Sonifying Performance Data to Facilitate Tuning of Complex Systems

Cody M. Henthorne

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Eli Tilevich, Chair

James Arthur

Ivica Bukvic

September 6, 2010

Blacksburg, Virginia

Keywords: Sonification, Performance Tuning, Empirical Study, Human Factors, J2EE, Enterprise Application Servers,

Sonifying Performance Data to Facilitate Tuning of Complex Systems

Cody M. Henthorne

ABSTRACT

In the modern computing landscape, the challenge of tuning software systems is exacerbated by the necessity to accommodate multiple divergent execution environments and stakeholders. Achieving optimal performance requires a different configuration for every combination of hardware setups and business requirements. In addition, the state of the art in system tuning can involve complex statistical models and tools which require deep expertise not commonly possessed by the average software engineer.

As an alternative approach to performance tuning, this thesis puts forward the use of sonification—conveying information via non-speech audio—to aid software engineers in tuning complex systems. In particular, this thesis designs, develops, and evaluates a tuning system that interactively (i.e., in response to user actions) sonifies the performance metrics of a computer system. This thesis demonstrates that *interactive sonification can effectively guide software engineers through performance tuning of a computer system.*

To that end, a scientific survey can determine which sound characteristics (e.g., loudness, panning, pitch, tempo, etc.) are best suited to express information to the engineer. These characteristics were used to create a proof-of-concept tuning system that was applied to tune the parameters of a real world enterprise application server. Equipped with the tuning system, engineers—not experts in enterprise computing nor performance tuning—were able to tune the server, so that its resulting performance surpasses that exhibited under the standard configuration. The results indicate that sound-based tuning approaches can provide valuable solutions to the challenges of configuring complex computer systems.

This work is based on a paper accepted to ACM Onward! 2010 [27].

Acknowledgments

My graduate student career lasted four years and occurred on two campuses. During that time I relied on the support and love from multiple people in my life. I firmly believe that without this unconditional support I would not have completed this research or manuscript. I would like to use this space to acknowledge and thank those who have helped me be successful in my graduate life and in my life as a whole.

First and foremost, I would like to thank my parents, because if it was not for them I wouldn't even be here in the first place. I'd like to thank my Mom for making sure I got all my homework done, and turned in all of the end of the year projects, even if she didn't know about them until three days before they were due. I'd like to thank my Dad for teaching me discipline and providing guidance whenever I needed it. My parents have always been there for me and supported me in my decisions and I love them very much.

Without my best friend, my high school sweetheart, my significant other, my wife, Irina Henthorne, I would not be the better man I am today. Irina has been a guiding light in my life for almost an entire decade and she has never lost faith in me or my ability to complete my academic and personal pursuits. I am grateful and thankful everyday that I get to come home from work and receive her unconditional love. I'd also like to thank my in-laws, the Zabel family, for supporting me and especially for being there for my wife and I when things got a bit to hectic.

Rick Swink and I have been best friends since we were two and even managed to stay friends after rooming together during our freshmen and sophomore years. Hanging out with Rick was my way to completely get away from research and school. He helped keep me sane and at the same time keep me motivated to complete my research. Rick has also provided me with a home away from home by giving me a place to crash when I visited Blacksburg. Additionally, I would like to thank the entire Swink family for practically being a second family to me while growing up. As a college student, you always enjoy a free meal and good company, and the Swinks gave both to me on multiple occasions.

My personal shrink and friend, Kevin Perrine, was always just a phone call away when I needed to vent or needed reassurance. I can't even begin to count how many times I relied on Kevin to enjoy the good times and help get me through the tough times, but if I had to guess it would definitely be more than 42. Kevin also constantly demonstrates to me that it is important to remember there is more to life than school and work.

When I wasn't stuck in the CRC, it was pivotal that I relax and enjoy the down time. I could always count on my roommates Nick Huth and Brian Wright to help in my non-academic endeavors while at Tech. They were always up for a round or two of video games, a set in Wallyball, or a Macadoo's Hindenburg challege. Just like Rick and Kevin, Nick and Brian were vital in keeping me sane while at school.

Of course, I have to thank my advisor, Dr. Eli Tilevich, for giving me the research bug. When I first started my graduate work I had no intention of writing a thesis. As this manuscript shows otherwise, Eli convinced me that the research track was the better way to go, and I have to agree with him. Eli has taught me many things over the years, some academic and some not, like what it really means to be a strong candidate. He has helped me grow as a researcher and writer. Eli also showed the most patience I have ever seen exhibited by a person. It took me four years to complete this degree and Eli was always encouraging me

along the way. I don't know how many other professors out there would have stuck it out. I'd also like to thank my fellow research group members when I was at Tech, Wesley Tansey and Andrew Hall.

This work would not have been possible without the help of Ico Bukvic and Pardha Pyla. Ico's abilities and insights when it came to music and sound manipulation were invaluable. I met Pardha when he was the TA for one of my first graduate classes at Tech. It just so happened that his PhD work was related and depended on data gathered during the class. I'm grateful that we were able to "turn the tables" and leverage Pardha's HCI skills to help guide this research.

I'd also like to thank Dr. James Arthur for taking an interest in this research and participating on my committee. Additionally, I'd like to thank all the staff in the Virginia Tech Computer Science department that help make the department as a whole run smoothly as well as help me push through paperwork.

While at Tech my roommates were my support group, after I left and started working at BBN, my co-workers took on that role. My co-workers at BBN put up with participating in my survey and experiments and most of all put up with me. I could always count on one of the guys listening to me rant, complain about classes, or talk about the headaches of completing this degree. I can only hope that I can extend the same courtesy to them as they did to me as they complete their degrees.

Finally, I would like to thank my employer, Raytheon BBN Technologies, for providing me an opportunity to apply and grow my abilities, and for providing financial aid.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software engineers have long recognized the utility and value of sound as a cognitive tool for understanding their programs. For example, hearing a cooling fan suddenly turning on or running for extended periods of time can indicate a portion of processor intensive program code. Additionally, an engineer can listen to the sound created by spinning media to detect the presence of disk thrashing. Seasoned software engineers tell anecdotes about using electromagnetic interference picked up by an AM radio to learn properties of executing programs. Thus, sound has always been used by engineers to help understand program execution.

More importantly, it is not just the fact that computers make sound, but that the sound emitted conveys useful information to the programmer. While the examples above show the use of sound produced as an incidental byproduct of the physical workings of a computer, it is possible to intentionally use sound to convey information. The utilization of non-speech sound to convey information is called *sonification* [43]. Additionally, *interactive sonification* is the continuous sonification of information where the information and resulting sonifications are immediately influenced by the actions of the listener [3, 66].

1.1 Sonification as an Under Explored Aid to Software Engineering

As a discipline, Software Engineering strives to improve the software artifact under development. What makes Software Engineering as much of an art as a science is that the exact meaning of *improving an artifact* is defined in the "eyes of the beholder." Common objectives for such improvements include shortening development time, reducing the number of bugs, improving maintainability through revisions, or increasing performance [8]. Following software engineering principals for a professional software company are a means to increase profits by increasing programmer productivity. Therefore, software engineering research aims at discovering novel approaches to achieve these objectives.

Software engineering entails multiple processes or activities that engineers can follow to create software that satisfies the needs of all the important stakeholders [25]. Key software engineering activities include but are not limited to design, implementation, testing, maintenance, and configuration. To explain the contributions of this research, software engineering activities have been grouped into two types.

In the first type of activity, the engineer reifies abstract information into concrete software engineering artifacts from existing understanding (e.g., creating class diagrams to document an engineer's mental model of the envisioned system, generating requirements based on the understanding of stakeholder needs, etc.). In the second type of activity, the engineer gleans new information from existing software artifacts (e.g., determining memory usage through the output of performance profilers, comprehending source code developed by other engineers, etc.). Both activities can be helped through novel approaches, techniques, and tools. However, it is the second types of activities that involves complex cognitive processes, which can be helped through approaches and tools broadly known as *cognitive aids*. A

cognitive aid helps a human user to better utilize her natural senses to perform a given task.

Vision has been used to great success to help with various cognitive processes, including those of common software engineering tasks. However, why should one not engage senses other than vision to help accomplish complex software engineering tasks? In particular, can any other senses be effectively engaged to provide useful cognitive aids? Additionally, there is an estimated 161 million people worldwide that are either blind or visually impaired [57] and could benefit from non-vision based cognitive aids like sonifications [74].

Recent research in sonification techniques have shown sonification's usefulness as a cognitive aid by representing financial data [32], "displaying" complex multivariate systems [23], aiding artificial life research [21], and exploring large data sets [29, 40]. Furthermore, other research has shown the applicability of sonification-based cognitive aids to facilitate some software engineering tasks such as comprehending source code [30, 6, 5] and understanding software architectures [7].

However the full potential of sonification as a cognitive aid remains under explored, and important questions remain unanswered. For example, how can sonification be used to facilitate understanding of a program's dynamic runtime behavior like optimal performance? Or, which sound characteristics are interpreted uniformly by a substantial percentage of software engineers? And finally, what software engineering techniques can be used to integrate sonification into software?

While these are just some of the questions related to software engineering and sonification, the goal of this thesis is to explore sonification as an aid for performance tuning and begin to answer the questions above by:

- Showing that interactive sonification can effectively guide software engineers through performance tuning of a running computer system.

- Investigating the underlying sonification characteristics which are uniformly understood by most software engineers.
- Deriving an iterative approach to building sonification-based software.

1.2 Thesis Objectives and Motivations

1.2.1 Using Interactive Sonification to Facilitate Performance Tuning

A system's performance depends as much on the efficiency of its design as it does on the precision of its configuration. Performance tuning, a key facet of configuration and can be categorized as a second type of software engineering activity, is the process of choosing configuration parameters that achieve the desired level of some performance characteristic. For example, properly choosing the size for a server's thread pool can increase throughput or decrease latency.

Formally, the problem of performance tuning can be expressed as a performance function $PF(cp_1, ..., cp_n, p_1...p_n)$, where cp_x represents a constant parameter that is imposed by the execution environment, and p_x represents a user controlled configuration parameter. The values cp_x and p_x can affect the PF's range. Thus, performance tuning becomes the act of choosing the proper values for the user controlled configuration parameters, p_x, which best maximizes the PF for a given set of constant parameters, cp_x. For example, the range of a server's PF can be the number of transactions processed per second; its example constant parameters include the number of processor cores, processor speed, and network bandwidth; and its example configuration parameters include timeout threshold, available memory, and the thread pool's size.

Performance tuning is notoriously hard due to the need to accommodate multiple deployment environments (constant parameters) and business requirements. Furthermore, pinpointing the exact source of inefficiency in a complex system can be non-trivial, and it is often unclear how the value of a configuration parameter affects system performance. Finally, the large number of performance metrics generated through benchmarking a system can often be difficult to understand or correlate to other performance metrics. As such, performance tuning fits within the second type of software engineering process as described earlier and as will be shown, can benefit from the use of sonification.

As stated previously, engineers have already used the incidental sound created by the mechanical workings of the computer to provide basic understanding of program performance. But what if executing programs could provide more meaningful performance information to the programmer through sound via sonification? Furthermore, what if the sonifications used were specially designed so that multiple performance indicators could be effectively discerned by the engineer? This would indicate the level of different performance characteristics at the same time to engineer. Further, what if interactive sonification was used to immediately reflect if performance was affected by the engineer changing a configuration parameter. In reference to the formal definition above, could a tuning system use sound to guide the user to adjust configuration parameters, so that their values would appropriately increase or decrease the range?

To answer these questions and to begin answering the more general software engineering questions posed above, it was necessary to design, develop, and test a proof-of-concept sonification-based tuning system called SoniTune. SoniTune is a novel tuning system that reifies a general software architecture that integrates four major components: a system under test, a workload generator, a tuning user interface, and a sound rendering engine. Specifically, SoniTune focuses on tuning a real world enterprise system—the GlassFish Application Server

[54]—which is widely used in real world installations. Utilizing the SoniTune prototype, evaluation subjects were able to successfully tune GlassFish to achieve a level of performance higher than that provided by the standard configuration.

1.2.2 Investigating Sonification Characteristics

However, prior to developing and evaluating SoniTune, it was necessary to determine the accuracy with which engineers can perceive sound properties as the means to convey performance information. This is necessary for two reasons, first, to obtain consistent results between different engineers. Second, an individual's response to particular sounds is subjective; it depends on factors including one's cultural background, level of music sophistication or training, and frequency of exposure to non-speech audio [26]. Since professional software developers come from all walks of life and have varying cultural and ethnic backgrounds, a sonification approach must be as broadly applicable as possible to be beneficial. This entails choosing those sound characteristic which are interpreted uniformly by a substantial percentage of software developers.

Therefore, one must ask which sound characteristics are most suitable for the task at hand, performance tuning. Although prior research has explored using sound and music to represent program execution [5, 22, 68], such results have to be confirmed experimentally to determine their effectiveness specifically as tuning aids. A systematic study was required to gather empirical evidence assessing the fundamental affordances of sound characteristics and their effectiveness when applied to performance tuning. The study involved more than 30 engineers with different levels of programming and music expertise. A statistical analysis of the participant responses revealed that there are indeed sound properties that are more effective for performance related sonifications, specifically pitch and panning. Using these re-

sults, it was possible for SoniTune to effectively convey performance information with sound for a broad range of software engineers with varying backgrounds.

1.2.3 Engineering Sonification-based Software

Only recently has research begun to explore the application of sonification to software engineering processes. Additionally, even less research has been conducted to explore which software engineering processes are best suited for developing software which uses sonifications. Since one of the products of this work is a proof-of-concept sonification-based performance tuning system, it provides a concrete example to abstract out potential software engineering processes. While one data point is not enough to establish a complete abstraction or process, the insights gathered from this particular experiment contribute to the growing body of knowledge so that software engineering processes can be created or adapted for building sonification-based systems in the future.

1.3 Thesis Statement

This thesis asserts the following statement:

> *Interactive sonification can effectively guide software engineers through performance tuning of a computer system.*

1.4 Contributions

This thesis makes the following contributions:

1. A systematic user study that assessed the accuracy of sonification to convey information; the results of the study can inform software system designers who want to integrate sonification into their systems.

2. A software architecture for tuning software systems with sound; which integrates a system under test, a workload generator, a tuning user interface, and a sonification engine.

3. A proof-of-concept prototype that demonstrates the utility of our approach and its applicability to real world systems.

4. An iterative software engineering process for developing effective sonification based tuning systems.

1.5 Overview of Thesis

Creating SoniTune consisted of multiple, iterative steps, each intended to answer the questions provided throughout this chapter, Chapter 1. Prior to developing SoniTune, an investigation was necessary to establish the fundamental affordances of sounds to a general population of software engineers. The design and goals of the investigation are described in Chapter 2 and the analysis of the resulting data are presented in Chapter 3. With the insights from the case study, the SoniTune prototype was built on a general sonification-based tuning system architecture and implemented for tuning a real world system as described in 4. A complete description of the SoniTune demonstration and the proof-of-concept's initial results is provided in Chapter 5. An iterative based software engineering process derived from the experience of building SoniTune and potential future opportunities for merging sonifications and software engineering are detailed Chapter 6. Finally, related work is dis-

cussed in Chapter 7 and the concluding remarks and future work are presented in Chapter 8.

Chapter 2

Investigation of Underlying Sonification Parameters

An individual's response to particular sounds is subjective; it depends on factors including one's cultural background, level of music sophistication or training, and frequency of exposure to non-speech audio [26]. Since professional software developers come from all walks of life and have varying cultural and ethnic backgrounds, a sonification approach must be as broadly applicable as possible to be beneficial. This entails choosing those sound characteristic which are interpreted uniformly by a substantial percentage of software engineers.

Therefore, one must ask which sound characteristics are most suitable for the task at hand, performance tuning. To that end, a systematic study was conducted that gathered empirical evidence assessing the fundamental affordances of sound characteristics and their effectiveness when applied to performance tuning.

2.1 Sound Characteristics

Sound characteristics provide a mechanism to describe sound and separate one sound from another. While multiple sound characteristics exist, the study primarily focused and evaluated five characteristics, two of which are combined as they influence each other:

Pitch/Tempo - "Pitch is defined as a sensory characteristic arising out of frequency, which may assign to a tone a position in a musical scale" [53] and "tempo is a term used to designate the rate of movement of the [sound]" [53]. For example, when the playback of a song is fast forwarded ("the chipmunk effect") the pitch and tempo of the music increases; when the playback of a song is slowed down, the pitch and tempo of the music decreases. For clarity, the reminder of this thesis will refer to the pairing of pitch and tempo as only pitch.

Loudness - "Loudness of a sound is the magnitude of the auditory sensation produced by the sound" [53]. Most people relate loudness to volume. While this is incorrect as volume combines multiple sound characteristic in addition to loudness, it may be easier to relate loudness to volume.

Panning - Panning is the spatialization of a sound in physical space. For example, if a listener is wearing properly functioning stereo headphones and only hears a sound in their left ear, the sound is panned entirely to the left. If sound is only in their right ear, the sound is panned entirely to the right.

Timbre - "Timbre is that characteristic of a tone which depends on its harmonic structure. The timbre of a tone is expressed in the number, intensity, distribution, and phase relations of its components" [53]. Timbre can be thought of the characteristics of sound which distinguish it from other sounds. For example, people use timbre tell

voices apart.

2.2 Study Objectives and Questions

Although several prior studies assessed the fitness of sonification techniques for various software engineering tasks [5, 22, 68], this study focused specifically on how sonification can help software engineers conduct performance tuning. In particular, the intent of the survey was to answer the following questions:

1. Which sound characteristics are most suitable to facilitate perception and comprehension in a performance-tuning context?
2. Which sound characteristics are most effective in expressing whether a value is increasing or decreasing?
3. How effectively can a user understand a mapping of sound characteristics to the performance metrics of a running application?

Question 1 was directly related to the overall goal of the survey in determining which sound characteristics are most suitable for performance tuning.

Question 2 was specific to determining which sound characteristic is best at representing the typical changes in system performance data. Generally, knowing how system performance data fluctuates in an increasing and decreasing direction is helpful to determining the value of the current system configuration. For maximum accuracy to be achieved across different users, it is vital to determine which sound characteristic best represent these increasing and decreasing fluctuations.

Question 3 aimed to determine if software engineers are capable of interpreting sonifications of a simulated program execution when provided with a direct mapping of a specific performance metric to a specific sound characteristic. In order to create a useful performance tuning tool for the greatest number of software engineers, it was essential to ascertain if they would be capable of using such a tool, as well as, which sound characteristic would provide the highest accuracy.

2.3 Study Methodology and Design

The study was presented to the participants as a series of five sections: demographic, general, panning, mapping, and general feedback. Each section was specifically designed to help answer the questions above.

2.3.1 Demographic Section

The aptly named demographic section consisted of a series of multiple choice questions intended to gather non-identifiable demographic information about each participant. This information included:

- Years of programming experience
- Programming environment preference
- Debugging environment preference
- Use of profiling tools
- Musical background (including listening habits, musical education, and ability to play instruments)

- Music listening habits **while** programming
- Music genre preferences
- Left or Right handed, and
- Use of hearing aids or existence of hearing problems

2.3.2 General Section

The general section determined which sound attributes participants naturally map to increasing or decreasing values. Participants were asked to listen to eight 10-second audio clips and indicate if they felt the clip represented "something" increasing a lot, increasing a little, staying constant, decreasing a little, or decreasing a lot. Each clip exercised one of four different sound characteristics: pitch, loudness, panning (spatialization), and timbre. Questions 1 and 2 covered increasing and decreasing pitch. Questions 3 and 4 covered increasing and decreasing loudness. Questions 5 and 6 covered the planning transition from right to left and from left to right. Questions 7 and 8 covered the transition between sounds with similar timbre and the transition between sounds with drastically different timbre. The questions did not inform the participant which sound characteristic was being exercised or what should be interpreted as increasing or decreasing.

Without providing a direct mapping, the participants' answers are a natural inclination of how they perceive each sound characteristic in relation to increasing and decreasing values. Analysis of the general section provided the evidence for answering *Question 2*.

Table 2.1: Panning values for panning section questions.

Question	Left Channel	Right Channel
1	25%	75%
2	100%	0%
3	75%	25%
4	50%	50%
5	10%	90%

2.3.3 Panning Section

The panning section consisted of five questions which enforced a mapping of sound coming from the left channel as low values and sound coming from right channel as high values. Prior to starting the questions, the participants were provided with sample clips which provided a baseline as to the lowest value possible (100% left headphone speaker), the medium value (50% left headphone speaker and 50% right headphone speaker), and the highest value possible (100% right headphone speaker). Then, the participant was asked to listen to five short sound clips and indicate on a scale of 1 (lowest) to 100 (highest) what value they felt the clip represented. Each clip only played for a few seconds and the sound was constant (i.e., the percentage of sound from each speaker remained the same). The panning values of the five clips are described in Table 2.1.

The panning section was designed to help answer *Question 2* and *Question 3*. For *Question 2*, the panning section is a supplement to the general section with an emphasis on the panning sound characteristic. For *Question 3*, the panning section provides initial results as to how well software engineers can understand an implied mapping (panning values to a numerical scale) for a point time. While *Question 3* is more concerned with determining if software engineers can understand a running program where sonifications would change while the the program executed, the panning section still provides helpful information towards answering *Question 3*.

2.3.4 Mapping Section

The mapping section consisted of six questions which asked participants to interpret the sonification of three performance metrics of a simulated program execution. The performance metrics included disk usage, memory usage, and network usage. The same basic sounds with distinct timbres represented the three performance metrics throughout the section except for those questions where timbre was the sound characteristic being exercised. While the sounds stayed the same, the sound characteristics being exercised were changed for each question. Analysis of the mapping section provided the evidence for answering *Question 3*.

Question Design

Each question consisted of three parts: baseline clips, simulated performance sonification, and sonification interpretation. The baseline clips consisted of three sound clips that instructed the participant as to which sounds and changes in sound characteristics mapped to increasing and decreasing disk usage, memory usage, and network usage. The mapping was conveyed to the participant by each baseline clip sonifying one of the three performance metrics starting at the lowest possible value, increasing to the highest possible value, and then returning to the lowest value. Participants were encouraged to listen to the baseline clips multiple times until they were comfortable with understanding the sonification mapping.

The simulated performance sonification consisted of a 30 second clip to be interpreted and understood by the participant. The clip sonified the performance metrics in a way that was congruent to that of the baseline clips. The participant was allowed to the listen to the clip multiple times, but they were encouraged to only listen enough times to be comfortable in answering the questions in the final part, the sonification interpretation.

The sonification interpretation consisted of three sets of questions intended to evaluate the

participants comprehension of the performance sonification at the beginning, middle, and end of the clip. For each portion of the clip, the participant was asked to indicate where they felt the usage level for each resource (disk, memory, and network) was on a scale of 1 to 10.

Question Descriptions

The first three question of the mapping section leveraged three different sound characteristics to sonify performance metrics: Question 1 leveraged loudness, Question 2 leveraged pitch, and Question 3 leveraged timbre. All of the first three questions used panning to separate the three performance metrics. Memory was panned entirely to the left channel (100% left headphone speaker), disk was panned to the middle channel (50% left and 50% right speaker), and network was panned entirely to the right channel (100% right headphone speaker). Question 1 and Question 2 sonified data with proportional increase and decrease in loudness and pitch respectively as each sound charactertics lends itself easily to this form of manipulation. Question 3 sonified value changes via transitioning between sounds with drastically different timbres.

Question 4 combined the sonification techniques from the first three questions into one question. More precisely, memory was sonified by changing pitch and panned entirely to the left channel, disk usage was sonified by changing loudness and panned to the center channel, and network usage was sonified by changing timbre and panned entirely to the right channel.

Question 5 extended the panning section by mapping performance data to the panning location. As with the panning section, the left channel represented low values and right channel represented high values. Unlike the panning section, the panning values changed as each sonified performance metric changed throughout the 30 second clip.

Question 6 combined all of the sonification techniques from the previous five questions and used pitch, loudness, timbre, and panning to sonify the performance metrics. When any of the three performance metrics increased, that metric was sonified by increasing pitch, increasing loudness, changing timbre, and panning to the right. When any of the three performance metrics decreased, that metric was sonified by decreasing pitch, decreasing loudness, changing timbre, and panning to the left.

2.3.5 General Feedback Section

The general feedback section collected subjective comments and suggestions about the survey. This feedback was used to gather qualitative data from software engineers about using sonification in general and specifically for performance tuning. Analsysis of the general feedback section aided answering *Question 1* and *Question 3* by collecting qualitative information which was impossible to gather from the other sections.

2.4 Study Implementation and Deployment

In order to gather inputs from as large and diverse a population of participants as possible, the study was implemented and deployed as a Web-based survey. This would allow any software engineer around the world to partake in the survey providing the best chances for getting participants with different backgrounds. The survey was implemented as a Rich Internet Application constructed using the Adobe Flex Framework [1]. The sonification files used for the survey's questions were custom designed using Max/MSP [17] and were embedded into the survey as MP3s.

With the advantage of greater exposure, unfortunately, came the challenge of control over

the participant's environment. To achieve meaningful results under these constraints, a comprehensive configuration section was added to the survey. The configuration section guided the participants in selecting appropriate volume levels to be used throughout the survey. Additionally, a sample stereo audio clip was played to identify if the headphones being used were stereo headphones and if they were oriented properly (e.g., left speaker on left ear).

To comply with the requirements of the Virginia Tech Institutional Review Board, the survey was deployed and its results stored on a secure server, running Ubuntu Linux 9.04 and Apache 2.2. The public IP address of the server was given to participants in recruitment e-mails posted to graduate student mailing lists at five universities (Virginia Tech, Georgia Tech, University of Maryland, University of California in Irvine, and University of Rochester) and professional software developer mailing lists at four companies (Google, Amazon, Microsoft, and Raytheon BBN).

Chapter 3

Sonification Investigation Analysis and Results

The analysis and results of the sonification case study are broken into two sections. The analysis of the data produced by the study is in the analysis section, Section 3.1, and the insights and conclusions based on the analysis is in the results section, 3.2.

3.1 Analysis

Overall, 42 subjects participated in case study. However, only 33 of these participants successfully completed the survey. Nine participants' answers were removed from analysis due to incomplete answers or skipped questions. The survey framework tracked the number of times a participant listened to a sound clip for each question. Thus, it was possible to determine if the participant listened to a sound or not regardless of their answer.

If a participant did not listen to the necessary sound clips for a specific question, all of their

Table 3.1: Distribution of participants' programming experience.

Years of Exp.	**# of Participants**	**Percentage**
0-1 Years	1	3%
1-2 Years	3	9%
2-5 Years	7	21%
5+ Years	22	67%

Table 3.2: Distribution of participants' music experience

Music Exp.	**# of Participants**	**Percentage**
None	1	3%
Casual	4	12%
Intermediate	17	52%
Expert	11	33%

answers were excluded from analysis, including successfully completed questions. While this may be a strict requirement, it was necessary so that participants who gave up half way through the survey or blindly answered questions would not affect the overall analysis and results of the study. Except for this restriction, there were no additional criteria for excluding participant data from analysis.

After removing the nine sets of answers, the data from the remaining 33 participants was analyzed.

3.1.1 Demographics

The programming and music experience of the participants in the survey are summarized in Table 3.1 and Table 3.2 respectively. The programming and music experience are highlighted here as they provided the best description of the overall population of participants. For the full results from the demographics section of the survey please see Appendix A.

3.1.2 General Section

Analysis of the general section provided a distribution of participants who were asked to map increasing and decreasing values to particular changes in sound characteristics. As described in Section 2.3, the general section of the survey was a collection of multiple choice questions which asked participants to indicate how they felt changes in basic sound characteristics mapped to "something" increasing and decreasing.

Analysis of this section was fairly straightforward, as the distribution of participants that selected each choice for a given question was calculated. See Figure 3.1 for a summary of the analysis of the general section.

Due to the overwhelming number of similar responses, the analysis of the pitch increasing question (Q1) and pitch decreasing question (Q2) of the general section have been highlighted in separate column charts in Figure 3.2 and Figure 3.3 respectively.

3.1.3 Panning Section

The panning section was the first section of the survey which enforced a specific mapping from sonification to data value. The section consisted of five questions where the participant was asked to indicate on a scale of 1 to 100 the data value they felt the sonification represented. The analysis of this section involved comparing the expected value to the average value of all participants as seen in FIgure 3.4.

3.1.4 Mapping Section

The mapping section consisted of six major questions each with nine subquestions. These questions asked the participant to rate their observed usage levels of three resources (disk,

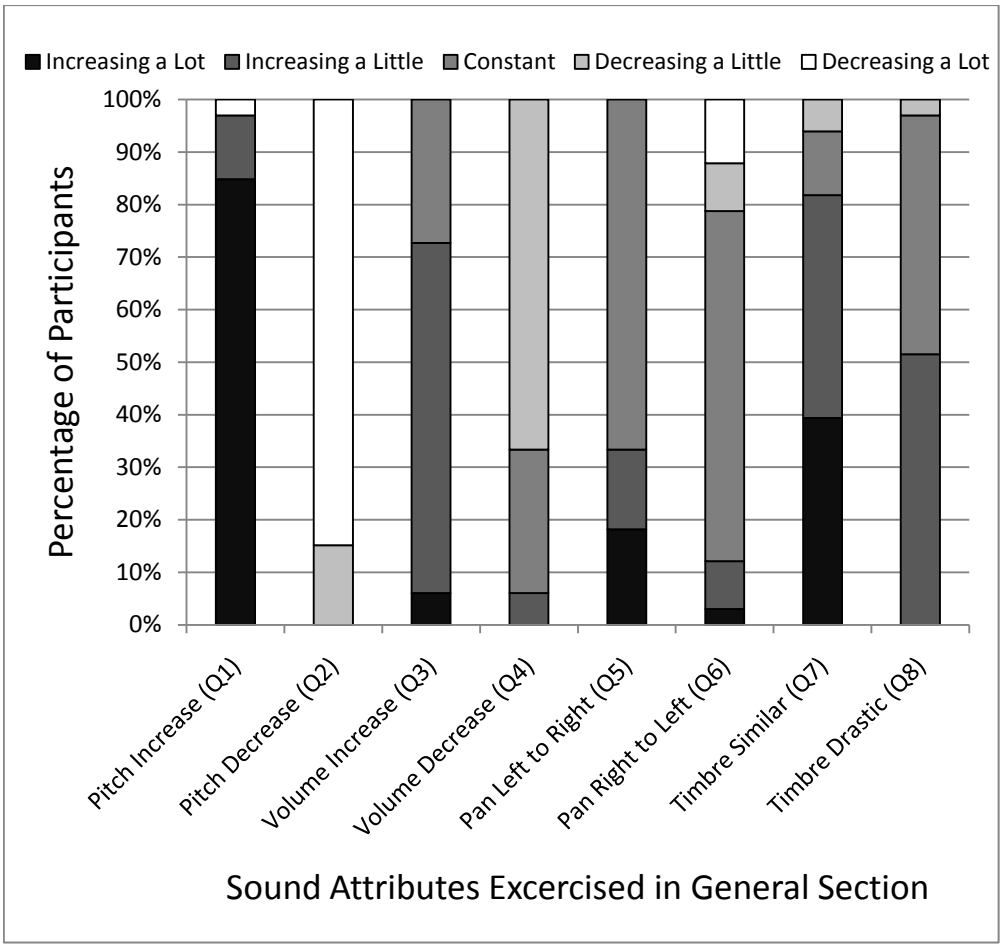

Figure 3.1: Distribution of participants' responses to all questions in the general section.

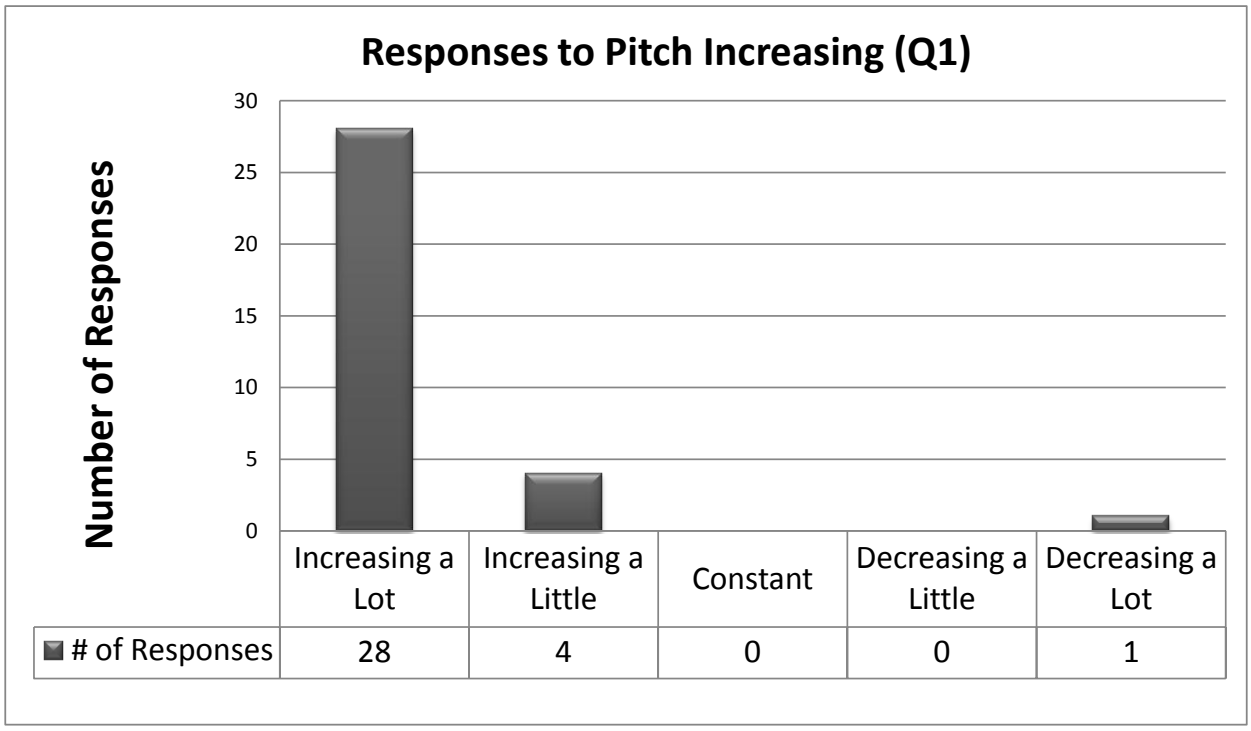

Figure 3.2: Distribution of participants' responses to the pitch increasing sonification in the general section.

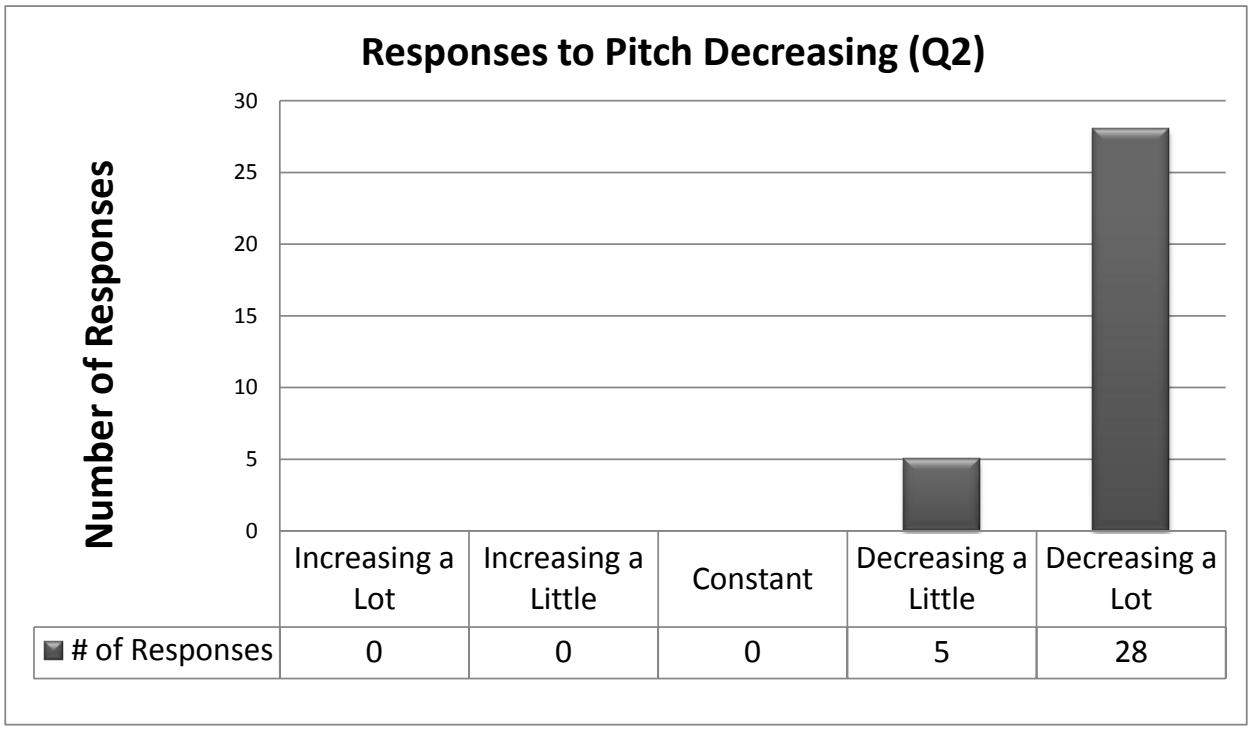

Figure 3.3: Distribution of participants' responses to the pitch decreasing sonification in the general section.

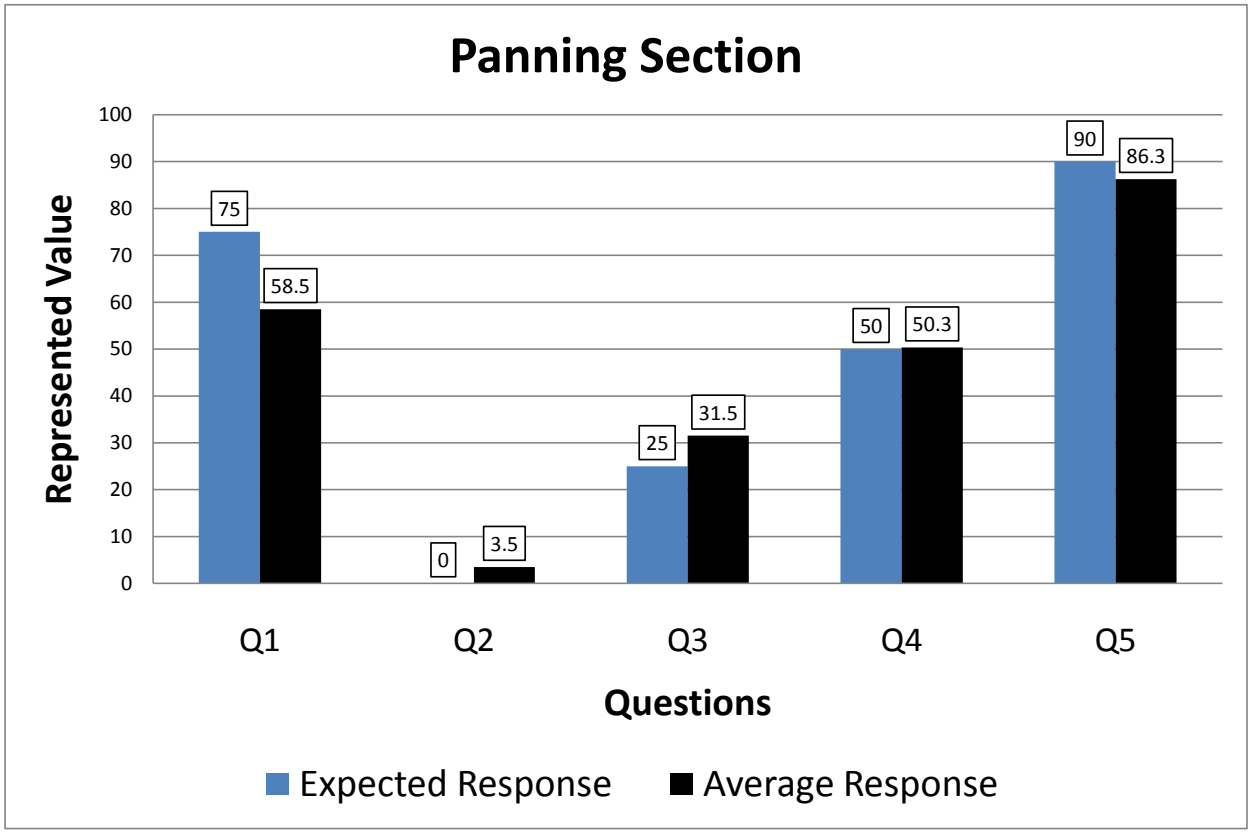

Figure 3.4: Expected value compared to the average.

memory, and network) on a scale from 1 to 10 during the beginning, middle, and end of a 30 second program sonification clip. For clarity, the nine subquestions when referenced individually, will be referred to as B-DISK, B-MEM, B-NET, M-DISK, M-MEM, M-NET, E-DISK, E-MEM, and E-NET for the remainder of this chapter.

Each question enforced a mapping of particular sound characteristics to resource usage. Since the data collected from this section was large, only analysis of panning question is included in this chapter as it showed the most promise. The question utilized panning from left to right and from right to left to show increasing and decreasing resource usage respectively. For the complete results from the mapping section please see Appendix A.

To analyze the questions within the mapping section the average error for each resource was calculated for each location in the clip (begining, middle, and end). The average error was calculated by taking the absolute value of the difference between the average of all the participants' responses and the expected answer for each subquestion. The error analysis is summarized as a column graph in Figure 3.5.

While the error analysis provides an estimate of the accuracy of using panning to sonify resource usage, it does not indicate any trends across the participant population. 95% confidence intervals were calculated for the average response of each subquestion. Figure 3.6 shows the confidence interval as error bars for the average participants' response for each subquestion along with the expected response.

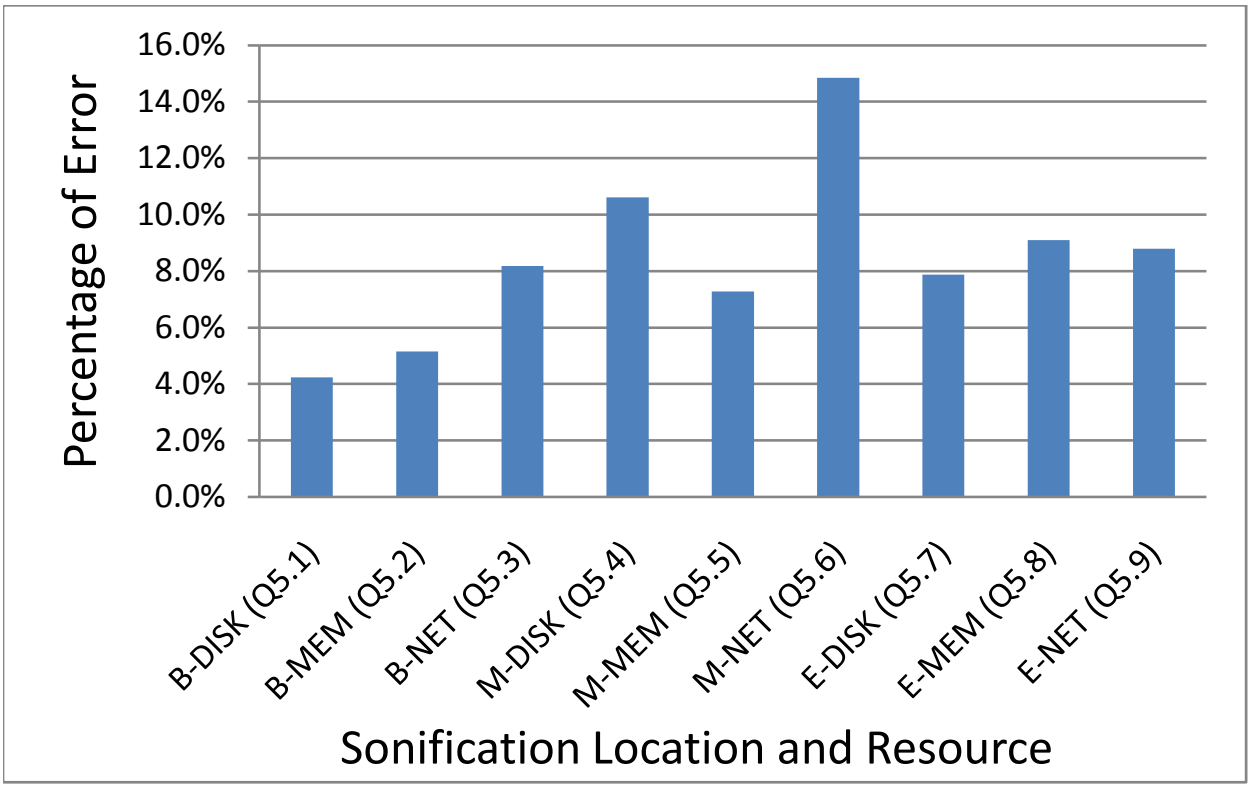

Figure 3.5: Error in mean participant vs. expected response for each panning subquestion. The persistently low error indicates that panning could provide a high level of accuracy for sonifying information.

3.2 Results

3.2.1 Quantitative Results

As is evident from the analysis of the general section, 85% of the participants mapped increasing pitch to "something" increasing a lot and 12% mapped increasing pitch to increasing a little. Additionally, 85% of participants mapped decreasing pitch to "something" decreasing a lot and 15% mapped decreasing pitch to decreasing a little. This is a strong indication that most participants already map increasing and decreasing pitch to increasing and decreasing values.

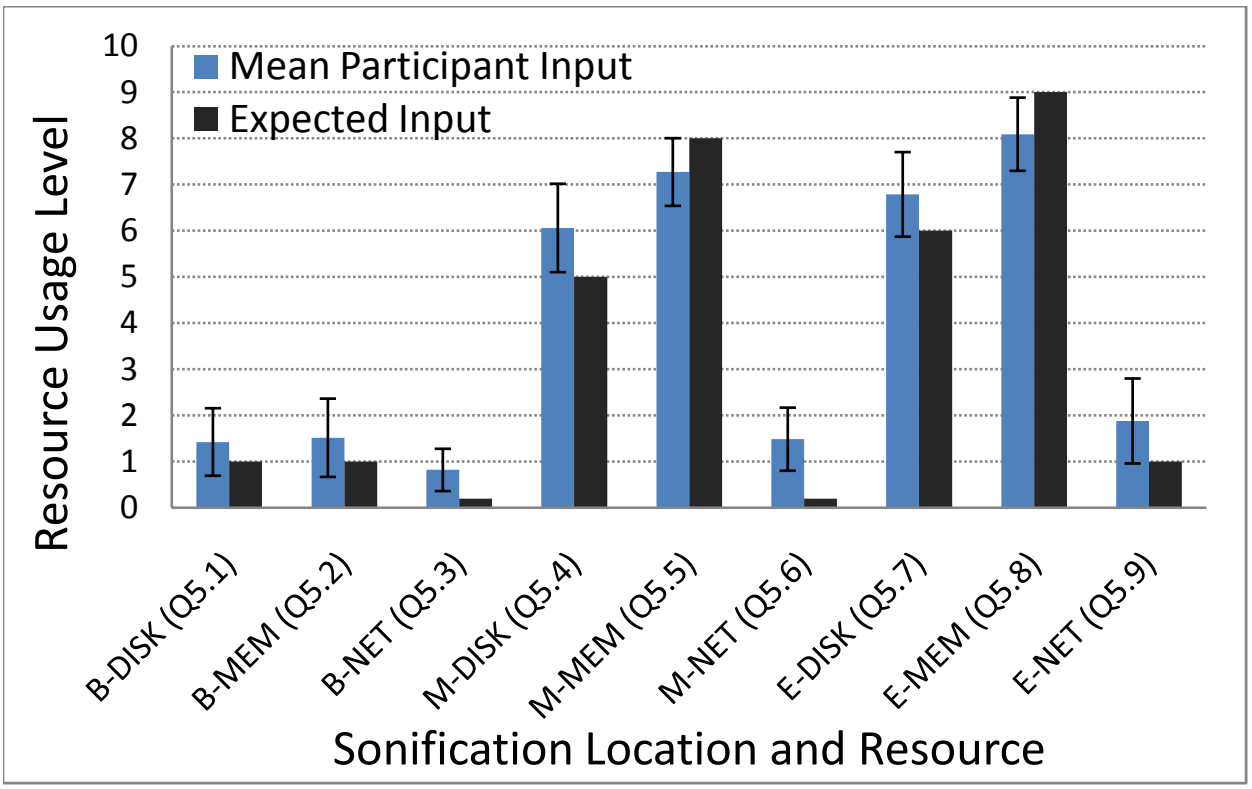

Figure 3.6: Expected vs. mean participant response with 95% confidence intervals as error bars for each panning subquestion. Since most expected responses fall within the confidence intervals, it shows that users will likely answer consistently and accurately.

The low error for using panning is an indicator that panning can provide a high level of accuracy for representing changes in sonified data, as long as the participant is aware of the mapping. Analysis of the general section, provided evidence that there is no apparent natural mapping from panning to resource usage; however, if a mapping of panning location to usage level is enforced, panning performs well as indicated by the analysis of the panning section and panning-based mapping question. Panning is further supported by the confidence interval analysis, as it shows a trend that users will answer consistently within an acceptable level of accuracy.

3.2.2 Qualitative Results

The final section of the survey provided a form for participants to provide subjective comments and feedback about the survey. The following are a selection of notable comments:

- "I especially liked...where panning indicated the usage level for each resource. It was easier to tell exactly how much of each resource was being used."
- "...on the whole it was much easier to pick out a specific position [panning location]."
- "For me - the ones which were most clear were those which used a left to right effect to convey information. Overall I like the idea and could see this being a useful tool for keeping track of system resources."

This strong preference for panning may stem from the user's having been bounded within the left and right limit. However, this is not the case for increasing and decreasing values of a characteristic such as pitch because pitch can keep on increasing without a hard ceiling; the inability of humans to sense frequencies above 20 kHz is the natural soft limit.

3.2.3 Conclusions and Insights

The study revealed multiple insights, some of which are far beyond the scope and purpose of performance tuning. However, for the purposes of conveying performance information, the survey clearly identified two sound characteristics that were shown to be most effective: pitch and panning. The study objectives were clearly met as all questions were answered:

Question 1 - Pitch was naturally mapped to increasing and decreasing values and when panning had an enforced mapping, it was accurate among a general population. Pitch and panning are the most suitable sound characteristics to facilitate perception and comprehension of performance metrics in a performance tuning context.

Question 2 - Pitch was interpreted uniformly by the participants as representing increasing and decreasing values with and without explicit mappings.

Question 3 - The analysis of the mapping section shows that users can effectively understand a mapping of sound characteristics to the performance metrics of a running application.

Chapter 4

Sonification-based Tuning System Architecture and SoniTune Implementation

4.1 Sonification-based Tuning System Architecture

The primary goal of the sonification-based tuning system architecture was to codify how one can create a modular, extensible, and interactive system for tuning complex computer systems using sonification. Using the architecture, one should be able to select component implementations to accommodate the requirements of different application domains.

Figure 4.1 depicts the overall architecture for interactive tuning systems that use sonification as a cognitive guide. The architecture integrates the following conceptual entities: system under test (SUT), workload generator, sonification engine, and tuning interface.

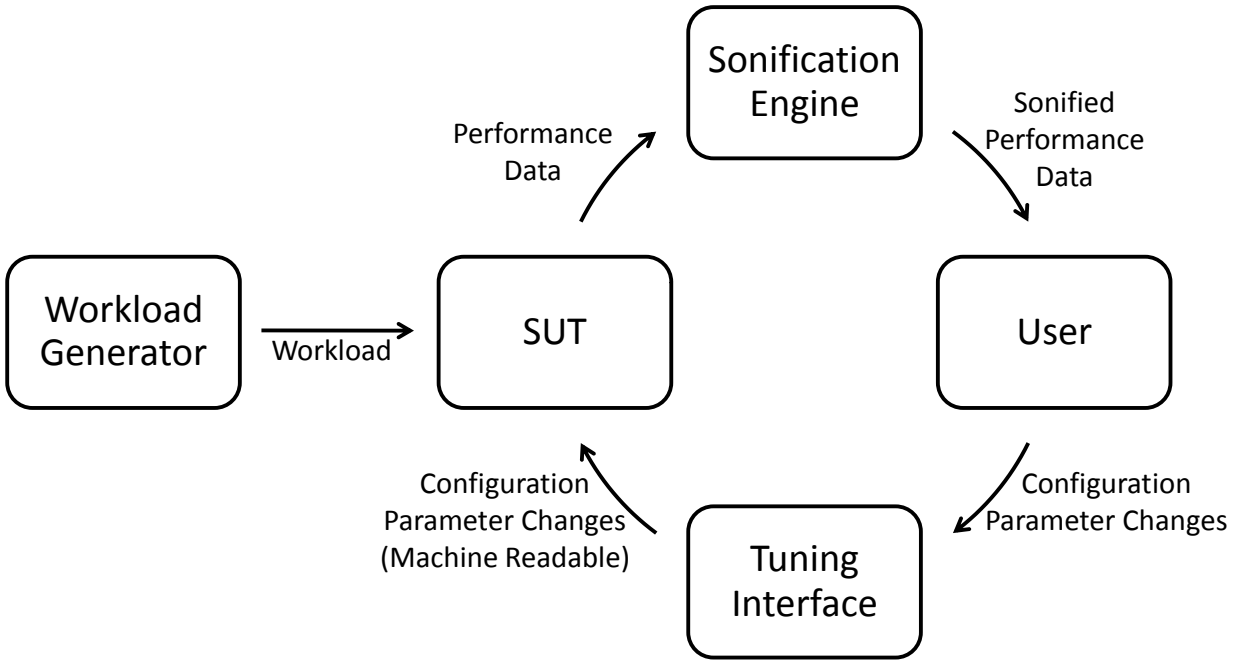

Figure 4.1: Software architecture

4.1.1 System Under Test (SUT)

The SUT is the system to be tuned. For a SUT to be tuned using an interactive approach, it must meet two requirements. First, the SUT must expose modifiable configuration parameters that can potentially affect some system performance characteristic while running. Second, the SUT must provide performance gathering facilities that can provide runtime information accurately with minimal interference on the SUT's execution.

While the first requirement of needing modifiable configuration parameters may seem obvious if a SUT is to be tuned, it is the fact that these parameters must be modifiable at runtime which makes the requirement difficult to meet. If the user is unable to modify the configuration parameters at runtime then the tuning process is no longer interactive. A non-interactive approach prevents the user, in this case, from immediately hearing the increase or decrease in system performance that their most recent change brought about.

The second requirement provides up-to-date runtime performance metrics to the sonification engine, described in Section 4.1.3, in quasi-real time. This allows any changes in performance due to the users configuration changes to be sonified almost immediately. However, the act of gathering performance metrics will in some way impact the performance of the SUT. The problem of modifying an object (SUT) by observing it (collecting performance data) is known in many disciplines as the observer effect [45]. While it is impossible to collect performance metrics without affecting the SUT at all is impossible, the tuning system will provide more actuate results if the impact is minimized.

4.1.2 Workload Generator

Performance tuning a SUT while it is idle or under no load is useless as it does not provide a representative case of when performance tuning is the most beneficial. It is necessary to place the SUT under the expected and sometimes heavier than expected load in order to achieve the best possible performance. The SUT is exercised by the workload generator, which emulates realistic usage scenarios that the SUT expects to receive when it is deployed.

The workload generator and SUT are tightly coupled with respect to the type of interaction that may occur between them. In essence, the workload generator and the SUT must be able to communicate via the same protocols and interfaces that the SUT exposes to users when it is deployed. For example, a workload generator for a exercising a web application and web server must be able to communicate via HTTP protocols over networking interfaces. Additionally, the same HTTP workload generator would not be suitable for exercising a relational database's low-level socket interfaces.

Overall, the purpose of generating workloads is to be able to tune a realistic system in an artificial (lab) environment, without compromising the accuracy of the end result.

4.1.3 Sonification Engine

The sonification engine is responsible for converting the performance metrics provided by the SUT into sonifications. As input, the sonification engine continuously takes the performance data from the SUT. The performance data must be normalized on a per-system basis as the range a sound characteristic can be used to sonify information is limited by a person's hearing. By the normalizing the performance data, the sonification engine is able to sonify a large range of performance metrics while remaining meaningful to the user.

As output, the sonification engine renders the normalized performance metrics into sonifications and plays them over the sound system to the user. The designer of the tuning system has significant leeway both in terms of what data they choose to sonify and how that data is sonified to the user. As was the case in regards to SoniTune, it may be necessary to conduct user studies so that the sonifications are appropriate for the target audience. Additionally, the sonification engine can be optionally combined with other cognitive aids, such as visual and haptic displays to potentially provide more information to the user. While the software architecture was designed with sonifications as the primary output to the user, it is flexible and modular enough to accommodate additional cognitive aids.

4.1.4 Tuning Interface

Since the software architecture is for interactive tuning, it must contain a user interface component. The tuning interface serves as the users entry point into the tuning process. The tuning interface exposes multiple configuration parameters to user and provides a straightforward way for the user the modify the parameters. For example, if one of the configuration parameters was the number of acceptor threads in a web server, the tuning interface could expose the value of the parameters as number in a text field and allow the user to adjust

the value via a slider.

Based on the cognitive feedback received from the sonification engine, the user interacts with the tuning interface adjusting the exposed configuration parameters as they see fit. When the user makes a change, the tuning interface is responsible for modifying the appropriate parameter in the SUT's configuration. Once the configuration changes are made, any changes in the SUT's performance will be sonified by the sonification engine and the user will immediately know how the change effected the system.

Overall, the purpose of the tuning interface is to expose to the end user a collection of intuitive controls for manipulating the SUT's configuration parameters and to relay the user's input into the actual SUT's configuration settings.

4.2 SoniTune Implementation

SoniTune, while specific to GlassFish, was built upon the general sonification-based tuning architecture described above. SoniTune implemented the architecture and leveraged the insights from the case study described in Chapters 2 and 2. The four components of the architecture were instantiated as follows: GlassFish web application server and a J2EE reference web application acted as the system under test, IBM Rational Performance Tester acted as the workload generator, a custom application built in Max/MSP acted as the sonification engine, and a custom GUI built in Java acted as the tuning interface. Each of these components are described in more detail in the following sections and an overall view of the SoniTune system in relation to the architecture can be seen in Figure 4.2.

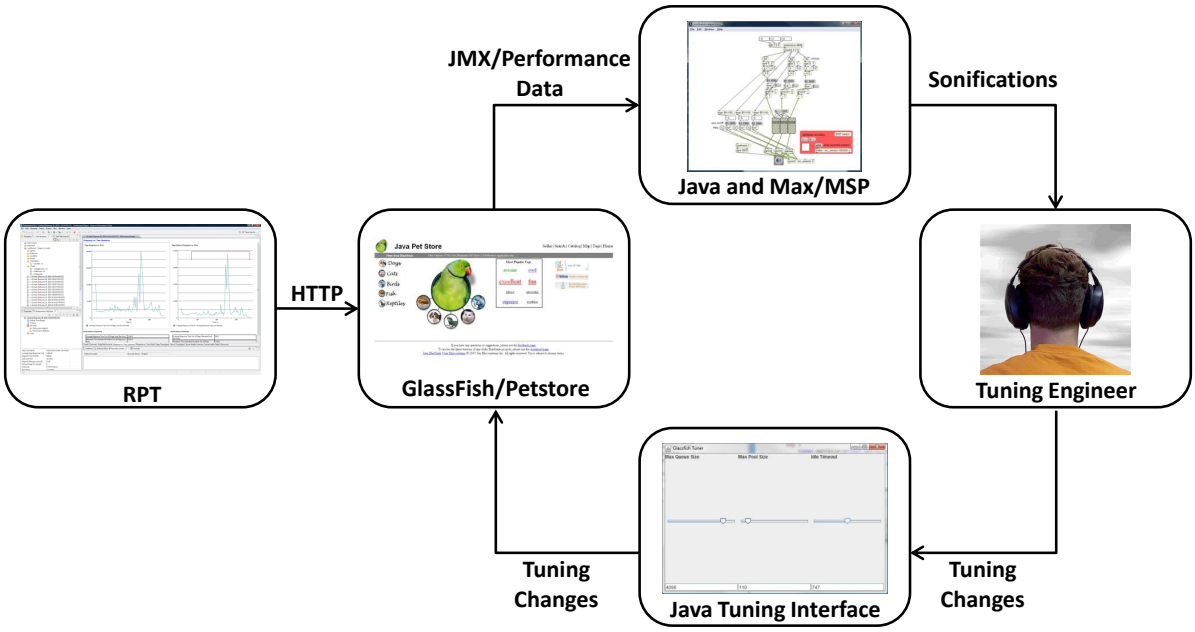

Figure 4.2: Instantiation of our software architecture for our proof-of-concept demonstration.

4.2.1 System Under Test (SUT)

SoniTune was designed to aid tuning a real-world commercial application server, an essential and representative part of modern enterprise infrastructure. An application server provides a runtime environment for enterprise applications typically accessed by hundreds of users at once. The performance and reliability requirements imposed on an application server are comparable to that of an operating system. SoniTune was specifically designed to work with Oracle GlassFish v3.0 [54], which has been successfully deployed in commercial installations for many years all around the world [55].

An application server was chosen because optimally tuning an application server is essential to meet the business requirements of an enterprise application [41, 61]. However, the definition of optimality differs for each hosted enterprise application and expected workload. As stated earlier, this situational optimality that depends on deployment and business re-

quirements contribute to the difficulty of performance tuning and could benefit from tuning tools.

An application server is a framework for enterprise applications, without an application, the server is practically useless. For SoniTune, the application was the Blueprints J2EE 5 reference application, PetStore 2.0 [33]. The PetStore application is representative of modern J2EE technologies and has been used as an evaluation subject in several performance tuning related research projects and demonstrations [13, 39]. PetStore encompasses the functionality of a typical e-commerce application, in which users can browse, select, sell, and purchase pets. See Figure 4.3 for a screen shot of the PetStore application running on the GlassFish server.

As stated in Section 4.1.1 there are two requirements that a SUT must meet to be applicable to the software framework, it must expose runtime modifiable configuration parameters and provide performance metrics with minimal impact on the SUT. GlassFish meets both of these requirements via Java Management Extensions (JMX). JMX provides a standardized avenue through which external clients send configuration changes and obtain information about a running system [48]. External applications, specifically the sonification engine and the tuning interface, can access GlashFish performance data and set configuration parameters. For example, GlassFish exposes processor usage, page request rate, and connection queue status through JMX.

4.2.2 Workload Generator

Since the SUT, GlassFish, is a Web server-based system, the workload generator emulates clients accessing the SUT remotely across the network using the HTTP network protocol. IBM Rational Performance Tester (RPT) [31] was used to produce a realistic workload for

Figure 4.3: Screenshot of the PetStore J2EE reference application running on the GlassFish application server.

a typical e-commerce application. RPT is a flagship web application testing tool from IBM used by enterprises to verify whether their applications can handle potential workloads.

RPT provides a range of test generation tools used to create virtual users. In this case, the actions of the virtual users were generated by recording various interactions between a real user and the website. RPT is capable of analyzing the recordings and generating scripts to exercise the server in a way that mimics the real user. To generate the realistic load for the SoniTune proof-of-concept, three types of virtual users were created:

1. A virtual user solely interested in browsing available dogs for purchase. The user used the various browsing interfaces provided by the PetStore application to look for dogs.

2. A virtual user with no particular goal other than browsing all available pets. The user used the various browsing interfaces provided by the PetStore application to look for all types of pets including lizards, turtles, dogs, and cats.

3. A virtual user interested in searching for particular types of pets available. The user used the search interface provided by the PetStore application to search for specific types of pets.

Using the three types of virtual users, a complete test suite was created that was made up of 75 virtual users. The 75 users were composed of 25% dog browsers, 25% general browsers, and 50% specific searchers. The test suite simulated the 75 users over a five minute time frame with the users starting up gradually over the first minute. The test suite was designed to exercise the functionality of the SUT under various frequencies commonly observed in practice. Figure 4.4 is a screen shot of the RPT test suite configuration.

Additionally, a slight variance was introduced into the timing between requests for each virtual user by randomly increasing or decreasing the delay by 0% to 5%. The slight ran-

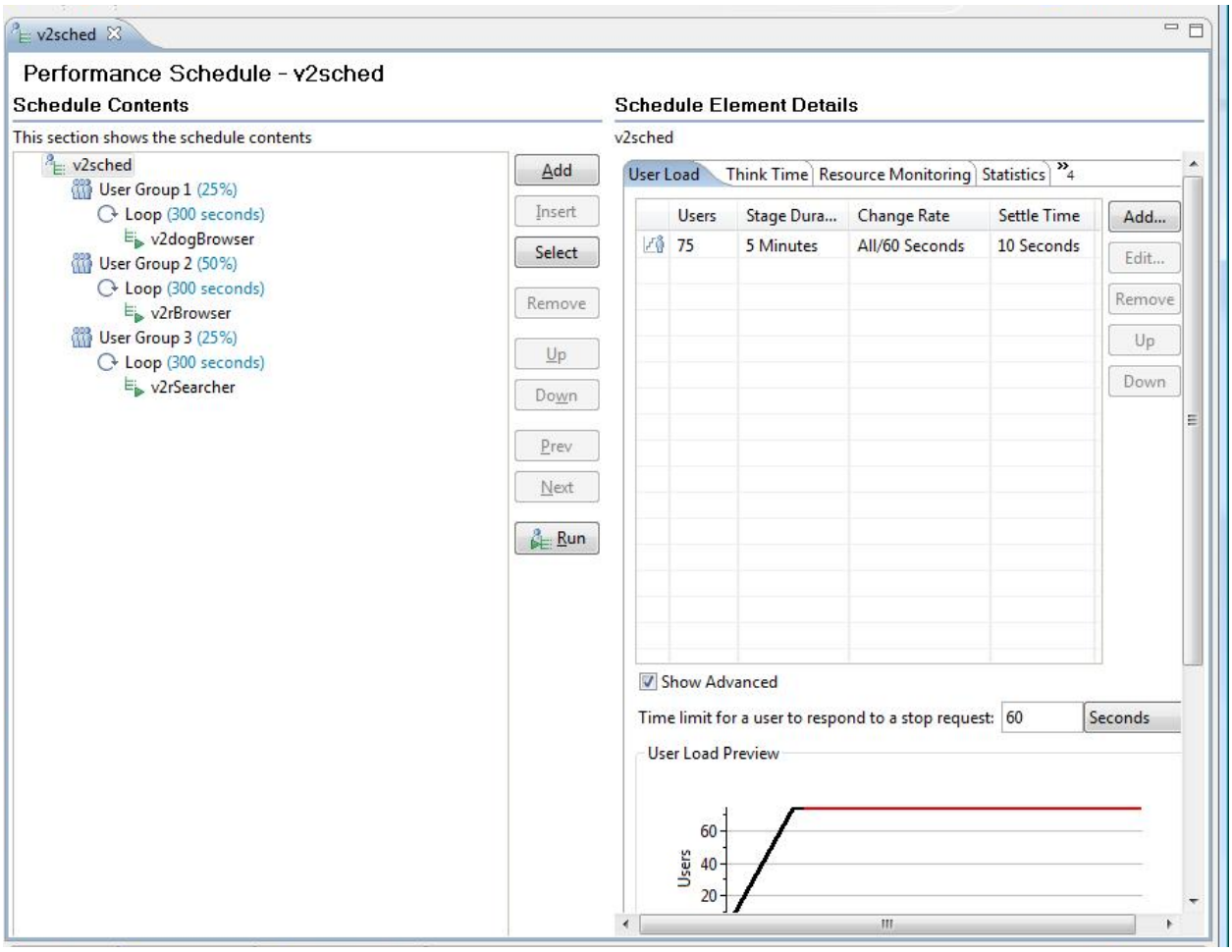

Figure 4.4: Screenshot of IBM Rational Performance Tester showing the testing schedule used to simulate a realistic load on GlassFish.

domness in page requests prevents all the users of a specific group from accessing the web server at the same time, however, the randomness is limited and thus allows for repeatable testing where the user load is always within an acceptable range.

4.2.3 Sonification Engine

The sonification engine is comprised of two components, a performance metric collector written in Java, and a sonification renderer written as a Max/MSP patch [17]. The performance

metric collector collected performance data from a running GlashFish instance via JMX. Specifically, the performance data collected was:

Average Requests Completed per Second - The average requests completed per second is one of the most important metrics for web application servers as it provides a clear window into how quickly and efficiently the server is able to respond to users.

CPU Usage - CPU usage is the percentage of system CPU time used by GlassFish as it processes requests.

Connection Queue Length - Connection queue length is the number of requests that are waiting to be processed by GlassFish and can be thought of a rough measurement of user request wait time.

These performance metrics were chosen for their availability, ease in computing, and by examining the metrics used in other web application based tuning approaches [39, 13].

The performance collector gathered information from the running GlassFish instance by polling the JMX interfaces once every second. The data was then normalized based on the expected minimum and maximum values. The expected ranges were calculated based on the system capabilities and experimentation. Once the data was collected and normalized, the performance collector dispatched the data as sequences of integers that were received, interpreted, and sonified by the Max/MSP sonification renderer component. The data was sent to the sonification renderer via a simple communication protocol over TCP sockets. The protocol consisted of space delimiting the set of performance metrics and terminating the sequence with a semi-colon and new line character.

The sonification renderer was built as a Max/MSP patch (Figure 4.5) with support from the Digital Interactive Sound and Intermedia Studio at Virginia Tech [70]. Max/MSP is a

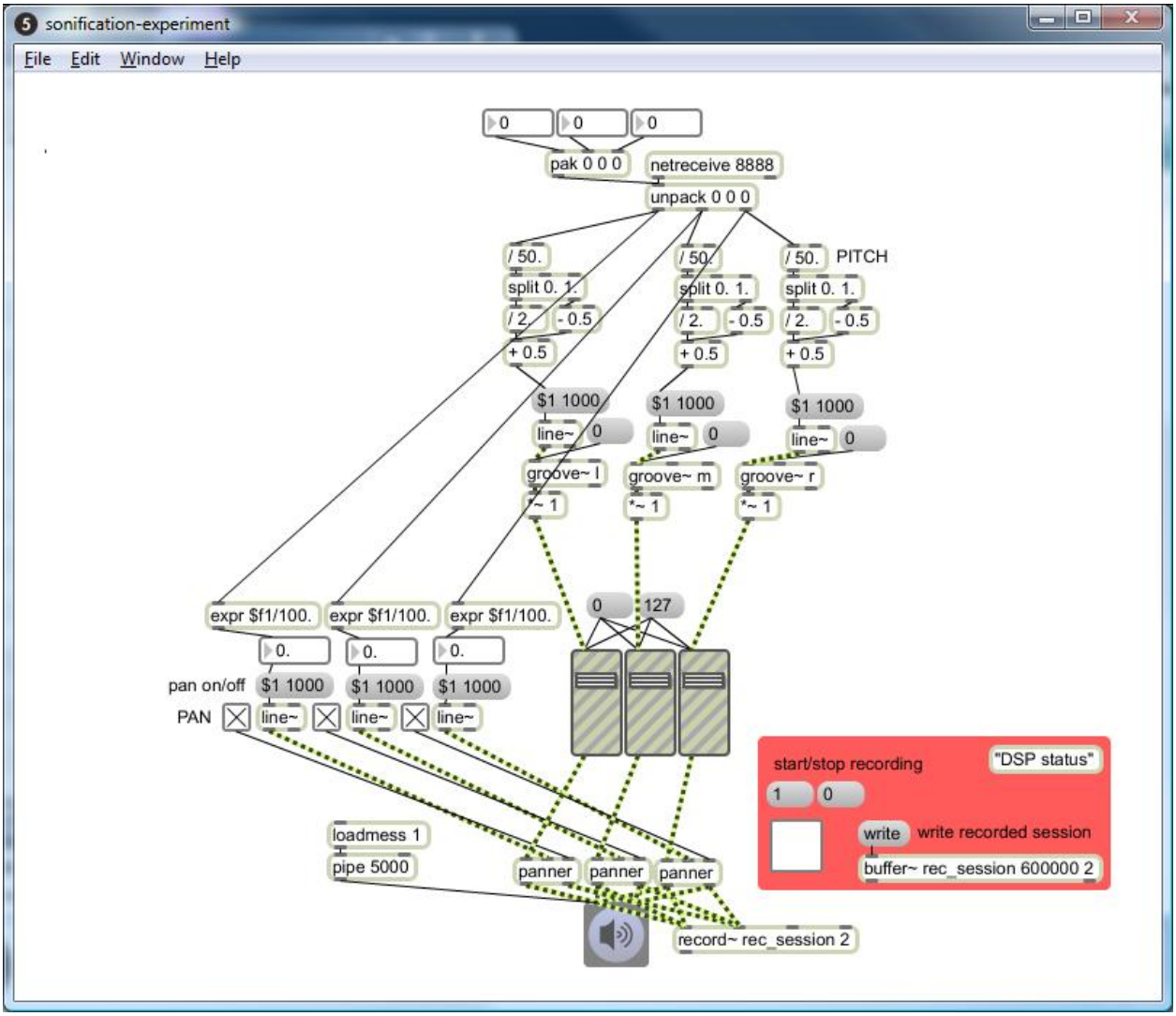

Figure 4.5: Screenshot of the Max/MSP patch.

visual programming language and framework that provides a high level of support for sound generation, modification, and recording. Additionally, Max/MSP supports TCP sockets which made receiving performance metrics from the performance collector straightforward. The sonification renderer constantly played the sonifications and would transition to the newly received values so as not to disrupt the playback to the user. Since three performance metrics were sonified and the playback was constant, it was necessary to pursue two key goals while designing the sonifications for the renderer:

1. The sonifications must not be "tiring" for the listener during a prolonged tuning session
2. The sonifications' spectral differences must make them sufficiently distinct from each other.

Therefore, the sonification renderer made use of three distinct, naturally-occurring sounds that were designed as follows:

1. A sustained chord as if it were played by a string instrument
2. A low hum of a cooling fan
3. A filtered sound of a mechanical hard drive

The three sound clips represented the three performance metrics that were sonified. The sustained chord represented CPU usage, the cooling fan represented completed requests per second, and the filtered mechanical hard drive represented connection queue length. The use of three distinct sounds provided the necessary spectral differences so that the users could lock in on a sound if they so desired. Naturally occurring sounds were chosen to prevent the listener from becoming tired or annoyed by the sonifications. However, it should be noted that listening to any sound in a repeated fashion becomes tiring given enough time.

As the analysis of the case study indicated, pitch and panning were the two sound characteristics that proved most effective for the definitive majority of the study's participants. To take advantage of this observation, the range of a performance metric was sonified such that it was proportionally mapped to a corresponding pitch range. That is, as the performance metric increased the pitch increased, and conversely, as the performance metric decreased the pitch decreased. Thus, there was a direct correspondence between the value of a performance metric and the pitch of the sound used to represent it.

From the psychoacoustics perspective, pitch is the human interpretation of sound frequency, which allows a human listener to observe the relevant changes in octaves. Since the range the human ear is capable of hearing is relatively limited, the sonification renderer used a range of two octaves that provided significantly noticeable difference in sounds while limiting their distortion. By limiting the pitch range and thus the resulting distortion, the original sounds (e.g., the string instrument, fan, and hard disk) were still distinguishable from each other even at the very lowest and highest ends of the performance metric's range.

Since the analysis of the case study showed that using pitch alone would not provide enough accuracy, the sonification renderer also mapped the performance metric to a panning location. In this case, the sonification renderer sonified the range of a performance metric by assigning the left channel to reflect the smaller values and assigning the right channel to reflect the higher values. The panning sonification used by SoniTune was exactly the same as that experimented with in the case study's mapping questions.

As an example sonification, consider a performance metric with an expected range from 0 to 100. If the sonification renderer receives a value of zero from the performance collector, the metric would be sonified by playing the sound clip which represents the metric solely from the left channel and with the pitch being one octave lower than the original. A metric value of 50 will result in a sonification playback with no modification to the pitch and equal

presence in both the left and right channels. Finally, a metric value of 100 will cause the sonification renderer to increase the pitch of the original sound by one octave and pan the sound entirely to the right channel. Any values between 0 and 50 or 50 and 100 are sonified as a gradient between the two respective values.

The design for SoniTune relied on two components to provide all the capabilities for the sonification engine, but this is not necessary. However, having separate components, one for gathering performance data and one for producing sonifications provides greater modularity. That is, the performance collector can be designed to work best with the SUT and the sonification engine can be designed to work best with the expected deployment situation.

In the case of SoniTune, a Java-based performance collector was used since GlasshFish outputs performance data over JMX (a Java centric technology). Since Java does not provide the best framework for sonifications, it was more efficient to use a platform like Max/MSP to sonify the data. However, neither component was dependent on the other as the protocol for communicating between the two could be implemented in any language or framework. The modular design allowed for the ability to use the best tools for the job, and provide greater independence between gathering and sonifying data.

4.2.4 Tuning Interface

The tuning interface provides a simple, intuitive interface that allows the user to adjust three configuration parameters in the running GlassFish application server:

1. **Acceptor Threads -** The acceptor threads are responsible for handling the initial connection to the application server from a client. Once the connection is established the acceptor thread hands the connection off to a request handling thread.

Figure 4.6: Screenshot of the Java-based tuning interface.

2. **Max Pool Size -** The max pool size is the maximum number of request handling threads that can be made available to handle request received by the acceptor threads.

3. **Idle Timeout -** The idle timeout is the length of time in seconds that a request handling thread may remain idle (not processing a request) before being killed by the system.

These configuration parameters were chosen because they could be modified while the GlassFish instance was running without causing significant overhead. Additionally, these parameters have also been used in other performance tuning demonstrations [39, 13].

The tuning interface exposed the configuration parameters to the user through a Java GUI, developed in Eclipse [65], that involved three combinations of a slider and text box. Each parameter had a corresponding slider and text box combo. The slider was used to set the value and the text box was used to display the value back to the user for verification. A screen shot of the GUI can be seen in Figure 4.6. By exposing the parameters in such a straightforward interface, it removes the complexity of navigating the complex configuration menus provided by GlassFish. This simplification allows engineers who are not experts in GlassFish to configure the system.

However, using an independent tuning interface over the built-in GlassFish interfaces also forces the implementation of changing the configuration parameters. To solve this problem, the tuning interface uses the same JMX protocols as the sonification engine uses to gather performance data. JMX is a bidirectional interface and provides the ability to set values in the system as well as read them. When the user makes a change to a configuration parameter in the GUI, the change will be immediately sent to the running GlassFish instance. Since nothing is free, a slight overhead will be incurred as the system adjusts the parameters and begins executing under the new configuration.

Chapter 5

The SoniTune Proof-of-Concept Demonstration and Feasibility Assessment

5.1 SoniTune Pilot Study and Evaluation

The SoniTune pilot study was the culmination of the case study which evaluated sound characteristics, the design of the general sonification-based tuning architecture, and the architecture's implementation into the proof-of-concept SoniTune system. The purpose of this pilot study was to demonstrate that *sonification-based tuning systems can enable non-experts in either the tuned application's domain or tuning itself to function as effective systems engineers*. Showing the feasibility of achieving this objective, thereby moving closer to proving the main conjecture of this investigation. The design, execution, and analysis of the SoniTune pilot study describd in the following sections.

5.2 Pilot Study Setup

The pilot study was setup to simulate the performance tuning of an application server, specifically GlassFish, as deployed on a resource constrained environment. To accommodate this, GlassFish was installed and configured in a VMWare virtual machine running Windows XP with Service Pack 3. The virtual machine was configured with 1GB of memory and a bridged network connection to the host machine. The host machine was running Windows Vista 64-bit, with a 3GHz AMD Phenom II X4 940 processor with virtualization technology, and 4GB of RAM.

As discussed in the implementation of SoniTune in Section 4.2, the workload generator, IBM RPT, the sonification engine, Max/MSP patch, and the tuning interface, Java GUI also need to be running to complete the cycle outlined in the architecture. RPT was started on the same host machine and communicated through the virtual networking interface to the SUT, GlassFish. The sonification engine and tuning interface were started on the system to be used by the participant and communicated with GlassFish over TCP sockets using the JMX protocol. The participant was presented with the tuning interface and listened to the output of the sonification engine with stereo headphones.

5.3 Pilot Study Design

The pilot study was designed to take approximately 30 minutes to complete and consisted of three stages. Stage 1 was the training stage and allowed the participant to become accustomed to the system, tuning interface, and sonifications. An explanation of the study was given to the participant along with a rough overview of what the participant would be doing. The participant was then exposed to the tuning interface and given a brief explanation

of the available configuration parameters. The configuration parameters were the number of acceptor threads, max thread pool size, and max idle timeout as described in Section 4.2.4. Great care was taken to not recommend or give specific guidelines regarding how the parameters' values should be adjusted.

Then the sonifications used in the experiment, as designed in Section 4.2.3, were explained and played for the participant. This portion of Stage 1 was very important as the entire experiment relied on properly establishing the mapping from sonifications to performance metrics in the participant's mind. The participants were allowed to the listen to the individual sonifications for each of three sonified performance metrics: requests per second, CPU usage, and connection queue length. The full range of the sonification was played from low to high and then high to low so that the participant would be exposed to how the changing performance data affected the sonifications. Additionally, it provided the bounds of the sonification by indicating the lowest and highest values possible. Once the participant was comfortable with the individual sonifications, all three sonifications were played at the same time to help prepare the participant for Stage 2.

Stage 2 consisted of an explanation of the participant's goals and the main procedure of the pilot study, tuning the GlashFish application server. The participant was presented with the goal of tuning GlassFish in order to maximize the requests per second performance metric. Auxiliary objectives were to maximize the CPU usage and minimize the connection queue length.

The main procedure of the pilot study included two tuning attempts, during which the participant attempted to achieve the aforementioned tuning objective while being guided by the sonifications. The reason for including two attempts was to present an opportunity for the participant to become more comfortable with the SoniTune tuning system. Even though all the sonifications were played during Stage 1 of the experiment, the sonifications were

played under a controlled environment and did not require the participant to understand the sonifications as they were played. During the first attempt the participant mainly listened to how the application server was running to get an idea of the current system operating parameters and adjusted the tuning parameters to see how each parameter effected the performance.

After the completion of the first attempt, the GlassFish server was restarted and the participants began their second and final attempt. Each attempt lasted for six and half minutes, and only the results from the second attempt were recorded for future analysis. To ensure greater accuracy, the GlassFish server was restarted between attempts but the configuration parameters' values remained the same from the end of the first attempt to the beginning of second.

After completing Stage 2, Stage 3 involved the participant completing a brief background survey similar to the demographics section of the case study (Section 2.3.1) and an exit survey. The exit survey was given to the participant so they could have a chance to explain the overall experience, and what they did and did not like about the process. Data collected in Stage 3 provided subjective information from the participants much in the same way as the general feedback section of the case study (Section 2.3.5).

5.4 Pilot Study Participants

The participants in the pilot study were four professional software developers, employed by a large technology R&D firm. Although quite experienced in their core domain, they only had a cursory knowledge with enterprise infrastructure and vague familiarity with configuring and tunning application servers. Because all the subjects happened to have participated in the initial sonification investigation case survey, they were familiar with the general concept

of conveying computing information with sound.

Participant 1 had 5+ years of programming experience, prefers programming in simple text editors but debugging in full IDEs, has experience in performance monitoring, intermediate music background, and sometimes listens to music while programming. Participant 2 had 5+ years of programming experience, prefers developing and debugging in full IDEs, has no experience with performance monitoring, intermediate music background, and sometimes listens to music while programming. Participant 3 had 2-5 years of programming experience, prefers programming in full IDEs but only uses output statements for debugging, has no experience with performance monitoring tools, no musical background, and never listens to music while programming. Participant 4 had 2-5 years of programming experience, prefers developing and debugging in full IDEs, has no experience with performance monitoring tools, casual music background, and rarely listens to music while programming.

5.5 Analysis and Evaluation

All four participants were able to successfully complete the entire experiment as described in Section 5.3. As expected, some participants were able to perform better than others, however, all the participants were able to improve the performance of the GlassFish application server above that of the standard configuration for some period of time.

Figure 5.1 shows the requests per second performance measurements for all four participants during the final two and half minutes of the second attempt. The vertical dashed line designates the time point at which the RPT workload was slowing down its simulation, which reduced the number of virtual users accessing the web application. As such, one would expect the number of requests per second to decline as fewer virtual users were accessing the website.

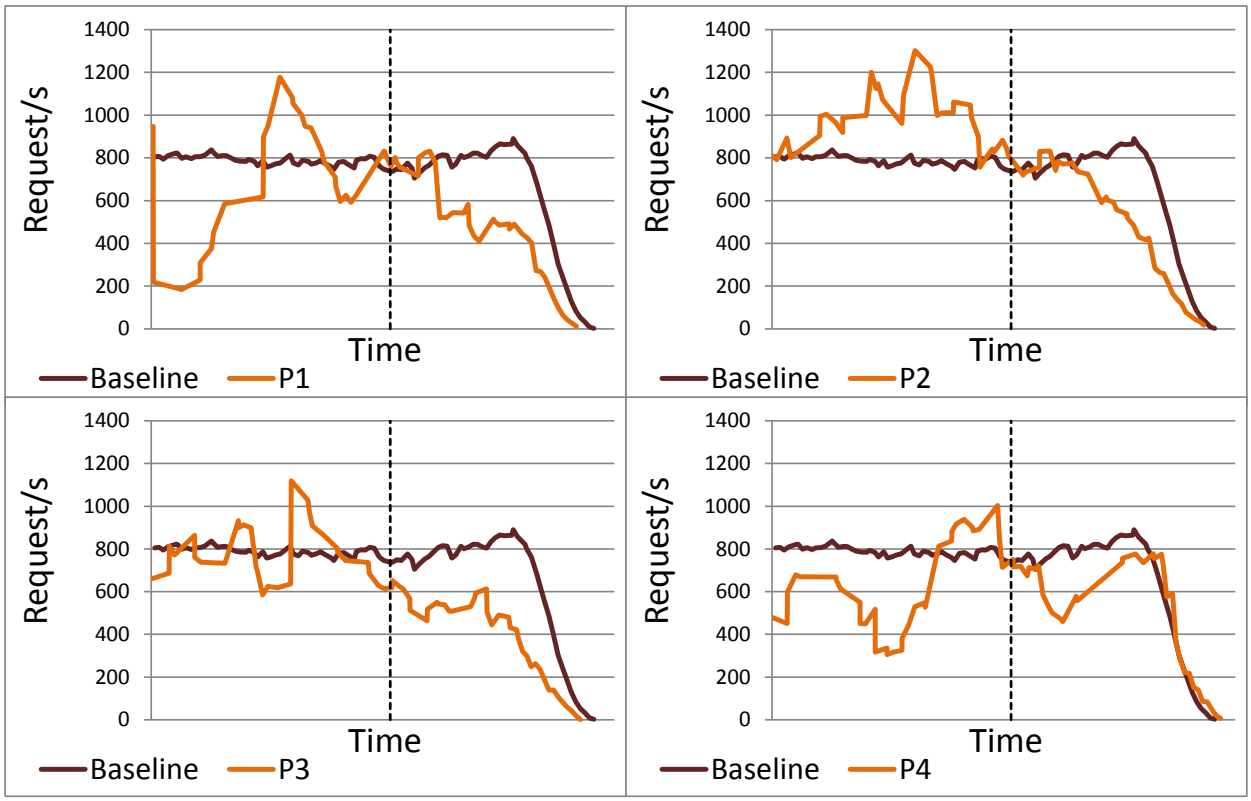

Figure 5.1: Graphs depicting the four participants' requests/second compared to GlassFish's standard configuration. The data starts at two and half minutes prior to the end of their second run. The vertical dashed line indicates when virtual users began logging off.

The analysis of the results shows that all the subjects were able to specify a tuning configuration that led to a performance level higher than that of the standard configuration. However, the degree and duration to which they were able to outperform the standard configuration differed between the participants.

Participant 1 (P1) was able to outperform the standard configuration by the end of the second attempt, but with more quality fluctuations in the intermediate configurations. Towards the end of the second attempt, P1 was able to specify a very high performing configuration which exceeded the standard configuration by almost 400 more requests per second. Following the peak in performance, P1 made a poor configuration adjustment which caused the performance to drop. However, as indicated by the quick return to increasing performance, P1 was able to course correct just prior to the experiment ending.

Participant 2 (P2) consistently outperformed the standard configuration during the final moments of their second attempt. P2's configuration was able to maintain an increase of 200 requests per second more than the standard configuration with peaks of processing over 400 more request per second than the standard configuration. P2 was by far the best performer of all the participants.

While Participant 1 and Participant 2 performed rather well, Participant 3 (P3) and Participant 4 (P4) both struggled with the exercise, potentially due to a longer learning curve. Although consistently improving their configuration, they took significantly longer to become comfortable with the SoniTune tuning system. One way to explain this disparity is that programmers' auditory perception and multitasking abilities tend to differ. P3 explicitly stated that he "was not a good multitasker" and was not even able to listen to music while programming.

Due to the performance of P1 and P2, it is possible that P3's and P4's performance could

have been improved to a greater degree had the attempt been longer. Based on the trajectory of their results and over coming the learning curve, it is possible that they would be able to eventually achieve the level of performance comparable to that of P1 and P2.

Overall, the results have moved us closer to proving the conjecture regarding the effectiveness of sound as a means to convey performance information to aid tuning. None of the participants had experience in performance tuning application servers and had no knowledge of GlassFish or the PetStore application enterprise application. However, all of the participants were able to tune the system within minutes of being exposed to the tuning sonifications and tuning interface. As indicated by the performance of all the participants, especially P1 and P2, it is definitely feasible for non-experts in both tuning and application servers to use sonified performance metrics to improve performance.

It is certainly worth investigating further the exact human and technical factors that affect the effectiveness of sonification as a cognitive tool.

Chapter 6

Sonification and Software Engineering

6.1 Iterative Approach to Developing Sonification-based Software

It is common for software engineering processes to stress the need to learn from previous efforts. For example, hundreds of organizations use the Capability Maturity Model Integration (CMMI) that emphasizes a lessons learned phase at the end of a project that records the successful and unsuccessful actions taken during the project [14, 49]. In that vein, an iterative approach to developing software with sonifications was gleaned from the design and development of SoniTune. While the approach described below is based on a single data point, it contributes to the overall body of knowledge concerned with integrating sonifications into software, and may help support future approaches.

6.1.1 Overview

An iterative development process starts with an initial planning phase and then enters a phase of repeated cycles of (1) planning and requirements; (2) analysis, design, and implementation; and (3) testing and evaluation. Once completed, the final product is deployed or delivered [63]. While SoniTune was developed with no conscious concern for processes or methodologies, as is the case with most research projects, the development ended up following a rough iterative process regardless. There was an obvious initial planning phase, followed by multiple iterations of selecting sonifications based on requirements, testing, and analysis, and concluded with the "deployment" of the SoniTune proof-of-concept. The following sections provide additional information on a general, iterative based approach to developing software which leverages sonifications.

6.1.2 Initial Planning

The initial planning phase involves selecting the types and characteristics of the data to be sonified. Data comes in many shapes and sizes and requires different types of sonifications. It is crucial to identify the types and characteristics of data early so that one can properly design the corresponding sonifications. In the case of SoniTune, performance metrics were identified early as the data to be sonified. Performance metrics are typically continuous values that will increase and decrease over the duration of a program's execution. The characteristics of performance metrics will enforce requirements on the types of sonifications that can be used.

6.1.3 Planning and Requirements

After determining the characteristics of the data to be sonified, it is necessary to determine how those characteristics will impact the available sonifications that can be used. For example, if the data is only available at set intervals, a rhythm style sonification based on the data availability intervals may be more applicable than a constantly playing sonification. The initial mapping of sonification to data characteristics is important, however, later iterations of this step allow for changing or tweaking the sonifications based on feedback from the testing and evaluation steps.

6.1.4 Analysis, Design, and Implementation

With data characteristics and sonification requirements (potentially revised sonifications if this is not the first cycle) properly identified, the next step is to analyze the requirements, design the software and sonifications, and then implement the design. Depending on the maturity of the sonifications chosen by analyzing the sonification requirements, it may be necessary to design and implement a sonification survey. Such a survey could leverage existing results from similar studies, but conducting a survey specifically targeted to your needs and potential users can result in more applicable sonifications. The case study conducted in developing SoniTune served this purpose by targeting software engineers in particular with the goal of finding those sonifications best suited for sonifying performance data.

If a survey is not needed, or this is a later iteration of the process, the necessary sonifications are designed or improved upon during this step. Improvements, in the form of sonification redesign and implementation, come from re-evaluating the data and sonification requirements based on feedback from the testing and evaluation steps. With every iteration of this process, the sonifications and resulting application should become better suited for the target users.

The development and implementation of the SoniTune prototype could be considered a second iteration of this step as designing and implementing the case study would have been the first.

6.1.5 Testing and Evaluation

The effectiveness of a sonification cannot be calculated by a typical suite of unit tests. The testing and evaluation step requires exercising the system with a sample of potential users and evaluating their comprehension level and the overall system usability. If the current iteration cycle involves a sonification study, testing and evaluation refers to administering the survey or study and evaluating the results respectively.

When there is no need for a sonification study or when completing later iterations, testing and evaluation involves conducting experiments with the current version of the software with expected users. The sonifications must be evaluated by testing the sonifications with potential users and examining the user's level of comprehension. In the case of SoniTune, the first iteration of this steps was administering and analyzing the case study, and later iterations involved the proof-of-concept demonstration with the four professional software engineers.

6.2 Looking Onward with Sonification and Software Engineering

Software engineering is an ever evolving collection of architectures, designs, processes, procedures, methods, and many more like minded components all with the intent of producing software and software related artifacts. The reason why we have an ever growing collection

of tools and procedures is the same as it was 20 plus years ago when Frederick P. Brooks, Jr. proclaimed there is "no silver bullet" to software engineering [11]. Basically, Brooks states that there is no single development that will promise improvement for any and every software engineering endeavor. Moreover, his statement has been reaffirmed during a celebratory panel at the 22nd International Conference on Object-Oriented Programming, Systems, Languages, and Applications in 2007 [64].

Still with no silver bullet in sight, it is very important for every software engineer to have a diverse toolbox of varying tools, approaches, and the like for tackling problems on a case by case basis. This thesis has shown that sonification deserves to be among those tools for performance tuning, however, there is potential for it to be applicable to multiple aspects of software engineering. As stated in the Chapter 1, this thesis divides software engineering into two categories: a process of information generation and a process of understanding existing information.

The definition of sonification implies that information must exist so that it may be sonified. Due to this fact, there is no apparent application of sonification to the information generation side of software engineering. However, there is great potential for applying sonification to the understanding of existing information. In truth, some work has already been done on applying sonification to multiple facets in software engineering as discussed in Chapter 7 on related work. Nevertheless, work on sonification as an asset to software engineering is under explored. The following sections provide a brief exploration into how sonification can be used or be further explored in three common areas of software engineering based on the insights and lessons learned while this research was conducted.

Implementation

While implementation can primarily be considered an information generation process with engineers implementing interfaces and fulfilling designs, there are some aspects which could benefit from sonifications. For example, engineers do not always start the implementation of new features from a clean slate. It is very common for engineerings to come onto a project and be asked to implement new features for a pre-existing system. However, before they can begin, they must take the time to learn and understand what is already available and what they can leverage. This process is known as program comprehension, and sonification can become an excellent tool to aid the process as early work has already shown [30, 5]. Sonifying program structure, call graph trees, or the results of any form of static analysis could provide invaluable cognitive aids to engineers that could improve productivity or reduce time to completion.

Testing

Testing often consists of engineers running suites of unit tests and then processing the results of which tests passed and which failed. This information could be sonified and presented to the engineer in such a way that they could listen to the tests results and know which tests failed and the severity of failing a particular test. For example, if all the tests passed successfully the sonification would be a pleasant, melodious tune. However, if one or more tests failed, the melodious tune would be interrupted by off-tune notes or non-music sounds such as dogs barking. The sonification of failure could be specific to individual tests so that the engineer would know exactly which test failed and caused the pleasant tune to be interrupted.

Deployment

The SoniTune prototype is directly applicable to this area of software engineering. It is of great benefit to engineers as well as professional organizations to be able to successfully tune complex software systems without requiring expertise in tuning or the system to be tuned. Companies like JBoss by Red Hat [34] are able to sustain an open-source development model while remaining profitable because one needs specialized expertise to ensure that JBoss products perform optimally in a customer's deployment environment. JBoss developers can function as high paid consultants responsible for configuring and tuning their software for optimal performance. Having tools to prevent the need to outsource configuration to consultants has the potential to save time and money.

Additionally, a mechanic can listen to a car to determine if it is running optimally, but can an engineer listen to a deployed software system to determine if it is continually operating optimally? Sonifications could be leveraged to sonify the health of a running system and be used to indicate when a system should be restarted or reconfigured due to a change in load.

Chapter 7

Related Work

7.1 Performance Tuning Approaches and Tools

Existing approaches to performance tuning take advantage of performance profiling [38, 82], visualization [76], and automation using complex statistical models [13]. Because performance tuning remains a standing challenge in the face of the increasing complexity of modern systems, there is great potential benefit in exploring novel tuning approaches that utilize untapped resources such as sonification.

Jovic and Hauswirth [38] presents an approach to tuning Java GUI applications that profiles event listener latency. Another profiling approach to analyzing performance and tuning applications is presented by Zagha et al. [82]. They provide a framework of counters and profiling tools for tuning systems based on the MIPS R10000 processor.

Other approaches have leveraged tools and visualizations to aid tuning. Mellor-Crummey et al. [50] developed application specific tools to aid performance tuning. Walker et al. [76] visualized large amounts of data collected during the execution of an object-oriented system

at the architectural level. Chassin et al. [19] visualize a variety of thread interactions to facilitate tuning multi-threaded parallel applications. Jones et al. [36] used profiling and visualizations to aid tuning of parallel applications.

Krogmann et al. [44] have combined software cartography visualizations and performance tuning to provided an approach that allows engineers to map the cryptic results of testing into design decisions that increase performance.

Other approaches have leveraged statistical and stochastic tools to automate tuning instead of simply aiding a manual process. Chen et al. [13] utilize the Markov decision process and a reinforcement learning strategy to discover a system's optimal configuration. Additionally, similar to our work, they also validate their approach by tuning an older version of the J2EE PetStore application. Tiwari et al. [67] present a parallel algorithm for automatically tuning parallel applications. Their work utilizes the automated tuning framework, Active Harmony [16], presented by Ţăpuş et al. as a framework for automated runtime tuning.

SoniTune relies on the ability to change the conifugration parameters of a running system. Tuning a system while it is running is known as dynamic tuning and is used widely in performance tuning. Felber et al. present an approach to improving transactional memory performance by leveraging an automated dynamic tuning system. The approach "searches" for an optimal configuration by dynamically changing configuration parameters. While the tuned system is different, this is very similar to SoniTune except the engineer is required to change the parameters manually.

7.2 Sonification

7.2.1 General Sonification

A good reference that outlines key concepts of sonification research and design is an article by Walker and Nees [75] that defines an *auditory display* as using sound to convey information and *sonification* as an auditory display consisting of non-speech audio. Another definition of sonification is provided by Kramer at al. [43], who define it as "the transformation of data relations into perceived relations in an acoustic signal for the purposes of facilitating communication or interpretation."

Most of sonification research is concerned with identifying those scenarios for which auditory displays provide effective solutions [59, 42, 35]. For example, auditory displays have been particularly effective in rendering complex data patterns and events requiring the user's immediate attention [24, 52]. This is because human hearing excels at identifying temporal information.

A Sonification Design Map presented by deCampo [18] and Sonificaiton Design Patterns by Barrass [4] shows quantitative relationships between non-speech auditory displays. A traditional classification of sonification approaches includes audification [42], parameter mapping, and model-based [28]. Categorizing sonification approaches on the basis of their respective data representations results in continuous, discrete, and model-based data. According to this categorization, the sonification approach that we used in this work is model-based. This representation uses a model based on the properties of the data to mediate between the sonified data and the sound. Because the model captures the domain knowledge of the sonified data, it can be applied to different types of datasets.

Sonifications have also been applied when visualizations are restricted due to physical con-

straints, such as screen size. Brewster [10] showed that sonifications can increase usability of mobile platforms which lack screen space due to their size. Walker et al. [72] sonified the progress of background processes on mobile devices to aid multi-tasking.

7.2.2 Human Capabilities

The approach presented in this thesis and SoniTune prototype sonified multiple data streams. Sonification of multiple data streams relies upon the concept of aurual focus [58], or the human ability to segregate different sources of sound in loud or noisy environments (e.g., having a conversation in a loud restaurant). Additionally, some research has indicated that it is possible for some people to be capable of understanding multiple sonifications better than listening to only one sonification [2].

However, while the average person is capable of processing multiple stimuli at once, "Miller's Law" indicates that the most objects one can hold in working memory is seven plus or minus two [51]. Unlike visualizations which can remain on the screen and provide additional time to understand, sonifications are typically not repeat and occur at a point in time. Thus, it is important to not over stimulate the listener

It has also been shown that in some situations, humans are able to understanding auditory input rather than visual input. Repp et al. [56] conducted an experiment that determined humans were more attracted to auditory rhythms than visual rhythms. Thus, if the data to be sonified relies on some sort of rhythm (e.g., heartbeats to indicate active state in a distributed system) sonifications can be a better approach over visualizations.

7.2.3 Sonification Frameworks

Pauletto et al. [60] present initial work on their "Interactive Sonification Toolkit." The toolkit was designed to aid engineers in rapidly developing interactive sonifications for large, static data sets. Additionally, work by Walker [73] has resulted in a multi-purpose sonification toolkit, called Sonification Sandbox that allows dynamic customization of static data sonifications. The toolkit allows users to import data from a file and then alter the sonifications in addition to select portions of the data to sonify. Sonification Sandbox was constructed by combining the key features of multiple sonifications frameworks [81, 62, 37]

Many of the available sonification frameworks are targeted at sonifying static data (i.e., data that can be imported and processed from a file). The architecture and proof-of-concept implementation presented in this thesis relies heavily on the ability to sonify performance information in quasi-real time. Thus, the approach is truly interactive as changes to the data (e.g., changes in performance) are immediately sonified to the tuner.

7.3 Sonifications and Software Engineering

Several prior approaches have used auditory displays to convey information about computer applications. Vickers and Alty [68] use music to communicate information about which programming language structures are used, to express how programs behave at runtime, and to find potential program defects [69]. They have created the CAITLIN system to aurolize Turbo Pascal programs. One of the most salient insights of their investigation is that music can successfully communicate useful information about computer programs for all users, even for those who have not been formally musically trained.

When representing program constructs using both speech and non-speech audio, Finlayson

and Mellish [22] concluded that the two modalities should be used together for maximum benefit. To improve program understanding, Berman and Gallagher [6] sonify program slices.

Compared to the aforementioned auditory displays approaches to convey program information, this work focuses on understanding the performance of a computer application that is treat as a black-box. That is, the external behavior of an application is sonified without any regard for its source code or any other software artifacts. Furthermore, the sonifications are significantly less-structured than music tunes. Finally, the sonifications change interactively in response to tuning actions of the user.

7.4 Sonifications and Software Visualization

Another cognitive aid that can assist the user in tuning complex systems is information visualization, which Card et. al. defined as "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition [12]." The subset of information visualization that concerns program details, including structure, algorithms, execution, and evolution, is called software visualization [20].

In relation to visual displays, auditory displays tend to complement each other. One can substitute a visual display with an auditory one when no visual display is available, some comprehensive task fully engages vision [80], or vision suffers from an information overload [9]. Experiments in psychology have shown that sound enhances visual perceptions [71], and this was leveraged by tool which supplemented a visual display with an audio cue [15]. Additionally, several other studies have shown that informartion presented in auditory form can enhance visualizations [77, 79, 78].

Visualizations can be considered as being complementary to the sonification approach out-

lined in this thesis. That is, by presenting this approach, yet another cognitive aid can help the user to deal with the challenges of performance tuning. Performance sonification is yet another tool in the performance tuner's toolbox.

Chapter 8

Thesis Contributions, Conclusions, and Future Work

8.1 Thesis Contributions

This thesis has established that sonification can provide a valuable cognitive aid to assist performance tuning. A case study was conducted to determine the fundamental affordances of sounds to a general population of software engineers, a proof-of-concept sonification-based tuning system, SoniTune, was built based on a general sonification-based tuning architecture, a pilot study was conducted to show the feasibility of the SoniTune system, and a collection of lessons learned and a potential iterative software engineering processes was described.

1. *A systematic user study that assessed the accuracy of sonification to convey information.* The results of the study can inform software system designers who want to integrate sonification into their systems. As shown in this thesis, the results were used to develop a sonification-based tuning system which leveraged both panning and pitch

manipulation to represent increasing and decreasing values in performance metrics. However, there is no reason why these results could not be used in other tools or further analyzed for additional underlying trends. Additionally, the case study provided a good starting point to launch other studies which can be focused on particular sound characteristic or a particular group instead of assessing multiple sound characteristics at once.

2. *A software architecture for tuning software systems with sound; which integrates a system under test, a workload generator, a tuning user interface, and a sonification engine.* The software architecture incorporated a cycle which allowed the sonification engine to play performance sonifications a user, the user to interpret and act on those sonifications by adjusting the system under test's configuration parameters via the tuning interface, and then having the new performance metrics be sonified back to the user. All this while the workload generator simulated typical load on the system under test to provide realistic operating conditions.

3. *A proof-of-concept prototype that demonstrates the utility and feasibility of our approach and its applicability to real world systems.* The proof-of-concept sonification-based tuning system, SoniTune, implemented the general architecture and was shown to guide non-experts in tuning or GlashFish to tune the application server so that it outperformed the base configuration.

4. *An iterative software engineering process for developing effective sonification-based tuning systems.* As with any new approach, it is necessary to document successes and failures so that future work can leverage the triumphs and avoid the pitfalls of past work. The successes and failures through the design and development of SoniTune have been documented as an iterative approach to developing software systems which involve tuning system.

8.2 Future Work

With the success in showing the feasibility of the sonification-based tuning system, several possible future work directions can be taken.

8.2.1 Additional Case Study Analysis and Studies

Additional analysis of the results of the case study can be conducted to further investigate underlying trends between similar participants. If noticeable differences exist for specific groups of participants, for example inexperienced musicians, sonification tools in general can take this into consideration and be customizable to accommodate specific users.

Since this study was conducted with the goal of examining multiple sound characteristics to determine the best candidates, additional studies could be conducted to further investigate particular sound characteristics. Also, additional studies could be conducted which involve some form of training to help eliminate potential side effects caused by the participants being unfamiliar with interpreting sonifications.

8.2.2 Improving the Tuning Interface by Embedding Domain Knowledge

The purposes of this research was aimed at assessing the effectiveness of sonification and wanted to isolate the results from the influence of any domain-specific knowledge. Thus, in designing the tuning system for the demonstration experiment, the tuning interface did not embed any domain-specific knowledge and exposed the tuning parameters as is, without putting tight constraints on the range that the user could select.

Even though all the participants discovered a configuration that led to a throughput higher than that of the standard configuration, the tuning interface could have been restricted by taking advantage of any of the tuning guidelines mentioned in the GlassFish manual. For example, one of these guidelines prescribes that the number of acceptor threads should be roughly equal to the number of CPU cores available to the process. This guideline, or domain knowledge, could be embedded into the tuning system by restricting the allowable range for setting the number of acceptor threads in the tuning interface. Enhancing a tuning system with domain knowledge can be particularly beneficial for less experienced or novice engineers.

8.2.3 Applying Sonification to Other Software Engineering Tasks

As stated previously, the results of the case study can be used to create approaches that facilitate software engineering tasks other than tuning. This thesis has established the effectiveness of certain sound characteristics to convey performance metrics to the general population of software engineers. With that in mind, it would be possible to revisit important software engineering challenges such as program evolution [46], analysis, and maintenance management [47].

8.2.4 Deeper Evaluation of Sonification-Based Tuning

The purpose of the pilot study with the proof-of-concept tool, SoniTune, was to demonstrate the feasibility of guiding software engineers through performance tuning via sonification-based tuning systems, a goal that has been fulfilled. However, the evaluation only showed how the participants compared to GlassFish's standard configuration. Nevertheless, further investigating its effectiveness in relation to other tuning approaches would help demonstrate

the applicability of such an approach.

8.2.5 Combing and Evaluating Sonification with Other Cognitive Aids

Sonification is just one cognitive aid that can be leveraged to help software engineers. Perhaps the most popular and widely used cognitive aid is data visualization. Sonification and visualization have their respective strengths and weaknesses, and can be utilized to convey different kinds of information. Future research in evaluating sonifications in direct comparison to visualizations and other cognitive aids would help determine which aids are best at conveying information in certain situations.

Additionally, once weakness and strengths of different aids have been determined, it would be possible to combine aids in ways that their strengths complement each other and reduce the overall weakness. Combining cognitive aids together could provide powerful synergies for the next generation of tuning tools.

8.3 Conclusions

This thesis has presented research into sonification-based performance tuning systems with the goal of showing the feasibility of such a system as a helpful cognitive aid. Through the case study conducted and the pilot study of SoniTune, this thesis showed that interactive sonification can effectively guide software engineers through performance tuning of a computer system. While sonification research in mainstream applications is still young, this work provides valuable insights that can be used by future researchers in the areas of sonification and performance tuning.

The ever increasing complexity of modern computer systems calls for new and more powerful approaches to accommodate these systems for the needs of real users. In that light, tuning systems for optimal performance has come to the forefront of industrial software engineering. This thesis has presented a novel approach that leverages sonification to facilitate performance tuning of complex computer systems. We believe that this approach can lay the foundation for a new generation of approaches and tools that use sonification. This powerful cognitive aid can help tame the complexity of engineering computer systems of today and tomorrow.

Bibliography

[1] Adobe. Adobe Flex. http://www-sjc0.adobe.com/products/flex/, 2010.

[2] 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.

[3] T. H. Andy Hunt. The importance of interaction in sonification. In *Proceedings of International Conference on Auditory Display*, 2004.

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

[5] 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, 2006.

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

[7] L. I. Berman and K. B. Gallagher. Using sound to understand software architecture. In *SIGDOC '09: Proceedings of the 27th ACM international conference on Design of communication*, pages 127–134, New York, NY, USA, 2009. ACM.

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

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

[10] S. Brewster. Overcoming the lack of screen space on mobile computers. *Personal and Ubiquitous Computing*, 6(3):188–205, 2002. 10.1007/s007790200019.

[11] F. P. Brooks. No silver bullet: Essence and accidents of software engineering. *Computing*, 1987.

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

[13] H. Chen, G. Jiang, H. Zhang, and K. Yoshihira. Boosting the performance of computing systems through adaptive configuration tuning. In *SAC '09: Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 1045–1049, New York, NY, USA, 2009. ACM.

[14] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[15] 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.

[16] C. Ţăpuş, I.-H. Chung, and J. K. Hollingsworth. Active harmony: towards automated performance tuning. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE con-*

ference on Supercomputing, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[17] Cycling '74 Inc. Max/MSP. http://www.cycling74.com, 2010.

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

[19] J. C. de Kergommeaux, B. Stein, and P. E. Bernard. Paj, an interactive visualization tool for tuning multi-threaded parallel applications. *Parallel Computing*, 26(10):1253 – 1274, 2000.

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

[21] A. Eldridge. Issues in auditory display. *Artificial Life*, 12(2):259–274, 2006.

[22] 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.

[23] 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.

[24] 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.

[25] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[26] D. Hargreaves. *The developmental psychology of music*. Cambridge University Press, 1986.

[27] C. Henthorne and E. Tilevich. Sonifying performance data to facilitate tuning of complex systems. In *Onward! 2010 in the proceedings of the SPLASH 2010.*, 2010.

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

[29] T. Hermann and H. Ritter. Listen to your data: Model-based sonification for data analysis. In *International Institute for Advanced Studies in System Research and Cybernetics*, pages 189–194, 1999.

[30] K. Hussein, E. Tilevich, I. I. Bukvic, and S. Kim. Sonification design guidelines to enhance program comprehension. In *ICPC '09: Proceedings of the* 17^{th} *IEEE International Conference on Program Comprehensio n*, 2009.

[31] IBM. IBM Rational Performance Tester. http://www-01.ibm.com/software/awdtools/tester/performance/, 2010.

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

[33] Java BluePrints. Java Pet Store Demo. https://blueprints.dev.java.net/petstore/, 2010.

[34] JBoss by Red Hat. Jboss. http://www.jboss.com/, 2010.

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

[36] D. Jones, Jr., S. Marlow, and S. Singh. Parallel performance tuning for haskell. In *Haskell '09: Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*, pages 81–92, New York, NY, USA, 2009. ACM.

[37] A. Joseph and S. Lodha. Musart: Musical audio transfer function real-time toolkit. In *Proceedings of the 2002 International Conference on Auditory Display (ICAD)*, 2002.

[38] M. Jovic and M. Hauswirth. Measuring the performance of interactive applications with listener latency profiling. In *PPPJ '08: Proceedings of the International Symposium on Principles and Practice of Programming in Java*, pages 137–146, New York, NY, USA, 2008. ACM.

[39] K. Juse, S. Kounev, and A. Buchmann. Petstore-ws: Measuring the performance implications of web services. In *Proceedings of the 29th International Conference of the Computer Measurement Group on Resource Management and Performance Evaluation of Enterprise Computing Systems*, pages 113–123, 2003.

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

[41] P. Killelea. *Web Performance Tuning, 2nd Edition.* O'Reilly, 2002.

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

[43] 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.

[44] K. Krogmann, C. M. Schweda, S. Buckl, M. Kuperberg, A. Martens, and F. Matthes. Improved feedback for architectural performance prediction using software cartography visualizations. In *QoSA '09: Proceedings of the 5th International Conference on the Quality of Software Architectures*, pages 52–69, Berlin, Heidelberg, 2009. Springer-Verlag.

[45] S. M. Lambert Surhone, Miriam Timpledon. *Unusual Software Bug: Software Bug, Computer Bug, Command-Query Separation, Incertainty Principle, Copenhagen Interpretation, Observer Effect (Physics)*. Betascript Publishing, 2010.

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

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

[48] McManus, Eamonn. Java Management Extensions Specification (JSR 3). http://www.jcp.org/en/jsr/summary?id=3, 2010.

[49] S. C. Mellon. Published appraisal results, 2010. http://sas.sei.cmu.edu/pars/pars.aspx.

[50] J. Mellor-Crummey, R. Fowler, and D. Whalley. Tools for application-oriented performance tuning. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 154–165, New York, NY, USA, 2001. ACM.

[51] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 1956.

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

[53] H. F. Olson. *Music, Physics and Engineering*. Dover Publications, 1967.

[54] Oracle. Glassfish - Open Source Application Server, 2010. https://glassfish.dev.java.net/.

[55] Oracle. Glassfish usage map, 2010. http://maps.glassfish.org/server/.

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

[57] 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.

[58] N. Roman, D. Wang, and G. Brown. Speech segregation based on sound localization. *The Journal of the Acoustical Society of America*, 114:2236–2252, 2003.

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

[60] A. H. Sandra Pauletto. A toolkit for interactive sonification. In *Proceedings of International Workshop on Interactive Sonification*, 2004.

[61] J. Shirazi. *Java Performance Tuning, 2nd Edition*. O'Reilly, 2002.

[62] T. H. A. J. S.K. Lodha, J. Beahan and B. Zne-Ulman. Muse: A musical data sonification toolkit. In *Proceedings of the 1997 International Conference on Auditory Display (ICAD)*, 1997.

[63] I. Sommerville. *Software Engineering, 8th Edition*. Pearson Education Limited, Essex, England, UK, 2007.

[64] D. M. Steven Fraser. No silver bullet: Software engineering reloaded. *IEEE Software*, 25(1):91–94, 2008.

[65] The Eclipse Foundation. Eclipse - an open development platform, 2010. http://www.eclipse.org.

[66] A. H. Thomas Hermann. The discipline of interactive sonification. In *Proceedings of International Workshop on Interactive Sonification*, 2004.

[67] A. Tiwari, V. Tabatabaee, and J. K. Hollingsworth. Tuning parallel applications in parallel. *Parallel Comput.*, 35(8-9):475–492, 2009.

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

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

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

[71] 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.

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

[73] B. Walker and J. Cothran. Sonification sandbox: A graphical toolkit for auditory graphs. In *Proceedings of the 2003 International Conference on Auditory Display (ICAD)*, 2003.

[74] B. Walker and D. Lane. Psychophysical scaling of sonification mappings: A comparison of visually impaired and sighted listeners. In *Proceedings of the International Conference of Auditory Display (ICAD 2001)*, 2001.

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

[76] R. J. Walker, G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak. Visualizing dynamic software system information through high-level models. In *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 271–283, New York, NY, USA, 1998. ACM.

[77] 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.

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

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

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

[81] C. Wilson and S. Lodha. Listen: A data sonification toolkit. In *Proceedings of the 1996 International Conference on Auditory Display (ICAD)*, 1996.

[82] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz. Performance analysis using the mips r10000 performance counters. In *Supercomputing '96: Proceedings of the 1996*

ACM/IEEE conference on Supercomputing, page 16, Washington, DC, USA, 1996. IEEE Computer Society.

Appendix A

Complete Results of Sonification Case Study

The following sections provide all the results from case study conducted on the fundamental affordances of sound characteristics in relation to software engineers. When appropriate, data was aggregated into summary tables. Data which could not be sensibly consolidated is provided as tables of anonymized results.

A.1 Demographic Section

Table A.1: Question 1: Programming Experience

Years of Exp.	**# of Participants**	**Percentage**
0-1 Years	1	3%
1-2 Years	3	9%
2-5 Years	7	21%
5+ Years	22	67%

Table A.2: Question 2: Programming Environment Preference

Environment	**# of Participants**	**Percentage**
Text Editor	6	18%
Enhanced Text Editor	7	21%
Full IDE	18	55%
Other	2	6%

Table A.3: Question 3: Debugging Environment Preference

Environment	**# of Participants**	**Percentage**
Output Statements	9	27%
Command-Line (gdb)	6	18%
Visual/IDE	15	46%
Other	3	9%

Table A.4: Question 4: Profiling Tool Usage

Usage	**# of Participants**	**Percentage**
Yes	7	21%
No	26	79%

Table A.5: Question 5: Music Background

Music Exp.	**# of Participants**	**Percentage**
None	1	3%
Casual	4	12%
Intermediate	17	52%
Expert	11	33%

Table A.6: Question 6: Music Listening Frequency while Programming

Frequency	**# of Participants**	**Percentage**
Never	4	12%
Rarely	6	18%
Sometimes	8	25%
Often	9	27%
Always	6	18%

Table A.7: Question 7: Music Genre Preference

Genre	**# of Participants**	**Percentage**
Pop	14	42%
Rock	25	76%
Indie	14	42%
Classic Rock	16	48%
Classical	18	55%
Country	4	12%
Rap/Hip Hop	10	30%
Blues	6	18%
Metal	8	24%
Other	13	39%

Table A.8: Question 8: Right or Left Handed

Hand	**# of Participants**	**Percentage**
Right	6	18%
Left	27	82%
Ambidexerous	0	0%

Table A.9: Question 9: Hearing Aid or Hearing Problems

Hand	**# of Participants**	**Percentage**
Yes	0	0%
No	33	100%

A.2 General Section

Table A.10: Question 1: Pitch Increasing

Value Change	**# of Participants**	**Percentage**
Increasing Alot	28	84.84849%
Increasing a Little	4	12.121212%
Constant	0	0.0%
Decreasing a Little	0	0.0%
Decreasing Alot	1	3.030303%

Table A.11: Question 2: Pitch Decreasing

Value Change	**# of Participants**	**Percentage**
Increasing Alot	0	0.0%
Increasing a Little	0	0.0%
Constant	0	0.0%
Decreasing a Little	5	15.151516%
Decreasing Alot	28	84.84849%

Table A.12: Question 3: Loudness Increasing

Value Change	**# of Participants**	**Percentage**
Increasing Alot	2	6.060606%
Increasing a Little	22	66.66667%
Constant	9	27.272728%
Decreasing a Little	0	0.0%
Decreasing Alot	0	0.0%

Table A.13: Question 4: Loudness Decreasing

Value Change	**# of Participants**	**Percentage**
Increasing Alot	0	0.0%
Increasing a Little	2	6.060606%
Constant	9	27.272728%
Decreasing a Little	22	66.66667%
Decreasing Alot	0	0.0%

Table A.14: Question 5: Pan Left to Right

Value Change	**# of Participants**	**Percentage**
Increasing Alot	6	18.181818%
Increasing a Little	5	15.151516%
Constant	22	66.66667%
Decreasing a Little	0	0.0%
Decreasing Alot	0	0.0%

Table A.15: Question 6: Pan Right to Left

Value Change	**# of Participants**	**Percentage**
Increasing Alot	1	3.030303%
Increasing a Little	3	9.090909%
Constant	22	66.66667%
Decreasing a Little	3	9.090909%
Decreasing Alot	4	12.121212%

Table A.16: Question 7: Timbre Change Between Dissimilar Sounds

Value Change	**# of Participants**	**Percentage**
Increasing Alot	13	39.39394%
Increasing a Little	14	42.424244%
Constant	4	12.121212%
Decreasing a Little	2	6.060606%
Decreasing Alot	0	0.0%

Table A.17: Question 8: Timbre Change Between Similar Sounds

Value Change	**# of Participants**	**Percentage**
Increasing Alot	0	0.0%
Increasing a Little	17	51.515152%
Constant	15	45.454548%
Decreasing a Little	1	3.030303%
Decreasing Alot	0	0.0%

A.3 Panning Section

Table A.18: Individual Responses to Panning Questions

Participant	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
P1	56	0	15	48	100
P2	52	0	30	52	78
P3	69	0	37	50	84
P4	58	0	45	51	63
P5	50	0	31	59	100
P6	52	0	28	54	93
P7	65	0	40	47	71
P8	54	0	18	38	91
P9	56	0	16	53	94
P10	67	51	68	87	53
P11	51	0	42	52	100
P12	48	0	48	50	59
P13	66	0	20	42	86
P14	70	0	12	48	100
P15	61	0	37	51	100
P16	52	0	24	49	70
P17	72	7	41	53	86
P18	57	0	40	53	89
P19	65	0	30	52	100
P20	100	0	25	51	76
P21	51	1	26	43	57
P22	59	0	25	44	98
P23	68	0	39	52	100
P24	50	2	56	66	100
P25	55	0	39	50	100
P26	70	0	27	59	100
P27	52	0	35	59	70
P28	52	0	20	43	100
P29	73	0	25	53	95
P30	1	49	0	0	63
P31	53	6	25	47	89
P32	70	0	31	54	100
P33	56	0	45	51	82

A.4 Mapping Section

Part. - Participant

PBC - Playback Count

BHD - Beginning Hard drive

BMEM - Beginning Memory

BNET - Beginning Network

MHD - Middle Hard drive

MMEM - Middle Memory

MNET - Middle Network

EHD - Ending Hard drive

EMEM - Ending Memory

ENET - Ending Network

Table A.19: Question 1

Part.	**PBC**	**BHD**	**BMEM**	**BNET**	**MHD**	**MMEM**	**MNET**	**EHD**	**EMEM**	**ENET**
P1	2	8	10	0	1	6	5	4	2	8
P2	6	6	5	0	0	3	5	5	0	8
P3	2	9	7	0	3	7	9	8	7	9
P4	1	10	7	0	5	7	10	0	6	10
P5	4	4	7	1	0	10	3	4	10	10
P6	6	2	2	0	2	4	5	0	0	8
P7	3	5	10	0	0	5	7	1	4	10
P8	3	2	10	0	1	3	7	0	3	10
P9	1	5	3	0	0	5	0	4	0	8
P10	2	7	7	0	1	5	7	0	3	10
P11	3	8	5	0	1	5	9	8	3	7
P12	3	5	5	0	1	5	4	0	5	10
P13	2	5	7	0	0	6	0	5	5	8
P14	2	8	10	0	2	5	3	2	4	6
P15	1	10	10	0	3	10	5	0	10	10
P16	2	5	2	0	1	4	2	0	2	4
P17	1	5	7	0	2	8	4	0	8	6
P18	3	10	9	0	5	5	5	0	5	10
P19	2	10	8	0	6	10	3	0	5	10
P20	3	10	10	0	0	5	5	5	2	10
P21	2	7	5	0	2	1	3	0	3	9
P22	2	8	10	0	1	9	0	1	9	10
P23	2	3	1	0	2	1	8	0	0	10
P24	1	3	6	0	1	6	5	1	6	9
P25	2	10	5	0	0	10	3	1	10	10
P26	3	10	10	0	0	3	8	0	2	10
P27	3	7	9	0	1	8	8	1	8	9
P28	3	0	1	0	4	1	9	0	5	10
P29	1	5	9	0	2	6	4	3	5	10
P30	4	1	1	1	1	5	3	3	2	8
P31	4	6	8	0	1	5	4	2	3	6
P32	2	5	10	0	0	5	0	5	3	10
P33	1	1	10	0	2	10	1	3	10	9

Table A.20: Question 2

Part.	PBC	BHD	BMEM	BNET	MHD	MMEM	MNET	EHD	EMEM	ENET
P1	1	10	2	2	2	4	8	0	7	7
P2	5	8	5	2	3	5	8	2	5	7
P3	2	8	8	2	2	8	8	2	8	9
P4	2	9	6	0	4	9	7	1	5	8
P5	5	10	10	0	2	10	0	7	10	0
P6	4	10	5	7	2	6	6	2	6	6
P7	3	9	5	5	3	5	8	2	5	8
P8	2	10	3	3	2	4	4	1	7	5
P9	1	7	7	4	5	8	6	2	8	5
P10	2	8	5	5	2	4	9	1	4	9
P11	2	9	6	8	4	5	10	1	8	10
P12	4	10	5	5	0	5	10	0	5	10
P13	2	10	6	3	3	5	6	3	5	10
P14	2	10	0	3	5	10	10	1	8	8
P15	1	10	10	10	6	10	10	7	10	10
P16	3	5	3	0	5	4	0	4	6	0
P17	1	4	4	2	0	7	3	0	7	3
P18	2	8	4	1	1	3	3	0	0	3
P19	5	10	5	0	0	3	8	0	3	10
P20	3	10	10	5	3	5	10	0	5	8
P21	1	10	5	2	2	2	6	1	3	6
P22	2	9	2	2	2	3	5	3	3	6
P23	4	10	4	7	3	4	10	2	4	9
P24	2	9	2	0	3	5	7	2	5	3
P25	3	10	5	0	3	5	3	0	5	5
P26	1	10	10	0	2	10	0	2	10	0
P27	2	8	0	2	1	4	8	0	2	9
P28	5	10	5	0	5	10	0	2	9	0
P29	1	10	4	1	5	5	9	5	6	5
P30	3	5	2	2	3	5	5	2	7	7
P31	2	5	5	5	5	5	5	5	5	5
P32	2	10	10	0	0	0	3	0	0	7
P33	2	5	2	1	4	2	4	1	7	9

Table A.21: Question 3

Part.	**PBC**	**BHD**	**BMEM**	**BNET**	**MHD**	**MMEM**	**MNET**	**EHD**	**EMEM**	**ENET**
P1	1	3	0	5	5	5	0	7	7	1
P2	2	1	5	6	5	5	5	5	5	7
P3	4	7	2	3	7	7	7	8	8	3
P4	1	5	3	8	7	7	3	6	10	2
P5	1	3	2	5	6	9	10	6	10	3
P6	3	3	1	6	6	7	3	6	6	2
P7	6	5	0	7	8	6	6	8	7	4
P8	1	7	1	4	10	8	2	9	8	0
P9	1	3	0	8	6	4	6	3	9	2
P10	3	3	3	7	5	6	6	5	7	6
P11	1	4	5	7	7	8	2	10	8	2
P12	1	3	0	3	5	3	3	5	5	0
P13	3	7	1	3	7	6	2	7	6	2
P14	2	4	2	2	10	8	5	10	10	4
P15	1	10	0	10	10	5	10	10	10	5
P16	3	3	2	4	5	5	5	5	7	2
P17	1	3	1	2	4	1	3	6	2	4
P18	2	0	0	3	9	8	3	10	10	0
P19	3	0	2	10	8	8	0	10	5	3
P20	4	2	0	2	7	5	2	7	5	0
P21	1	1	2	2	5	5	2	5	8	1
P22	1	0	0	10	7	9	10	8	8	2
P23	1	5	2	6	5	4	4	5	7	1
P24	4	3	2	5	8	8	5	8	8	2
P25	2	2	1	5	5	5	5	5	5	0
P26	2	0	0	5	5	8	6	8	10	2
P27	1	1	0	8	7	6	3	7	1	7
P28	1	5	0	5	5	5	5	5	10	0
P29	1	3	3	3	5	6	9	5	10	8
P30	1	5	5	6	5	5	1	7	8	2
P31	2	3	3	1	6	5	2	7	6	2
P32	1	10	0	0	10	10	9	10	10	0
P33	2	2	2	3	5	6	3	5	9	3

Table A.22: Question 4

Part.	**PBC**	**BHD**	**BMEM**	**BNET**	**MHD**	**MMEM**	**MNET**	**EHD**	**EMEM**	**ENET**
P1	1	8	3	3	1	10	1	8	0	2
P2	2	5	5	6	3	8	6	7	3	6
P3	1	7	5	8	2	10	9	7	2	4
P4	1	7	2	7	2	9	4	9	2	2
P5	1	3	3	3	1	6	4	2	10	2
P6	3	5	5	5	0	9	4	5	0	5
P7	3	8	4	7	2	10	8	9	2	7
P8	2	9	2	6	2	10	6	8	2	10
P9	1	3	3	3	1	5	3	8	2	4
P10	1	5	6	6	4	9	6	6	2	6
P11	1	8	3	5	2	10	4	5	6	2
P12	2	5	6	1	0	10	0	5	0	0
P13	2	6	7	3	1	10	6	7	2	4
P14	2	10	4	4	1	10	3	10	2	4
P15	1	10	10	10	5	10	5	10	6	5
P16	3	4	4	2	4	0	5	6	8	5
P17	1	5	3	2	1	10	3	8	0	3
P18	1	10	0	3	3	10	5	10	2	2
P19	1	10	5	10	0	10	5	4	1	10
P20	2	8	5	5	2	9	3	9	1	3
P21	1	5	2	5	1	9	4	3	8	3
P22	2	5	0	7	1	9	7	8	0	3
P23	1	3	1	3	1	7	4	6	2	5
P24	1	5	5	0	1	8	5	9	5	5
P25	3	9	2	4	0	8	4	9	3	4
P26	1	10	2	10	2	10	7	10	0	3
P27	1	5	3	10	1	9	10	6	3	9
P28	2	5	5	5	1	10	5	9	1	5
P29	1	5	6	5	2	9	5	9	1	5
P30	2	2	5	4	1	7	1	8	1	1
P31	1	7	5	7	2	7	4	6	1	7
P32	1	10	5	5	0	10	5	10	0	2
P33	2	2	4	4	2	9	4	2	5	4

Table A.23: Question 5

Part.	PBC	BHD	BMEM	BNET	MHD	MMEM	MNET	EHD	EMEM	ENET
P1	1	0	0	0	8	8	0	8	8	0
P2	2	2	5	2	4	4	4	3	3	3
P3	1	1	1	1	7	7	2	9	9	2
P4	1	0	0	0	6	7	0	5	10	0
P5	1	2	2	1	2	5	3	7	5	6
P6	2	0	0	0	5	5	1	8	8	0
P7	1	0	0	0	10	10	0	5	9	0
P8	1	0	0	0	10	10	0	4	10	0
P9	1	2	2	2	6	6	2	10	8	2
P10	2	2	2	1	8	8	1	5	8	1
P11	1	5	7	2	7	6	7	10	8	10
P12	1	0	0	0	5	5	0	10	10	0
P13	2	0	0	0	4	8	0	8	9	0
P14	1	2	2	2	6	10	1	9	10	2
P15	1	8	10	0	8	10	5	8	10	8
P16	3	2	2	1	7	7	3	5	9	2
P17	1	0	0	0	10	10	0	3	10	0
P18	1	0	0	0	9	9	0	5	5	0
P19	1	0	0	0	0	10	0	10	10	0
P20	1	0	0	0	8	9	5	1	9	5
P21	1	5	1	5	2	5	4	2	3	5
P22	1	0	0	0	5	9	0	8	10	0
P23	1	0	0	0	5	6	0	8	8	1
P24	1	1	1	1	5	8	1	9	9	0
P25	1	0	0	0	5	5	0	10	7	0
P26	1	0	0	0	7	10	0	7	10	0
P27	1	0	0	0	0	8	0	8	8	0
P28	2	5	5	2	8	8	5	8	8	1
P29	1	0	0	0	10	5	0	5	10	0
P30	2	5	5	5	4	4	2	5	5	5
P31	1	4	4	1	4	4	2	3	2	3
P32	1	0	0	0	10	5	0	10	10	5
P33	2	1	1	1	5	9	1	8	9	1

Table A.24: Question 6

Part.	PBC	BHD	BMEM	BNET	MHD	MMEM	MNET	EHD	EMEM	ENET
P1	1	10	0	10	0	8	0	10	0	1
P2	1	6	0	5	2	6	2	8	0	1
P3	2	8	2	7	2	7	4	8	2	2
P4	1	5	5	5	0	5	2	7	0	2
P5	1	5	5	5	2	1	5	10	8	2
P6	3	9	0	2	0	9	0	9	0	0
P7	2	9	0	9	0	9	2	9	0	2
P8	2	8	0	8	0	10	7	10	0	2
P9	1	5	0	5	0	5	2	9	0	2
P10	4	9	0	9	2	8	3	8	0	2
P11	1	8	0	7	2	7	3	9	2	2
P12	1	5	5	5	0	8	0	9	0	1
P13	2	8	2	8	2	7	2	7	1	2
P14	3	5	1	4	0	10	0	10	0	0
P15	1	10	0	10	0	10	0	10	0	0
P16	2	5	5	5	5	5	5	5	5	5
P17	1	5	5	5	2	10	4	10	3	2
P18	2	8	0	8	1	8	3	9	0	1
P19	1	10	0	10	0	10	0	10	0	5
P20	1	9	0	9	1	8	5	10	0	0
P21	1	8	1	8	0	8	4	10	0	0
P22	2	10	2	10	0	10	4	9	1	1
P23	1	6	0	8	0	6	5	8	0	2
P24	1	8	0	5	1	7	1	9	1	3
P25	1	10	0	10	0	8	5	10	3	0
P26	1	5	5	5	0	10	0	10	0	0
P27	1	5	5	5	0	9	5	8	0	0
P28	1	9	10	10	5	5	10	10	0	1
P29	1	5	5	5	0	10	3	8	4	1
P30	1	4	4	2	2	7	3	9	1	4
P31	5	8	0	8	4	6	0	8	0	0
P32	1	10	6	10	1	9	0	10	0	3
P33	4	5	4	4	1	9	4	9	1	4