Telling Software Evolution Stories With Sonification

Carmen Armenti, Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland

***Abstract*—The comprehension of software evolution remains one of the most challenging and time-intensive tasks in software development, further complicated by the sheer size and complexity of systems. Researchers have addressed the problem in several ways, using both static and dynamic analysis. Visualization has proven to be a promising technique, and over the years a myriad of approaches in 2D and 3D have been proposed, even extending in more recent times to virtual reality (VR). All leverage the most used human sense, vision.**

We present an approach that leverages our second most used sense, hearing, by mapping the information related to software evolution onto sounds, thus opening up an under-explored domain, the one of software sonification. Data about software evolution is intrinsically centered around time, which is one of the things that hearing as a sense is good at: processing information sequentially.

Our approach, implemented in a tool, enables sonifying the evolution of the files belonging to a software repository over time, utilizing harmonic chord progressions to represent code changes and identifying developers through musical instruments. We illustrate the feasibility of our approach with a case study and report on insights and reflections.

***Index Terms*—software evolution, software sonification, program comprehension**

I. Introduction

Program comprehension has been defined as a process where "to acquire new knowledge, knowledge is required" [1]. The understanding of software evolution is key for program comprehension. This process requires both general and software-specific knowledge. The former depends on the developers that possess it independently of the piece of software; the latter is their level of understanding and knowledge of the software under analysis. Program comprehension takes up a significant amount of time [2]. As software artifacts grow in size and complexity, analyzing their evolution becomes challenging and time-consuming, as first observed by Belady and Lehman [3], [4].

The human understanding process requires cognition strategies [1], [5] and is highly sensitive to the form in which information is conveyed to the senses [6]. The key is to leverage the perceiving phase of the human processing of sensory information to ease the understanding of complex software-related information. In the case of software visualization, the underlying theory is Gestalt psychology and pre-attentive visual processing, which emphasizes the human's mind ability to construct perceptions and abstract notions from low-level sensations in their totality, rather than focusing on individual pieces separately. Researchers have proposed several approaches to present software-related information visually. The nature of software systems is intangible, disguising software complexity [7].

By using visual metaphors, software visualization makes software tangible and reifies complex software-related concepts [8], [9], intending to amplify the cognition of abstract information through visual representations [10], avoiding cognitive overload, and supporting developers in program comprehension tasks [11].

Visual channels have been widely used to communicate information, starting with 2D graphical views [12]–[16], then expanding to 3D [17], and now extending into virtual reality (VR) [18]–[20]. Animations have also been employed for analytical [21]–[25] and educational purposes [26], [27], where the inherent dynamics of software is depicted.

As the perception of visual information is organized according to Gestalt principles, so can the perception of sound, and thus music [28]. The use of sound to reify information goes under the name of **Sonification**, *i.e., "the use of non-speech audio to convey information or perceptualize data"* [29]. Sound is apt to convey temporal information of changing events [30]. Humans process sequential information while *hearing* information [31]: The way in which sound is organized is processed earlier than the sound itself [32].

Our central idea is to leverage the fact that software evolution is driven by time-based changes, and that sound is inherently tangled to time: A perfect match.

However, sound as a communication medium has not been exploited as extensively as visuals in the field of program comprehension. The rationale for this is twofold: it pertains to the complexities of music and its theoretical framework, and it relates to the classical format of the scientific article, which is suboptimal for presenting such research. Nonetheless, we propose an approach to sonify periods of software development.

Our mapping of software information onto sound attributes focuses on the files that change within a given time period. We represent files as musical triads, *i.e.,* chords composed of three notes. Files are initially assigned a chord that encodes the number of lines of code (LOC) that they are composed of in a version at a specific moment in time (*i.e.,* in a commit). They undergo different types of change over their evolution, *e.g.,* additions, deletions, renaming, etc. We represent each change as a harmonic variation of the initial chord. As a result, the evolution of single files is sonified as harmonic chord progressions. Musical instruments playing the chords symbolize the developers who changed the files: one developer plays the piano, another the strings, etc.

Our idea stems from the combination of a sonification technique and a storytelling practice, which shapes the aural representation within a structured and sequential narrative.

The harmonic chord progressions that depict file evolution are designed following a *"setup, confrontation, resolution"* structure that is representative of the art of storytelling: While the setup is the initial chord definition, the confrontation is driven by additions and deletions on a file; finally, the resolution softens the tension generated by the modifications, converging to the initial chord the file was mapped to.

By leveraging the storytelling theory's emphasis on temporal event progressions and resolution, we implemented a parameter-mapping sonification approach that transforms the information about the sequential changes of file versions into auditive narratives.

II. Related Work

The origins of sonification can be traced back to 1954 when Pollack and Ficks undertook a quantitative study to investigate the feasibility of conveying information through multidimensional auditory cues [33]. Their results showed that the multidimensionality of sonification (the use of multiple parameters of sound) outperformed unidimensional displays.

In 2002, Vickers and Alty investigated the power of using sonification to communicate information. Their studies demonstrated that music can serve as a communication medium and that listeners do not require musical training to understand and benefit from auditory representations [34].

Brown and Hershberger leveraged music to ease the comprehension of algorithms [35]–[37]. Sonnenwald *et al.* proposed InfoSound, a prototype system that allows the creation of auditory interfaces, using music and sound effects to describe application program events that are difficult to detect visually using graphical interfaces [38]. Similarly, Baecker and Buchanan, with LogoMedia, expanded the program auralization approaches to clarify program behavior. They developed a system that maps program events to sound, allowing the execution of programs to be listened to [39].

Vickers and Alty developed CAITLIN, a tool that maps Turbo Pascal programs to musical motifs, to assist programmers during debugging tasks [40]. Berman and Gallagher developed a sonification mapping to describe the low-level architecture of Java programs and showed its usefulness in static program comprehension [41].

Researchers have proved that conveying additional information through sound, when coupled with visualization, can enhance the visual display to facilitate program comprehension tasks for developers [42]. Boccuzzo and Gall combined their visualization tool, CocoViz, with auditory icons to facilitate program comprehension [43]. North *et al.* proposed GitVS, a visualization augmented with sound. They showed that for some tasks, the sound makes the understanding of version histories easier [44]. McIntosh *et al.* proposed a sonification approach to highlight software evolution phenomena, focusing on the artistic qualities of the generated music [45].

When taking a broad look at the field, the heterogeneity of the research so far is striking. The "anything goes" approach is due to how little of the field has been explored so far, where the need for a more systematic approach is certainly due.

III. The Mapping of Visuals and Sounds

Among the approaches that have tackled the problem of understanding software evolution, software visualization has proven beneficial. The tactic of conveying information using visual aids and metaphors helps in comprehending troublesome and entangled information that is usually produced while developing. Humans process visual information better: Not only the majority of the information transmitted to our brain is visual, but our brain also processes images 60k times faster than text [46]. When leveraging the properties of visual elements—such as shape, color, dimensions, and borders (color and thickness)—as well as their organization in space (the layout), pre-attentive processing allows for the accumulation of all available information, highlighting what is most important [47].

The Concept of Mapping. Conveying information through a medium requires a mapping from the source domain (*e.g.,* the software system under analysis) to the target domain (*e.g.,* the visual world): The aids used to describe concepts from the original model need to have counterparts in the target domain. For instance, polymetric views are a valid example of mapping information visually [14]: They summarize relevant information from the source domain visually, directly relating to the information the visualization depicts. Metaphors are useful as well [20], as they provide the user with a concept of familiarity with the information displayed, which reduces the distance between the abstract concept being represented and its understanding, thus easing the process of comprehending the source domain model [18].

Sound Properties. Employing sonification is rather different: The auditory domain is rich in faceted properties that can be exploited. To begin with, a single sound has multiple properties that describe it if analyzed by itself: frequency and volume that give it "voice", and duration, that make the sound "alive". If other entities, *i.e.,* people, are considered, then sounds also have a spatial location, that defines the spatial neighborhood where the vibration is heard, and a timber, *i.e.,* the object or instrument that gives character to the sound by interacting with it physically. Taking into account music theory, the fan of properties unfolds: Sound frequencies are musical pitches (or notes) that are organized in octaves, that establish the range and pitch of audible frequencies for each instrument (*e.g.,* on a piano, octaves define the length of the keyboard). The relationships between pitches define musical scales and intervals: sequences of notes following a defined pattern, and the distance between two notes, respectively.

While discussing musical scales and intervals, harmony theory comes into play. Musical pitches not only are related to each other from a physical distance point of view but also from a theoretical one. Notes have harmonic relationships with each other. The concept of musical key arises: a group of pitches that form the base for a musical composition.

All of the above applies to a single instrument. If multiple instruments are considered simultaneously, concepts from musical composition theory must also be considered.

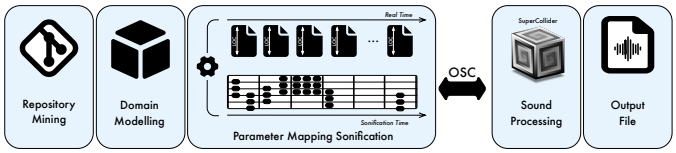

Fig. 1. Overview of the approach.

Telling Stories with Sonification. Figure 1 depicts the overall approach. We mine a collection of repositories from GitHub and applied Parameter Mapping Sonification [48] to our modeled data. Once the parameters are set, we perform sound processing relying on SuperCollider, which our model interacts with through the Open Sound Control (OSC) protocol. SuperCollider allows the generation of .wav files and the interaction in real time with the sonified model.

Time plays an important role when software systems are analyzed over their evolution. Software is made of components and analytics that are *involuted* in the time dimension since they are part of an evolutive process. Thus, their contextualization over the time dimension is essential. The first mapping we apply is from the real-time to the sonification time.

Time Discretization. The period selected is mapped to a defined and settable duration. The definition of the beats per minute (bpm) determines the tempo (or rhythm) of the "piece of music", and allows to set a scaling factor: one sonification-time second is mapped to *n* real-time seconds, hours, or days.

Time Manipulation. The discretization of time allows the organization of the series of file modifications (*i.e.,* the file versions modified in sequential commits) in the sonification timeline. However, commits do not last in time: they last moments. We dilate the commit time by temporarily distributing the modifications of a single commit over a "commit period", *i.e.,* the time elapsed between two sequential commits.

Sonification Mapping. Musical pieces possess both the syntax and semantics of "sound and silence". Here we present how we established these two components.

Syntactic Mappings. The timelines of software systems are not uniformly distributed: active and idle periods alternate. Accordingly, our mapping of time is a representation of alternating sounds and pauses (*i.e.,* periods of silence).

The file versions in a commit are associated with a sound whose duration depends on the period between two commits and the number of modifications in the current commit. The more file versions are changed in a commit, the shorter the pauses and the longer the sounds. For instance, if a commit modifies a significant number of file versions and the commit period is short, the sounds play for their entire sonification-time duration, and pauses are not represented. Sounds and pauses are organized in measures, which are the single unit of time used to organize music: Every measure features a specific number of beats (*i.e.,* notes) played at a specific tempo (*i.e.,* speed). For example, in a piece of music lasting 4 minutes at 120 bpm, every measure lasts 2 seconds and contains the notes and pauses whose total duration amounts to 2 seconds. Our sounds represent file versions in commits and play simultaneously during sonification.

Semantic Mappings. We map file versions onto triads. The LOC of the file when it is added to the repository (first version) determines the first note of the triad for that file.

A musical chord is a group of notes played simultaneously. A triad is a musical chord of three notes with fixed note distance between each other, depending on the type of triad (*e.g.,* major, minor, etc.). The first note of the triad is the *tonic note*. If it corresponds to the first note of the corresponding scale (each chord has a pertaining musical scale), then the chord is named *tonic chord*. The pitch of the tonic note is determined by mapping the LOC of the pertaining file version proportionally to the longest file version in the commit. The other two notes that are part of the triads are determined by following the standard pattern for major triads.

File Changes Mapping. We source information from GitHub repositories. The components of interest are *files*, which undergo modifications during software evolution. We leverage music intervals to encode this piece of information: An interval is a difference in pitch between two notes and an interval between two chords is defined by the difference in pitch between the tonic of each chord.

The histories of files are represented by harmonic progressions of chords that represent their changes over the selected development period. Git primarily distinguishes changes in files that are added, modified, renamed, or deleted. We preprocess the information about LOC of consecutive versions and establish distinctions of these macro groups: (1) Modified files encode the information about lines changed, but not the *overall* type of the change. We classify them as additive or subtractive, based on the LOC delta between versions. (2) The concept of "renaming" in git is identical to the one of "moving": we specify this difference. (3) We indicate the type of file versions tracked by merge commits as "merge".

Inspired by the *"setup, confrontation, resolution"* structure, we aim at conveying the narrative of consecutive changes on the same file by mapping types of modification to *types of chord*. In music theory, chords and notes are ruled by a key: the group of notes that can be used by the piece of music. The harmony theory categorizes chords within the same key into *tonal groups*, defining three categories: (a) the *tonic* group, that conveys an emotional sense of *stability*, (b) the *subdominant* group, that expresses *change*, and (c) the *dominant* group, that triggers *tension*. The chord built on top of the tonic belongs to the tonic group. We pick one chord for each tonal group to convey a taste of setup, confrontation, and resolution, associating each version modification type with a chord (we call them *types of chord*).

Newly added files are initially mapped to a chord depending on its LOC: this is a chord from the tonic group. Also, the deletion of files is mapped to a chord of this group, as in musical practices every piece starts and ends on the same note. We link additive files to the concept of "tension"; subtractive files are linked to a chord from the tonic group, but the feeling of this type of chord is similar to those that convey tension.

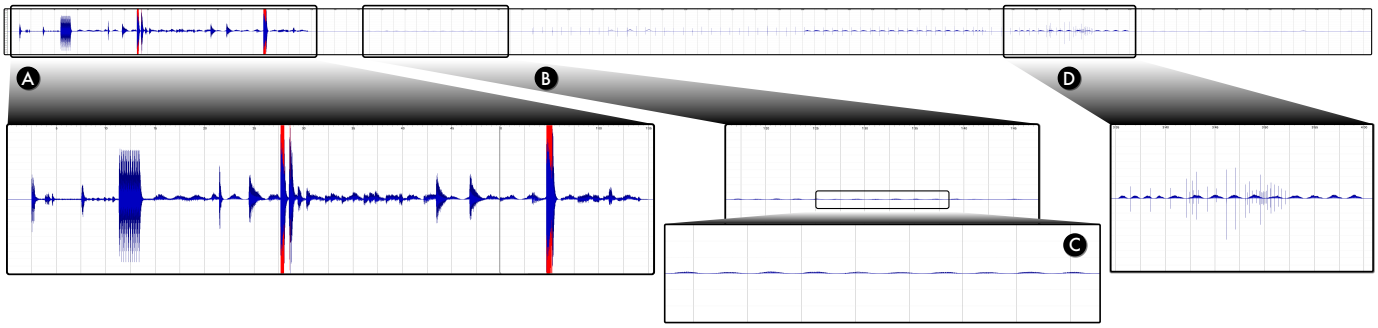

Fig. 2. Soundwave of the first month of development of the JetUML project.

Although tonic chords are typically considered to express stability, they are influenced by the tension of the nearby chords. Moved, renamed, or merged file versions are represented with a sense of "change", which adds a little strain to the stability conveyed by the tonic group's chords, but less than the tension represented by the chords in the dominant group. Every modification is a harmonic variation of the first chord (tonic chord), thus the history of each file is represented by a series of harmonic transitions.

We map developers to instruments. Our approach retrieves the top three developers based on the number of commits and associates them from the most to the least active to piano, strings, and percussions. All the other developers are mapped to a unique instrument, the flute. The last step is to define synthesizers in SuperCollider along with Open Sound Control (OSC) protocol methods for communicating with the SuperCollider server.

IV. Case Study

We illustrate our sonification approach on an example from a public repository of Java code: JetUML.[1]

We configure our sonification with a duration of 4 minutes and 120 bpm. We sonify the first month of development of the project: from January to February 2015.

Given the initial configuration, 1 sonification-time second corresponds to about 3 real-time hours. The period consists of 145 commits from 6 different developers and 890 file versions. Figure 2 depicts the soundwave of the sonification, where the periods we draw some conclusions from are highlighted. The piece of music can be heard on SoundCloud.[2]

Identifiability and Patterns of Developers. We use 4 instruments to map the developers that were active in the selected timeframe. Despite mapping many frequencies together, the instruments remain recognizable.

The LOC-to-frequencies mapping exhibits a positive correlation; hence, low notes indicate a small amount of LOC in the file, whereas high pitches reveal the opposite. During minute 1 (A in Figure 2), the piano produces strong chords and the strings have a softer touch playing high notes.

[1]See on GitHub: https://github.com/prmr/JetUML

[2]Hear on SoundCloud: https://on.soundcloud.com/WEWb2SRDS7Jdz5nJ8

This indicates that the "piano developer" modified smaller versions than those on which the developer mapped to the strings worked. Overall, the piano and strings were the most active developers throughout the month, while the flutes and percussions contributed briefly and at defined times.

Frequency and Magnitude of Changes. By hearing the track, it is clear who committed and when: there are patterns related to the frequency with which developers used to commit and the magnitude of the changes they performed. In the first period of development of about a week, the activity is more agitated (A in Figure 2). Then the activity shifts to a more relaxed rhythm: In the second minute of the track pauses alternate with delicate frequencies played by flutes (windy-like sounds). While the magnitude appears to be little, the regularity of modifications is consistent as it is evident by looking at the pitch picks in Figure 2 (B and C).

Between 3:35 and 4:00 (D in Figure 2), there is an overlap of patterns between the percussions (soft ticks) and the flutes. While the magnitude of the changes on the files sounds quite similar, as the frequencies they play are not much distant from each other, their "commit-regularity" is different. The percussions commit more changes as their rhythm is more persistent (many quick chords), while the strings commit less as their performance in the commit period lasts more time.

V. Discussion, Future Work and Conclusion

We present an approach to understanding software evolution using sonification, built upon music theory. While the approach is certainly in its early stages and will require much refinement, the preliminary results are surprisingly good at intuitively conveying complex information: 4 minutes of sound (not yet music, which is part of our future work) summarize the development activity of six people working on almost 1.000 file versions over 1 month. Hearing is a powerful sense, sound is a powerful medium. Based on prior research on developers' questions when changing codebases [49], our next step is to leverage software sonification and visualization to help developers analyze software evolution.

Acknowledgements. The authors would like to thank the Swiss National Science Foundation (SNSF) for the support via the project "FORCE" (SNF Project No. 232141) and the Swiss Group for Original and Outside-the-box Software Engineering (CHOOSE) for sponsoring the trip to the conference.

References

[1] A. Von Mayrhauser and A. M. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, 1995.

[2] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer–An investigation of how developers spend their time," in *Proceedings of ICPC 2015 (International conference on Program Comprehension)*. IEEE, 2015.

[3] L. A. Belady and M. Lehman, *Program evolution: Processes of software change*. Academic Press, 1985.

[4] M. M. Lehman, "Laws of software evolution revisited," in *Proceedings of EWSPT 1996 (European workshop on software process technology)*. Springer, 1996.

[5] H. A. Simon and A. Newell, "Human problem solving: The state of the theory in 1970," *American psychologist*, vol. 26, no. 2, 1971.

[6] J. Feldman, "The neural binding problem(s)," *Cognitive Neurodynamics*, vol. 7, 2013.

[7] T. Ball and S. G. Eick, "Software visualization in the large," *Computer*, vol. 29, no. 4, 1996.

[8] S. Benford, C. Brown, G. Reynard, and C. Greenhalgh, "Shared spaces: Transportation, artificiality, and spatiality," in *Proceedings of CSCW 1996 (Conference on Computer Supported Cooperative Work)*. ACM, 1996.

[9] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, 2009.

[10] S. K. Card, J. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[11] M.-A. Storey, "Theories, methods and tools in program comprehension: Past, present and future," in *Proceedings of IWPC 2005 (International Workshop on Program Comprehension)*. IEEE, 2005.

[12] N. J. Agouf, S. Ducasse, A. Etien, and M. Lanza, "A new generation of class blueprint," in *Proceedings of VISSOFT 2022 (Working Conference on Software Visualization)*. IEEE, 2022.

[13] S. Eick, J. Steffen, and E. Sumner, "Seesoft–A tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, 1992.

[14] M. Lanza and S. Ducasse, "Polymetric views–A lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, 2003.

[15] M. Lanza, "CodeCrawler–Polymetric views in action," in *Proceedings of ASE 2004 (International Conference on Automated Software Engineering)*. IEEE, 2004.

[16] M.-A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu, and M. Musen, "SHriMP views: An interactive environment for information visualization and navigation," in *Proceedings of CHI EA 2002 (Human Factors in Computing Systems)*. New York, NY, USA: ACM, 2022.

[17] R. Wettel and M. Lanza, "CodeCity: 3D visualization of large-scale software," in *Proceedings of ICSE 2008 (International Conference on Software Engineering)*. ACM, 2008.

[18] M. Lanza and R. Wettel, "Program comprehension through software habitability," in *Proceedings of ICPC 2007 (International Conference on Program Comprehension)*. IEEE CS, 2007.

[19] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. González-Barahona, and M. Lanza, "CodeCity: On-screen or in virtual reality?" in *Proceedings of VISSOFT 2021 (Working Conference on Software Visualization)*. IEEE, 2021.

[20] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza, "On the use of virtual reality in software visualization: The case of the city metaphor," *Information and Software Technology*, vol. 114, 2019.

[21] M. F. Kleyn and P. C. Gingrich, "GraphTrace–Understanding object-oriented systems using concurrently animated views," in *Proceedings of OOPSLA 1988 (Object-Oriented Programming Systems, Languages and Applications)*. ACM, 1988.

[22] G. Occhipinti, C. Nagy, R. Minelli, and M. Lanza, "SYN: Ultra-scale software evolution comprehension," in *Proceedings of ICPC 2023 (International Conference on Program Comprehension)*. IEEE, 2023.

[23] C. M. Taylor and M. Munro, "Revision towers," in *Proceedings of VISSOFT 2002 (International Workshop on Visualizing Software for Understanding and Analysis)*. IEEE, 2002.

[24] C. Armenti and M. Lanza, "Using animations to understand commits," in *Proceedings of ICSME 2024 (International Conference on Software Maintenance and Evolution)*. IEEE, 2024.

[25] ——, "Using interactive animations to analyze fine-grained software evolution," in *Proceedings of VISSOFT 2024 (Working Conference on Software Visualization)*. IEEE, 2024.

[26] J. T. Stasko, "Tango: A framework and system for algorithm animation," *ACM SIGCHI Bulletin*, vol. 21, no. 3, 1990.

[27] L. Végh and V. Stoffová, "Algorithm animations for teaching and learning the main ideas of basic sortings," *Informatics in Education*, vol. 16, no. 1, 2017.

[28] A. Schindler, M. Herdener, and A. Bartels, "Coding of melodic gestalt in human auditory cortex," *Cerebral Cortex*, vol. 23, no. 12, 2013.

[29] G. Kramer, "Auditory display: Sonification, audification, and auditory interfaces," 1994.

[30] W. W. Gaver, "The SonicFinder: An interface that uses auditory icons," *Human–Computer Interaction*, vol. 4, no. 1, 1989.

[31] C. M. Conway, D. B. Pisoni, and W. G. Kronenberger, "The importance of sound for cognitive sequencing abilities: The auditory scaffolding hypothesis," *Current directions in psychological science*, vol. 18, no. 5, 2009.

[32] I. Nelken, "Music and the auditory brain: Where is the connection?" *Frontiers in Human Neuroscience*, vol. 5, 2011.

[33] I. Pollack and L. Ficks, "Information of elementary multidimensional auditory displays," *The Journal of the Acoustical Society of America*, vol. 26, no. 2, 1954.

[34] P. Vickers and J. L. Alty, "Using music to communicate computing information," *Interacting with Computers*, vol. 14, no. 5, 2002.

[35] M. H. Brown and J. Hershberger, "Color and sound in algorithm animation," *Computer*, vol. 25, no. 12, 1992.

[36] M. H. Brown, "Exploring algorithms using Balsa-II," *Computer*, vol. 21, no. 5, 1988.

[37] J. A. Jackson and J. M. Francioni, "Aural signatures of parallel programs," in *Proceedings of HICSS 1992 (Hawaii International Conference on System Sciences)*, vol. 2. IEEE, 1992.

[38] D. H. Sonnenwald, B. Gopinath, G. O. Haberman, W. M. Keese, and J. S. Myers, "InfoSound: An audio aid to program comprehension," in *Proceedings of HICSS 1990 (Hawaii International Conference on System Sciences)*, vol. 2. IEEE, 1990.

[39] C. J. DiGiano and R. M. Baecker, "Program auralization: Sound enhancements to the programming environment," in *Proceedings of GI 1992 (Conference on Graphics, Visualization & HCI)*. Morgan Kaufmann Publishers Inc., 1992.

[40] P. Vickers and J. Alty, "Caitlin: A musical program auralisation tool to assist novice programmers with debugging," 1996.

[41] L. I. Berman and K. B. Gallagher, "Using sound to understand software architecture," in *Proceedings of SIGDOC 2009 (International conference on Design of communication)*. ACM, 2009.

[42] K. Hussein, E. Tilevich, I. I. Bukvic, and S. Kim, "Sonification design guidelines to enhance program comprehension," in *Proceedings of ICPC 2009 (International Conference on Program Comprehension)*. IEEE, 2009.

[43] S. Boccuzzo and H. C. Gall, "Software visualization with audio supported cognitive glyphs," in *Proceedings of ICSME 2008 (International Conference on Software Maintenance)*. IEEE, 2008.

[44] K. J. North, S. Bolan, A. Sarma, and M. B. Cohen, "GitSonifier: Using sound to portray developer conflict history," in *Proceedings of ESEC/FSE 2015 (Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM, 2015.

[45] S. McIntosh, K. Legere, and A. E. Hassan, "Orchestrating change: An artistic representation of software evolution," in *Proceedings of CSMR 2014 (Conference on Software Maintenance, Reengineering, and Reverse Engineering)*. IEEE, 2014.

[46] M. C. Potter, B. Wyble, C. E. Hagmann, and E. S. McCourt, "Detecting meaning in rsvp at 13 ms per picture," *Attention, Perception, & Psychophysics*, vol. 76, 2014.

[47] A. Treisman, "Preattentive processing in vision," *Computer vision, graphics, and image processing*, vol. 31, no. 2, 1985.

[48] T. Hermann, A. Hunt, J. G. Neuhoff *et al.*, *The Sonification Handbook*. Logos Verlag Berlin, 2011, vol. 1.

[49] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of FSE 2006 (14th International Symposium on Foundations of Software Engineering)*. ACM, 2006.