Some Heuristics for the Development of Music Education Software: First Steps Towards a Methodology

Luciano Vargas Flores, Rosa Maria Vicari, Marcelo Soares Pimenta

Laboratório de Computação & Música – Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{lvf,rosa,mpimenta}@inf.ufrgs.br

Abstract. *Our Computer Music group has been researching and developing software for Music Education since 1995. In search for a specific development methodology for Music Education software, we have collected some heuristics that are very likely to constitute its basis. These heuristics are presented and discussed in this paper. Our set of heuristics cannot be exhaustive, but it does cover important issues of Music Education software development, and we hope they may even help other development teams.*

1. Introduction

In the last years, our work has been oriented towards the construction of models, concepts and tools for computer-based Music Education. In the beginning, this work was being conducted in a very unsystematic way with respect to software engineering methodologies. As a consequence, some problems began to emerge at that time, motivating us for the creation and proposal of a specific development methodology for Music Education Software (MES). These problems were related to issues such as *software quality* improvement, *remote cooperation* support, support for *developers from outside the Computer Science area* and *nonspecificity of usual software development methodologies*. In addition, we found that the *expertise our research group acquired* after these years of dedication could be very useful if registered and made available to the interested public.

MES development is essentially an interdisciplinary team activity. Therefore, it must be practiced as a principles-based approach, which we believe are essential to the *multi-person construction of multidisciplinary environments*. It is our view that such principles are much more important than any particular notation or methodology. In fact, these principles will enable the software designer to evaluate various techniques, tools and possible technological solutions, and to apply them when they are appropriate.

Such motivation led us to begin a careful inspection of our practices, trying to apply more systematic methods and then to organize the satisfactory results as a *set of heuristics for MES*. Such methods should integrate the coming methodology in order to suit it for the Music Education domain. Consequently, two main questions guided us in that investigation:

- How to build educational software, in general?
- How to build Music Education software, specifically?

In the present paper we will share some of these heuristics already found and being used in our projects. It's still not the final proposal of our development methodology, but rather a collection of important issues regarding MES development and some suggestions that address these issues. Surprisingly (or not!), many good solutions are proposed by researchers of our own country, Brazil. To make comprehension easier, they will be presented in the same sequence in which they would be applied to a real development project, following a software life cycle. We hope this

paper may resume some contributions from the field of Software Engineering to the Computer Music science's educational discipline.

The paper is structured as follows: In section 2 we present some issues to be considered before starting the developing process; section 3 describes heuristics for the analysis and design phases of the software life cycle; in section 4, implementation phase issues are discussed; section 5 suggests some techniques to be used in the evaluation phase and, finally, in section 6 some conclusive notes are presented.

2. Preliminary considerations

MES is *educational software*, and one must agree that this kind of software has some differences in comparison with more usual software. These differences refer to:

- Its *users*: students and teachers;
- Its *user interface*: should be more adaptive to the user than the generic user interfaces of other software;
- The *involved disciplines*: Cognitive Sciences, besides Computer Science and the specific domain treated in the software; and
- The *tasks* it helps the user to perform: the ones involved in teaching-learning, which is a delicate process that, in order to be successful, should avoid interference caused by problems in software usability.

As a consequence of dealing with the knowledge domain of Music, *MES* presents even more specific characteristics, such as an inherent employment of multimedia, a demand for a better quality of the employed sound/music media (audio medium) and a demand for correct synchronization between audio and visual media.

So, the process of *developing software for Music Education* has itself to be different from the ones for other software, considering the idiosyncrasies mentioned above. Thus, next we will present some overall concepts about this process, which are consequences of observing the particular requirements of MES.

2.1. Interdisciplinarity

One of the first things to have in mind, in MES development, is that one will be dealing with an *interdisciplinary* task. This interdisciplinarity occurs in two main levels: 1) a *knowledge interchange between the members* of the development team, which come from different knowledge domains (the multiple disciplines involved), may be noticed; and 2) the *expertise of each member* may be itself interdisciplinary.

In the kind of software discussed here, there are basically three generic disciplines involved: *Computer Science*, *Music* and *Education*. So, the members of a MES development project should come at least from these three areas. But they may also be specialized in disciplines that have their origin in the interconnections between these basic areas. Such are interdisciplinary knowledge domains that are very contributive in projects like these. For instance, in our group's latest projects, there have took part Computer Science experts in *Computer Music*, *Computers in Education*, *Human-Computer Interaction* and *Multimedia*, and musicians with expertise in *Music Education* and in the *Cognitive Psychology of Music*. A good discussion of our interdisciplinary experiences may be read in Krüger (et al., 1999).

Finally, sometimes forming such a group of experts in interdisciplinary domains is not possible. This could be the case of interested musicians and Music teachers who don't have contact with the Computer Science community. If this is the circumstance, then at least the developers of one area should talk to consultants or try to acquire knowledge themselves (like from books, magazines and symposia) in the other areas involved. Our experience shows that it will be difficult to get good results without following this interdisciplinary working approach.

2.2. Software life cycle

Another important issue to consider, before starting a MES development process, is the notion that, as for any kind of software, this process comprises several stages, or phases. Traditionally, these phases are: 1) requirements analysis and specification; 2) design; 3) implementation; 4) verification and validation; and 5) maintenance. Together, they make up what is called the *software life cycle* (Ghezzi et al., 1991, p.6-8; Sommerville, 1982, p.3-5).

We found recently a good work by Winckler, Nemetz and Lima (2000), where they suggest an *extension of the User-Centered Design* (UCD), considering the *differences between ordinary users and the student*, and so adapting that method to support the development of educational software.

This was done by extending two phases of the UCD software life cycle (Figure 1):

- The initial phase, that in UCD corresponds to "knowing the users and their tasks", was extended into *"knowing the users, the students and their tasks"*. Students are users with special needs, they should not be treated like ordinary users. Such special needs refer to the definition of the software's content and pedagogical approach. Besides, such a software may have different groups of users, some of them might be students, some not, like teachers, for instance. Therefore, in this initial phase, the different user groups and their necessities should be identified.
- The last phase, known in UCD as "usability evaluation", was extended as *"multidisciplinary evaluation"*. The authors have thought that, in the case of educational software, a broader evaluation should be carried out. One should evaluate, for instance, the promotion of students' interest, fitness of the chosen content and pedagogical approach.

Actually, in this work by Winckler and his colleagues, they propose the use of their extension on the development of the software's *user interface* (UI). But they defend that "one of the most important aspects of educational software is the interaction between the student and the system, and the dialogue that's established between the parts. So, to separate what is the software's UI from the software itself is often not so easy" (idem, p.19). In fact, their extended UCD life cycle may be applied to the whole educational software development process. One may even identify to which traditional phases of a software life cycle correspond the phases on the modified UCD life cycle, as seen in Figure 1 (traditional phases are italicized).

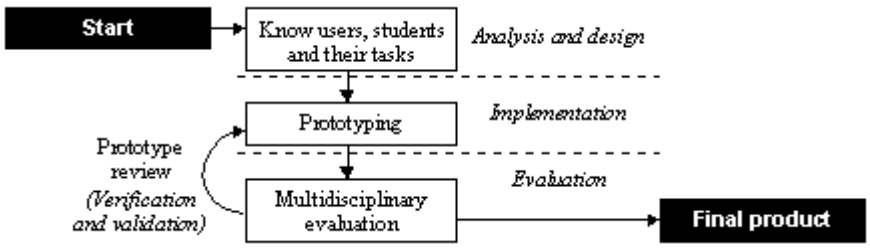

Figure 1 - Modified UCD life cycle, with corresponding phases of a traditional software life cycle.

3. Analysis and design

The requirements analysis and system design phases, together, match the "knowing the users, the students and their tasks" phase of the described modified UCD life cycle. Usually, in this phase, the software is planned, or designed. This design is based upon the software's requirements that, in turn, are defined by the developers after a context analysis. But what could be different in this process for *educational software* development?

In this case, the context analysis, for instance, should observe that the *users* are students and teachers, the *tasks* they perform are the ones involved in a teaching-learning process and the *context* in which this happens is (usually) an educational organization (school, university, college, etc.). Besides the user profile (that must include which are the users: students, teachers or both), should be verified some other context issues, like how the students' learning is evaluated, what

pedagogical approach does the school adopt, what is the school's hardware infrastructure (computer labs), and cultural aspects.

Questions that define the *new task* (because software usage will change the way the teaching-learning process was being carried out) should, in the same way, consider educational issues, like what *pedagogical approach* to adopt. This question, for example, helps to determine which kind of software to develop (which teaching-learning strategies it will apply). Gamez (1998, ch.4) suggests an agreement between the two main pedagogical approaches, behaviourism and cognitivism, and different kinds of software: tutorial and "drill and practice" systems follow an algorithmic approach, represented by behaviourist learning theories; simulation, educational games, exploration and discovery micro-worlds, and expert systems follow an heuristic approach, represented by cognitivist learning theories; and finally, Intelligent Tutoring Systems (ITS) combine both approaches.

It is obvious that Music plays an important role in MES, more than in other multimedia software, and therefore should not be forgotten in the requirements specification. Thus, one should define: which audio quality the Music educational task requires; which technologies should be used for sound emissions (e.g. digitized audio, MIDI, etc.); which Music notation to use (based on the one already known by the aimed students); and if the system has time constraints (Has the rhythm to be perfectly reproduced? Sometimes not, like when the content deals just with note pitches. Should musical events be perfectly synchronized with visual elements?). Metaphors to be used on the UI's visual design, for user interaction with the Music, must reflect what the user knows about this domain (that's why we need to do an extensive context analysis and user profiling). For example, guitar students may prefer to see chords represented (and to manipulate such chords) in a guitar tablature, rather than in a keyboard-like visual representation.

To help the definition of the software's *requirements* we suggest one to consult the recommendations' (guidelines, rules or heuristics) literature. "Guidelines characterize themselves as being the publishing of sets of suggestive (...) rules for the design of interactive systems. In general, such are very extensive publications, constituted of generic recommendations, empirically derived or validated by the scientific community" (Valiati et al., 2000). There are already such publications that aim *educational software* development. Our group suggests those proposed by Gamez (1998) and Valiati (2000), and one specific for MES, by Krüger (2000). All these are meant for software evaluation purposes, but may be used "a priori", in the following design process.

In software design, besides following guidelines, we've been having success in using Swanwick's (1979) *C(L)A(S)P Model* to define which musical activities to offer to the users. According to this author, in Music Education one should care to promote various musical experiences, in which the students could play several roles. His model classifies such experiences into five areas: Composition, (Literature studies), Audition, (Skill acquisition) and Performance. Brackets mean that those activities are secondary to the educational process, since they result in knowledge *about* Music, and not in a direct involvement *with* Music as the remaining ones. So, when designing MES, our group tries to offer most of these activities as system modules (or UI presentation units), as seen in Krüger (et al., 1999).

4. Implementation

This phase corresponds to the modified UCD life cycle's "prototyping" phase. Here, the developers will use the software's analysis and design resulting specifications. With this documentation, they should build a first prototype of the software, which is then evaluated, reviewed and redesigned, in a *cyclic development process*. In each cycle, new functionality is added and evaluated problems solved, originating several versions that evolve into the final product.

For this phase, what we verified is that *multimedia authoring tools* may be acceptably used. One of their advantages is that they incorporate very high level programming languages, along with visual programming, allowing even inexperienced programmers to build satisfactory systems. They are also fast-prototyping tools, and fit well in the prototyping cycle, since one may build a first, functionally poor version of the software, and continually add functionality until it turns into the final product.

A disadvantage of such tools, on the other hand, is that in some cases they may demand some kind of adaptation to fulfill musical requirements. They offer only high level of abstraction functions to the programmer and so, for Music, they usually only offer sound file manipulation functions (e.g. play, stop, pause, etc.). Such functions will be useless if someone needs to work at the level of musical events (handling MIDI events, for example). Consequently, it may be needed to extend the musical capabilities of such environments with libraries written in other programming languages. Flores (et al., 2000) describes such a case, in which a software built in Macromedia's ToolBook™ environment demanded a MIDI events handling DLL written in Borland's Delphi™.

5. Evaluation

In dealing with MES, the "multidisciplinary evaluation" phase of the modified UCD life cycle suggests that one should evaluate all the prototype's different aspects (usability, underlying pedagogic theory, etc.).

Experiments have shown that combining several evaluation techniques helps to identify a greater number of problems in a wider set of domains. For instance, Nemetz (et al., 1997) says that some usability problems are only identified in user tests. In an experiment with one of our software, user tests and heuristic evaluation, with both Music and usability experts, were made (Winckler et al., 1998). Many problems concerning the musical domain were identified by the Music experts, but these proved to be not as effective as the usability experts in identifying UI usability problems, showing that usability experts cannot be replaced.

A satisfactory combination for our projects' purposes has been the same used in Winckler's experiment: user tests; heuristic evaluation (Nielsen, 1997) with usability experts; and heuristic evaluation with Music experts. But we encourage the trial of other evaluation techniques as well (e.g. the ones described by Nielsen & Mack, 1994).

We may also not forget to mention the use of guidelines in an "a posteriori" checking of the software's attributes. As said before, there are already such guidelines specific for educational software (Gamez, 1998; Valiati, 2000), and even a script for MES assessment (Krüger, 2000).

6. Conclusions

Research about the use of *computers in education* already exceeded the questions concerning their utility for educational purposes (since, today, this utility is incontestable), and now investigate how they may be used to improve the teaching-learning process (Gamez, 1998, ch.4). Yet, it seems to us that the *development of educational software* is still rarely following any software engineering methodology. In fact, it is even difficult to set up such a methodology, because the development procedures, in this case, are very user/task/context-dependant, and this makes it hard for one to define any generic solutions. On the other hand, our research group's experience is verifying that some practices indeed help developers in building useful and usable educational software. Such practices were suggested in the heuristics presented in this paper.

Clearly, heuristics are not sufficient to drive the development of MES. In fact, these heuristics are general and abstract hints based on our group's experience in developing such software. Indeed, in order to improve the quality of MES, the software designer should be equipped with appropriate *methods* and specific *techniques* that help to incorporate the desired properties into processes and products. When methods and techniques are packaged together, they form a *methodology*, which purpose is to promote a certain approach to problem solving by preselecting the methods and techniques to be used. *Tools*, in turn, are developed to support the application of methods, techniques and methodologies. Our ultimate goal is to continue our work by searching other elements to complete our MES development environment: methods, techniques, methodologies and tools. In our point of view, however, such heuristics are much more understandable and easy to apply than any particular technique or methodology. In fact, these heuristics will enable the software designers to evaluate various techniques, tools and possible technological solutions and apply them when they are appropriate.

We hope this work may contribute to improve software development quality in this specific area and, finally, to improve Music teaching-learning as a whole.

References

Flores, L. V.; Viccari, R. M.; Pimenta, M. S. (2000). Extending the Musical Capabilities of a Multimedia Authoring Environment. In: Brazilian Symposium on Computer Music, 7., 2000, Curitiba, Brazil. *Electronic Proceedings (CD-ROM) of the XX Congresso Nacional da Sociedade Brasileira de Computação.* Curitiba, Brazil: Champagnat.

Gamez, L. (1998). *TICESE - Educational Software Ergonomic Compliance Inspection Technique.* M.Sc. thesis. Guimarães, Portugal: Universidade do Minho. Also available through HTTP at www.labiutil.inf.ufsc.br/estilo/Ticese.htm (In portuguese).

Ghezzi, C. et al. (1991). *Fundamentals of Software Engineering.* Englewood Cliffs, New Jersey: Prentice-Hall.

Krüger, S. E.; Fritsch, E. F.; Flores, L. V.; Grandi, R. H.; Santos, T. R.; Hentschke, L.; Viccari, R. M. (1999). Developing a Software for Music Education: An Interdisciplinary Project. In: Brazilian Symposium on Computer Music, 6., 1999, Rio de Janeiro. *Proceedings of the XIX Congresso Nacional da Sociedade Brasileira de Computação, v.3.* Rio de Janeiro: EntreLugar. p.251-264.

Krüger, S. E. (2000). *Development, Testing and Proposal of a Music Education Software Evaluation Script.* M.Sc. thesis. Porto Alegre, Brazil: Programa de Pós-Graduação em Música / UFRGS. (In portuguese).

Nemetz, F.; Winckler M. A. A.; Lima, J. V. (1997). Evaluating Evaluation Methods for Hypermedia Applications. Short Paper. In: *Electronic Proceedings (CD-ROM) of ED-MEDIA & ED-TELECOM.* Calgary, CA.

Nielsen, J. & Mack, R. L. (eds) (1994). *Usability Inspection Methods.* New York: John Wiley & Sons.

Nielsen, Jakob (1997). *Heuristic Evaluation.* Available through HTTP at www.useit.com/papers/heuristic/ (last accessed in Jan/1997).

Sommerville, I. (1982). *Software Engineering.* London: Addison-Wesley.

Swanwick, K. (1979). *A Basis for Music Education.* London: Routledge.

Valiati, E. R. A.; Levacov, M.; Lima, J. V.; Pimenta, M. S. (2000). Guia-GEPESE: User Interface Guidelines for Educational Software. In: *Electronic Proceedings of the II Simposio Internacional de Informática Educativa - SIIE'2000*, Puertollano (Ciudad Real), Spain: November.

Valiati, E. R. A. (2000). *Guidelines for the Development of Interfaces with Usability in Informative Hypertext/Hypermedia Educational Software.* M.Sc. thesis. Porto Alegre, Brazil: Programa de Pós-Graduação em Computação / UFRGS. (In portuguese).

Winckler, M. A. A.; Nemetz, F.; Lima, J. V. (1998). Case Study in Applying the Heuristic Evaluation Method to a Multidisciplinary Project. In: IHC'98, I Workshop of Human Factors in Computer Systems, 1998, Maringá, Brazil. *Proceedings...* Maringá, Brazil. (In portuguese).

Winckler, M. A. A.; Nemetz, F.; Lima, J. V. (2000). Learner-Computer Interaction: Interface Development and Evaluation Methods. In: *Digital Technology in Education* (presented at the IV Workshop Informática na Educação, Porto Alegre, Brazil: PGIE/UFRGS, sep/2000), chapter 1. Porto Alegre, Brazil: Gráfica UFRGS. (In portuguese). (To appear).