Sonification Design Guidelines to Enhance Program Comprehension

Khaled Hussein[1], Eli Tilevich[1], Ivica Ico Bukvic[2]
[1]Department of Computer Science
[2]Department of Music
Virginia Tech, Blacksburg, VA 24061
{khussein,tilevich,ico}@vt.edu

SooBeen Kim
Wellesley College
Wellesley, MA 02481
skim10@wellesley.edu

Abstract

Faced with the challenges of understanding the source code of a program, software developers are assisted by a wealth of software visualization research. This work explores how visualization can be supplemented by sonification as a cognitive tool for code comprehension. By engaging the programmer's auditory senses, sonification can improve the utility of program comprehension tools. This paper reports on our experiences of creating and evaluating a program comprehension prototype tool that employs sonification to assist program understanding by rendering sonic cues. Our empirical evaluation of the efficacy of information sonification indicates that this cognitive aid can effectively complement visualization when trying to understand an unfamiliar code base. Based on our experiences, we then propose a set of guidelines for the design of a new generation of tools that increase their information utility by combining visualization and sonification.

1. Introduction

The source code comprising modern software systems is among the most complex products of human intellect. Software maintenance is concerned with correcting problems and enhancing existing features in released software and constitutes the largest portion of the total software development effort and cost [6, 7, 28, 29, 55]. An essential prerequisite for maintaining a software system is understanding its source code. As a result, software comprehension has long been recognized as one of the most critical and time-consuming software development activities [33, 30, 47, 36].

Research efforts aiming at addressing challenges of software comprehension falls into two general categories. Some researchers study different cognitive factors that affect source code understandability, including readability, documentation, evolution, etc. Other researchers explore new techniques and tools that simplify program understanding and evaluate them empirically.

One of the traditional cognitive aids employed to improve program understanding is software visualization [39]. Several research studies, however, have shown that an additional information channel provided by sound can enhance a visual or haptic display [53, 52, 51]. To that end, this paper explores how a program comprehension tool can use sound to facilitate program comprehension and how vision should be supplemented with sound to achieve maximum benefit. Specifically, this work presents an approach to enhancing an Integrated Development Environment (IDE) with an auditory display. We report on our experiences of adding the ability to render sonic cues to Eclipse [42], so that sonification could supplement visualization to assist program understanding. Using the enhanced IDE, we then conducted a controlled experiment to assess the efficacy of sonification as a cognitive aid that assists program comprehension. In addition, we have reviewed a substantial body of the research literature in data sonification and program comprehension to understand the potential for these two research areas to cross-pollinate. Finally, based on the results of our study and the insights from the literature review, we have created a set of guidelines for guiding the creation of techniques and tools that harmoniously combine vision and sound to help the programmer in understanding an unfamiliar codebase.

Hence, this work makes no claims regarding the superiority of sonifciation over visualization or vice versa as a cognitive aid for program comprehension. Instead, we argue that these two presentation techniques should be combined to achieve maximum benefit for a large and diverse population of software developers. Although the general idea of adding sound to enhance a visual display has been studied in the past, the novelty of this work lies in applying this idea to the problem of program comprehension.

Although this paper reports on the initial results of an ongoing investigation, we believe that it makes the following contributions:

- An approach to adding the ability to render sonic cues to an existing Integrated Development Environment (IDE).

- A controlled experiment that demonstrates how sonification can be a viable tool for program comprehension.

- A set of guidelines for the design of software comprehension tools that increase their information utility by combining sonification and visualization.

The rest of this paper is organized as follows. Section 2 provides a background and discusses the state of the art in program comprehension, software visualization, and information sonification. Section 3 presents our approach to sonifying an IDEs and an implementation realized as an Eclipse plug-in. Section 4 details the controlled study we conducted. Section 5 presents a set of guidelines for constructing program comprehension tools that combine visualization and sonification. Section 6 outlines future work direction. Section 7 presents concluding remarks.

2 Background and Related Work

Program comprehension has long been recognized as an essential part of the software development process [47]. The effectiveness of program comprehension depends on a variety of diverse factors, ranging from source code readability to how effectively structural program information is retrieved and exposed [43]. In the following, we first give an overview of program comprehension and its challenges. Then we outline how software visualization has been employed to aid program comprehension. Finally, we introduce information sonification and its main concepts used in the paper.

2.1 Program Comprehension

In a comprehensive review, Storey [40] describes the state of the art in program comprehension, including its main cognitive theories, tooling strategies, and future trends. Of particular importance to this paper are cognitive models, the mental processes and information structures leading to a particular mental model, and their influence on software tools for program comprehension. One of the insights communicated by Storey is the need for more cognitive support in order to leverage the established cognitive theories. Among the main factors affecting the formation of the mental model is information representation. The following summarizes the related state of the art in representing information visually and sonically.

2.2 Software Visualization

Card et al defined *information visualization* as "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition" [9]. Software visualization is a subset of information visualizations that is primarily concerned with program details such as structure, algorithms, execution, and evolution [14]. Therefore, several researchers developed software visualizations that support program comprehension.

One of the main source code visualization techniques is the line-oriented technique, in which the visualization tool represents every single line in the source code. Eric et al proposed *Seesoft* [14] as a line-oriented source code visualization that was enhanced by several other tools such as *Augur* [18], *Aspect Browser* [19], and *Tarantula* [24].

In addition to line-oriented visualizations, researchers developed a variety of source code visualizations that study the source code from different perspectives. For example, *Visual Code Navigator* [31] is an example of block-oriented source code visualizations, in which it creates an annotated syntax tree of the source files and represents code structure in cushions. The SHriMP (Simple Hierarchical Multi-Perspective) tool employs both single view and multiple-view visualization techniques to expose program information at different levels of abstraction [41, 56].

Hundhausen et al investigate a number of software visualization effectiveness theories in program comprehension [21]. One of the theories of interest is *Dual-coding*, which emphasizes encoding information in both verbal mode such as textual displays and non-verbal mode such as pictures and icons. In our case, we are interested in adding another encoding mode, auditory representations.

2.3 Information Sonification

A recent article by Walker and Nees [50] provides an overview of the main concepts of sonification research and design. They define an *auditory display* as using sound to convey information and *sonification* as an auditory display that uses non-speech audio. According to Kramer at al., sonification is "the transformation of data relations into perceived relations in an acoustic signal for the purposes of facilitating communication or interpretation" [27].

A large body of sonification research has identified a set of scenarios, in which auditory displays are most effective [37, 26, 23]. For one, human hearing tends to be well-equipped to identify temporal information, making auditory displays particularly effective for rendering complex data patterns and events that require the user's immediate attention [17, 32]. In addition, an auditory display can be employed as a substitute when a visual display is not available, vision has already been engaged in some comprehension task [54], or vision has been overloaded with information [8]. Hearing has also been found to be better fit to process multiple concurrent inputs [16]. Finally, supplementing a visual display with an audio cue may increase tolerance for error [11].

deCampo [13] presents a Sonification Design Map that shows quantitative relationships between non-speech auditory displays. He starts with a traditional classification of sonification approaches (i.e., audification [26], parameter mapping, and model-based [20]) and then proposes to categorize sonification approaches on the bases of their respective data representations: continuous, discrete point data, and model-based. The sonification approach employed for this work falls in the model-based data representation, which mediates between the sonified data and the sound through a model based on the properties of the data. The model captures the domain knowledge of the sonified data and thus can be applied to different types of datasets.

When applied to computing, several prior approaches have used an auditory display to convey information about computer programs. Vickers and Alty [44] investigate how music can be used to communicate information about programming language structures, program runtime behavior, and locating bugs [45]. Their CAITLIN system aurolizes Turbo Pascal programs. This investigation has demonstrated that music can be a successful communication device, even for users who have not been formally trained in music. Compared to their work, this paper focuses on understanding computer code manipulated through an IDE, using sonifications that are significantly less-structured than music tunes, and finally employs sonification interactively (i.e., the sonic cues are rendered in response to specific user UI actions).

Finlayson and Mellish [15] have investigated approaches to representing programming constructs using speech and non-speech audio, concluding that the two modalities should be used together for maximum benefit. Berman and Gallagher [4] sonify program slices to improve program understanding. By contrast, this work focuses on interactive sonification, rendered in response to specific IDE user actions.

Considering the complexity of understanding modern programs and the need for new approaches to facilitate the task, the idea of using sound to aid software comprehension has been remarkably unexplored. In 2006, a working session titled "The Sound of Software: Using Sonification to Aid Comprehension" was held at the 14^{th} IEEE International Conference on Program Comprehension [3]. Nevertheless, to the best of our knowledge, three years later, this is the first publication citing that working session. One could see that the program comprehension community so far has not followed upon the ideas explored during that session. We believe that the main obstacle hindering the adoption of sound as a medium to aid program comprehension is that pursuing this research requires a multi-disciplinary team, with expertise in both music and software technologies. Unfortunately, few computer scientists possess enough expertise (or even interest) in music technologies, and few music technologies possess enough expertise in computing to be able to exchange ideas required for creating new technologies. This work is a result of an interdisciplinary collaboration, and next we report on some of its initial results.

3 Enhancing an IDE with Sonification Capacities

Developing large software projects usually involves using a number of visual tools, including IDEs, source control plugins, and file managers. Empirical evaluations and the practitioner's experiences alike show that using IDEs can significantly increase programmer's productivity. Microsoft Visual Studio and Eclipse are two examples of commercial and open source IDEs, respectively. In this work, we used Eclipse as our experimentation platform, an IDE to be enhanced with sonification capabilities.

Max/MSP [12] is a visual IDE designed specifically for audio and music-oriented applications. Created by musicians for musicians, it has since grown to encapsulate Quicktime and OpenGL, as well as to offer embedding of mainstream programming languages, including Java, JavaScript, Python, and Lua. MAX/MSP offers a large library of abstractions through a free SDK. In the area of music and interactive multimedia, MAX/MSP facilitates rapid prototyping of ideas and concepts, arguably one of its greatest advantages. It is the strengths of MAX/MSP as a digital signal processing engine and its ability to interface with other applications seamlessly that influenced our decision to use Max/MSP as the audio platform for the experiment.

Integrating Eclipse with sonic cues requires that Eclipse communicate with MAX/MSP. The Eclipse IDE supports a plug-in architecture that makes it possible to extend the IDE with additional capabilities in a systematic way. To achieve maximum flexibility, we put in place a simple client-server communication model using TCP sockets between our Eclipse plug-in and the MAX/MSP engine. To that end, we used a MAX/MSP package called *jitter* that provides scripts for processing network transactions. Specifically, the plugin sends numeric values representing program information to MAX/MSP, which then processes the numeric values to render the corresponding sonic cues. Because the MAX/MSP server is decoupled from the Eclipse plug-in client, the server could be reused for enhancing other IDEs with sonic cues in a cost-effective manner. Figure 1 demonstrates how our Eclipse plug-in communicates with the MAX/MSP server.

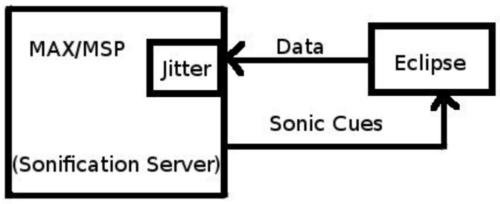


Figure 1. Enhancing Eclipse with sonification rendered through MAX/MSP using a client/server architecture.

4 Empirical Evaluation

The hypothesis under study in this experiment is that *as a cognitive aid to assist source code comprehension, information sonification can be as effective as information visualization*. To that end, effective in this experiment is measured in terms of reaction, comprehension, and user preference. Chewar et al defines reaction as *the response time to a notification* and comprehension as *situation awareness caused by accumulative perception of elements in the system* [10]. Positive results of this experiment could yield new insights about the use of sonification in creating program comprehension tools.

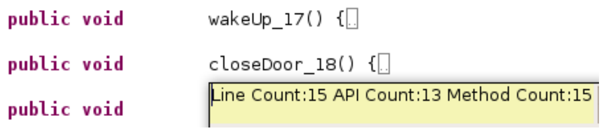
For this experiment, we visualized and sonified three pieces of program information:

1. the number of lines of code in a method;
2. the total number of method calls in a method;
3. a given API usage by a method (e.g., a total number of calling methods in a given package such as `java.utils.io`)

One could argue about the pragmatic value of helping the programmer understand these specific pieces of program information. Since our primary goal was to come up with a proof of concept, thus setting up a platform for further investigation, we may have traded some utility for simplicity. The details of the controlled experiment are presented next.

4.1 Experimental Design

To visualize and sonify the pieces of program information described above, we developed an Eclipse plugin. The plugin can be configured to render visualization and ignore sonification or vice versa. When the programmer hovers over a method, the plugin collects information about this method. Then, based on the representation mode, it either generates a visualization or sonification. When the plugin is in visualization mode, it generates a yellow text box similar to the one that displays JavaDoc documentation. This text box contains 3 numbers that represent the number of code lines, total number of method calls in this method, and the total number of calls to a `java.utils.io` methods respectively as shown in Figure 2.



```
public void        wakeUp_17() {
public void        closeDoor_18() {
public void        Line Count:15 API Count:13 Method Count:15
```

Figure 2. Eclipse plugin visualization of method information

When the plugin is in the sonification mode, it generates three sonic cues representing the 3 numbers collected. The sonic cues used were as follows: rain from the left speaker, water stream from the right speaker, and cello from the center. 3 shows the association of these sonic cues and the sonified information. The sonic cue volume increases with the increased numeric value of the represented information. However, we made sure that the volume will not exceed the comfortable zone for the average human hearing.

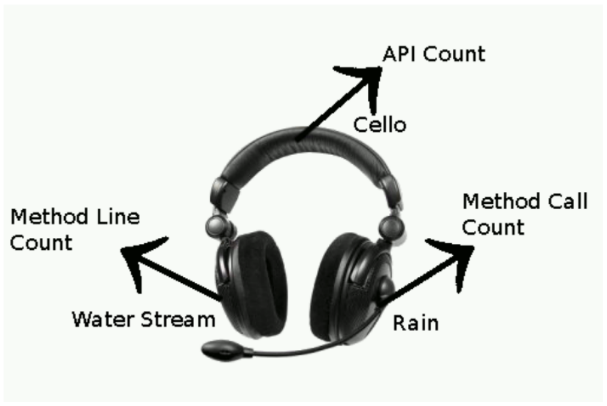


Figure 3. Association of sonic cues to sonified information

In this experiment, participants answered questions about an unfamiliar code base assisted first by a visualization and then by a sonification of the program's information or vice versa, with the order altered for each new participant. To avoid biased results, we created two similar Java source code files to be used for both visualization and sonification. Each source file contained 24 methods. The order of the methods in each source code file is changed and the method names are modified, but the average LOC per method, the average calls to the API, and the average method calls per method were kept constant.

4.1.1 Participants

For the study, we recruited 10 volunteers, 9 of whom were computer science undergraduate students from different universities, participating in a summer research program at Virginia Tech. One participant was a graduate student. The only prerequisite to participate in the study was to have a basic-to-intermediate experience using Java. The average age of participants was 21.4 years old (ranging from 19 to 24 years old). Although the demographics of the participants may have an effect on the generality of our findings, it is likely that the identified trends will persevere for more experienced programmers and larger code bases. The experiment sessions were conducted on an individual basis, and an experimenter was present during the sessions.

4.1.2 Materials

As a venue for the experiment, we used the DISIS laboratory [46], a specialized laboratory for conducting multimedia experiments. The laboratory walls are sound-proof to minimize any noise interference from outside. The standard equipment in the lab includes standard Windows and Apple workstation, with dual monitors. For the experiment, we used a workstation with Intel Pentium IV 1.8 GHz, 512 MB RAM, and 19“ dual monitor running at 1024x1024. The monitors were arranged, so that when a participant is sitting, the primary monitor would display the source code, and the secondary monitor would display the questions to be answered. The setup is shown in 4.

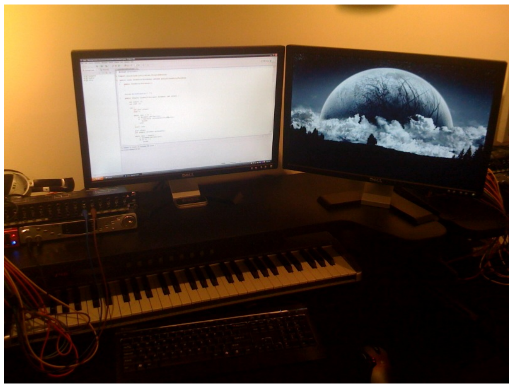


Figure 4. Experimental Setup in DISIS

4.1.3 Training Session

The experimenter provided a ten-minute training for the Eclipse plugin for each participant. During the training session, the participants were asked to interact with a training source code in order to get familiar with the sonic and visual cues. The training source code file contains 21 methods, in which each 3 methods are 3 examples for visualizing or sonifying a combination of method lines count, API usage count, or method calls count. At all times, the methods were collapsed, so that the participant would not look at the source code of each method, but rather focus on interacting with the visualization and the sonification to answer the questions. The participants were encouraged to adjust the volume on the head speakers to comfortable levels. However, the volume could still be adjusted during the experiment if needed.

Which 3 methods have the largest number of method calls?
Which are the 3 longest methods?
Which 3 methods have the most API usage?
Compare the lines of method and the number of method calls in (methodName)
Choose the correctly sorted list of characteristics of (methodName)

Table 1. Reaction Questions

Which method has the most lines of code?
Which method has the least API usage (not including methods with 0 API usage)?
Which method has smallest number of lines and method calls combined?
Which method did you perceive as the most crucial method of this program?
Which method did you perceive as the least crucial method of this program?

Table 2. Comprehension Questions

4.1.4 Procedure

Each participant was asked to fill a consent form. Then he/she was given a ten-minute training session. Then the participant was asked to start the experiment with either one of the following conditions:

Visual condition–the participant interacts with the source code using our Eclipse plugin started in the visualization mode. All methods were collapsed at all times. Participants interacted with the source code in order to answer the questions displayed on the secondary monitor. The questions were divided into two sections. While the first section contained 5 questions targeting the participant's reaction, the second section contained another 5 questions targeting the participant's comprehension. Table 1 and Table 2 list the questions used in the reaction and comprehension sections. The questions focus on the relation between the observed phenomena rather than on identifying discrete values.

Audio condition–the participant interacts with the source code using our Eclipse plugin started in the sonification mode. Similar to the visualization mode, all methods were collapsed at all time. Unlike visualization mode, yellow notifications were invisible.

In both conditions, when the participants were done with answering the questions in the reaction section, the experimenter instructed them to stop interacting with the source code, although they still could look at the method's names. Then participants would start answering the comprehension questions without source code interaction. At the end of each condition, the experimenter asked participants to relax for a couple of minute until the mode of the plugin was switched.

At the end of the experiment, the participants were asked a number of questions about his/her preferences. All participants were allowed and even encouraged to freely give comments at the end of each preference question. The experiment lasted for approximately 30 minutes per participant on average.

4.1.5 Variables

The experiment employs both independent and dependent variables. The independent variables are the specific visualization or sonification used to help a participant to answer particular questions. The dependent variables include the correctness, comprehension, user's response time, and user preference. To measure user preference, we asked each participant to fill a questionnaire using a subjective rating scale from 1 to 5. The following section presents and discusses our results.

4.2 Results

The results obtained from performing the empirical evaluation mentioned above are summarized in the following.

Correctness:

In this measure, we were interested in comparing the number of correct answers between visualization and sonification. Interestingly, we found 1-1 correlation between the two conditions. This implies that users were able to react the same way when using sonification or visualization. Based on the users' feedback, using visualization or sonification was easy to answer the questions. However, the practicality of using visualization is different than using sonification as demonstrated in the following.

Comprehension:

In this measure, we were interested in measuring the effect of using visualization or sonification on the users' awareness of the program information, including their memorization. After analyzing the data we found that users were able to answer all the comprehension questions, without interacting with the source code, correctly in both conditions. However, we have noticed differences in the participant average response time to the comprehension questions. Although we do not have significant time differences, users took from 5 to 11 seconds more to answer the questions when they used sonification. Interestingly, 4 of the participants mentioned that *"using sonification was easier and faster, but using visualization was more accurate"*.

Which method would you use if you're asked to answer the same questions again? Why?
How easy was it to distinguish between the 3 different pieces of data? (1=easy, 5=difficult)
Which method did you perceive as the least crucial method of this program?

Table 3. Preference Questions

User preference:

Table 3 shows the preference questions that participants had to answer at the end of each condition. Although the results show the participants' preference for visualization as an aid for answering the questions. their comments demonstrate that they are quite interested in using sonification. One explanation about the participants being somewhat skeptical about the practicality of sonification, is the sheer novelty of the approach. One participant stated this sentiment as follows *"[I prefer] visual, because I can easily read and understand. Using sound is too new for me."*

4.3 Discussion

Based on the results described above, there is strong evidence that sonification could be as effective as visualization if used in the right context and with the right program information data. All the participants found our visualization helpful in understanding the code. Based on some of the comments, we have reason to believe that supplementing visualizations with sonification will prove beneficial; it would expand the simultaneous perception channels, thus increasing the amount of information processed concurrently. For example, one of the participants stated that *"For the ones where I could scan them, I'd use the sound method. That way I was only concerned with looking at the methods and I could let my hearing comprehend the details. Visually, I'd have to keep track of both; which method I was looking at and the details. Splitting my cognitive resources in a good way, I guess."*.

The results of our experiment are not surprising, as they coincide with the documented results of other researchers who use sonification to represent information [1, 13, 16, 17, 25]. Sonically enhanced visualizations are definitely worth exploring, as they could lay the foundation for a new generation of program comprehension tools. To aid in the development of such tools, the following section discusses our proposed guidelines for using sonification in program comprehension tasks.

5 Designing Guidelines for Combining Information Visualization and Sonification in Program Comprehension Tools

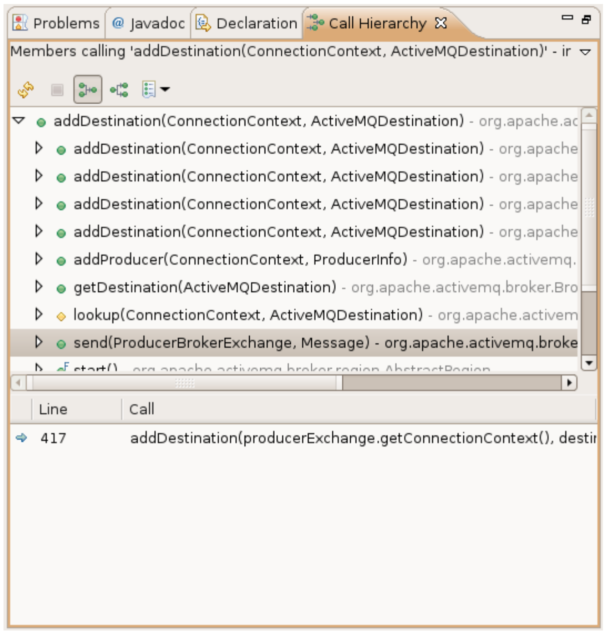
Based on our experiment and drawing on the insights from the data sonification research literature, we next propose a set of guidelines that will inform the design of program comprehension tools that combine visualization and sonification. These tools are likely to improve their information utility as compared to existing tools that use visualization as their only cognitive aid. We stop short of following a pattern-oriented methodology to describe these guidelines. Even though design patterns have been used to identify sonification best practices [2, 38], our guidelines aim at addressing the challenges that pertain to combining visualization and sonification in a single program comprehension tool. Therefore, at this point, we feel that more empirical evidence is needed to determine whether our guidelines are indeed patterns. In the following presentation, we first introduce a general principle and then illustrate it with a hypothetical example of applying the principle to building or enhancing a program comprehension tool.

5.1 Add sonification to simplify visualizations

Information visualization can become overly complex and unwieldy in several ways. For one, visualization can change the displayed content in the user's focal view, which can be disturbing for the user's attention management. Moreover, any visualization is necessarily confined by the number of available display pixels. Thus, a visualization designer has limited screen real estate available for displaying their visual representations. Therefore, we propose that sonification be used as an avenue for addressing these inherent shortcomings of information visualization.

For example, consider the Call Hierarchy View in Eclipse [42]. Upon requesting a call hierarchy on a given method, Eclipse generates a tree visualization of a call graph shown in Figure 5. The visualization, however, provides no information about the depth of the generated call graph. Assuming that the programmer needs to know how deep the tree is, she can only do so by expanding the call hierarchy tree, which may prove cumbersome and ineffective. Expanding a long tree, for example, is likely to conceal important nodes from the programmer's focal view. Adding a special visual cue that can show the depth of the tree is likely to clutter the tree visualization.

By contrast, a fairly straightforward sonification could effectively help address the issues above by issuing a sonic cue that represents the depth of the tree via a different volume or pitch. The programmer then can hover over a method and immediately recognize the depth of its call path. This additional cognitive aid has the potential to minimize the required interaction time by providing the user with useful information without complicating the existing tree visualization.



```
Members calling 'addDestination(ConnectionContext, ActiveMQDestination)' - ir
▽ ● addDestination(ConnectionContext, ActiveMQDestination) - org.apache.ac
  ▷ ● addDestination(ConnectionContext, ActiveMQDestination) - org.apache
  ▷ ● addDestination(ConnectionContext, ActiveMQDestination) - org.apache
  ▷ ● addDestination(ConnectionContext, ActiveMQDestination) - org.apache
  ▷ ● addDestination(ConnectionContext, ActiveMQDestination) - org.apache
  ▷ ● addProducer(ConnectionContext, ProducerInfo) - org.apache.activemq.
  ▷ ● getDestination(ActiveMQDestination) - org.apache.activemq.broker.Bro
  ▷ ◆ lookup(ConnectionContext, ActiveMQDestination) - org.apache.activem
  ▷ ● send(ProducerBrokerExchange, Message) - org.apache.activemq.broke
  ▷ start() - org.apache.activemq.broker.region.AbstractRegion

   Line     Call
 ⇨ 417      addDestination(producerExchange.getConnectionContext(), desti
```

Figure 5. Call Hierarchy View in Eclipse

5.2 Increase visual perception speed and accuracy by adding sonification

A study by Vroomen and de Gelder [48] has shown that the use of auditory cues can enhance visual perception. Although one could argue that few situations in software development warrant the programmer reacting immediately to some development-related event, such situations do occur, particularly in large software development projects that involve multiple developers. For example, multiple concurrent edits of the same source file complicates the subsequent merging of the changes. Because automated merge tools are often incapable of handling the resulting complexity, manual editing becomes necessary, a tedious and error prone activity that is better avoided. Although a visualization could convey the information about the developers who have checked out a source file for modification purposes (e.g., FASTDash [5], a new developer starting to modify a file is a real time event that has to be communicated.

We argue that a sonic cue could effectively supplement the information already conveyed by a visualization such as FASTDash. For example, when the programmer opens a file, a sonic cue representing the number of concurrent edits can be rendered. This additional cognitive aid could help the programmer keep her attention on the coding task at hand by conveying this information without having to switch one's attention to the visualization. Upon receiving the sonic cue, the programmer will be able to use the existing visualization to obtain more details about the concurrent edits.

5.3 Add sonification to present multiple information pieces simultaneously

The use of sound can improve comprehension and lower cognitive load when one has to monitor multiple information sources updated concurrently. Specifically, a visualization coupled with or relegated to aural cues has been found to be more efficient and less error prone [16, 49, 22]. Additionally, some researchers even suggest that listeners are capable of monitoring multiple audio streams better than single streams [1]. Musical scores, in particular, are known to convey lots of concurrent information together with their mutual relationships. In particular, rhythm is much more pronounced aurally than it is visually [34], which can facilitate more accurate cognition while lowering the overall cognitive load. For example, the part of our controlled experiment that dealt with the number of lines could have used an attack-based multiple beats sonification, compressed in a short time interval. Thus, if several different sources of information about a program have to be conveyed concurrently, a program comprehension tool could employ both visual and aural cues.

5.4 Use sonification to summarize information

Real-life programs often have large and complex code bases, with vast amounts of information that the programmer is expected to understand. Several software visualization techniques have been proposed to address the challenges of representing large data sets. Nevertheless, with the increased types and variety of information that has to be conveyed, software visualizations can quickly become extremely complex to provide relevant data summaries. For example, *linking and brushing* enables effective interactions with a visualization to retrieve manageable portions of information that can be displayed on a single screen. However, even a provided portion may still contain too much details, not all of which is relevant. For example, Eclipse text editors always display all the content of a source file.[1] For some program comprehension task, such as understanding which source files exceed a given number of lines or some cyclomatic complexity metrics, seeing an entire file is unnecessary and counterproductive.

[1]Eclipse presents a collapsed view of methods and classes, but still all the classes and methods are displayed.

A simple sonification could effectively complement a visualization by providing a summary for large volumes of information. For example, a sonic cue could be used to express a relative length or a cyclomatic complexity metrics of selected source files.

5.5 Interchange visualization and sonification to improve effectiveness

The information that can prove valuable when understanding an unfamiliar code base pertains to different properties of the code, all of which can use different visualizations. For example, Eclipse uses a call hierarchy view for call graphs, a class hierarchy view for class structures, and source control view for project repositories. The programmer, however, can make use of only a limited number of visualizations at a time.

Although audio may not necessarily convey more information, it covers more ground spatially (360 degrees in 3D while visual perception is anterior in nature), and as such could provide more distinguishable entry points for monitoring. This ability, however, also depends on other factors, including the specific sound and spatialization technology in use. A study by Kaper et al. [25] has observed that at times sound superseded visuals in terms of the amount of conveyed detail, while at other times the situation was the opposite. This insight, in and of itself, suggests that under certain circumstances, one could design aural cues, so that they are indeed more effective than their visual counterparts utilized in the same scenario. In particular, a program comprehension tool could replace multiple visualizations that fall short of improving comprehension with multiple sonifications at will, thus improving the overall cognitive utility of the tool.

5.6 Alternate visualization and sonification to improve accessibility

The importance of accommodating users with disabilities has been widely recognized. It has been estimated that 161 million people worldwide are either blind or visually impaired [35]. Some programmers could have either visual or auditory impairments. Thus, visual or auditory representations of the information could supplement the otherwise unaccessible cues for impaired users.

6 Future Work

This paper introduces a number of design guidelines for supplementing information visualization with sonification in program comprehension tools. As opposed to the somewhat simplistic design that we had to follow to create our prototype experimentation tool, the next logical step is to leverage our guidelines to create a realistic program comprehension tool that combines visualization and sonification. In designing this tool, it will be worth investigating the different types of information that can be provided to the programmer and various combinations and interactions of visual and audio representations. The effectiveness of the tool will be evaluated empirically, possibly leading to other guidelines and insights.

7 Conclusion

As software is getting more complex, the task of understanding code bases is becoming more difficult, requiring better tools and approaches. This paper represents an attempt to enrich visualization-based program comprehension tools with sonification. We have constructed and evaluated a prototype tool that supplements visual program information by rendering sonic cues. The empirical evaluation of our prototype indicates that information sonification can be at least as effective as information visualization at different levels, including correctness and comprehension.

We also have reason to believe that combining the different program representations in the same tool will require new guidelines and cognitive theories. As a first step in this direction, we have proposed a set of guidelines that can guide the design of next-generation program comprehension tools, combining information visualization and sonification. We hope that these guidelines will prove useful for researchers and practitioners alike, charged with the difficult challenge of reducing the burden of program understanding.

Acknowledgments The authors would like to thank Pardha S. Pyla and the anonymous reviewers, whose comments have helped improve the paper. SooBeen Kim was supported by the NSF (grant NSF-IIS-0552723).

References

[1] J. Anderson and P. Sanderson. Designing sonification for effective attentional control in complex work domains. In *Proceedings of the 48th Annual Meeting of the Human Factors and Ergonomics Society (HFES2004)*, pages 1818–1822, 2004.

[2] S. Barrass. Sonification design patterns. In *Proceedings of the 2003 International Conference on Auditory Display*, pages 170–175.

[3] L. Berman, S. Danicic, K. Gallagher, and N. Gold. The sound of software: Using sonification to aid comprehension. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 225–229, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[4] L. I. Berman and K. B. Gallagher. Listening to program slices. In *Proceedings of the 12th International Conference on Auditory Display (ICAD)*, 2006.

[5] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322, New York, NY, USA, 2007. ACM.

[6] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, 2001.

[7] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, 1981.

[8] S. Brewster. Using non-speech sound to overcome information overload. *Displays*, 17(3-4):179–189, 1997.

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

[10] C. M. Chewar, D. S. McCrickard, and A. G. Sutcliffe. Unpacking critical parameters for interface design: evaluating notification systems with the irc framework. In *DIS '04: Proceedings of the 5th conference on Designing interactive systems*, pages 279–288, New York, NY, USA, 2004. ACM.

[11] B. Connell, M. Jones, R. Mace, J. Mueller, A. Mullick, E. Ostroff, J. Sanford, E. Steinfield, M. Story, and G. Vanderheiden. The principles of universal design, version 2.0. *North Carolina State University, The Center for Universal Design, Raleigh*, 1997.

[12] Cycling '74 Inc. Max/MSP. `http://www.cycling74.com`, 2008.

[13] A. de Campo. Toward a data sonification design space map. *Proceedings of the International Conference on Auditory Display (ICAD)*, pages 342–347, 2007.

[14] S. G. Eick, J. L. Steffen, and J. Eric E. Sumner. Seesoft—a tool for visualizing line oriented software statistics. pages 419–430, 1999.

[15] J. L. Finlayson and C. Mellish. The 'audioview' - providing a glance at Java source code. In *Proceedings of the 11th International Conference on Auditory Display (ICAD)*, 2005.

[16] W. Fitch and G. Kramer. Sonifying the body electric: Superiority of an auditory over a visual display in a complex, multivariate system. *In Kramer G. (ed) Auditory Display: Sonification, Audification and Auditory Interfaces. SFI Studies in the Sciences of Complexity*, 18:307–326, 1994.

[17] J. Flowers, D. Buhman, and K. Turnage. Cross-modal equivalence of visual and auditory scatterplots for exploring bivariate data samples. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 39(3):341–351, 1997.

[18] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 387–396, Washington, DC, USA, 2004. IEEE Computer Society.

[19] W. G. Griswold, J. J. Yuan, and Y. Kato. Exploiting the map metaphor in a tool for software evolution. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 265–274, Washington, DC, USA, 2001. IEEE Computer Society.

[20] T. Hermann. *Sonification for exploratory data analysis–demonstrations and sound examples*. PhD thesis, Bielefeld University, Bielefeld, Germany, 2002.

[21] C. Hundhausen, S. Douglas, and J. Stasko. A meta-study of algorithm visualization effectiveness. In *Journal of Visual Languagues and Computing*, 2002.

[22] P. Janata and E. Childs. Marketbuzz: Sonification of real-time financial data. In *Proceedsing of the International Conference of Auditory Display (ICAD 2004)*, 2004.

[23] G. Johannsen. Auditory displays in human–machine interfaces. *Proceedings of the IEEE*, 92(4):742–758, 2004.

[24] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, New York, NY, USA, 2002. ACM.

[25] H. Kaper, E. Wiebel, and S. Tipei. Data Sonification and Sound Visualization. *Computing in Science & Engineering*, pages 48–58, 1999.

[26] G. Kramer. An introduction to auditory display. *Auditory Display: Sonification, Audification, and Auditory Interfaces*, pages 1–78, 1994.

[27] G. Kramer, B. Walker, T. Bonebright, P. Cook, J. Flowers, N. Miner, J. Neuhoff, et al. Sonification Report: Status of the Field and Research Agenda. 1999. *Prepared for the National Science Foundation by members of the International Community for Auditory Display*, 1999.

[28] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[29] B. Lientz and E. Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1980.

[30] J. Lionel E. Deimel. The uses of program reading. *SIGCSE Bull.*, 17(2):5–14, 1985.

[31] G. Lommerse, F. Nossin, L. Voinea, and A. Telea. The visual code navigator: An interactive toolset for source code investigation. In *INFOVIS '05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, page 4, Washington, DC, USA, 2005. IEEE Computer Society.

[32] B. Moore. *An introduction to the psychology of hearing*. Academic Press San Diego, Calif, 2003.

[33] D. R. Raymond. Reading source code. In *CASCON '91: Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research*, pages 3–16. IBM Press, 1991.

[34] B. Repp and A. Penel. Rhythmic movement is attracted more strongly to auditory than to visual rhythms. *Psychological Research*, 68(4):252–270, 2004.

[35] S. Resnikoff, D. Pascolini, D. Etya'ale, I. Kocur, R. Pararajasegaram, G. Pokharel, and S. Mariotti. Global data on visual impairment in the year 2002. *Bulletin of the World Health Organization*, 82:844–851, 2004.

[36] S. Rugaber. The use of domain knowledge in program understanding. *Annals of Software Engineering*, 9(1):143–192, 2000. 10.1023/A:1018976708691.

[37] M. Sanders and E. McCormick. *Human Factors in Engineering and Design*. McGraw-Hill Science/Engineering/Math, 1993.

[38] S.Barrass. Sonification design patterns. `http://c2.com/cgi/wiki?SonificationDesignPatterns`, 2006.

[39] J. Stasko, J. Domingue, M. Brown, and B. Price. *Software Visualization*. MIT Press, 1998.

[40] M. Storey. Theories, methods and tools in program comprehension: past, present and future. In *Proceedings of the 13th International Workshop on Program Comprehension IWPC 2005*, pages 181–191, 2005.

[41] M.-A. Storey and H. Muller. Manipulating and documenting software structures using shrimp views. In *Software Maintenance, 1995. Proceedings., International Conference on*, pages 275–284, Oct 1995.

[42] The Eclipse Foundation. Eclipse - an open development platform, 2008. `http://www.eclipse.org`.

[43] S. R. Tilley, D. B. Smith, and S. Paul. Towards a framework for program understanding. In *WPC '96: Proceedings of the 4th International Workshop on Program Comprehension (IWPC '96)*, page 19, Washington, DC, USA, 1996. IEEE Computer Society.

[44] P. Vickers and J. Alty. Using music to communicate computing information. *Interacting with Computers*, 14(5):435–456, 2002.

[45] P. Vickers and J. L. Alty. When bugs sing. *Interacting With Computers*, 14:793 – 819, 2002.

[46] Virginia Tech. Digital interactive sound & intermedia studio. `http://disis.music.vt.edu/main/index.html`, 2008.

[47] A. von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.

[48] J. Vroomen and B. de Gelder. Sound Enhances Visual Perception: Cross-Modal Effects of Auditory Organization on Vision. *Journal of Experimental Psychology Human Perception and Performance*, 26(5):1583–1590, 2000.

[49] A. Walker and S. Brewster. Spatial audio in small screen device displays. *Personal and Ubiquitous Computing*, 4(2):144–154, 2000.

[50] B. Walker and M. Nees. *Handbook of Sonification, In T. Hermann, A. Hunt, & J. Neuhoff (Eds.)*. New York: Academic Press, 2009.

[51] Y. Wang, S. Celebrini, Y. Trotter, and P. Barone. Visuo-auditory interactions in the primary visual cortex of the behaving monkey. electrophysiological evidence. *BMC Neuroscience*, 2008.

[52] C. Wickens, S. Gordon, and Y. Liu. *An introduction to human factors engineering*. New York: Addison Wesley Longman, 1998.

[53] C. D. Wickens. *Processing resources in attention, In R. Parasuraman & R. Davies (eds.), Varieties of attention,*, pages 63–101. New York: Academic Press, 1984.

[54] C. D. Wickens and Y. Liu. Codes and modalities in multiple resources: a success and a qualification. *Hum. Factors*, 30(5):599–616, 1988.

[55] N. Wilde, P. Matthews, and R. Huitt. Maintaining object-oriented software. *IEEE Software*, 10(1):75–80, 1993.

[56] J. Wu and M.-A. D. Storey. A multi-perspective software visualization environment. In *CASCON '00: Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, page 15. IBM Press, 2000.