Software Evolution Sonification

Pedro O. Raimundo, Sandro S. Andrade, Renato Novais

GSORT Distributed Systems Group
Federal Institute of Education, Science, and Technology of Bahia
Av. Araújo Pinho, 39. Canela. Salvador – Bahia

{pedrooraimundo, sandroandrade, renato}@ifba.edu.br

Abstract. *Program comprehension is one of the most challenging tasks undertaken by software developers. Achieving a firm grasp on the software's structure, behavior and evolution directly from its development artifacts is usually a time-consuming and challenging task. Software visualization tools have effectively been used to assist developers on these tasks, motivated by the use of images as outstanding medium for knowledge dissemination. Under such perspective, software sonification tools emerge as a novel approach to convey temporal and concurrent streams of information due to the their inherently temporal nature. In this work, we describe how software evolution information can be effectively conveyed by audio streams; how music, software architecture concepts and techniques come together to achieve such means; and present a framework for sonification of extracted evolutionary metrics from software repositories.*

1. Introduction

Comprehending computer programs is a notoriously difficult task that involves gathering information from diverse sources (source code, documentation, runtime behavior, version history, just to mention a few) and gets progressively harder as the program's size and complexity grow [Stefik et al. 2011]. Synthesizing that information to tackle the development process effectively and efficiently is an endeavor that requires time, experience and, more often than not, peer support.

Tools are usually employed in order to help the decision making and understanding of the developers. Such tools range from built-in Integrated Development Environment (IDE) helpers and code metric viewers to complex software visualization solutions, focusing on conveying information to the user through the computer screen using tables, charts, drawings or animations due to their higher apprehension, if compared with a naive representation of the same data.

While such approaches are helpful in the comprehension process, aural representations of software structure and behavior have been shown to excel at representing structural [Vickers 1999], relational [Berman 2011] and parallel or rapidly changing behavioral data [Sonnenwald et al. 1990] even for individuals without extensive musical background [Berman 2011, Vickers and Alty 2002, Vickers and Alty 1998].

Software evolution adds another dimension to the problem since it forces the subject to add another layer of understanding to the structural aspects of a software program – the temporal layer. Common analysis requires, for example, information about the relationship between components of a system and lower-level information such as lines of code and fan-in/fan-out values per component. Evolutionary analysis can, for instance,

consume the results of the regular analysis procedures to yield higher-level information such as architectural drift in a given time slice, through the use of the project's meta-data (number of commits, bugs reported, time between releases, etc). Evolutionary analysis can also provide environment-related information which is particularly useful to manage large program evolution and continuing change [Lehman 1980] in computer systems.

In order to convey such information through the means of aural representations it is possible to use any kind of sound. Vickers and Alty proposed that these constructs are more effective if they follow culturally-appropriate styles, conceived analogously to words, which carry meaning to an individual native to or familiar with a specific language. Furthermore, they also found that western musical forms (based on the seven-note diatonic scale) are more readily recognized around the world [Vickers and Alty 2002].

We present a novel approach to transmit information about software evolution by exploiting sound's uniquely temporal nature and aural events such as melody, harmony and rhythm. Different events are used because each event has a distinct impact on the listener and they can be mixed and matched together to convey different streams of information, without being confusing or overwhelming.

The key contributions of this work are as follows. First, we propose a sonification framework in which the evolutionary aspects of a piece of software can be represented as sound streams, in an unobtrusive and noninvasive way. Second, we ran initial validation studies to assert that sonifications carry sufficient meaning to represent the evolutionary aspects of both large and small software projects.

The rest of this work is organized as follows. Section 2 details the pre-existing technologies and techniques utilized to reify the artifacts of this work, and summarizes the developed procedure. Section 3 briefly enumerates some of the implementation details and tools adopted in the framework's development. Section 4 details the validation procedure and presents an analysis of the obtained results. Section 5 summarizes the content and purpose of this paper, epitomizes the main ideas presented herein, and highlight some venues of future studies and experiments in this area.

2. Proposed Approach

This paper proposes a sonification framework that sonifies software evolution by following three main steps: source code retrieval, data extraction, and sound synthesis. The following sections detail the solution's general architecture and the sonification process.

2.1. Proposed Architecture

Figure 1 depicts the structural architectural view of the proposed framework, presenting three specialized software modules and the interactions between them and their shared working set. This approach allows each module to evolve and adapt independently, as long as the format of their shared data and their communication interfaces remains unchanged, and also enables independent reuse of each module. Details on the actual sonification process can be found in Sections 2.5 and 3.

2.2. Source Code Retrieval

Version control systems (VCS) create repositories by storing an initial version of the source code and then successive versions of the files by using delta compression. This

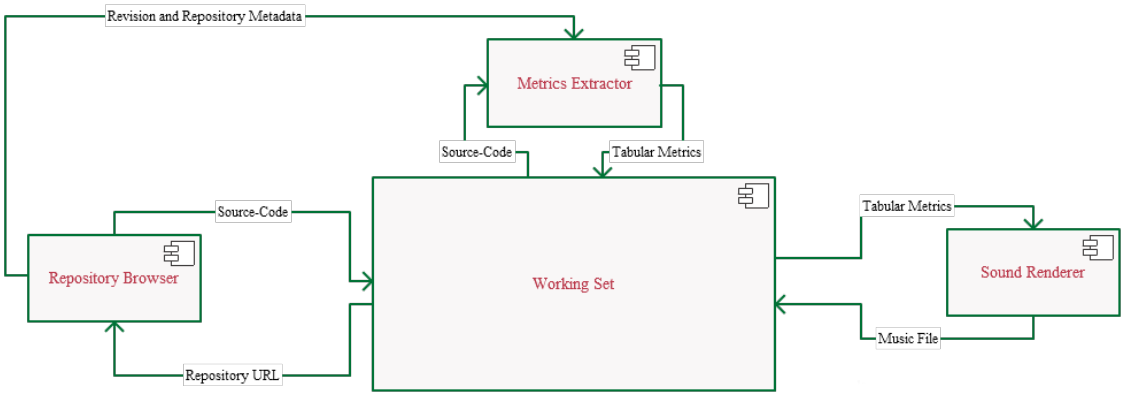

Figure 1. Structural view of the proposed framework.

allows for efficient storage and seamless switching between different snapshots of the versioned project.

Software evolution is an inherently temporal phenomenon. As such, it is fundamental to track the changes in a program's structure and code-metrics across a period of time to achieve a proper representation of it. This involves retrieving snapshots of the software's source code at different revisions. VCS greatly simplify this task since repositories created by such tools can be systematically transversed, processed and compared.

In order to extract meaningful data from software repositories and generate interesting sonifications, the metrics extractor module uses filters. While the framework includes some filters to demonstrate its capabilities, the ability to easily write and combine new filters is the highlight that allows the user to transverse source code repositories in order to achieve virtually any goal.

2.3. Data Extraction

The broad field of investigations that generally deal with extracting data from software repositories in order to uncover trends, relationships and extract pertinent information is called Mining Software Repositories (MSR). Software repositories refer, in this context, to the entirety of development artifacts that are produced during the process of software development and evolution (usually excluding by-products of the building process).

The content of these aggregated data sources exists throughout the entirety of the project's life cycle and carries a wealth of information that includes but is not limited to: the versions that the system has gone through, meta-data about the revisions of the software (as seen in Subsection 2.2), the rationale for project's architectural choices and discussions between the project's members. In this work, the focus is not on answering questions through MSR but using it to display the software evolutionary aspects focusing on the changes to properties rather than internal measuring.

Our approach does not, though, contemplate the specialized software evolution metrics developed by Lehman and Ramil in [Ramil and Lehman 2000]. The rationale is that those metrics focus heavily on cost-estimation and applied aspects of project management, whereas this work focuses on program comprehension.

2.4. Sound Synthesis

Martin Russ [Russ 2009] defines Sound Synthesis as:

> (...) the process of producing sound. It can reuse existing sounds by processing them, or it can generate sound electronically or mechanically. It may use mathematics, physics or even biology; and it brings together art and science in a mix of musical skill and technical expertise (...)

This is a broad, but sufficient, definition for the purposes of this work, in which the ultimate goal is to electronically generate sounds that are both meaningful and musical.

The artistic vein of sound and music allows sonifications to rouse specific emotions on the listener, the displays of technical expertise and emotion by musicians also greatly affect the impression left on the listener. While raw sound synthesis lacks these qualities, Vickers and Alty [Vickers and Alty 2013] go to great lengths to assert that the aesthetics and structural motifs of music itself determine the effects of sonifications on readers; along with cultural and psychological aspects and specific preferences of each subject.

In this work, sound is produced electronically through a tool for musical notation that utilizes its own Domain-Specific Language. This approach ensures that the aural metaphors developed can be systematically associated with the numbers that represent software metrics, without the element of *musical performance*.

2.5. Summarized Procedure

Working with the building blocks detailed above, the sonification process we propose herein can be summarized as the following automatic steps:

1. Download a source-code repository through a version control system;
2. Extract the desired metrics from the repository's meta-data or several versions of the source-code;
3. Process the extracted metrics along with a pre-defined sonification template.

Some implementation details of this process and the adopted technologies are detailed in the next section.

3. Implementation

The proposed framework was developed using the Java programming language, chosen due to its good balance of productivity, debuggability and the pre-existence of several libraries and components necessary for this implementation.

The initial implementation of the framework adopted Git as the Version Control System, the FreeMarker engine [Revusky et al. 2015] to process the sonification templates and the GNU Lilypond music notation tool [Nienhuys and Nieuwenhuizen 2015] to generate the aural output with the desired characteristics.

In order do run the exploratory study, presented in Section 4, the first implementation of the framework is a simple Java application that, given the URL of a Git repository, opens such a repository locally (downloading it over the Internet if not already present), extracts the desired metrics for a predefined period and generates the corresponding sonification of the extracted data in the *.ly* file format. Then, Lilypond processes this file and generates both MIDI and PDF files with the sonification results.

Figure 2. Graphical User Interface of the runner class.

This implementation includes extractors for two project metrics: commits per month and committers per month. In the context of open-source software, these two metrics provide insight on the overall health and the community around a software project.

The sonification template included maps the commits per month metric to a specific pitch, and the number of committers per month to a number of notes. Subsection 4.2 elaborates further on how this is represented on the finished sonification.

4. Validation (Exploratory Study)

A exploratory study was conducted to validate the proposed framework. The experimental procedure, their goals and our findings are detailed in the following subsections

4.1. Goals

The exploratory study was undertaken in order to assert whether or not the sonifications rendered from evolutionary data gathered from source-code repositories are meaningful and easily understood. Additionally, this exploratory study helps determine whether or not the basic implementation of the framework has enough assets to render useful sonifications.

4.2. Procedure

For this exploratory study, two *open-source* office suites were selected as subjects: the KOffice suite – whose developed ceased in 2013 and the LibreOffice suite – whose development is still well underway and is largely utilized by the open-source community.

We selected the period from *01/01/2010* to *31/04/2013* and generated a sonification of the interpolated evolutionary metrics. The seemingly arbitrary finish date for the sonification period corresponds to the month *before* the last non-automatic commit of the KOffice project, because KOffice's activity ceased midway through the month, whereas LibreOffice continued all throughout.

The monthly number of commits in each project was mapped to the pitch representing that specific month, while the number of committers for a given month corresponded to the number of notes it lasts for, as specified by the default template file. The extracted data was interpolated to make sure the minimum and maximum number of commits correspond to the pitches of C2 (two octaves below the middle C) and C8 (four octaves above the middle C), while the minimum and maximum number of committers correspond to, respectively, 2 and 8 notes; such interpolation strategy was utilized in the extractor classes to represent both projects in a similar perspective, even through their numbers were in slightly different orders of magnitude. Snippets of the musical scores for the sonifications can be seen in Figures 3 and 4.

The finished sonifications were qualitatively analyzed in order to determine if the desired mappings were correctly reproduced in the sonification, if both projects were

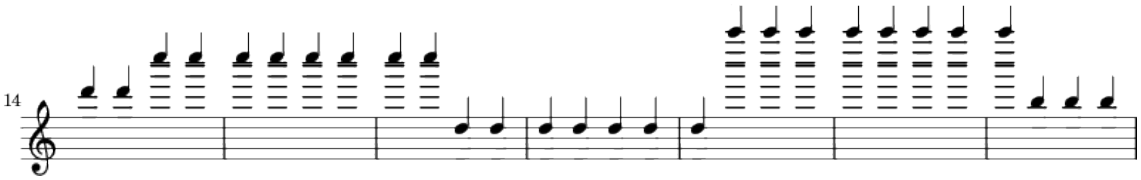

Figure 3. Snippet of the musical score for LibreOffice's sonification.

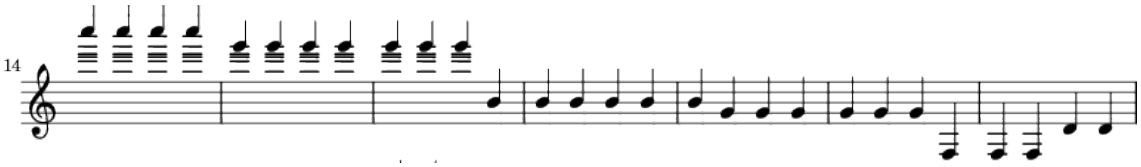

Figure 4. Snippet of the musical score for KOffice's sonification.

represented in a similar fashion, and if it is possible to get an insight on the project's health and activity through sonification alone. It should be noted that the musical scores are provided here as a printed alternative to the sound output, which is the focus of the work.

4.3. Results

A musical analysis of the sonification results is in order to further explain the impact of the elected aural metaphors in the process of program comprehension. In Figure 5, an annotated version of the previously shown snippet of LibreOffice's evolution is provided to support this analysis process.

First, let's take a look on the differences in pitch and their expected effects in the user of the auralizations. The blue and yellow highlights in Figure 5 correspond to the months with the least and the most commits, respectively; it is expected that this mapping feels the most natural to any individuals familiar with western forms of music, where higher tones are usually employed to rouse stimulant effects and lower tones inspire serenity and quietude.

The second aural event utilized in this mapping, number of notes, corresponds to the amount of people involved in a month's commits, and determines how many notes with the same pitch will be played in succession. This metaphor is closely related to sound's inherently temporal nature and, once trained, users should quickly internalize that more notes equals more people.

It should be noted that no additional effort is necessary to convey or understand temporal aspects in sonification. Once a user is trained to understand a specific set of

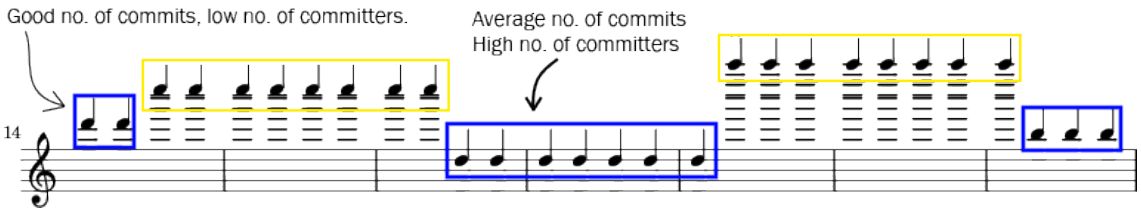

Figure 5. Anotated snippet of the musical score for LibreOffice's sonification.

metrics and aural events, the passing of the time is immediately perceived upon exposure to the sonifications.

One apparent shortcoming of these mappings is that two consecutive months with similar amounts of commits and different amounts of committers will show up as a single longer string of notes with similar pitch, this is by design because these larger repeated strings are *promptly* detected by users and may highlight a strong burst of activity or inactivity, a very important moment in the projects history or both. Months could be artificially separated by pauses or other musical elements, this was avoided until further empirical results confirm or deny the importance of this event.

While further experiments are necessary to confirm the effects of these mappings, previous studies have shown that musical auralizations with western characteristics are promptly apprehended by listeners of diverse backgrounds with or without previous musical training [Berman 2011, Vickers and Alty 2002].

Despite the project's numbers being in different orders of magnitude (LibreOffice has always had a lot more activity than KOffice), the sonifications were able to represent both projects in a comparable scale. If a software project evolves quickly and ascends to a larger scale, the latest sonification generated should be the one taken into consideration, since it will accurately represent this event by attributing lower pitches and less notes to the previous months, in contrast with the higher values of most recent ones. No additional training should be required to make use of the new sonifications as long as the mappings and metrics itself do not change.

Through analysis of the sonifications, some conclusions were drawn. First, it is possible to coherently map data to aural events. Second, through sonification it is possible to analyze large and small software projects under a similar perspective, preserving the evolutionary trends of each project and investing roughly the same amount of effort for each project.

From an evolutionary standpoint, the sonifications evidenced what could be an important pattern. In KOffice's sonification there were mostly extreme frequencies, meaning that there were many moments of heavy development and long periods of very modest development. In contrast, LibreOffice's sonification had mostly moderate frequencies, with the pitches varying in a roughly wavelike pattern, even in its most extreme moments, meaning that development sprints were cyclical and well-defined. More projects should be analyzed in order to confirm or deny the validity of these patterns. The *wav* files for the evolution sonification of LibreOffice and KOffice projects are available at `http://wiki.ifba.edu.br/sonification`.

5. Conclusion

In this work, we presented some of the pre-existing technologies and techniques that give grounds to the implementation of a sonification framework for software evolution, discussed several implementation details of the proposed tool and performed initial validation of the framework and its results, we ultimately conclude that sound is a medium worth investigating for the transmission of evolutionary aspects of software.

Future efforts planned in this line of work include a systematic literature review of all the studies that were revealed in this work, to foster future research efforts in the field, and further experimental works to assert the effectiveness of the proposed sonification

strategy, minimize the risks to the validation presented here, and try to uncover further evolutionary patterns that are better exposed by aural metaphors.

Acknowledgements

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES[1]), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08; and the Fraunhofer Project Center for Software and Systems Engineering at UFBA[2].

References

Berman, L. (2011). *Program Comprehension Through Sonification*. PhD thesis, Durham University.

Lehman, M. M. (1980). On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221.

Nienhuys, H.-W. and Nieuwenhuizen, J. (1996–2015). GNU LilyPond. `http://lilypond.org/`.

Ramil, J. F. and Lehman, M. M. (2000). Metrics of software evolution as effort predictors - A case study. In *2000 International Conference on Software Maintenance, ICSM 2000, San Jose, California, USA, October 11-14, 2000*, pages 163–172. IEEE Computer Society.

Revusky, J., Szegedi, A., and Dékány, D. (2002–2015). FreeMarker. `http://freemarker.org/`.

Russ, M. (2009). Chapter 1 - Background. In Russ, M., editor, *Sound Synthesis and Sampling (Third Edition)*, Music Technology, pages 3 – 86. Focal Press, Oxford, third edition edition.

Sonnenwald, D., Gopinath, B., Haberman, G., Keese, W., and Myers, J. (1990). *InfoSound: An audio aid to program comprehension*, volume 2, pages 541–546. Publ by Western Periodicals Co.

Stefik, A., Hundhausen, C. D., and Patterson, R. (2011). An empirical investigation into the design of auditory cues to enhance computer program comprehension. *Int. J. Hum.-Comput. Stud.*, 69(12):820–838.

Vickers, P. (1999). *CAITLIN : implementation of a musical program auralisation system to study the effects on debugging tasks as performed by novice Pascal programmers*. PhD thesis, Loughborough University.

Vickers, P. and Alty, J. (1998). Towards some organising principles for musical program auralisations. In *Proceedings of the Fifth International Conference on Auditory Display*.

Vickers, P. and Alty, J. L. (2002). Musical program auralisation: a structured approach to motif design. *Interacting with Computers*, 14(5):457–485.

Vickers, P. and Alty, J. L. (2013). The well-tempered compiler? the aesthetics of program auralization. *CoRR*, abs/1311.5434.

[1] www.ines.org.br

[2] http://wiki.dcc.ufba.br/FPC