. (1992). Color and sound in algorithm animation. *IEEE Computer* **25**(12), pp. 52-63.

Carver, S., and McCoy (1988). Learning and Transfer of Debugging Skills: applying task analysis to curriculum design and assessment. In Mayer R.E. (ed.), *Teaching and Learning Computer Programming, Multiple research perspectives*. Lawrence Erlbaum Associates, Inc., chapter 11.

Clements, D.H. and Guilo, D. (1984). Effects of computer programming on young children. *Proceedings of the 1984 Logo Conference*.

Dijkstra, E.W. (1989). On the cruelty of really teaching computing science. *CACM* **32**(12), Dec. 1989, pp. 1398-1400.

Foss, R. (1989). Music in computer science courses. Using inexpensive, exciting technology to teach programming principles. *SIGCSE Bulletin*, **21**(4), pp. 57-64.

Francioni, J. M., Jackson, J. A. and Albright, L. (1991). The Sounds of Parallel Programs. *Proceedings of the 6th Distributed Memory Computing Conference*, pp. 570-577.

Franklin, J. (2001). Computer Generated Music as a Teaching Aid For First Year Computing. *JCSC*, **16**(4), pp. 10-20.

Guzdial, M. and Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM*, **45**(4), pp. 17-21. Available on the web at http://www.cc.gatech.edu/gvu/people/Faculty/Mark.Guzdial.html

Hamer, J. (2004).An Approach to Teaching Design Patterns using Musical Composition. *Proceedings of ITiCSE'04* June 28–30, 2004, Leeds, UK, pp. 156-160.

Hazzan, O. and Lapidot, T. (2004). Construction of a professional perception in the "Methods of Teaching Computer Science" course. *Inroads – the SIGCSE Bulletin* **36**(2). pp. 57-61.

Hotchkiss, R.S. and Wampler, C.L. (1991). The Auditorialization of Scientific Information. *Proceedings of Supercomputing*, ACM Press, pp. 453-461.

Lapidot, T. (1996-a). Multimedia and its Contribution to the Teaching of Recursion. *Hebetim*, No. 7, pp. 50-52 (in Hebrew).
Lapidot, T. (1996-b) Musical Debugging. *Hebetim*, No. 8, pp. 26-30 (in Hebrew).
Lapidot, T. and Hazzan, O. (2003). Methods of Teaching Computer Science course for prospective teachers. *inroads – the SIGCSE Bulletin* **35**(4), pp. 29-34.
Levy, D. (1995). Programming Principles in a Musical Environment. *Hebetim*, No. 5, pp. 34-41 (in Hebrew).
Lieberman, H. (1997). The debugging scandal and what to do about it (special section). *Communications of ACM*, **40**(4), pp. 27–29.
Papert, S. (1980). *Mindstorms: Children, Computers and Powerful ideas*. Basic Books Inc.
Perkins, D.N. and Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In Soloway E. and Iyengar S. (eds.), *Empirical Studies of Programmers*. Norwood, NJ: Ablex Publishing Co., pp. 213-229.
Resnick, M., Bruckman, A. and Martin, F. (1996). Pianos not stereos: Creating computational construction kits. *Interactions*, **3**(6), pp. 41-50.
Rubinstein, R. (1995).Computer Science Projects with Music. *Proceedings of SIGCSE'95*, March 1995, Nashville, TN USA, pp. 287-291.
Spohrer, J.G. and Soloway, E. (1986). Analyzing the high frequency bugs in novice programs. In Soloway E. and Iyengar S. (eds.), *Empirical Studies of Programmers*. Norwood, NJ: Ablex. pp. 230-251.
Vickers, P. (1999). *CAITLIN: Implementation of a musical program auralization system to study the effects on debugging tasks as performed by novice Pascal programmers.* A doctoral thesis, Loughborough university. Available on the web at http://computing.unn.ac.uk/staff/cgpv1/caitlin/index.htm.