Art and Understanding through Code Visualisation

Arrian Purcell

A subthesis submitted in partial fulfillment of the degree of
Bachelor of Software Engineering (Honours) at
The Department of Computer Science
Australian National University

October 2014

Sections of this thesis are to appear in: Arrian Purcell, Henry Gardner, and Ben Swift, *Visualising a Live Coding Arts Process*, OzCHI 2014 Proceedings (forthcoming).

Except where otherwise indicated, this thesis is my own original work.

Arrian Purcell
24 October 2014

"If you look around at the world and where technological surprises are happening, one place is in the art world – artists who are using tech to create new experiences."

– Peter Lee, *Head of Microsoft Research*, July 2014

Acknowledgements

This has been an incredibly interesting and challenging year. I have many to thank for their support and guidance through the challenges encountered.

I would like to thank my supervisors, Henry Gardner and Ben Swift for their motivating words and inspiring dreams of what can be achieved through visualisation, live coding, and the combination of the arts and sciences. In particular, my appreciation goes to Henry for his positivity, guiding me and pushing me when I needed it the most. My appreciation goes to Ben for his dedication and patience in seeing this project succeed and the many hours he has spent in support of this.

My appreciation goes to those who have kept the office environment interesting. Charles, for the conversation and support particularly during the critical periods. Torben and Matt, for the warm welcome and interesting perspectives.

Finally, I would like to thank my family, friends and work colleagues who have put up with my absence or vacant stares throughout this project. In particular I would like to thank Emily and Damien for their support of the project and making the honours environment interesting; Ben Frommel, for his support through the project, despite my absence, the adventures we have had and adventures still to come; and finally, my mother for an overwhelming level of support throughout the year. Thank you all.

Abstract

The visualisation of source code has been examined as a method of enhancing audience experience within "live coding". Following an exploratory field study examining the experience of the audience observing source code projections, two process-driven source code visualisation techniques were incorporated into a live coding system. These visualisations were evaluated within a user study during a series of live coding performances, examining the self-reported experiential dimensions of *understanding* and *enjoyment*. Through a process of refinement, a follow-up user study was conducted on a second iteration of the visualisation techniques comparing visualisations with no visualisations to determine if the visualisations had enhanced the audience's experience.

The exploratory field study examined the validity of the dimensions of *understanding* and *enjoyment* for evaluating visualisations within live coding and indicated the potential for visualisations to enhance these dimensions through the live coding performance. Results of the initial user study identified the ability to manipulate the dimensions of understanding and enjoyment through the *beginning*, *middle* and *end* of the live coding performance. Finally, the follow-up user study determined the effectiveness of the visualisations over no visualisations within live coding. Minor increases in both understanding and enjoyment were identified for these visualisations, with an identified increase in understanding for those initially with lower understanding and an increase to enjoyment throughout the middle stages of the live coding performance. Results of the three studies are indicative of the potential for visualisations within live coding to enhance the experience of the audience.

Contents

List of Figures

List of Tables

List of Acronyms

IDE integrated development environment

OSC Open Sound Control

SoW state of the world

SoC state of the code

SDLC software development life cycle

SLOC source lines of code

VU volume unit

HCI human-computer interface

Chapter 1

Introduction

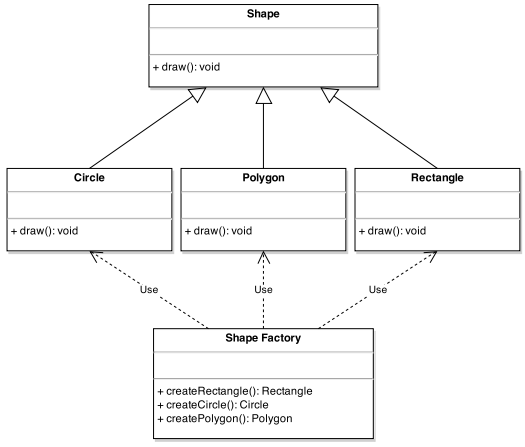
> *"In spite of progress in restricting and simplifying the structures of software, they remain inherently unvisualizable, thus depriving the mind of some of its most powerful conceptual tools. This lack not only impedes the process of design within one mind, it severely hinders communication among minds."*
>
> – Frederick Brooks, *The Mythical Man-Month*, p.185

A fundamental challenge of modern software is of utility. Questions regarding how software interacts with the world and how those that use the software benefit from it drive much of modern software engineering research. There is no doubt software is becoming more useful. However, software is also becoming more complex and the methods of interacting with software are diversifying.

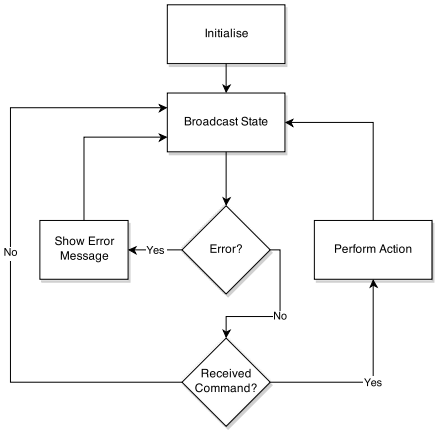
Live coding is one area in which the utility of software has not been thoroughly examined. Live coding, the artistic process of programming for an audience, asks the performer (the live coder) to show the audience their screens. This results in many live coding performances projecting raw source code for the audience. Immediately, this raises two questions: "why show the screen at all?" for audiences that have no programming experience; and "what can be gained by showing the source code?". Answering these questions would not only have immediate benefit in understanding how to enhance the audience's experience during live coding performances but could also have the wider benefit of communicating the programming process.

Visualisations have been suggested as a method of assisting the audience in the appreciation and comprehension of live coding performances. Historically in software, visualisations have attempted to provide a higher level of understanding of the structure of programs but have rarely been effective in communicating the fundamental processes of software development. *The Mythical Man-Month* goes so far as to suggest that "software is invisible and unvisualizable"[Brooks 1995] sparking debate as to the effectiveness of current software visualisation techniques and whether software can be visualised at all.

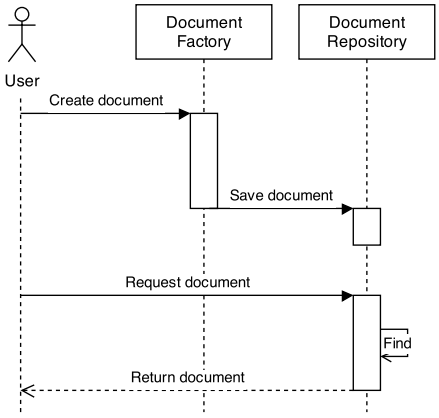
There has been a push to develop effective software visualisations as the audience of software becomes more diverse, multidisciplinary teams within the industry become the standard, and multimedia and the arts become more focussed on effective software development. Live coding presents a space in which a diverse audience and dynamic software is common and in which visualisations could provide significant benefit



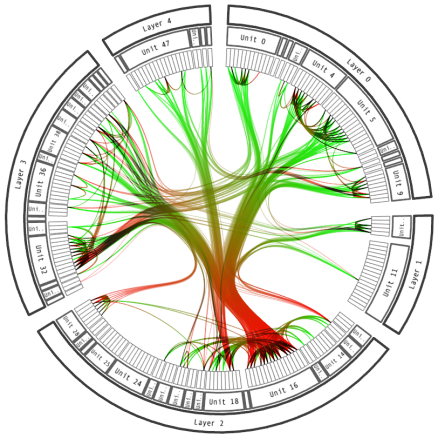
(a) Class diagram



(b) Flow chart



(c) Sequence diagram



(d) Edge bundled graph [Holten 2006]

Figure 1.1: Existing software diagramming and graphing techniques.

to the experience of observers. Live coding is a means of combining the artistic goal of enjoyment and the educational aspects associated with programming with visualisations to effectively communicate software.

Recently, the development of live coding has presented a unique application space for visualisations and the potential for effective communication directly with audiences. These new developments could challenge the long-held belief that "software is invisible and unvisualizable".

This thesis investigates the proposition that "code visualisations improve the experience of observers" in the setting of live coding. More specifically, this thesis investigates the question: "can the application of visualisation techniques to live coding enhance audience experience by increasing understanding and enjoyment?". These questions were examined through a process of prototype development, user study evaluation and refinement.

1.1 Background

For most of its history, program source code has been displayed as simple text. This is due to the expressiveness of the text format and despite the inefficiencies of representing algorithms and abstractions in natural language.

Recently, due to ever increasing programming language complexity, increasing screen fidelity and increasing computational power, code annotations and syntax highlighting have become commonplace. Nevertheless, these visual enhancements rarely provide information beyond the basic grammar of the language they are intended to augment. The limitations of this approach are becoming ever more apparent as programming languages and interactive programming environments move towards the need for real-time comprehension and a need to understand the source code within the context of a running program.

Historically, source code diagrams have attempted to display the high level structure of source code. For example, class diagrams (see Figure 1.1a) are commonly used to document the structure of a program whereas flowcharts (see Figure 1.1b) have been used extensively to represent simple software behaviours. Sequence diagrams (see Figure 1.1c) have allowed the visualisation of interacting objects and actors in a software system, putting the focus on the users of the software system rather than the structure of the program. Edge bundling of software call hierarchies [Holten 2006; Zhou et al. 2013] (see Figure 1.1d) has allowed the visualisation of the lifecycle of a program, showing which software components were called at what time and the volume of calls made.

1.1.1 Hot Swapping Source Code

Various modern software environments now support the concept of hot swapping source code. Within software environments, hot swapping refers to the process of modifying program source code at runtime to achieve desired behaviour without interrupting or restarting the system. Examples of languages and platforms supporting this workflow include Java with HotSwap or JRebel [ZeroTurnaround 2014] and Python, in addition to a variety of other custom platforms and modifications (e.g. [Würthinger et al. 2011]). This workflow has also inspired the programming practice of "live coding".

Live coding further develops the concept of hot swapping source code, applying it to live musical and visual performance. In live coding a programmer performs for an audience. The audience observes the programmer as they modify an active program to produce music and visuals. Some of the most important and influential live coding environments over the last decade have included SuperCollider [McCartney 2013], ChucK [Wang 2008] and Extempore [Sorensen 2013]. These environments have a growing following as the field of live coding matures.

Figure 1.2: A typical live coder projection includes the raw source code of the musical performance such as the screen pictured above. Audiences observe the changes to the source code on a projector screen while listening to the musical output. In the source code pictured here, the live coder is making changes to the musical "bassline" during a performance.

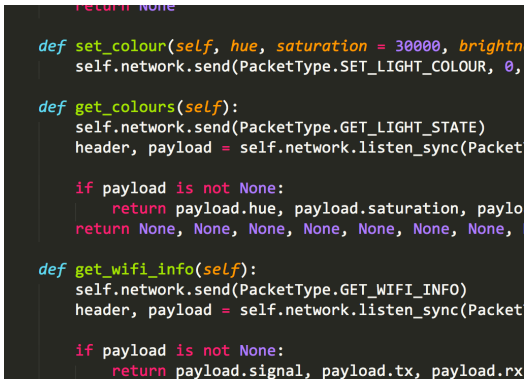
1.1.2 Live Coding

Live coding can be broadly defined as writing a program while it runs [Ward et al. 2004]. More specifically, live coding is identified as the artistic process of musical and visual expression through programming [Collins and McLean 2003]. Figure 1.2 shows the standard screen projected for a live coding audience. The audience views the raw source code and the programming process as the programmer makes changes to the source code.

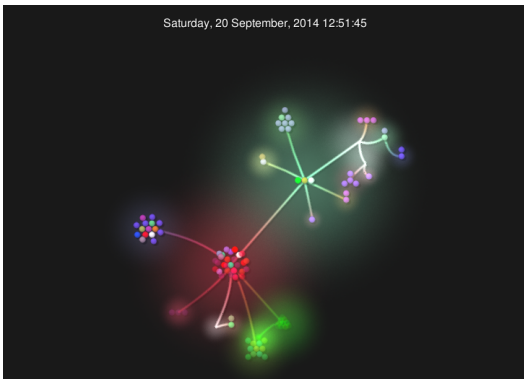
Live coding emphasises the concept of *liveness*, a concept covered comprehensively within the literature [Auslander 2008; Masura 2007]. Liveness is foundational to the performance aspect of live coding in which the mantra is "show us your screens" [Toplap 2010], asserting that all live coders should display their raw program source code to the audience.

A unique opportunity is provided through live coding to combine both source code and software visualisation techniques [Mclean et al. 2010]. This is due to the approach of live coding involving effective sensory communication, a goal of transparency of the coding process, and direct manipulation and refinement of the running program.

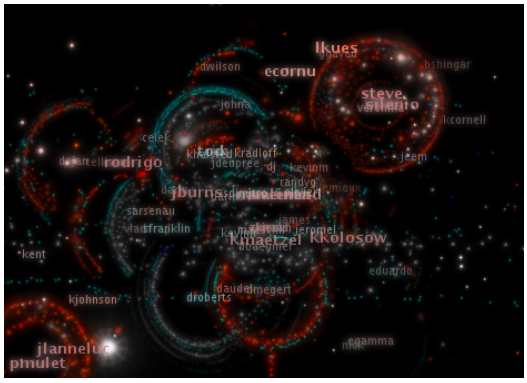
Firstly, live coding is built around communicating visually and audibly. This extends beyond the physical process of programming. Human input is visible in the creative process, demonstrating a link between physical actions and artistic output [McLean and Wiggins 2010].



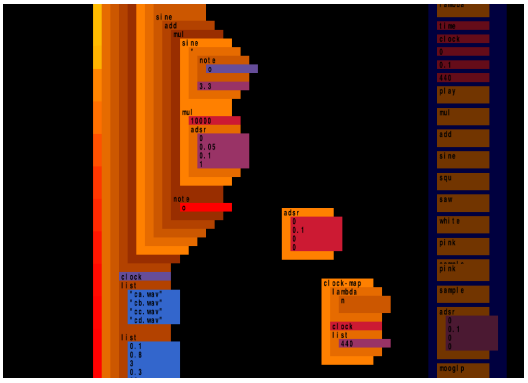
(a) Syntax Highlighting



(b) Gource [Caudwell 2010]



(c) Code Swarm [Ogawa 2012]



(d) Scheme Bricks [Mclean et al. 2010]

Figure 1.3: Existing software visualisations techniques.

Secondly, live coding has a history of exposing audiences to source code with the goal of ensuring the transparency of the coding process [Collins 2011; Mclean et al. 2010]. It provides a space in which there is a direct mapping from the human interaction with the source code to the musical or visual output [McLean and Wiggins 2010]. This relationship allows for visuals to map the interaction with the output.

Lastly, live coding allows for direct manipulation and refinement of the running program [Swift et al. 2013]. This provides the capacity for uninterrupted visualisation of the dynamic aspects of the program combined with the static nature of the source code. Similarly, manipulation and refinement of the running program allows for manipulation of the software visualisation [Mclean et al. 2010].

1.2 Related Work

As noted above, a number of existing software visualisation techniques exist. Some of the most common software visualisations include static diagrams (see Figure 1.1) and syntax highlighting (see Figure 1.3a). These techniques generally model the structure of the program to allow for more effective navigation around structures and take

advantage of visual memory.

While diagrams and syntax highlighting provide information regarding the structure of source code, the fundamental nature of software is to change [Brooks 1995]. Understanding these changes requires visualising the software development process. Techniques to visualise the changes occurring during the software development process include Gource [Caudwell 2010] (Figure 1.3b) and Code Swarm [Ogawa 2012] (Figure 1.3c). These visualisation techniques show historic source code changes, based on source code repository data.

Showing historic source code changes allows the process of programming to be understood. However, this information is not immediately useful to programmers or observers. Visualisation of active software has shown potential in providing useful and interesting information, allowing the programmer to take actions based on the visualisations and allowing the observers to make relevant judgement of what actions the programmer is taking. Visualisation techniques such as inline annotations [Swift et al. 2013; Beck et al. 2013] provide direct feedback useful to both the programmer and observers, and technologies such as Light Table [Kodowa 2014] with an inline interactive terminal and immediate visual feedback after hot swapping source code have further identified how useful visual feedback can be.

A number of attempts have been made in visualising active software within live coding. Examples of existing software visualisations within live coding include Scheme Bricks (Figure 1.3d), Daisy Chain and Betablocker [Mclean et al. 2010]. These visualisation techniques suggest potential within live coding for effective process-driven visuals to enhance the observer's experience. However, no systematic studies of process-driven visualisations have been conducted within live coding.

1.3 Structure

This thesis discusses software visualisations which have been applied to live coding through a process of design iteration and evaluation in collaboration with a live coding artist. Three studies were administered, including one field study and two laboratory studies as a means to determine if understanding and enjoyment could be influenced by the introduction of visualisations to live coding.

Chapter 2 of this thesis summarises the literature including the basis for and the direction of this thesis. The current state of software visualisations and the state of the application of visualisations to live coding are examined with particular reference to the goals and future directions of the field.

Chapter 3 discusses the initial exploratory field study conducted to investigate existing perception and understanding of the live coding process. The field study explored the current state of live coding practice, examining the process of live coding, the effect on the audiences of live coding and the expectations of the audience.

Chapter 4 discusses the first iteration of the visualisation prototype developed following the results of the exploratory field study. The process of development is described with particular focus on software design within the space of an artistic

process. This process involved collaboration with a live coding artist to develop software visualisations appropriate for the space of live coding.

Chapter 5 summarises the initial user study conducted with the visualisation prototype. This study compared visualisation features in reference to enhancing the experience of the live coding audience. The study used surveys and video-cued recall techniques to examine live coding in the context of an artistic practice.

Chapter 6 discusses refinement of the visualisation prototype driven by the results of the initial user study. The initial user study identified a number of areas of improvement within the first iteration of the visualisation prototype and provided direction for following prototypes. The visualisation refinement took these concepts and applied them to the first iteration of the visualisation prototype.

Chapter 7 discusses the follow-up user study conducted to analyse the refined visualisations. A similar approach to the initial user study was taken, examining live coding audiences within the context of a live coding performance. However, in this case the examination compared the refined visualisation prototype with presenting just the plain source code within the live coding performance.

Chapter 8 summarises the results of the user studies, contributions, limitations and future work. Results of the field study and the two user studies are compared and the findings are discussed. Following this, the contributions made to the field of software visualisation and the visualisation of live coding are listed. Limitations with the contributions and limitations of the studies are then discussed. Future work in the field of software visualisation and the visualisation of live code is discussed with particular reference to developments of the methodology, improvements to the user studies and application spaces of the visualisation techniques discussed. A reflection on the process of combining a scientific approach with the arts is presented, followed by some closing words to conclude the thesis.

Chapter 2

Literature Review

Visualisation is building momentum within the space of live coding. This chapter seeks to identify the reason for this momentum and identify the purpose and potential for visualisations within this field.

2.1 Software

Understanding changing software has been identified as one of the most important goals in software engineering practice [Tao et al. 2012]. It is the nature of software to change [Brooks 1995] and there is a need for not only the programmer to understand the software but also for knowledge transfer to take place between those modifying the software and those impacted by the changes [Tao et al. 2012].

Programming languages are the formal languages of software. These languages are typically represented by source code in a plain text format. However, plain text format is limited, requiring an interpretation step (parsing and compilation) to achieve a functioning program that can be understood by the computer [Badros 2000]. The same occurs while programming. The programmer needs to comprehend the source code in order to make informed changes. In this case, the programmer conducts the interpretation step within the brain, rather than the computer, through a process of hypothesis creation, confirmation and refinement [Brooks 1983].

The steps involved in, and the issues with, interpreting and comprehending source code have been comprehensively examined within the literature (see [Novais et al. 2013; Mclean et al. 2010; Brooks 1995; Desmond et al. 2006; Rajlich and Wilde 2002]). Nevertheless, although many studies discuss the limitations of text-based source code, comparatively few have conducted empirical user studies examining the effectiveness of alternative representations of source code.

The concept of alternative source code representations is not new. Alternative representations of the source code include diagrams [Rumbaugh et al. 2004], visual languages [Cox 2007] and combinations of the two (e.g. [Lucanin and Fabek 2011]). Modern software development environments may also include tools that allow for alternative representations of code [Cox 2007]. Diagrams, visual languages and modern software development environments vary greatly in relation to the source code with visualisations representing many levels of abstraction [Jerding and Rugaber 1997].

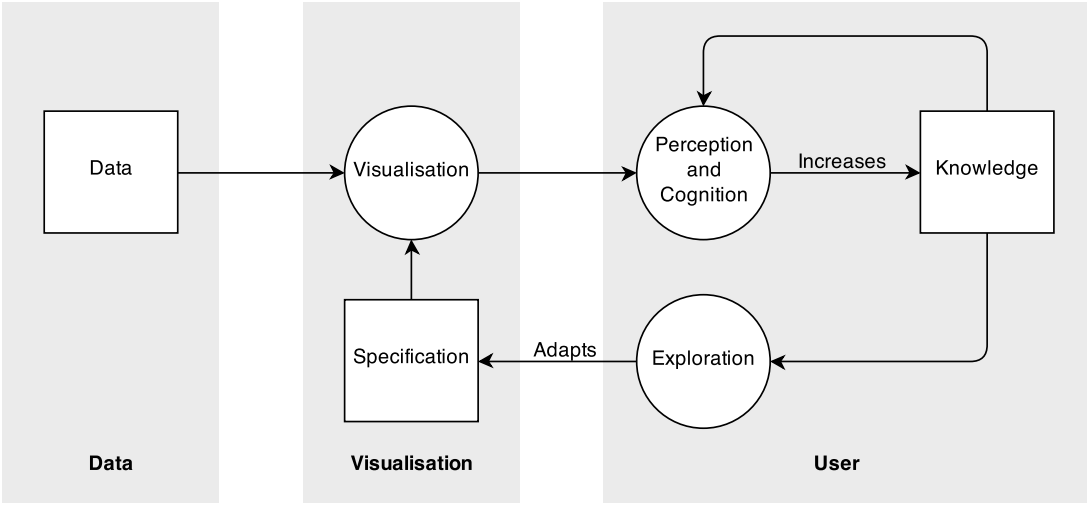


Figure 2.1: Generic model of visualisation [van Wijk 2005].

Despite the differences in the level of abstraction and goal of the alternative representations, these approaches are all related through their common use of visualisation techniques.

2.2 Visualisation

Visualisation is widely understood as "the use of computer-supported, interactive, visual representations of data to amplify cognition" [Card et al. 1999]. Further extensions of this definition discuss the need to perform cognitive work more efficiently [Ware 2013] and the need to transfer knowledge [Burkhard 2004]. These definitions are summarised in the model shown in Figure 2.1. This model shows the feedback loop attributed to a change in knowledge. Changes to knowledge impact perception but may also impact the user's direction of exploration of the visualisation.

Software visualisation is the process of representing the characteristics of computer programs visually [Stasko and Patterson 1992] in order to improve understanding [Diehl 2007]. The advantage of providing a visual representation over the more traditional text-based representation is that the text-based approach does not take full advantage of human visual information processing capabilities [Myers 1990].

Initial efforts to classify software visualisations identified two axes: whether the visualisation illustrated the code, the data or the algorithm; and whether the visualisation was static or dynamic [Myers 1990]. Taxonomies characterised software visualisations according to the aspect of the program, the abstractness of the visualisation, the animation and the automation of the visualisation [Stasko and Patterson 1992].

Although it is the nature of software to change, static diagrams have traditionally been used to represent software systems visually. These diagrams typically show the structure (class diagrams) or function (state diagrams) of a software system at a specific

moment in time [Rumbaugh et al. 2004]. The usefulness of these diagrams lies in their ability to represent fundamental structures within the program more concisely than the source code itself. However, it is not possible to represent a program's behaviour using these types of static diagrams [Baecker 1998] and it is difficult to represent the evolution of the software using nothing but static diagrams.

Approaches to dealing with the visualisation of software evolution and realtime feedback have been integrated into modern software development environments. Modern software development systems and integrated development environments (IDEs) may feature syntax highlighting to assist in comprehension [Chen and Marx 2005; Reis and Cartwright 2004], source code annotations, and the generation of visualisations based on the source code [Hendrix et al. 2004], and allow the programmer to interact with the running program. These modern IDEs emphasise the relationship between the source code and the running program but are still fundamentally tools for source code manipulation.

Only the initial steps have been taken in order to implement and evaluate methods of communicating source code intent outside the field of programmer comprehension. Studies hint at the limitations of static diagrams, visual languages and modern software development environments and identify the need for alternative software representations and evaluation of their effectiveness for those with limited programming knowledge and experience.

Visualisations have the capacity to present information more effectively than traditional programming languages. Nevertheless, software visualisations still require significant development to provide a benefit to the observers in the understanding of the complexity of software [Baecker and Price 1998]. Effective software visualisations contribute to making software easier to understand, reflecting the software's history through the lifecycle, facilitating the transfer of knowledge from the programmer to the observer, making important structures visible and managing software complexity [Baecker and Price 1998].

In a process-oriented activity such as live coding, different code visualisation techniques are necessary [Mclean et al. 2010; Magnusson 2011]. However, these academic treatments of code visualisation in live coding adopt a survey-based approach, and the techniques discussed have not been subject to empirical evaluation.

2.3 The Programmer and the Observer

The relationship between the programmer and the observer has significant implications not just within the field of live coding but also within software engineering practice.

Commonly within live coding, the live coding artist is performing in front of an audience. This setup includes the live coder as the programmer and the audience as a passive observer. Similar structures occur within software engineering practice in pair programming and with the concept of the maintenance programmer [Robson et al. 1991]. In this case, rather than a passive role, the observer takes the active role of communicating mistakes or considering new ideas. Due to the similarities between

these two fields, there is potential to apply software engineering techniques to the live coding process, and the potential to generalise the examination of live coding to software engineering practice.

In the literature, live coding audiences are yet to be surveyed as to whether the projected code within a live coding performance gives a greater sense of communication between the programmer and observer or a greater sense of alienation [McLean 2011]. Similarly, studies are yet to be conducted to determine if source code is the best way of communicating the programming process.

2.4 Measuring Audiences' Experiences

There is currently a search for software visualisations that increase enjoyment and understanding [Mclean et al. 2010]. Here enjoyment refers to the perceived benefit gained from observing the visualisations regardless of the level of understanding. Understanding refers to the ability of an observer or audience to comprehend the abstract thinking process of the programmer.

Enjoyment is the most common reaction to the positive effect of media [Vorderer et al. 2004]. Increased enjoyment is usually characterised by "pleasurable affective response to a stimulus" [Green et al. 2004] and is one of the fundamental goals of the modern media entertainment industry. Enjoyment, and the closely related concept of entertainment, are characterised or caused by a wide range of emotional responses including exhilaration, suspense, aesthetic appeal and a sense of achievement [Vorderer et al. 2004]. The literature identifies that further investigation into how design impacts enjoyment is needed [Reed et al. 1999].

Understanding of a particular program is said to be achieved when the program can be explained "in terms that are qualitatively different from the tokens used to construct the source code" [Biggerstaff 1994]. Source code comprehension has a large body of research both within the understanding of text-based source code and within the space of software visualisation [Kot 2005]. In the space of live coding, this understanding is required to avoid a sense of distraction or exclusion [Mclean et al. 2010]. Measuring understanding could identify methods to reduce distraction and exclusion and provide a method to evaluate live coding.

Enjoyment and understanding align closely with the concepts of aestheticism and didacticism. Didacticism refers to the ultimate goal of teaching. This approach mirrors the concept of understanding from the previous section. A didactic approach intends to increase the level of understanding. Aestheticism refers to the ultimate goal of appealing to the senses. This approach mirrors the concept of enjoyment from the previous section. An aesthetic approach intends to increase visualisation usability and increase retention [Cawthon and Moere 2007].

A wide variety of terms exist for describing phenomena related to aestheticism and didacticism. Expressiveness and effectiveness [Hundhausen 1996; Hundhausen et al. 2002], the aesthetic and the pragmatic [Angeli et al. 2006], and the artistic and the pragmatic [Kosara 2007] are similar concepts and comprehensively covered in regards

to software visualisation, music visualisation, usability and preference.

Software visualisations within the space of live coding have the potential to manipulate these two variables [Bell 2014; Mclean et al. 2010]. For example, by increasing visual interest it may be possible to increase aesthetic appeal, though increasing aesthetic appeal may reduce didacticism with increased visual complexity causing confusion.

The educational aspects of software visualisations have been examined in a number of studies [Baecker 1998; Hundhausen and Brown 2007] though few have applied these visualisations to areas outside the field of software engineering and fewer still have investigated software visualisations targeted at those with no programming experience.

There has been an initial examination of the aesthetics in graph drawing [Purchase et al. 1996; Purchase and McGill 2001] and within the space of information visualisations [Cawthon and Moere 2007; Bell 2013], though no studies have examined the effectiveness of process-driven software visualisations. This is particularly the case for the examination of live coded visualisations and the effect on audiences.

Some frameworks for both aesthetic evaluation of visualisations [Cawthon and Moere 2007; Purchase et al. 1996] and didactic evaluation of visualisations [van Wijk 2005] have been developed, though a thorough evaluation of the combination of the two concepts has not been considered.

2.5 Future Directions

Chapter 1 identified the need for enhancing audience's experience within live coding. The literature discussed here demonstrates that those involved in live coding are looking for new and interesting ways of approaching this problem. The visualisation of software is not a new technique for enhancing the audience's experience but no empirical evaluations of the application of visualisation to live coding have been conducted.

Regarding the visualisation of live code, the most important questions identified include "how can visualisations be applied to live coding?" and "what should be used as the metric to measure the success of the application of visualisations?". Furthermore, the limitations of the current methods of software visualisation in general have been identified including how to best visualise the process of programming, how to visualise the evolution of software [Gall et al. 1999] and how to improve understanding and enjoyment of programming.

The model in Figure 2.1 and the investigation of the literature in this chapter identify some limitations in the current understanding and evaluation of visualisations. Firstly, within this model, a gain in knowledge is the only measurement of value and the only identified method for increased cognitive performance and increased exploration. Secondly, this model of visualisation inherently describes *interactive* visualisations, where increased exploration allows for the adaptation of the specification. Many visualisations may not allow for an adaptation step to take place.

The literature has identified that there may be more metrics appropriate for the evaluation of visualisations. Furthermore, live coding appears to be a space in which

it may be possible to apply this model as a *process* in the development of visualisations. Perhaps the exploration phases of this model could be adapted to meet the challenge presented within live coding to develop methods of source code visualisation that enhance the audience's experience, iteratively developing and adapting effective visualisation techniques.

Questions identified through the introduction (Chapter 1) and the literature review include, "why show the source code at all during a live coding performance?" and "are there better methods of demonstrating the programming process?". Systematic and empirical studies are yet to be conducted within live coding to answer these questions. Similarly, no studies have been conducted to evaluate effective metrics for analysis of the live coding process and the display of source code or visuals.

In an attempt to answer some of the questions raised, an exploratory field study was conducted (see Chapter 3) in an effort to define the direction of the development and implementation of software visualisations within live coding.

Chapter 3

Exploratory Field Study

In order to determine a strategy for visualising source code an exploratory field study was conducted at the "You Are Here" arts festival in Canberra (see Figure 3.1). The literature identified the need to re-examine existing models of visualisation and develop a strategy for implementing visualisations within the space of live coding. To this end, an exploratory field study examined the existing understanding and enjoyment (as discussed in [Mclean et al. 2010]) of a live coding audience via a survey distributed after a performance. This examined the live coding performance with only the source code projected without any visualisation. This would provide a baseline for the addition of visualisations to live coding.

In addition to the survey distributed during the performance a set of follow-up email-based interviews were conducted. The purpose of these interviews was to gain insight into the audience's current understanding and enjoyment of the live coding process. Additionally, the relationship between enjoyment and understanding was to be examined. It was hoped that the examination of these factors would further inform the development of visualisations targeted at similar audiences and lead to a more general software visualisation strategy.

3.1 Method

Audience members were asked to fill out a survey (see Appendix A) regarding their perception of and response to the projection of the computer code during the performance. Each audience member was asked to indicate which of a number of curves or trajectories best represented their *enjoyment* and *understanding* of the performer's actions through the performance. These trajectories allowed for "high", "medium", and "low" levels of enjoyment and understanding for the self-determined "beginning", "middle" and "end" of the performance (see Figure 3.2). Additional questions addressed the audience's sense of "liveness" of the performance (c.f. [Auslander 2008]) and whether the projected code was confusing.

Follow-up email questions were distributed after the performance to a number of those in attendance. The follow-up email asked two questions: "What did you *understand* about what was going on with the code being projected? In particular, what did you understand about the relationship between the code and the music?"

Figure 3.1: The standard live coding setup includes a laptop connected to a projector (pictured in foreground) displaying program source code (pictured in background) to an audience.

and “What would you like to understand more about the code in order to enjoy the performance more?”.

3.2 Participants

In total, thirteen survey responses were received. 38% of the respondents stated that they had high exposure to programming through work, study or their hobbies, 31% stated that they had some experience and 31% stated that they had no experience with it.

77% stated that they listen to large amounts of music compared to 23% that stated they only listened to a small amount. 54% stated that they performed music regularly, 16% stated that they performed only occasionally and 31% stated that they had never performed music. Of the respondents, 69% had never been to a live coding performance before.

3.3 Results

3.3.1 Survey

Of the thirteen survey responses received, six audience members showed a high level of enjoyment throughout the whole performance, while the remaining seven responses showed alternating levels of enjoyment. No audience members indicated a low level of

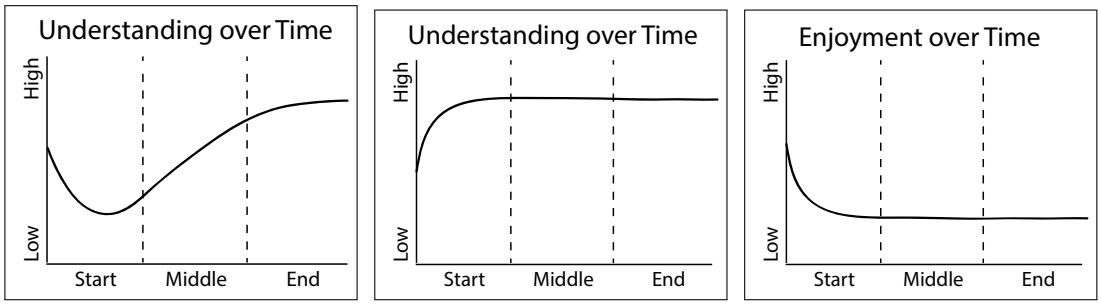


Figure 3.2: Three examples of curves provided to the audience during the exploratory field study. A total of five curve options were provided for both dimensions of understanding and enjoyment. For the understanding dimension, the survey question asked the participants to: "circle the image that best represents your understanding of the relationship between the visuals and the music through the performance." For the enjoyment dimension, the survey question asked the participants to: "circle the image that best represents your enjoyment through the performance."

enjoyment throughout the performance. Only two of the thirteen respondents indicated that they understood the relationship between the code projections and the music throughout the performance. Three of the six respondents who reported a high level of enjoyment throughout the performance also indicated an increase in understanding (from low to high) as the performance progressed, although a Chi-square analysis revealed no significant relationship between enjoyment and understanding.

When asked if the projected code helped to communicate a sense of liveness, nine members of the audience indicated that the projected code helped whereas four members of the audience indicated that the projected code had no effect on their sense of liveness.

Regarding confusion, five members of the audience stated that they found no aspects of the visuals confusing. For the members of the audience that found aspects of the performance confusing, aspects identified as confusing included the small font, the difficulty linking changes in the source code to changes in the sound and the flicking between screens. Four members of the audience did not respond to the question.

3.3.2 Follow-up Interview

The follow-up interview identified some gaps in understanding from the survey. In total, three email responses were received. When asked what these audience members felt they understood about the performance, all three responses indicated high understanding of the initial process of building the music from scratch and developing the musical forms to be modified throughout the performance. One of the responses indicated that they could see changes being made but could not identify which elements within the source code related to which elements in the music. Notably, one audience member stated that they didn't "have a big picture of what all the code looks like" and this caused them difficulty in understanding the live coding process.

Figure 3.3: A live coder performs at the *You Are Here* arts festival in Canberra.

The second follow-up interview question asked what should be made clearer within the space of live coding prompting a variety of responses. Two audience members stated that more understanding of the ideas or source code changes would be desirable. However, one of the responses indicated that a higher understanding of the live coding performance may result in lower enjoyment and suggested that a higher understanding may take focus away from the enjoyable parts of the performance.

Results of the follow-up email interview are available in Appendix B.

3.4 Discussion

A large proportion of the audience indicated that the code was confusing. This confusion was confirmed within the follow-up interviews. Despite this confusion, the projected code still conveyed that the programmer was modifying the music, as indicated by the positive contribution to liveness by the source code identified by 70% of the audience. This was also confirmed within the follow-up interviews in which the interviewees indicated that during the early stages of the performance, the relationship between writing the code and triggering the music was clear.

Some audience members (40%) stated that they found elements of the source code projections confusing and only a very small number of respondents claimed to have actually understood what the programmer was doing. The existence of a small cohort

of respondents whose understanding increased through the performance and whose enjoyment remained high suggested that it might be possible to bring more audience members into this group.

3.4.1 Limitations

Due to the layout of the venue, in some locations the projection screen may have been difficult to see. Additionally, the projection screen surface was uneven with low contrast resulting in difficulties in seeing the projected source code, particularly punctuation. This may have contributed to confusion amongst those with some background in programming.

Results of this study were constrained by sample size. Although the follow-up interview identified some interesting problems within current live coding practice, it was determined that more could be gained from surveying a larger audience.

In addition, the survey was found to be too limited to make more than basic judgements regarding the audience-reported understanding and enjoyment. The survey question asking for the selection of a curve from five options that most closely matched enjoyment and understanding, although providing insight into the audience's opinion of the performance, did not allow for more specific analysis of the stages of the live coding performance.

3.5 Summary

This study identified the need to reduce confusion within the space of live coding. Understanding during the early stages of the performance was shown to be high but drifted as more code was added. During the middle and end of the performance, changes were apparent to the audience but the intention of these changes was not clear. The interviewees indicated that an increase in understanding might contribute positively to their experience.

Similarly, this study identified the need to maintain, and if possible increase, levels of enjoyment while attempting to increase understanding. Reported enjoyment was in general high. However, the follow-up interviews identified that some audience members may equally weight the benefits of enjoyment and understanding, suggesting that a focus only on increasing understanding may be detrimental to the live coding performance.

Taken as a whole, this study indicated the need to investigate methods of increasing understanding without reducing enjoyment. The three respondents whose understanding increased through the performance and whose enjoyment remained high indicated that it might be possible to have similar patterns across the wider audience. It was identified that live coding visualisation techniques might contribute to this goal.

Chapter 4

Visualisation Design

Following the exploratory field study (see Chapter 3), a strategy for visualisation prototyping and evaluation was developed. This strategy was based on the analysis of the literature and the evaluation of the results of the exploratory field study. A collaborative approach was taken, working with a live coder to integrate visuals into the live coding process. The rationale behind developing visualisations, the approach taken to develop the visualisations, and the resulting visualisations, are discussed in this chapter.

4.1 Rationale

The exploratory field study established and examined questions regarding the projection of source code during live coding performances including: "Is source code the best method of communicating the performance?" and "Are there alternative methods?". Results suggested that there was potential for visualisation of the live coding process and identified understanding and enjoyment as valid measures of the audience.

Evaluation of the literature had identified the need for visualisations that helped live coding audiences understand and enjoy the live coding performance. To this end, two sets of visualisations were developed. The first was intended to increase the enjoyment of the audience, enhancing aesthetic appeal and was labelled the "aesthetic visualisation". The second was intended to communicate the live coding process, help the audience to understand the programmatic aspects of the performance and was labelled the "didactic visualisation".

Application to a real live coding performance would be necessary to realistically evaluate the visualisations. To this end, discussions with the live coder identified that critical requirements of the implementation of visualisations would be speed, as the nature of live coding is realtime, and reliability, as a single failure in the visualisations could cause catastrophic failure of the live coding performance.

Typical live coding performances utilise around four instruments. This is not only to maintain musical consistency throughout the performance but also to allow for musical progression. It was identified that to accurately visualise a live coding performance, the visuals would need to have similar progression while using a visual palette of colours, shapes and animations to maintain consistency throughout the performance.

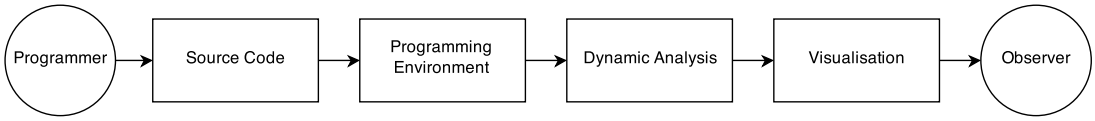


Figure 4.1: Knowledge flow from programmer to observer for the initial visualisation prototype.

4.2 Design

The initial iteration of the visualisation prototype consisted of two separate conditions. Each condition consisted of four visual progressions with the intention of representing the addition and removal of instruments during the live coding performance. Visualisations employed dynamic analysis of the running program to generate the visuals.

The intended model of knowledge flow from programmer to observer is shown in Figure 4.1. The programmer writing source code would activate the source code in the live coding programming environment. During dynamic analysis, information regarding the running program would be collected and organised into the current state of the running program. This dynamic information would then be visualised and projected onto a screen to be observed by an audience.

The first visualisation condition was labelled the "aesthetic" condition reflecting the intention to improve enjoyment during the live coding performance. The second visualisation condition was labelled the "didactic" condition with an intended goal of improving understanding during the live coding performance.

Guidelines from a variety of sources were adapted to develop these visualisations. Some of these guidelines are listed in Table 4.1 and are referred to in the following sections to identify the basis of design decisions.

A number of general standards exist for the design of interfaces and approaches for multimedia human-computer interface (HCI) design (see [ISO 2002] and [Bevan 2006]). However, as the space of the visualisation of live coding has not been fully defined, these standards have yet to be evaluated for relevance and effectiveness. In these visualisations, these standards were taken as guidelines to allow for the possibility of the emergence of new, effective visualisation techniques.

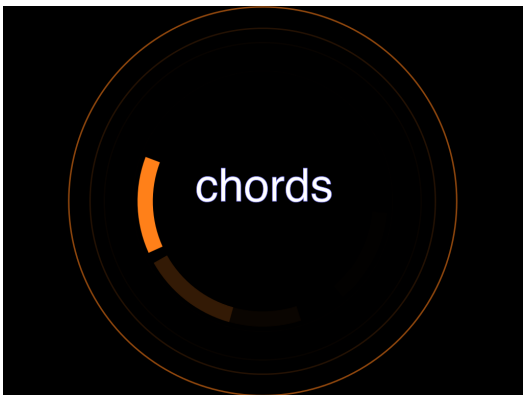
Colour schemes, although varying slightly throughout the visualisations, consisted of a colour palette of red, orange and blue with red and orange used heavily throughout the visualisations and blue used for emphasis and visual interest (Guidelines 4.4 and 4.2). Animations were used throughout to draw attention to parts of the visualisations (see Guideline 4.10).

4.2.1 Didactic Visualisation

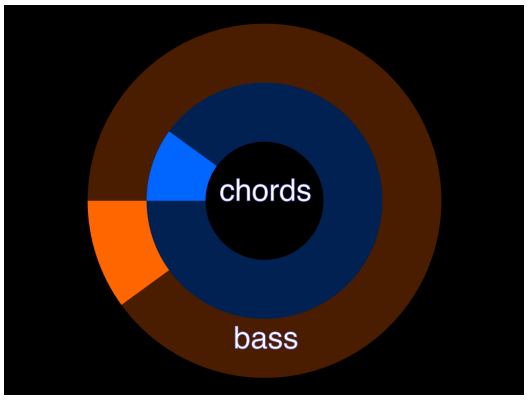
The didactic visualisation (shown in Figure 4.2) attempted to communicate *information* about the actions of the programmer, prominently displaying the *names* of the active (source code) functions and the "time until next execution" for each function. This

Guideline	**Reference**	**Identifier**
Design for important elements to be perceived quickly	[Ware 2013, p. 14]	(Guideline 4.1)
Use colour saturation to represent numerical values	[Ware 2013, p. 117]	(Guideline 4.2)
"Use strong preattentive cues before weak ones where ease of search is critical"	[Ware 2013, p. 156]	(Guideline 4.3)
Use the dominant variables of colour and size to help focus attention	[Borgo et al. 2013, p. 45]	(Guideline 4.4)
Use a distinctive feature channel for a popout effect	[Ware 2013, p. 157]	(Guideline 4.5)
Related symbols should be placed in close proximity	[Ware 2013, p. 181]	(Guideline 4.6)
Use words when the number of categories is few and space is available	[Ware 2013, p. 321]	(Guideline 4.7)
Text should appear "as close as possible to the related parts of a diagram"	[Ware 2013, p. 333]	(Guideline 4.8)
"A one-to-many mapping makes use of redundancies by mapping a data variable to multiple glyph channels. Such a mapping can reduce the risk of information loss by encoding important variables multiple times"	[Borgo et al. 2013, p. 52]	(Guideline 4.9)
"Important variables should be enhanced in the visualization… the mapping should guide the user's focus of attention"	[Borgo et al. 2013, p. 52]	(Guideline 4.10)

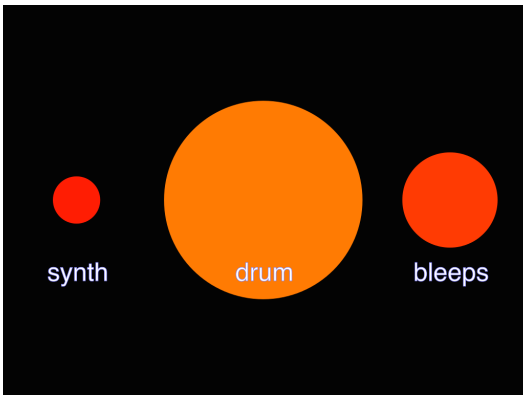
Table 4.1: Guidelines for design decisions.



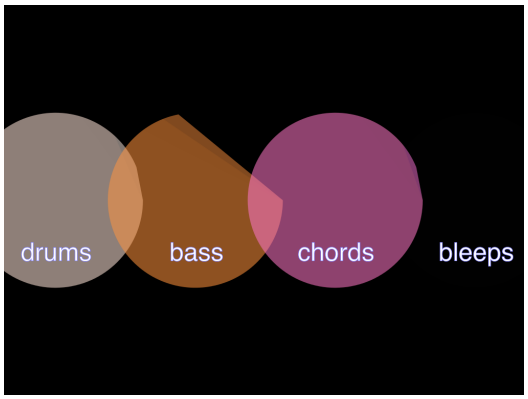
(a) Phase 1



(b) Phase 2



(c) Phase 3



(d) Phase 4

Figure 4.2: The four phases of the didactic visualisation.

"time until next execution" known as the "callback rate" was a primary feature of the programming environment and an essential part of musical live coding. Bright colours and solid shapes were used to ensure constant visibility and to communicate the intention of the underlying code (see Guidelines 4.1 and 4.4).

The stages of this visualisation attempted to address the issues indicated during the exploratory field study. These included the lack of an overview of the source code and program behaviour, the difficulty in linking changes to the source code and the flicking between source code screens. The didactic visualisations proceeded through four stages, with progression made depending on the number of active functions (instruments).

Phase one (see Figure 4.2a) displayed the name of the one active function in the centre of the screen (see Guideline 4.10). Two other major visual elements were present: a short circular line segment rotating and a pulsing circle. Both visual elements were mapped to the callback rate of the active function (see Guideline 4.9). Phase two (see Figure 4.2b) displayed the name of two active functions. Two ring-shaped visual elements represented the two functions. Each ring-shape was divided into ten segments with each segment coloured according to the callback progress of the active functions. Phase three (see Figure 4.2c) displayed three active functions as pulsing circles according to the progress of the active function callbacks. Finally, phase four (see Figure 4.2d) displayed four active functions as circles with a colour fill according to the progress of the active function callbacks.

4.2.2 Aesthetic Visualisation

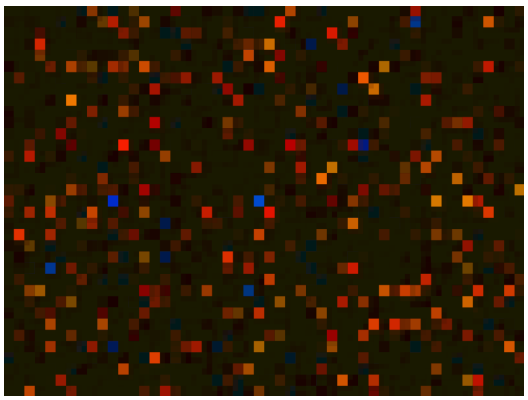
The aesthetic visualisation technique was designed to react to the programmer's activity in a more abstract way, to maximise aesthetic appeal [Cawthon and Moere 2007] and to engage the audience's interest. Although still based on the source code and the live coder's edits, the generation of shapes was driven by instrument volume and synchronised with the musical beat. These were secondary features of the programming language and features resulting from the output of the running program as opposed to the callback rate, a primary feature, used in the didactic visualisations.

The emphasis for the aesthetic visualisation was on the artistic appeal of the visuals (see Figure 4.3), including more variety in visual structure and colour. As in the didactic condition, the aesthetic visualisations proceeded through four stages, based on the number of active functions (instruments), but these visuals had no textual labels and they moved and interacted with each other over the entire projected scene.

As with the didactic visualisations, the aesthetic visualisations proceeded through four stages, again with progression made depending on the number of active functions (instruments).

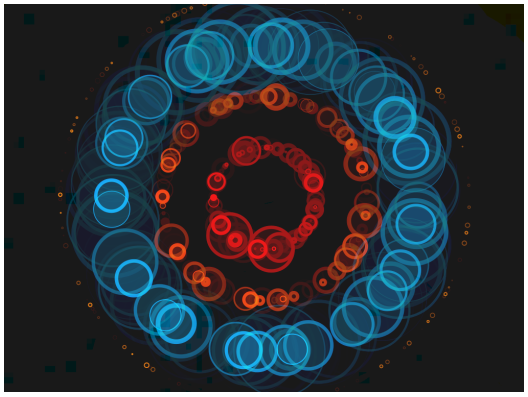
Phase one (see Figure 4.3a) displayed pulsing bars across the screen and indicated the active state of the instrument and the underlying active functions. The pulsing bars were matched to the tempo of the music. Phase two (see Figure 4.3b) displayed random small coloured squares across the screen, designed to match the musical aesthetic. Phase three (see Figure 4.3c) displayed groups of arcs, again designed to match the

(a) Phase 1



(b) Phase 2

(c) Phase 3



(d) Phase 4

Figure 4.3: The four phases of the aesthetic visualisation.

musical aesthetic. Finally, phase four (see Figure 4.3d) displayed four sets of rotating circles, with the size of each set based on the musical output (volume) of an associated function (instrument).

4.3 Development

Development of the visualisations took place in collaboration with a live coder. This collaboration ensured that the visualisations integrated with the live coding performance and that the performance aspects of live coding such as the workflow and the musical progression were not interrupted. Similarly, this collaboration allowed both the visualisations to be influenced by the musical structures common in live coding and the musical structures to be influenced by the expressiveness and limitations of the visualisations.

Development was conducted within the standard live coding environment, in Extempore. The prototype consisted of a set of functions to be used during the performance as a replacement for some existing structures. This consisted of a callback hook labelled *cv-callback* replacing Extempore's standard *callback* function. In Extempore, the *callback* function allows for repeatedly calling functions with a time based interval. This structure is fundamental to the musical aspects of live coding, allowing for the sequencing of musical structures and the patterns common in live coding music.

The *cv-callback* function allowed information regarding the dynamic aspects of the running program to be collected for display within the visualisations. Examples of these dynamic aspects are active function names, callback time, beat and duration information specific to each instrument.

Additionally, a volume unit (VU) meter function was adapted to the prototype to allow for collecting volume information specific to each active instrument. This would allow for the output of the musical structures to be collected and visualised.

Information from both the *cv-callback* and the VU meter were integrated into the visualisation technique. Specifically, the didactic visualisations made use of the *cv-callback* as more precise information about the active functions. The aesthetic visualisations, on the other hand, made use of the VU meter as more abstract high level information about the running functions and their output.

To develop the visualisations, Cairo [Worth and Esfahbod 2012] was used through Extempore bindings, as the supporting graphics library for visualisation implementation. Visualisations were written in pure xtlang, as part of the Extempore live coding environment. The final prototype consisted of a total of 1000 source lines of code (SLOC).

4.4 Summary

This chapter has described two sets of visualisations developed in collaboration with a live coding artist in an attempt to visualise the live coding process. The literature (see Chapter 2) had identified the need to evaluate the effect of visualisations on

audience experience within live coding in terms of the dimensions of enjoyment and understanding. The two sets of visualisations described here each target one of these dimensions with the aesthetic visualisations targeting enjoyment and the didactic visualisations targeting understanding.

However, evaluation of the two resulting sets of visualisations was required. Although based on the literature and the results of the field study, assumptions made over the course of the development process required careful evaluation within the application space of live coding to confirm or invalidate these visualisations. The following chapter (see Chapter 5) discusses the evaluation of these visualisations within the space of live coding, further informing future iterations of the visualisation technique and development methodology.

Chapter 5

User Study

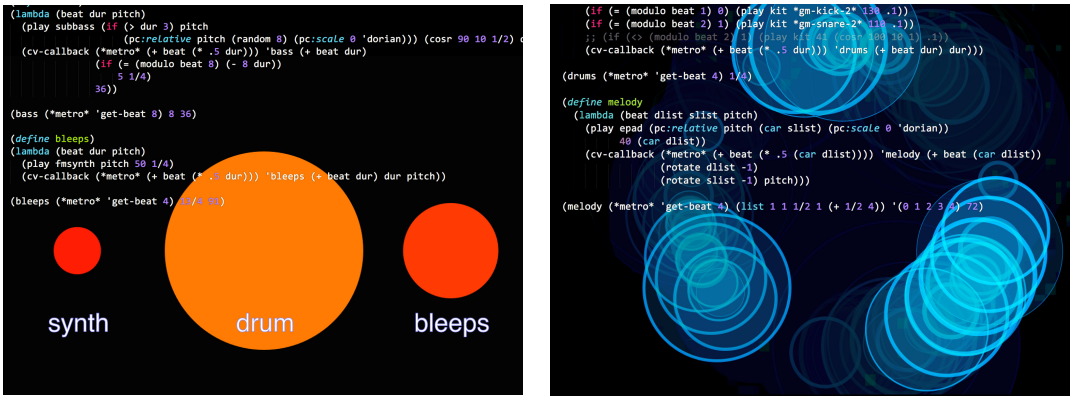
A laboratory study was conducted to test the impact of additional visual feedback (beyond the raw source code) on audience understanding and enjoyment in live coding. It was hypothesised that the didactic visualisation (see Section 4.2.1 and Figure 5.1a) approach would result in enhanced audience understanding, and a reduction in audience confusion through the performance. In contrast, it was predicted that the aesthetic visualisations (see Section 4.2.2 and Figure 5.1b) would positively influence audience enjoyment, both overall and over the course of the performance.

5.1 Method

Two independent audiences ($N = 19 + 22 = 41$) recruited through an on-campus advertisement (see Appendix C) each watched a live coder perform two ten-minute "sets": one accompanied by the didactic visuals, and one with the aesthetic. The order of presentation of the two visual conditions was swapped between the groups. The improvisational nature of a live coding performance makes "controlled" experiments difficult, but the live coding artist attempted (as much as possible) to do the same two performances for each group.

Over the course of these performances, each audience member completed a survey (see Appendix D) consisting of four sections: demographic information, their opinion of the first piece, their opinion of the second piece and questions about the performance overall. Similar to the first field trial, the survey primarily focussed on self-reported levels of "enjoyment" and "understanding" related to the visualisations specifically and also to the performance more generally. There was also a free-form question for suggested improvements to the visualisations.

After the laboratory study, to further inform the outcomes of the experiment, a video-cued recall [Suchman and Trigg 1992] interview was conducted using a video of the performance. Two of the performances were examined critically to assess the advantages and limitations of the visualisations.



(a) Didactic visualisation

(b) Aesthetic visualisation

Figure 5.1: Visualisations as they appeared during the study. Source code is overlayed on the visualisations.

5.2 Participants

Of the 41 participants who took part in the study over the two performances, 19 participants observed the first performance and 22 participants observed the second performance. The demographic makeup of the audiences was similar.

66% of the participants were male and most participants were aged between 18 and 32 (76%). As the study was conducted within the Computer Science Department, a large proportion of the participants were experienced with programming with 90% having current or previous experience with it. Nevertheless, only 15% of participants had previous experience with any of the Lisp style of languages, the style used within the performance.

Of the participants, 68% stated that they listened to a large amount of music though only about 15% of participants stated that they played an instrument or sang regularly. 78% of the participants had never seen a live coding performance before.

5.3 Results

The audience-reported enjoyment and understanding responses from the survey were evaluated for the two visualisation conditions as described below. A significance level of 0.05 was used for the Chi-squared analysis.

5.3.1 Enjoyment

Overall, the majority of the participants reported that both visualisation conditions had a positive effect on their **enjoyment** of the performance: 76% stated that the aesthetic visualisations improved their enjoyment and 56% stated that the didactic visualisations improved their enjoyment. No significant difference between the visualisation types regarding enjoyment was found ($\chi^2 = 3.7733, df = 2, p = 0.1516$).

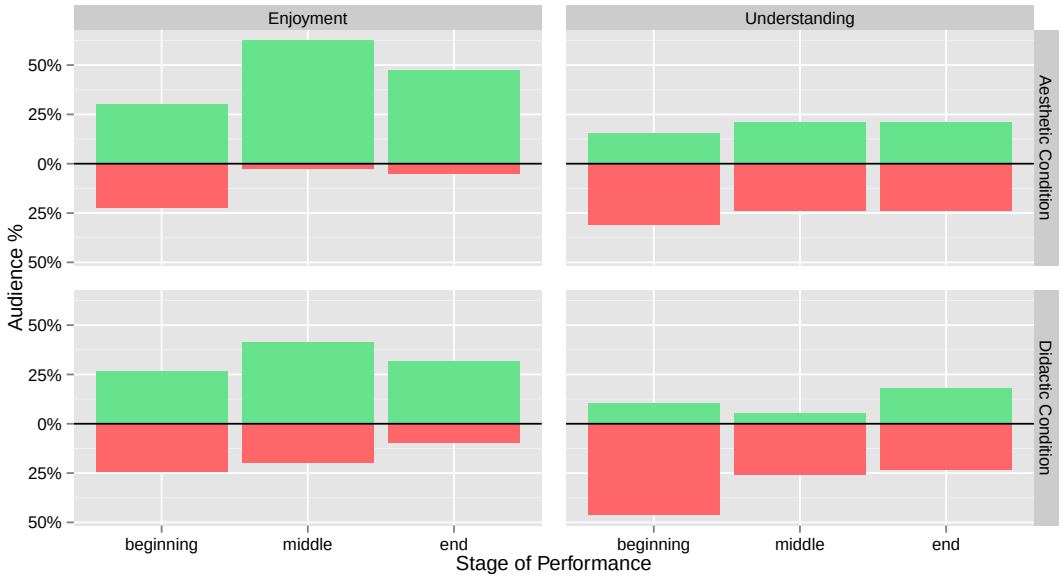


Figure 5.2: User study survey condition and dimension results. Percentage of the audience reporting "high" (green - above the line) and "low" (red - below the line) enjoyment and understanding over the beginning, middle and end stages of the performances for the aesthetic and didactic conditions. The remaining population, not shown here, reported "medium" levels of enjoyment or understanding.

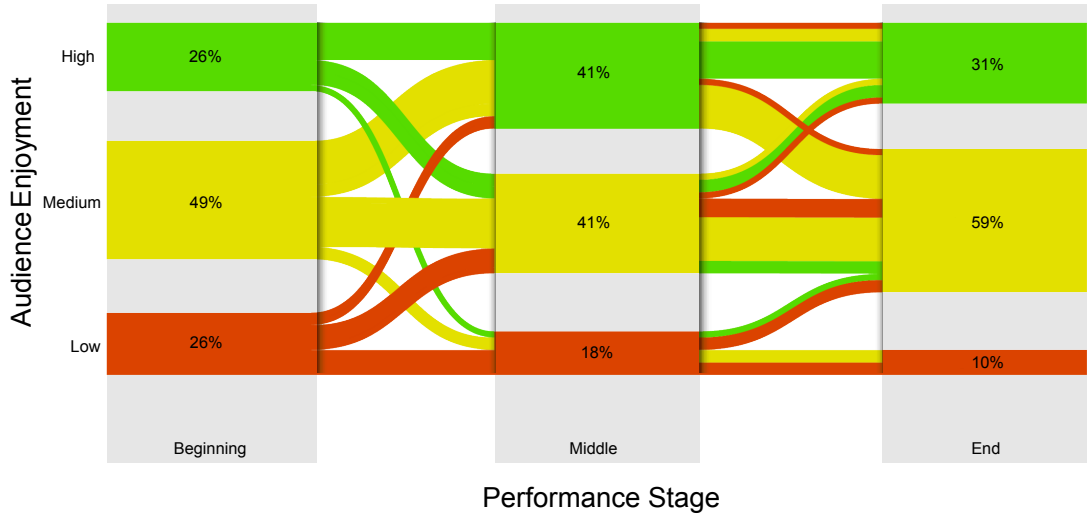
Participants were asked to rate their enjoyment during the (self-determined) "beginning", "middle" and "end" of the performances (see Figure 5.3). The aesthetic visualisation resulted in a larger percentage (around 60%) of the audience reporting high enjoyment during the middle of the performance compared to the didactic visualisation (around 40%). Notably, only a very small percentage reported low enjoyment for the aesthetic visualisations during the middle and end of the performance.

During the didactic performance, 15% of the audience stated that their enjoyment *increased* from the beginning of the performance and was steady thereafter. By contrast, 24% of the audience reported this pattern of enjoyment during the aesthetic performance. Approximately 30% of the audience of all (aesthetic and didactic) performances stated that their enjoyment remained steady throughout.

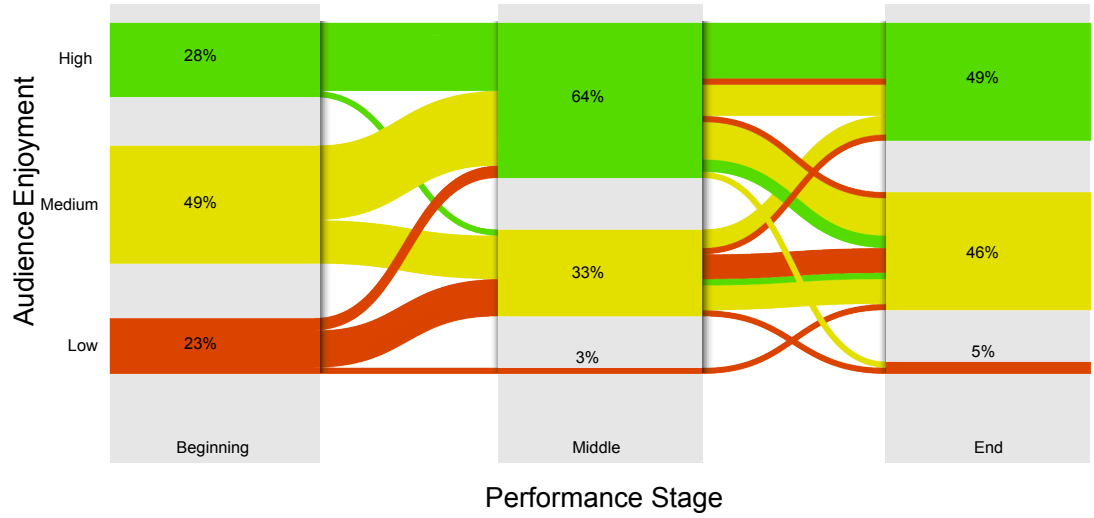
5.3.2 Understanding

In response to a specific survey question, 37% of participants stated that overall, the didactic visualisations "helped them to **understand** the code", compared to 12% of participants for the aesthetic visualisations. This was a significant difference between the visualisation conditions ($\chi^2 = 7.1986, df = 2, p = 0.02734$).

Again, participants were asked to rate their understanding during the (self-reported) "beginning", "middle" and "end" of the performance (see Figure 5.4). The aesthetic visualisation resulted in a smaller percentage of the audience (around 30%) reporting

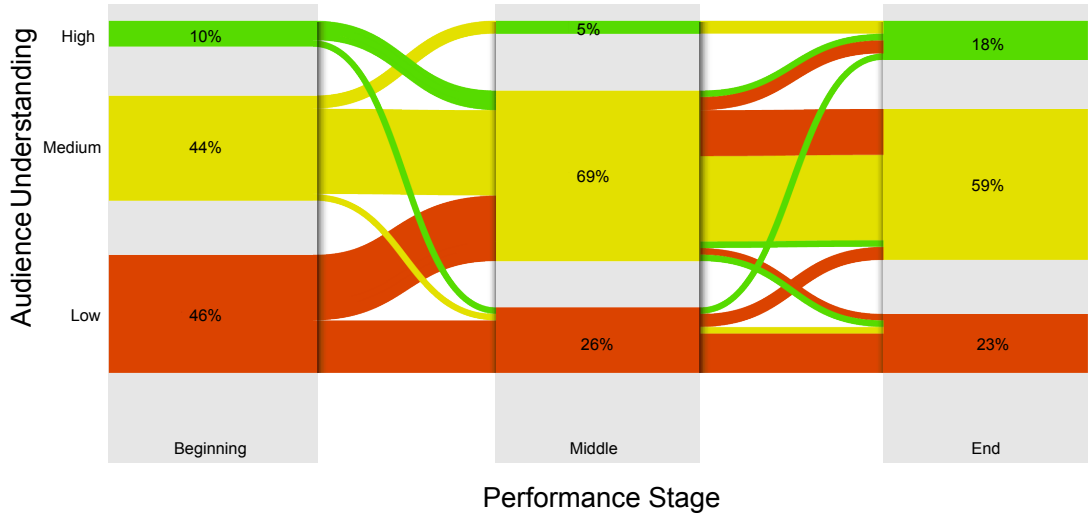


(a) Audience reported enjoyment level for the **didactic** condition.

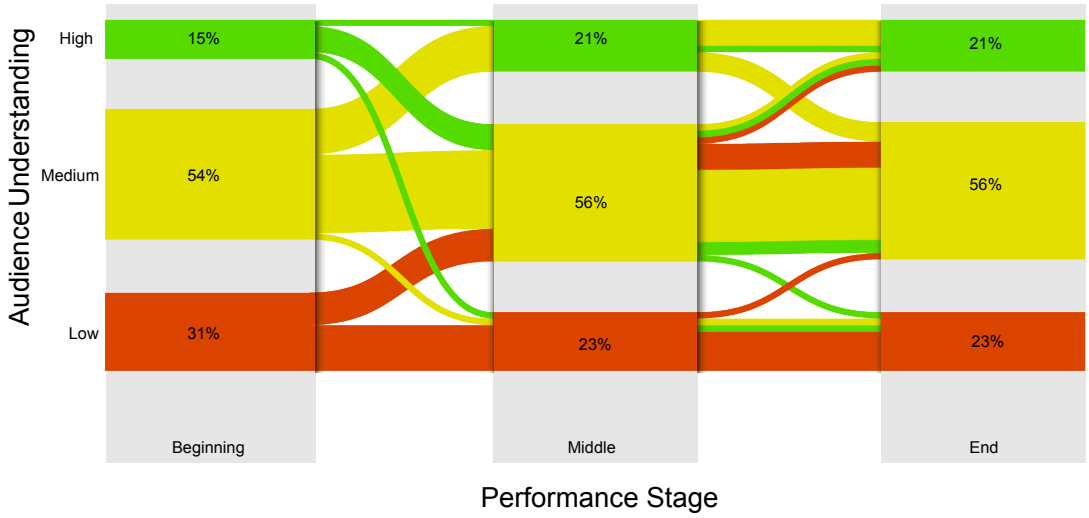


(b) Audience-reported enjoyment level for the **aesthetic** condition.

Figure 5.3: Audience reported enjoyment during the beginning, middle and end of the performance for the didactic and aesthetic visualisation conditions. Line width at each stage indicates proportion of the audience reporting high, medium or low enjoyment, and line colour connecting each section of the performance is determined by the enjoyment level at the *beginning* of the performance.



(a) Audience reported understanding level for the **didactic** condition.



(b) Audience-reported understanding level for the **aesthetic** condition.

Figure 5.4: Audience reported understanding during the beginning, middle and end of the performance for the didactic and aesthetic visualisation conditions. Line width at each stage indicates proportion of the audience reporting high, medium or low understanding, and line colour connecting each section of the performance is determined by the understanding level at the *beginning* of the performance.

low understanding at the beginning of the performance compared to the didactic visualisations (around 45%). However, by the end of the performances both distributions looked very similar indeed.

During the didactic and aesthetic performances, 49% and 44% respectively of the participants stated that their understanding *remained the same* throughout the performances. During the didactic performance, 10% of the audience reported a level of understanding that *trended downwards* (eg. high to low) compared to 20% of the audience during the aesthetic performance. However, this reported advantage of the didactic visualisations was offset by the reported audience understanding at the beginning of the performance where 44% indicated a low understanding with the didactic visualisations compared to only 30% with the aesthetic visualisations.

Overall, the survey results for audience understanding are complex, and reported levels of understanding fluctuated during the performances. Dramatically, Figure 5.4a shows that a very small proportion of the audience reported high understanding during the middle of the performances. One interpretation of this result might be that it took audience members some time to work out what the didactic visualisations were actually showing, and that this conflicted with the first impressions of what some audience members assumed (hence the decrease in levels of understanding from beginning to middle). However, once they finally understood the graphics some audience members were then able to better understand the live-coding performance.

5.3.3 Liveness

Participants were asked to discuss how each visualisation influenced or impacted the liveness of the performance. Concepts identified included positive and negative valence towards the didactic visualisations, positive and negative valence towards the aesthetic visualisations and the relevance of visual source code. The discussion included the positive and negative aspects of the didactic visualisations, the positive and negative aspects of the aesthetic visualisations, source code discussion, and statements indicating an understanding between the visuals and the source code.

Overall, 54% of the audience indicated negative valence towards the didactic visualisation regarding liveness, citing a variety of reasons including that the "visuals only responded to what was typed", that the "musical forms didn't occur at the most expected times" and that the visualisations "perhaps made the performance seem too polished".

On the other hand, 34% of the audience indicated positive valence towards the didactic visualisations suggesting that "it was easier to follow this visualisation than the code", "the visualisations clearly showed the changes being made to the code" and that these visualisations "helped more with communicating that the performance was live".

40% of the audience had negative valence towards the aesthetic visualisation's effect on the sense of liveness. Reasons cited include that the "influence is not clear" between the code and the visuals, that "the visualisation did not make much sense" and that the "visualisations did nothing to suggest that the performance was live".

32% had positive valence towards the aesthetic visualisation. Responses included that these visuals were "less dominating and more complementary" and that "the visualisations helped to show when a piece of code started working".

A relatively large proportion (28%) of the audience discussed the importance of the source code to the sense of liveness of the performance such as that the "code showed what the musician was doing physically". A further 5% of the audience demonstrated a deeper understanding of the live coding process such as "changing values produced changes in tone and speed of the music pitch".

5.3.4 Improvements

The audience was asked to indicate possible improvements to the visualisations. 40% of the audience suggested improvements that indicated a desire to see a better relationship between the visualisations and the music, with suggestions such as matching the visualisations to musical rhythms or pitch. For example, one audience member stated that the visualisations should "perhaps have a stronger correlation with hush tones and defined shapes, baselines with wide and soft shapes with animations that follow the beat more consistently".

Other members of the audience stated that they would like to see a better relationship between the code and the visuals. For example, one audience member stated that "it would be really cool once you edit a code line and then activate it for it to morph into a visualisation for as long as this line of code is active". Another audience member stated that "it would be better to not just relate the function to the sound but relate every beat to the sound or to the code responsible for that beat".

Other suggestions included increasing the readability of source code, increased immediacy of actions, increasing uniformity between the visualisations, and the use of images to assist with the understanding of high level concepts.

5.3.5 Follow-Up Interview

A follow-up interview was conducted with the live coder to further examine the visualisation approach taken and the overall usability of the visualisations. See Appendix E for the full interview transcript.

5.4 Discussion

The overall enjoyment of the visualisations was high, for both the aesthetic and didactic visualisations. Reported enjoyment of the aesthetic visualisations was higher than for the didactic visualisations but the trends across Figures 5.4b and 5.4a for understanding are complex.

As discussed above, the small number of high responses for understanding during the middle of the didactic performances, and the decreasing trend from high to middle level understanding from beginning to middle of the performances perhaps indicates a higher cognitive load for understanding the didactic visualisations themselves. In fact,

features of the didactic visualisation were reported to confuse some members of the audience, despite their stated aim of *assisting* audience understanding. One audience member even stated that they "found them distracting" and that they "preferred just to read the code".

The video-cued-recall interview indicated that the live coder's experience of the visualisations was fundamentally different to that of the audience. While many members of the audience reported that they drifted between focussing on the music, focussing on the visualisations and focussing on the code, the live coder reported that his focus was purely on the code and the music, rarely drifting. In one particular section of the interview, the live coder stated: "I definitely wasn't paying attention to them [the visualisations] on the day. In fact I tune them out as best I can because I am just trying to focus on the code". By contrast, one audience member stated that "you could see the code being written and the visualisations helped to show when a piece of code started working". Another audience member stated that "the visualisations were interesting but distracting". When asked if the visualisations were distracting, the live coder stated: "Ah, no. In general I'm just so focussed on the code".

5.4.1 Limitations

Several limitations of the investigation method were identified.

It should be noted that there are inherent difficulties in evaluating a live performance. Notably, physical factors such as constraints on the number of projectors, time during which to perform and live coder fatigue could have impacted the results. Similarly, factors such as musical preference, musical style and variations in the musical performance could have contributed to audience reception of the performance.

Due to the time and fatigue constraints of the live performance, no baseline for the study was set. Evaluation of the visualisations in comparison to a non-visualised live performance would be required to determine if the visualised live performance provided an additional benefit to understanding or enjoyment. In this case however, the comparison of the two live coding techniques provided a useful and valid basis on which to develop a second visualisation prototype for further evaluation.

The impact of testing effects on the validity of the study due to the order the two conditions were presented to the audience were mitigated through order balancing. However, differences between the live coding performances are inherent. The live coder attempted, to the best of his ability, to keep the performances consistent.

5.5 Summary

The differences between the experience of the live coder and the experience of the audience were identified during the follow-up interview with the live coder. The live coder had a mental image of the active code and the structure of the program that was not being communicated through the source code. Visualisations assisted the audience in some regards as to the state of the program, however, it was hypothesised that more

effective visualisation techniques might take advantage of the mental model of the live coder and attempt to align the audience with this mental model.

Results of this study indicate that there is a need for further refinement and evaluation of visualisations with specific focus on elucidating the state of the active program through the alignment of mental models between the live coder and the audience.

Chapter 6

Visualisation Refinement

Following the first user study (see Chapter 5), limitations of both the visualisations and the evaluation method were identified and corrective steps were taken. The steps taken to correct these limitations and the resulting refined visualisation are outlined in this chapter.

6.1 Rationale

The initial user study (see Chapter 5) identified some improvement in enjoyment through the use of aesthetic visualisations. However, the study had conflicting results regarding understanding, with audiences reporting high levels of understanding attributed to the didactic visualisations when asked directly, but with lower overall understanding during the beginning stages of the live coding performance (see Section 5.3.2). A new visualisation prototype was developed to bring together features identified as positively impacting enjoyment and understanding in one visualisation. This visualisation prototype built on a combination of the aesthetic and didactic visualisations evaluated in the previous study. The intention of this visualisation was to smooth the early transition into the more educational aspects of the visualisations through the aesthetic features identified.

Some technical limitations identified in the previous iteration of the visualisation prototype included incorrect timing and beat, no direct feedback from the programmer

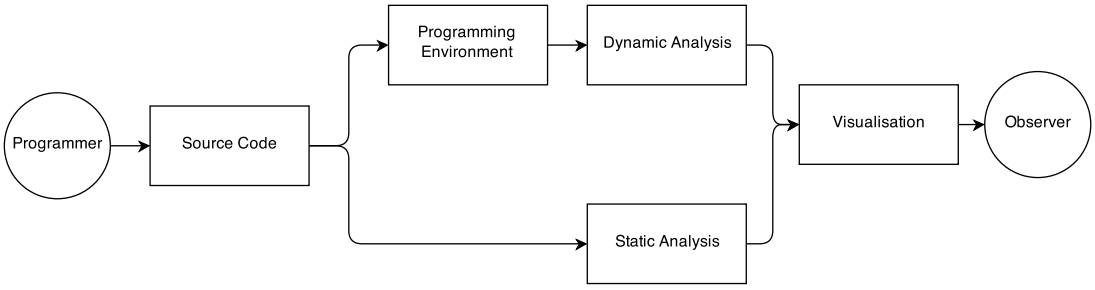


Figure 6.1: Knowledge flow from programmer to observer as directed by the visualisation technique employed.

Guideline	Reference	Identifier
Show relationships between entities using lines	[Ware 2013, p. 183]	(Guideline 6.1)
"Consider using pictorial icons for pedagogical purposes in infographics."	[Ware 2013, p. 320]	(Guideline 6.2)
"...relate every beat... to the code responsible for that beat"	(see Section 5.3.4)	(Guideline 6.3)
Respond to the programmer's typing	(see Section 5.3.3)	(Guideline 6.4)
Reduce visualisation distraction	(see Section 5.4)	(Guideline 6.5)

Table 6.1: Additional guidelines for refinement decisions.

and limited links between the visuals and the source code. The goal of this iteration of the visualisation prototype was to rectify these issues while aligning the mental model of the audience with that of the programmer.

6.2 Design

As with the previous visualisation iteration, guidelines were identified, adapted and applied to the design process. Some of these design guidelines are outlined in Table 6.1 and are referred to in subsequent sections of the thesis.

Three sources of data were to be combined and visualised: the state of the source code; the state of the running program; and the manipulation between the two by the live coder. To achieve this, three major components were required: an application manager; a code manager; and an editor plugin (see Figure 6.3). The following sections detail the implementation of these components.

The refined model of knowledge flow from programmer to observer is shown in Figure 6.1. As with the initial model of knowledge flow (see Figure 4.1), dynamic analysis of the programming environment occurs. In this case however, the dynamic analysis is combined with static analysis of the source code before mapping to a visualisation.

6.2.1 Visualisation Manager

The visualisation manager handled all visual elements on the screen. The Polycode [Safrin 2013] library was bound to the live coding environment allowing for

OSC Address	Arguments	Description
/interface/code	source_code	Sent when the source code within the text editor changes.
/interface/evaluate	evaluated_code	Sent when a block of source code within the text editor is evaluated (activated).
/interface/error	error_text	Sent when an error occurs within the text editor or the evaluation of source code fails.
/interface/focus	file_name	Sent when the file currently being edited changes.
/interface/cursor	cursor_position screen_minimum_position screen_maximum_position cursor_position_x cursor_position_y	Sent when the text editor cursor selection changes.

Table 6.2: The OSC protocol developed for standardised interaction between text editors and the visualisation.

more advanced two-dimensional graphical manipulations than the previous visualisation iteration through scene graph manipulation. Using this technique, the intention was to represent the state of the source code, the manipulation of the source code and the state of the active source code in the same visual environment.

6.2.2 Interface Manager

To gather information regarding the actions of the programmer, a method of interfacing with the live coder's programming environment (text editor) was required. This was achieved using a text editor plugin sending programmer actions over an Open Sound Control (OSC) protocol. The interactions sent included cursor movement, all source code changes, file focus and source code evaluation. This protocol was implemented as a standard way of communicating between the text editor in use and the visualisations. Table 6.2 describes the protocol.

Two text editor plugins implementing the protocol were developed including a plugin for EMACS [Stallman 1981] and a plugin for Sublime Text [Skinner 2013].

6.2.3 Code Manager

Both static analysis of source code and dynamic analysis (see [Eisenbarth et al. 2003] and [Jerding and Rugaber 1997]) of the program were combined into this iteration of the visualisation prototype in order to provide the audience with a link between the

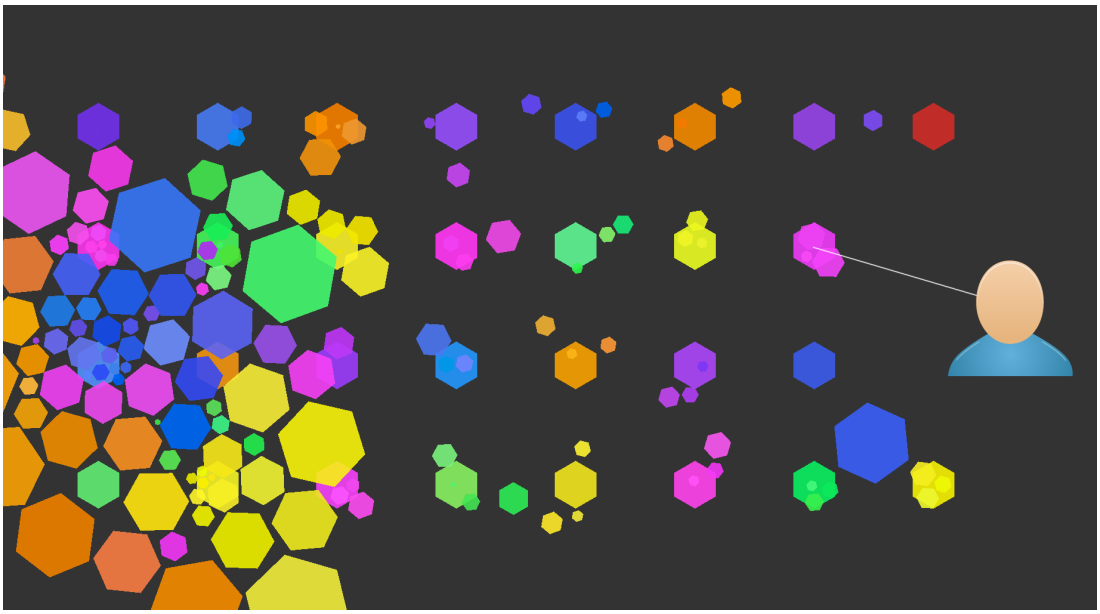


Figure 6.2: An example of the visualisation developed following the refinement process.

state of the world (SoW) (the current active program) and the state of the code (SoC) (the edits made to the source code not yet activated) [Swift et al. 2013].

Static analysis of the source code involved parsing the source code written by the programmer and analysing any changes made, mapping these changes to historic source code states. The source code written by the programmer was sent from an editor plugin (see 6.2.2) to the Extempore visualisation program and stored as the current SoC. Mappings to previous states were then made for each function within the source code.

Dynamic analysis of the running program involved providing mechanisms for the programmer to send running state information to be stored as the SoW. A callback hook was provided to be used when creating an instrument during performance. This hook would provide data on the names of the instruments and the state of the callbacks, and would allow links to be determined between the SoC and the SoW.

6.2.4 Mappings

A number of specific mappings were assigned to the visualisation. These visual mappings related directly to actions taken by the programmer, behaviour of the running program and representation of the source code.

Function count was mapped to the number of large visual elements on screen. When adding a new function, a new element was introduced. This mapping was intended to allow the audience to associate the simplified geometric shape to the source code and allow the audience to separate functions visually.

Each large visual element represented the state of active and inactive functions. In live coding, the programmer triggers functions to change between active and inactive

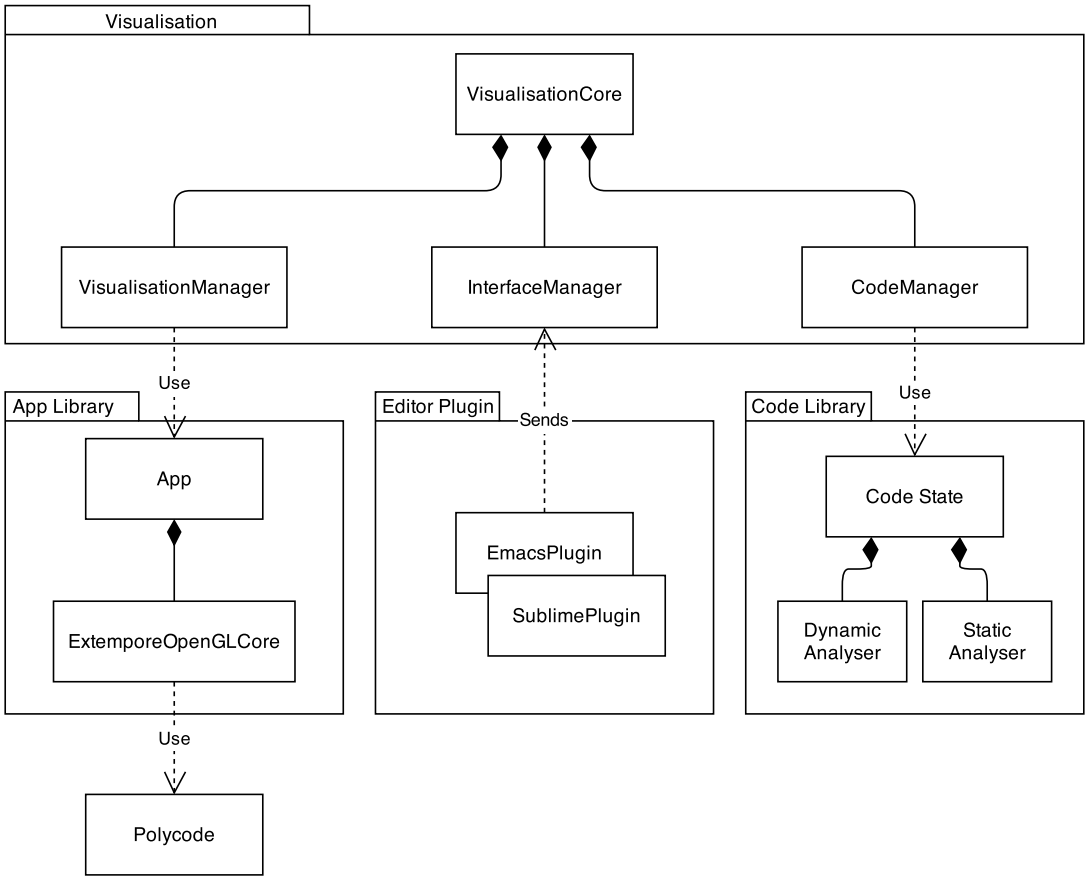


Figure 6.3: The refined visualisation class diagram. The three major components included the visualisation manager, the interface manager and the code manager. The visualisation manager handled the mapping from the state of the live coding program to the visual elements. The interface manager received details of the state of the text editor from a text editor plugin. Finally, the code manager collected data from both the running live coding program and the static source code retrieved from the text editor to combine into the current state of the live coding program.

states. The mapping represented this through the use of washed out grey and white shades while inactive, and vibrant, bright colours while active. In addition, the active functions were animated, with moving visual elements.

Function size and shape were represented visually through the use of "attractors" with each line of code mapped to one of the visual elements. Visual elements would grow and shrink according to the length of the line of source code directly mapping the typing actions of the programmer to visual elements.

To ensure the relationship between the actions of the programmer, the changes to the source code and the changes to the music were easily identified, an icon representing the programmer was displayed closest to the function being edited. This mapping intended to provide the audience with a better overview of the movements of the programmer among the functions than would otherwise be visible through the cursor alone.

6.3 Development

A similar development approach to the previous iteration of the visualisation prototype was taken. This process involved collaborating with a live coding artist to further develop visualisations appropriate for the live coding context.

A strategy for combining static source code and the dynamic program structures was developed. This required storing the state of each function within the program comprised of the function change history, active state, current state and editing meta-data. The function change history was extracted from source code sent from the text editor to the interface manager to be analysed.

Analysis of the source code involved parsing the source code and extracting functions and function names. It was determined early within the development process that full parsing of the syntax would not be necessary for informative visualisations. Instead, a heuristic line-by-line approach was implemented, separating functions in the source based on a set of simple rules. This approach allowed for managing syntax errors and incomplete syntax common while *writing* a program.

Development took an iterative approach, testing components thoroughly. Testing strategies employed included unit testing, integration testing and system testing. Low level unit tests were written for the three components of the visualisation system: the interface manager, the code manager and the visualisation manager. Integration testing was considered essential due to the large numbers of interacting components and was conducted throughout the development process. Finally, system testing (and to some extent, user testing) was conducted with the live coder to ensure correct functionality within the system as a whole.

6.4 Summary

In summary, this visualisation technique attempted to bring together effective features identified within the aesthetic and didactic visualisations of the previous iteration (see

Chapter 4). The prototype developed included three major components, consisting of the visualisation manager, the interface manager and the code manager. Technical limitations with the first iteration of the visualisation prototype were addressed and the state of the running program and the manipulation of the source code were mapped to visual elements. The effectiveness of aligning the mental models of the audience and the live coder was to be evaluated in a follow-up user study (see Chapter 7).

Chapter 7

Follow-Up User Study

A follow-up user study was conducted to evaluate the set of refined visualisations and to address some of the limitations identified in the first user study. This included providing a valid baseline for the study (the *no visualisation* condition) and reducing the visualisations to more meaningful representations.

It was hypothesised that the visualisation prototype would result in higher understanding at the end of the performance and that enjoyment would remain steady throughout both performances.

7.1 Method

Two independent audiences ($N = 14 + 11 = 25$) were recruited through on campus advertisement (see Appendix F). Each group was exposed to two live coding musical performances. One of the performances displayed only the source code of the performance while the other displayed the source code with the refined visualisation prototype as an underlay (see Chapter 6).

The first group was subjected to the *visualisation* condition, followed by the *no visualisation* condition. The conditions were swapped for the second group, with the audience exposed to the *no visualisation* condition first followed by the *visualisation* condition.

Again, over the course of these performances, each audience member completed a survey consisting of four sections: demographic information, their opinion of the first piece, their opinion of the second piece and questions about the performance overall.

Two additional questions were asked to get an impression of the audience's understanding at specific stages of the performance: "What do you think the performer was doing in the very early stages of the performance?" and "What do you think the performer was doing in the very last stages of the performance?". The results of these questions would then be compared to the reported levels of understanding through the performance to evaluate the accuracy of the measure.

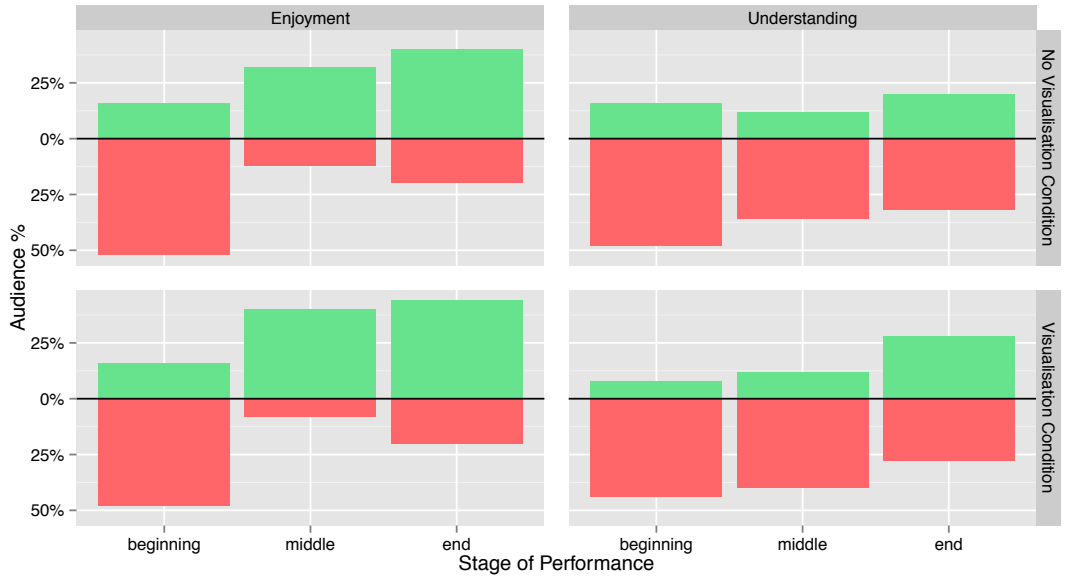


Figure 7.1: Percentage of the audience reporting "high" (green - above the line) and "low" (red - below the line) enjoyment and understanding over the beginning, middle and end stages of the performances for the no visualisation and visualisation conditions. The remaining population, not shown here, reported "medium" levels of enjoyment or understanding.

7.2 Participants

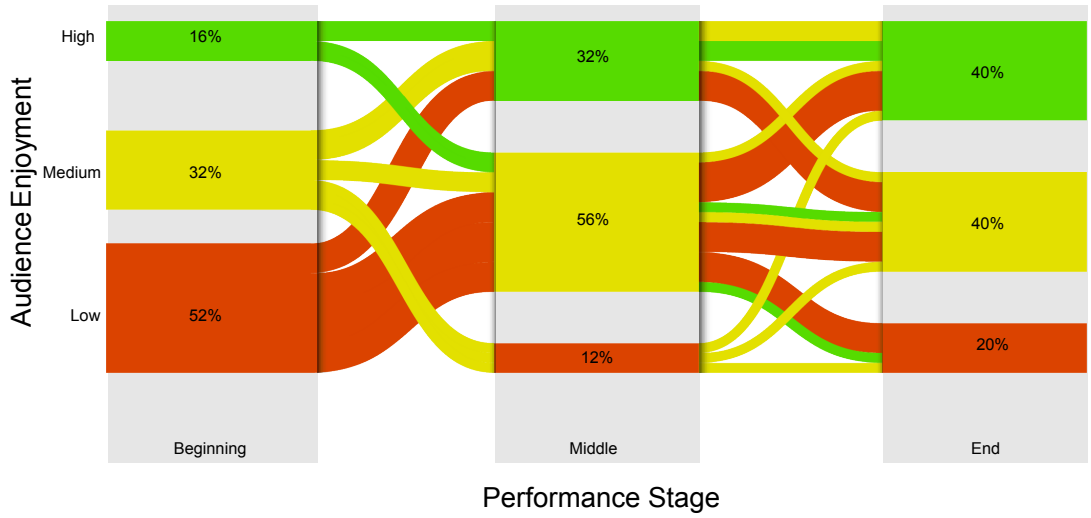
Of the 25 total participants over the two performances 12% were female. 64% of the audience had not been to a live coding performance before and 80% had not attended the previous live coding user study. The background of the audience included 56% that listened to music regularly, 28% that played an instrument and 72% who programmed for their hobby, job or study.

7.3 Results

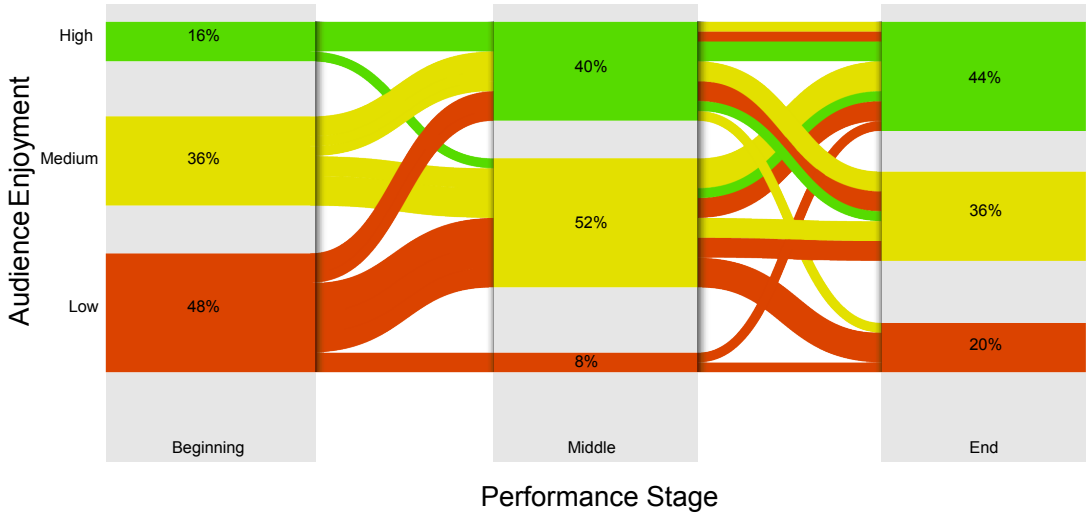
The audience-reported enjoyment and understanding responses from the survey were evaluated for the two conditions as described below. A summary of the results can be seen in Figure 7.1.

7.3.1 Enjoyment

Audiences were asked to state if they thought the source code projection helped their enjoyment of the performance and if they thought the visualisations helped their enjoyment of the performance. 60% of the audience stated that projection of the source code helped their enjoyment of the performance directly while 16% stated that the visualisations helped their enjoyment directly.



(a) Audience reported enjoyment level for the **no visualisation** condition.



(b) Audience-reported enjoyment level for the **visualisation** condition.

Figure 7.2: Audience reported enjoyment during the beginning, middle and end of the performance for the no visualisation and visualisation conditions. Line width at each stage indicates proportion of the audience reporting high, medium or low enjoyment, and line colour connecting each section of the performance is determined by the enjoyment level at the *beginning* of the performance.

Levels of enjoyment throughout the performance were surveyed for the *beginning*, *middle* and the *end* phases of the performance. A comparison of the enjoyment between the no visualisation condition and the visualisation condition is available in Figure 7.2a and Figure 7.2b.

Final levels of enjoyment between the two conditions differed only slightly. With no visualisation, 20% stated that they had low enjoyment, 40% stated that they had medium enjoyment and 40% stated that they had high enjoyment. With visualisations, 20% stated low enjoyment, 36% stated medium enjoyment and 44% stated high enjoyment.

7.3.2 Understanding

Audiences were asked to state if they thought the source code projection helped their understanding of the performance and if they thought the visualisations helped their understanding of the performance. Of the audience, 76% stated that projection of the code helped their understanding of the performance directly while 32% stated that the visualisations helped their understanding directly.

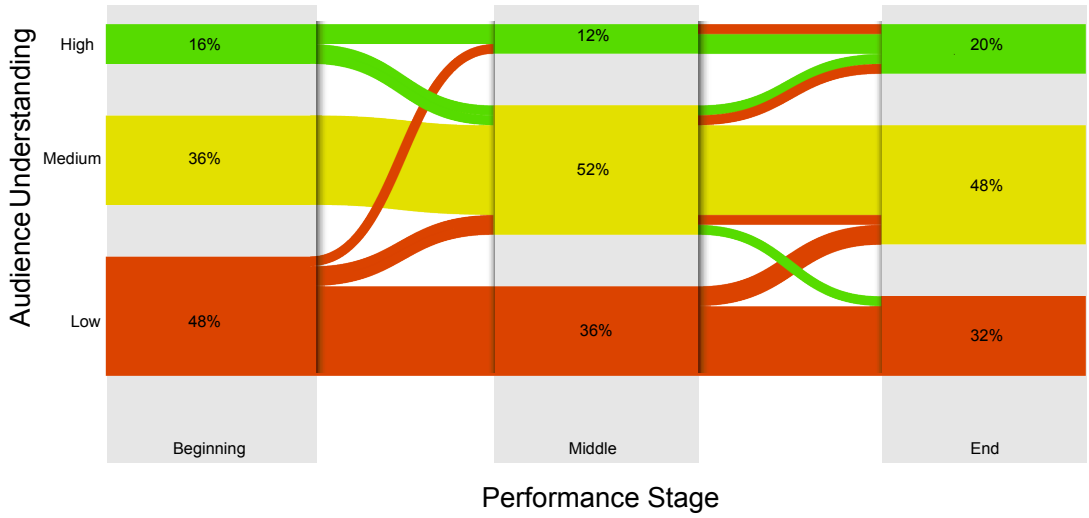
Levels of understanding throughout the performance were surveyed for the *beginning*, *middle* and the *end* phases of the performance. A comparison of the understanding between the no visualisation condition and the visualisation condition is available in Figure 7.3a and Figure 7.3b.

Levels of understanding also differed slightly between the two conditions. With no visualisation, 32% stated that they had low understanding, 48% stated that they had medium understanding and 20% stated that they had high understanding. With visualisations, 28% stated low understanding, 44% stated medium understanding and 28% stated high understanding.

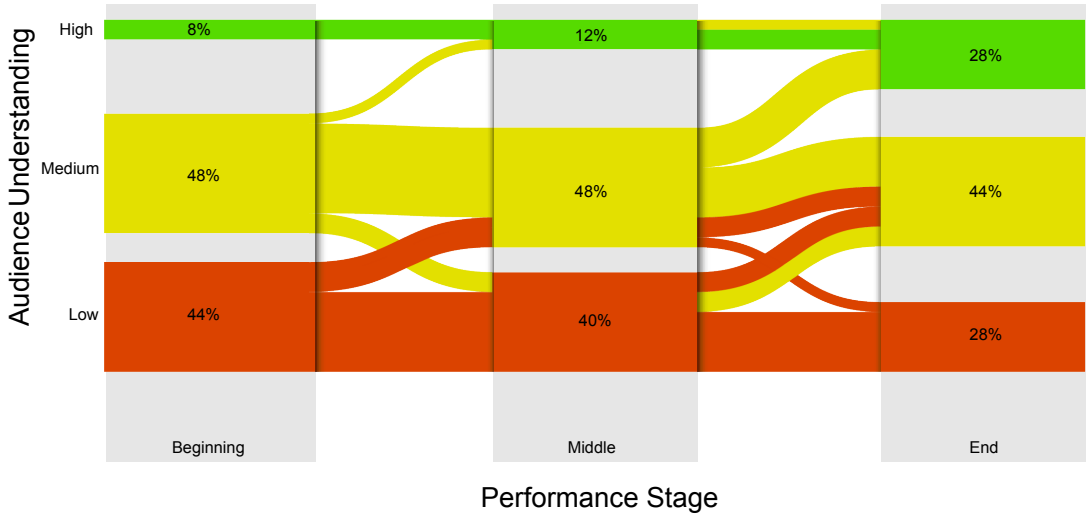
7.3.3 Liveness

After exposing the audience to both conditions, the audience was asked if the projected source code or visualisations helped to communicate the feeling that the performances were live. Of the twenty-five responses, eighteen indicated that the projection of the source code helped, though the audience overall was experienced in programming.

Results of this question reflected the audience opinion of the visualisations. Some responses indicated that: “Yes. Live code editing [helped to communicate the liveness], but visualisations showed what coder was looking at and editing as well as what was running.”. In contrast, members of the audience stated that the “visualisations [were] not really useful, didn’t seem in sync with the music and [had] no obvious meaning”. However, most of the audience (64%) focussed on the source code or musical aspects of the performance in their discussion.



(a) Audience reported understanding level for the **no visualisation** condition.



(b) Audience-reported understanding level for the **visualisation** condition.

Figure 7.3: Audience reported understanding during the beginning, middle and end of the performance for the no visualisation and visualisation conditions. Line width at each stage indicates proportion of the audience reporting high, medium or low understanding, and line colour connecting each section of the performance is determined by the understanding level at the *beginning* of the performance.

7.4 Discussion

Overall, only minor differences were observed between the no visualisation and visualisation conditions. Few differences in enjoyment were seen between the two conditions. Mirroring the previous user study, a pattern of lower understanding was seen with the visualisations during the beginning of the performance. This result supports the idea that visualisations targeting understanding take some time to understand and may only provide benefits in the long-term.

After each performance, two additional questions asked the audience to explain what they thought the live coder was doing at the beginning of the performance and to explain what they thought the live coder was doing at the end of the performance. Results of these questions indicate that many members of the audience had a good understanding of the musical aspects of the performance. Responses stating that the live coder was "adding instruments" and "setting up repetitive sequences to build on during the session" suggested a good overall comprehension of the early stages of the performance. Similarly, when asked about the last stages of the performance, responses stating that the live coder was "changing variables in order to add or remove instruments and alter the pace of the music" indicate a high level of understanding and experience with programming. A number of other responses indicated that understanding was low. For example, one audience member stated: "Sorry, I don't quite understand". Overall, the patterns of understanding reported over the three phases of the performance match the responses asking the audience to explain their understanding.

7.4.1 Comparison to Previous Study

Both studies were conducted under similar conditions and it is reasonable that the results of the studies could be directly compared. Comparing Figure 7.1 to Figure 5.2 reveals some interesting observations regarding the two user studies. It is immediately obvious that a larger proportion of audience members reported low understanding during the follow-up user study than the initial user study. Similar patterns are seen in the proportion of the audience reporting low enjoyment at the beginning of the performance.

The proportion of the audience reporting high levels of understanding is fairly similar across both studies with the exception of the end of the performance for the follow-up user study visualisation condition during which a larger proportion of the audience reported high understanding.

Across the aesthetic, didactic, visualisation, and no visualisation conditions, the aesthetic condition resulted in the highest levels of enjoyment. The visualisation, no visualisation and didactic conditions all performed poorly regarding understanding during the beginning of the performance.

7.4.2 Limitations

A number of factors were identified that could influence the validity of the user study results.

The visualisations suffered from being obscured by the source code displayed on the projection screen. However, compared to the initial user study, the visualisations were logically placed in relation to the source code, from top to bottom, in the order the functions appeared in the code to attempt to counteract this interference.

Nevertheless, one member of the audience stated that "the visualisation made sense of the separate parts [of the code] so you could track each easier but [the visualisations] were difficult to see". The difficulty in seeing the visualisations was discussed by a number of members of the audience, generally stating that the visualisations were projected too faintly behind the source code.

A large proportion of the audience stated that they had coding experience. This may have influenced the preference for seeing the source code during live coding performances. An audience experienced in coding may be less interested in viewing visualisations and have a preference for viewing code directly without anything interfering with the source code.

Finally, as with the first user study, musical and visual preference may have contributed to the results of the survey. Again, an attempt was made to mitigate this through similar musical styles through all performances.

7.5 Summary

This study identifies some small differences between live coding with and without visualisations. A larger proportion of the audience had high enjoyment during the middle of the performance when a visualisation was displayed than without. Similarly, the visualisations resulted in a small increase in the proportion of the audience reporting high understanding. However, this was offset by fewer reporting high understanding at the beginning of the performance.

Although some variation was found, the sample size limited the conclusions that could be made. A comparison with the previous user study suggested that the visualisations examined within this chapter did not quite capture the enjoyment patterns of the aesthetic visualisations. However, an increase in the proportion reporting high understanding for the visualisation condition examined in this chapter suggest that understanding may have been influenced despite the technical challenges encountered.

Chapter 8

Conclusion

This thesis has described studies that have been conducted to determine if the application of visualisation techniques to live coding can enhance audience experience. Results of the initial exploratory field study examined the live coding space without the application of visualisation techniques (see Chapter 3). Potential for increasing audience experience was identified and the dimensions of understanding and enjoyment were identified as effective for measuring changes to the audience's experience of live coding. Furthermore, this study and the related follow-up email interviews identified that audiences would respond to visualisations that identified the high level aspects of the source code and running program.

Results of the user study identified useful features of two sets of visualisations applied to live coding, an aesthetic approach and a didactic approach (see Chapter 5). The results of the survey and the results of the follow-up interview conducted with the live coder identified that the mental models of the audience and the live coder differed greatly and that effective visualisation techniques should identify and reduce the causes of the differences.

Results of the follow-up user study tentatively demonstrated that visualisations targeting *both* audience enjoyment and understanding provided enhancements to the audience experience (see Chapter 7). Analysis and comparison of the user study and the follow-up user study together suggested that visualisations targeting enjoyment may result in the most positive audience response.

8.1 Contributions

Throughout this investigation of the application of software visualisations to live coding, a number of contributions to the field have been identified. This section outlines these contributions.

Principally, a method of software visualisation has been identified and evaluated. Up to now there has been no attempt to evaluate visualisations within the space of live coding. The method of process-driven visualisation identified has included a combination of static and dynamic code analysis and presentation to audiences during live coding performances. This visualisation approach was built on design guidelines identified in the literature. This has allowed for the application of a novel approach to

live coding with a solid theoretical basis.

Through the development of the software visualisation prototypes, a strategy for developing visualisations using a software engineering approach has been identified. This approach has involved collaboration with a live coding artist with the application of design, iteration and validation approach through a variety of field and user studies.

The development of the software visualisation prototypes (see Chapters 4 and 6) and the application to the live coding space have allowed the development of a method of evaluating software visualisations within a live performance space. The more general implications of this method of visualisation evaluation may be useful in the evaluation of software developed for large audiences and audiences observing an individual developing software.

A conference short paper discussing the initial prototype (see Chapter 4), the user study and evaluation methodology (see Chapters 3 and 5) has been accepted to OzCHI. Anonymous references indicated that "the questions raised are worthwhile and interesting", "the paper describes interesting work" and that it is "an interesting paper, and certainly a novel contribution". The acceptance of this paper validates the contributions discussed as relevant to the current state of research within live coding and also the wider field of human-computer interaction.

Finally, visualisation infrastructure for the mapping of software state has been developed. This infrastructure combines two data sources of information: the state of the active program and the state of the source code. This infrastructure has shown the beginning of what is possible for live coding visualisation and has much potential to further develop informative and interesting visualisations.

8.2 Limitations

Associated with these contributions are some limitations within the visualisation design, development and evaluation methodology.

Identified limitations within the evaluation methodology result from a wide variety of sources throughout the field and user studies. The survey focussed heavily on self-reported enjoyment and understanding, and the audience's perception of the stages of the performance including the beginning, middle and end. Although the utility of these measures was identified early within the project, there still appears to be some variance in self-reporting.

All studies conducted were musical in nature and this musical nature presented challenges during survey design. Enjoyment of the performance appeared to centre around the audience's perception of the musical style and their own musical preference. Survey data was collected on audience musical experience. However, musical preference presents an additional challenge in comparing user study participants.

The maturity of the musical nature of live coding also directly impacts visual expressiveness. Music within live coding is far more mature and developed than visualisations and as indicated through these studies, visualisations still have much potential to develop further. It may be that because of the maturity of the musical

aspects of live coding, the music influenced perception far more than the visualisations. This was shown in the audience's written survey responses. Responses indicated that issues with the music were more prominent with larger percentages discussing musical limitations over visual limitations.

The exploration of understanding was based around a language with a functional syntax. Few audience members within the initial user study indicated that they had experience with the Lisp family of programming languages, a family commonly considered functional in nature, and it may be that there are differences between this family of languages and more common imperative or object-oriented styles of programming. This presents limitations in the generalisation of these visualisation techniques to other programming languages.

Finally, as hinted, there may be limitations in the generalisation of the evaluation methodology outside the field of live coding. These studies have explored concepts and challenges inherent to the application of science to the arts. While the techniques of scientifically evaluating the arts have been explored, application of these techniques to more traditional software development needs further investigation.

8.3 Future Work

The process of implementation and evaluation of visualisations conducted through this study are the first steps towards what is possible within live coding. The process of applying visualisations to live code has identified areas of future work and areas that would benefit from future systematic evaluation or development. This section considers these areas.

8.3.1 Evaluation Methodology

Future work will develop the methodology identified. Evaluation of visualisations with the dimensions of enjoyment and understanding, and during the three phases of live coding performances have been considered. Future work will develop and validate these concepts with large performance cohorts.

8.3.2 User Studies

There is still much work to be done in identifying what audiences really find interesting and worthwhile. This study has identified the need for taking an aesthetic approach in developing visualisations, in order to enhance the dimensions of enjoyment and understanding.

Further studies could identify useful aesthetic elements of the software visualisations presented. It may be possible, in future studies, to identify a taxonomy of aesthetic visual elements that enhance the audience experience. Similarly, further studies could identify elements of the didactic visualisations that contributed to decreased understanding and low enjoyment.

Further development of a combination of the aesthetic and didactic approaches, hinted in the follow-up user study, could develop visualisation techniques that consistently improve audience experience across both enjoyment and understanding throughout live coding performances.

Further development of the visualisation techniques and evaluation methodology could allow for more comprehensive user studies. Solving the technical challenges encountered during this set of studies and increasing differentiation between visual features are some obvious first steps in this direction. Similarly, improving upon the process based on the feedback of the audience (see Section 5.3.4) could be seen as some immediate steps to improve audience experience within live coding performances.

The differences between the nature of visualisations and the way in which they are presented should be investigated further. In this study, the distinction was blurred with the way in which the visualisations were presented potentially impacting the results. For example, suggestions were made to use two projectors during the live coding performance, with one projecting the live coder's source code and the other projecting a high level visualisation overview of the programming process and programming state. Similarly, during the follow-up user study (see Chapter 7), the visibility of the visualisations could have been improved. These are both examples of the *presentation* of the visuals impacting the results related to the *nature* of the visualisations. Future work will further distinguish the two concepts, with potential to greatly improve the validity of the results.

8.3.3 Application Spaces

The techniques and methodology described here present a number of opportunities for application within a wide variety of disciplines.

8.3.3.1 Live Coding

Immediately, the visualisations described here could be adapted to a wider range of live coding performance spaces. The visualisation techniques developed could provide an interactive interface to the live coder for manipulation during the performance. This technique could allow the programmer to provide a more expressive view of source code, providing relevant and timely details of code manipulation.

This technique would involve live coded visuals. There is much potential to harness the static and dynamic code structures extracted through these studies *during* the performance. While the visuals developed through this study were static in their expressive power, live coded visuals could provide more visual interest for the audience and allow for visuals to form dynamically with music. Manipulation of visual elements with a goal of enhancing audience experience throughout the performance would provide the basis for live coding as a true multimedia art form.

8.3.3.2 Education

Future work could examine the more pedagogical aspects identified through the application of visualisations to live coding. There is potential to assist audiences with little or no experience in coding to quickly understand high level concepts within the programming process. Applications within this space would apply visualisations as a means to communicate the programming process more effectively and apply visualisations as a means to more effective programming.

8.3.3.3 Software Engineering Practice

The visualisation methodology described here could be applied to software engineering practice. Software visualisations that interact directly with the programmer are yet to be adopted into mainstream development and there is potential to assist multidisciplinary teams by making the programming process more visible.

8.3.3.4 The Arts

The methodology applied through this study takes a step in the evaluation of an artistic process from a scientific perspective. This is an ongoing challenge and this investigation could be applied to a wider variety of performance arts or artistic processes. How to best apply the scientific process to the artistic process is one of the most challenging questions to come from this research, and further scientific understanding of the arts would be of significant benefit.

8.4 Reflection

The challenge of the application of science to the arts has been central throughout this thesis. In many ways, the studies conducted have examined how to approach this problem from both a software engineering perspective and a scientific research perspective. Careful consideration of the artistic process had to be taken into account when designing the software and the methodology. As much as possible, the approach taken attempted to avoid compromising either the artistic or the scientific approaches. Evidence of no compromise within the artistic aspects can be seen in the user studies (see Chapters 5 and 7) during which there was indication that the audience was still enjoying the live coding performance and was still focussing on the artistic musical and visual aspects of the performances. Similarly, in the interview with the live coder, the live coder identified that the visualisations did not interfere with the thought processes involved in live coding, stating that the visualisations were “tuned out”.

8.5 Final Words

This thesis has investigated the proposition that “code visualisations improve the experience of observers”. In this first empirical study of audience perception of code

visualisation in live coding, the studies carried out indicate that code visualisations do improve the experience and that there is potential for the further development of visualisations within this space. The application of visualisations to live coding and the subsequent evaluation has demonstrated that, even within the arts, a scientific analysis sensitive to the artistic goals can be conducted. Nevertheless, there is still significant opportunity to define the direction of visualisations within live coding and for visualisations to provide an enhanced experience of all software.

Appendix A

Field Study Survey

This survey was presented to the audience during the exploratory field study (see Chapter 3). The audience had the option of completing the survey either through a paper-based (as shown in this section) or online medium.

Live Coding Survey

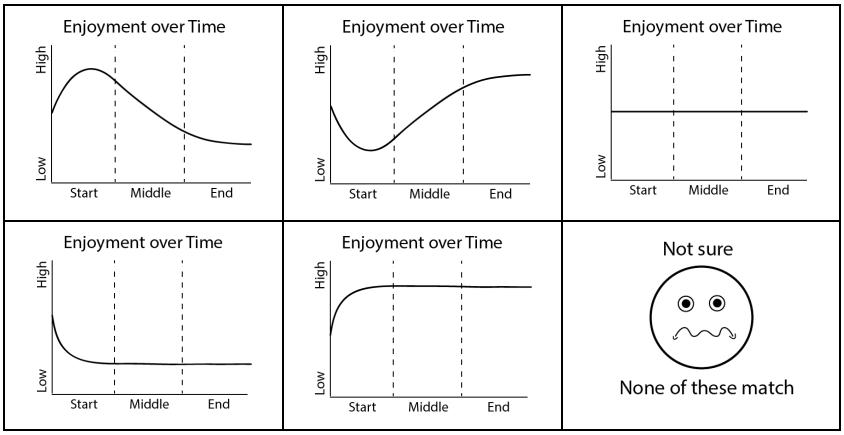
1. How many live coding performances have you been to?
   A. None - This is my first one
   B. Less than five
   C. Five or more

2. How much music do you listen to?
   A. Hardly any
   B. A little
   C. A large amount

3. Do you play an instrument or sing?
   A. No - I would not consider myself a musician or singer
   B. Occasionally or I have in the past
   C. Yes - I play or sing regularly

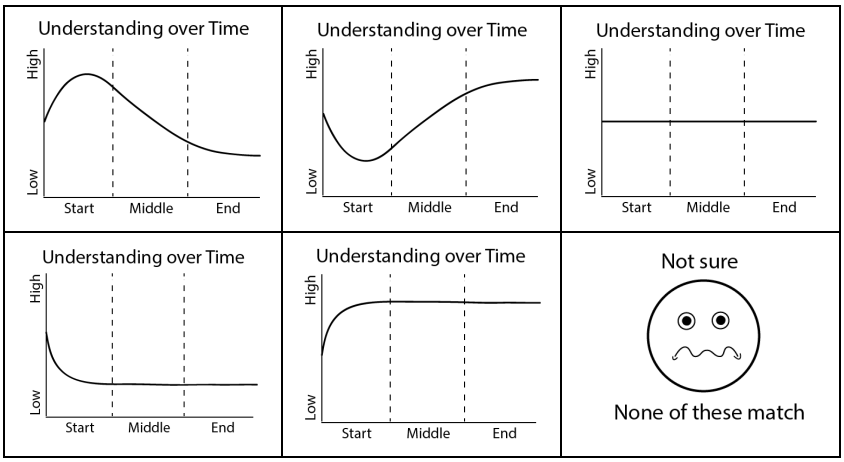
4. How much experience do you have with programming?
   A. I have no experience with it
   B. I have some experience programming
   C. I currently program for my study/hobby/work

5. Please circle the image below that best represents your **enjoyment** through the performance.



Please turn over to continue the survey ➡

6. Please circle the image below that best represents your **understanding** of the relationship between the **visuals** and the **music** through the performance.



7. This was a live performance! What effect did the visuals have on your sense of 'liveness' of the performance?

A. The visuals were distracting
B. They had no effect on my sense of 'liveness'
C. They helped

8. Were there any aspects of the visuals that you found confusing?

Thank you for your time!

Appendix B

Field Study Follow-Up Interviews

The initial exploratory field study (see Chapter 3) prompted a number of questions. These questions were examined through a number of email based follow-up interviews (for context, see Section 3.3.2). The responses to these follow-up interviews are presented below. Questions asked are italicised.

Interviewee 1

Question 1: What did you "understand" about what was going on with the code being projected? In particular, what did you understand about the relationship between the code and the music?

The code sets up a set of nested loops which are then modified by the composer in real time. This immediately leads to the danger of repetitive loops. It may be useful to have rhythms of 4 or 8 bar repetition as in Africa. Actually, this musical form lends itself to that type of rhythm and music. As soon as an organ comes in I am reminded of Mike Oldfield and am anxious that someone will say slightly distorted guitar. In jazz one improvises on standards so there is a very strong form (AABA etc) which, in general, is respected allowing the audience to deduce where they are in the piece. I had the feeling that the present way the code is used limited the music

Question 2: What would you like to understand more about the code in order to enjoy the performance more?

Yes, I think that if the audience were told what was happening or the ideas behind the constructs then I would be happier. Compared to jazz it is not note by note improvisation so an explanation of the limits and advantages would be useful.

Interviewee 2

Question 1: What did you "understand" about what was going on with the code being projected? In particular, what did you understand about the relationship between the code and the music?

In the beginning, I could tell from the silence and the live coding that it was being build, and sound by sound line by line was being added to as the piece grew. When [the live coder] went back in the code to change beats or melodies, I could tell something was being changed but wasn't tracking what or how.

Question 2: What would you like to understand more about the code in order to enjoy the performance more?

I feel like I already understood a rudimentary amount [...] which was enough to enjoy it. I feel like if I had more knowledge about code I would focus on that to the detriment of the music; and if I knew more about music then I may have focused on that to the detriment of my attention on the code. Considering my education, if I had not had the exposure to code and music through [...], then some basic rudimentary knowledge of code would have been good.

Interviewee 3

Question 1: What did you "understand" about what was going on with the code being projected? In particular, what did you understand about the relationship between the code and the music?

I understood that the music was being made from scratch and this was evident in the long silence before any sound is heard. I understand what sounds are being made based on the code names and that some of the numbers represent timing, volume and pitch. I still don't quite understand when the code is "ready" and starts working to make music. It has something to do with the highlighting the text, but that also confuses me. (I understand that most of the music is stored in the program as "sound bites" of real instruments, but sounds can also be made from scratch as mathematical wave functions [...]). Sometimes the coder scrolls up and down the screen to much and I get lost, I don't have a big picture of what all the code looks like.

Question 2: What would you like to understand more about the code in order to enjoy the performance more?

In some ways I would like to understand a little more about the code. It would be nice to have a director's commentary of what's going on behind the scenes, just so I can follow along with the changes that I can hear in the music as they are occurring. But I think I more enjoy just listening to the music, knowing broadly that a livecoder is manipulating code to make the sounds that I hear. I don't often like reading the code for the whole performance, maybe for a few minutes at a time, but then I like to switch off and just focus on what the musician is playing. I more often like to listen to the music and guess what the livecoder has done to make that changes (which is kind of backwards). I wouldn't mind having more understanding of the code on hand, but I probably wouldn't use the details of it during the whole performance every time.

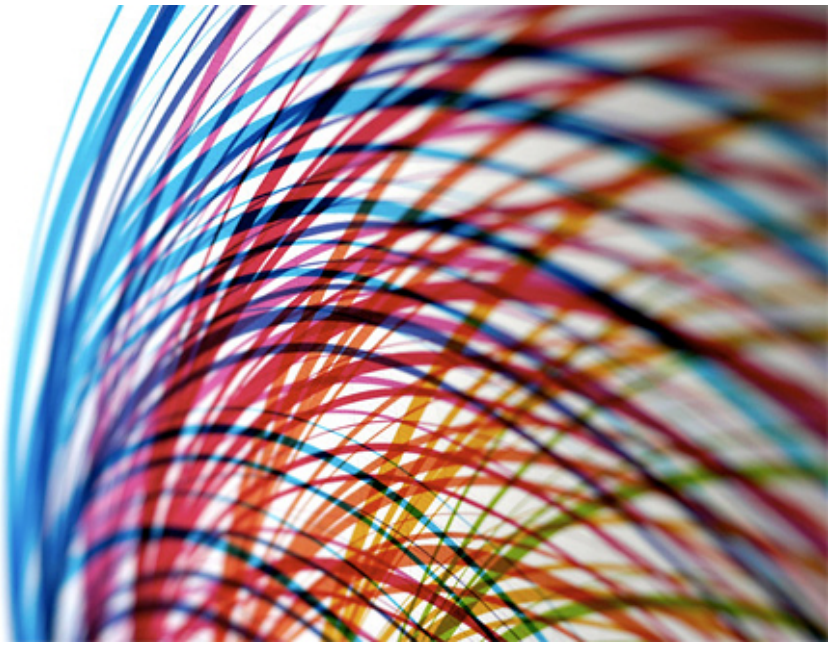
Appendix C

User Study Advertisement

The initial user study (see Chapter 5) was advertised throughout the university campus including through social groups and posters. The poster used to advertise the study is included below for reference.

Visualising Live Code

computer music and art performance



Live coding is the practice of using computer programming to improvise an artistic performance for an audience. We are conducting a research performance that will combine music with visuals through programming. Come along to one of the two sessions to find out more about live coding and help with the research.

30th May 2014

Session 1 at 3:30pm and Session 2 at 4:30pm

Building 108, CSIT, N101

bit.ly/visualising-live-code

Appendix D

User Study Survey

The initial user study (see Chapter 5) investigated the application of the developed visualisation to the space of live coding computer music performance. The survey used to evaluate these visualisations is presented below.

Live Coding Survey

Part A - Demographics

1. Age?
    a. 18-22
    b. 23-27
    c. 28-32
    d. 33-37
    e. 38-42
    f. 43-52
    g. 53-62
    h. 63+

2. Gender? _____

3. How many live coding performances have you been to?
    a. This is my first one
    b. I have been to one or two
    c. More than two. Please indicate the approximate number: _____

4. How much music do you regularly listen to?
    a. Hardly any
    b. A little
    c. A large amount

5. Do you play an instrument or sing?
    a. No - I would not consider myself a musician or singer
    b. Occasionally
    c. Yes - I play or sing regularly

6. How much experience do you have with programming?
    a. No experience
    b. Some experience
    c. I currently program for my study/hobby/work

7. Do you have much experience with the Lisp family of programming languages? (e.g. Scheme, Lisp, Clojure, Racket)
    a. Yes
    b. No
    c. I don't know what you're talking about

Part B - First Performance (Please complete after the first performance)

8. How would you rate your levels of **enjoyment** during the beginning, middle and end phases of this performance?
   Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:

   a. Beginning: Low Medium High

   b. Middle: Low Medium High

   c. End: Low Medium High

9. Did the projected visualisations help with your **enjoyment** of the code?
   a. Yes
   b. No
   c. No opinion

10. How would you rate your **understanding** of what the code was doing during each phase? Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:

    a. Beginning: Low Medium High

    b. Middle: Low Medium High

    c. End: Low Medium High

11. Did the projected visualisations help with your **understanding** of the code?
    a. Yes
    b. No
    c. No opinion

12. This was a live performance! Did the projected code and visualisations help communicate the feeling that the performance was *live*? If so, how?

Part C - Second Performance (Please complete after the second performance)

13. How would you rate your levels of **enjoyment** during the beginning, middle and end phases of this performance?
    Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:

    a. Beginning: Low Medium High

    b. Middle: Low Medium High

    c. End: Low Medium High

14. Did the projected visualisations help with your **enjoyment** of the code?
    a. Yes
    b. No
    c. No opinion

15. How would you rate your **understanding** of what the code was doing during each phase? Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:

    a. Beginning: Low Medium High

    b. Middle: Low Medium High

    c. End: Low Medium High

16. Did the projected visualisations help with your **understanding** of the code?
    a. Yes
    b. No
    c. No opinion

17. This was a live performance! Did the projected code and visualisations help communicate the feeling that the performance was *live*? If so, how?

Part D (Please complete after both performances)

18. Do you have any suggestions for how the visualisations for either performance might be improved? (Continue answer on the back of this sheet if you wish)

Appendix E

Live Coder Interview Transcript

This appendix contains the transcript of an interview conducted with the live coder (Ben Swift) following the first user study (see Chapter 5, specifically Section 5.3.5 for context). The first two performances were discussed with a goal of determining the differences in the mental model between the audience and the live coder.

Aesthetic Visualisation

Ben: So the first little synth thing in this I knew pretty much exactly what I wanted to do. I was going to do random pitches at 16th notes. I think I probably did this synth thing exactly the same in other performances.

Arrian: Is this a common technique you use?

B: Yeah, well. This was not even prepared specifically for this performance but I've done stuff like this in the past. This is a pretty common trick I use for getting up and running early.

I think the bass is playing at this point but I can't hear on these speakers.

A: How much planning went into this performance?

B: This performance was pretty safe. It was pretty simple and pretty preplanned. What I would say is that the form of all of the instruments are pretty standard for my live coding.

So for the bass to go through a list of pitches like that... I do that a lot.

For the drums I play with some sort of modulation of the sample slot, of the kit... I do that a lot.

I probably choose slightly different parameters each time through but, yeah, none of these things are really adventurous by the standards of my live coding.

A: Why is that?

B: Honestly, the main reason is that these things work well and I've tried other algorithms and they don't sound as good as these simpler algorithmic structures with judicious choice of parameters.

Though I feel that as an artist sure, as a programmer this is not so interesting because the algorithms are pretty safe.

There are a lot of people in computer music today that use fancy algorithms, they use cellular automata, they use generative algorithms but I think it works better if you use... I've had more artistic success using these simple algorithms and then just using my musical experience and intelligence to select good parameters...

A: Did you plan the performance around the visuals?

B: To be honest, I think in the second piece I tried to limit the callback rate because I knew that some of the visuals in the didactic setup worked better with a longer callback rate.

A: Do you think the visualisations held you back?

B: No, it certainly didn't hold me back. In some ways it is nice to have constraints.

I'm now going up an octave. It gives it a harder edge...

It is interesting that I'm very conscious of the hypermeter. I'm just grooving along and it just makes sense to evaluate in time.

A: Did you find this visualisation distracting?

B: Ah, no. In general I'm just so focussed on the code.

This little bit is adventurous. This drum bit I didn't know exactly what samples were in those slots. Before I was just guessing some numbers and picking numbers I thought might be good.

This bit is certainly more intense. This is more european house. I don't actually know if it is european house but I'm sure there is some name for this genre.

This end bit is a bit new. I hadn't planned to end it like that.

A: Did it occur by chance?

B: Not so much by chance. I got to the end and wasn't really sure how I would finish this. This is true making it up.

A: Were you happy with the performance?

B: Yeah.

I'm actually going diatonically out of the scale I was using for the whole piece.

Yeah, I hadn't planned to finish like that but... yeah, yeah, I'm reasonably happy with that.

B: I think in terms of the surprising stuff. There were no surprises when I started or added each new instrument. I knew pretty much exactly what I was going to do. I might not have had the exact parameters in mind. I would have put maybe a 60 instead of a 70. Even when I didn't have an exact number in mind I would have had an approximate number in mind... loud vs soft.

Once stuff is going then I think all bets are off and I at least don't really think through what I'm going to do after that. Generally I'll go back and start messing with stuff. In that case I went back and messed with the synth.

I did a bit of interesting stuff with the drums. I went from a more groovy and pretty standard drum beat to a heightened beat which definitely changed the mood of the piece.

I'm not unhappy with that. I'd probably do something different next time just because you do something different every time but that was one of the things that surprised me.

In general I was pretty happy with it. I think it is a good sound palette. The drums grooved pretty well which is an important thing. The bass line was pretty cool, though I couldn't hear it in that recording.

A: In terms of the visualisations, do you think they added anything to the piece?

B: I think they added something. I think they are just ambience. It is definitely cool to have that stuff going on that is a little visually interesting but I wasn't paying attention to them even then. I definitely wasn't paying attention to them on the day. In fact I tuned them out as best I can because I am just trying to focus on the code.

Like I was saying before there were times where it was hard to see the code under-

neath the visualisations.

A: Was this during the aesthetic or didactic performance?

B: I think it was more the didactic and I think it was the text in the didactic ones and not in the aesthetic.

I definitely like the visuals. They definitely add something and they don't take anything away even though I wasn't paying attention to them. I still see the text as the main thing but the visuals are gravy and that was nice gravy.

A: The audience was reasonably computer literature. Do you think they would have preferred to focus on the code, the music or the visuals?

B: That's a good question. I don't know. I'm really curious.

Focus is a funny thing. You rarely explicitly go "alright, I'm going to focus on blah and focus on blah". I think you drift. Probably at different points they were paying attention to each.

I think I'm a bad judge of what people pay attention to because I pay attention to completely different stuff when I watch it. What I pay attention to is probably completely unhelpful in terms of an indicator for what even a computer literate audience member would be paying attention to.

Didactic Visualisation

Ben: Now this one starts at a slower tempo. Half the speed... 60bpm vs 120bpm.

Arrian: Was this due to the nature of the visualisation?

B: No, that was just to be different. Both tempos work fine with the visualisations. In fact the visualisations pretty much work with whatever tempo.

So this one is a slower starting one. I still get it going fairly quickly.

A: Did the visualisations get in the way here?

B: It wasn't too bad because it's not over the top of where I was trying to work.

This is one of the things that I did have the visuals in mind when I put together this thing. Initially you have the fast spinning visuals. I knew that I was going to slow this

one down and go for two bars of eight beat long sustained chords. I knew that that would look cool as a slowly rotating thing.

In fact I stuffed it up there. It wasn't so much a typo as I did a tricky thing where I tried to have a couple of overlapping temporal recursions and filter out only the fast one keeping the slow one going. But that relies on changing the code once you've got it to the state you want.

I think in general with these parameters I was just messing around. I knew the general form.

I've got a couple of polyrhythms. I really like this bit. I think it works well.

A: Despite the timing issue with the visualisations?

B: Timing is off by half. We knew that was a problem. I still think it works pretty well. I think this one has real potential but it is disappointing that they didn't sync up.

This one is just grooving. I quite like the beat in this one.

A: Did the visuals affect your ability to see here?

B: I can't remember to be honest. It wasn't a big problem to be honest. It probably only happened once through all four pieces.

A: Were you tuning out these visuals during the performance?

B: Even when I'm watching it I'm tuning out the visuals but definitely during the performance.

I don't think this was planned. It is a fairly standard part of my live coding toolbox. I'm just changing the pitch. So it's to do with the bass. Quick ones and then long ones.

Then I changed the offset of the chords and the chords would come in staggered. I don't know how well this one worked in the end. I like bits of it but...

This one I think the stuff that is kind of up beat and is really groovy is easier to do than this.

This bit was disappointing. I had a cool ending in mind and then I stuffed it up here. I forgot to put the tick symbol.

A: Did you manage to pull off the cool ending in the second performance?

B: No, I tried to do the same thing and stuffed it up in the exact same way. That was interesting and frustrating. I had a cool ending that I just thought of that day where I was going to do some harmonic organ-y stuff, take it through the circle of fifths and do an interesting chord progression but really kind of draw one out for a slow finish. I just forgot to quote that symbol when I went from the minor to the major.

If I had other instruments covering me I could of started it again but since everything had died if I started it again it would have been really obvious so I decided in the moment that that is where I would finish it whereas I had one more minute planned. I started to go down a path where I had a minute more of material to finish it off but then just dogged it. Frustratingly I did not quite the same mistake but a similar sort of mistake in the second one. It's kind of really rare... obviously I make typos but I don't tend to make them in that way.

A: Was there some reason for the mistakes?

B: Not really.

A: Chance?

B: Yeah, just chance. Just life.

A: Was the main goal of this performance to entertain the audience... beyond the research?

B: Yeah, I think so. I always want the people to enjoy themselves. I tried to keep them pretty short. I think every little set was under ten minutes. If you're going to do a live coding set longer than ten minutes it needs to be bloody good. So in general I try to stay under ten minutes.

It's interesting, some of my earlier videos are longer than ten minutes and I watch them now and I think 'this drags on'. I'm much better at it now and I'm much better at making things happen quickly. I'm especially better at getting stuff up and running, partially because I'm an emacs guru and have all the snippet magic to make that happen but also you just learn the little extempore tricks and the general tricks for getting things up and running.

For example in the aesthetic set, there was probably stuff going after only 10 seconds. For this one there was probably stuff up before 30 seconds. I reckon you've got to get something up before 30 seconds.

A: Was boredom getting to you?

B: Not really. Certainly not in a big way. By the fourth I was like "I'm done, this

has been a lot of live coding, I'm sort of out of ideas".

It's not even fair to say 'out of ideas'. I had that cool idea about how I was going to finish it that I dogged both times. It's just exhausting. It really does take a lot of concentration.

I was done at the end and I was pretty happy it was done. I enjoyed it, I had a good time but I was glad it was done.

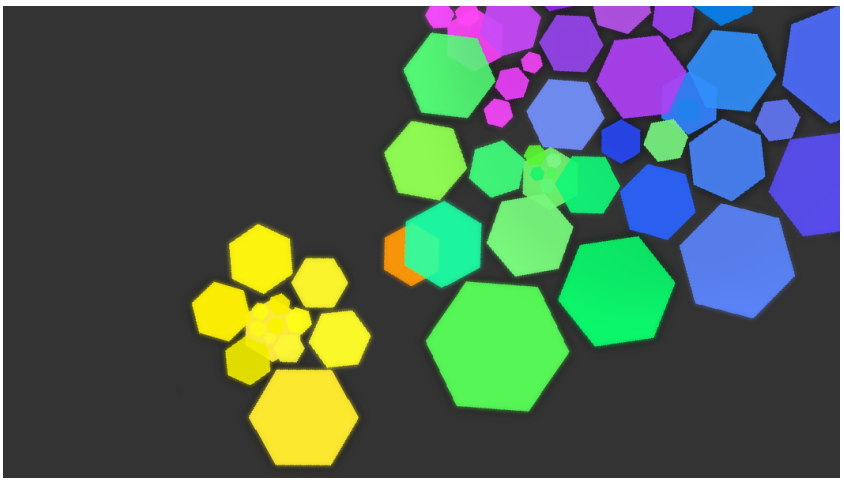
Appendix F

Follow-Up User Study Advertisement

The follow-up user study (see Chapter 7) was advertised throughout the university campus including through social groups and posters. The poster used to advertise the study is included below for reference.

Visualising Live Code

computer music and art performance

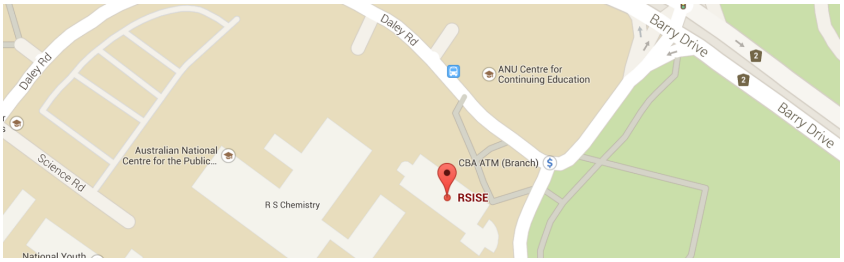


Live coding is the practice of using computer programming to improvise an artistic performance for an audience. We are conducting a research performance that will combine music with visuals through programming. Come along to one of the two sessions to find out more about live coding and help with the research.

30th September 2014

Session 1 at 3:30pm and Session 2 at 4:30pm

Building 115, RSISE, A105, Seminar Room



bit.ly/visualising-live-code-2

Appendix G

Follow-Up User Study Survey

The follow-up user study (see Chapter 7) investigated the differences between the application of no visualisations and visualisations to the space of live coding computer music performance. The survey used to evaluate the audience response to these conditions is presented below.

Visualising Live Code Survey

Part A - Demographics

1. Age? _____
2. Gender? _____
3. How many live coding performances have you been to before?
   - None
   - One
   - Many (Please specify: _____)
4. Did you attend the previous live coding study?
   - No
   - Yes
5. How much music do you regularly listen to?
   - Hardly any
   - A little
   - A large amount
6. Do you play an instrument or sing?
   - No - I would not consider myself a musician or singer
   - Occasionally
   - Yes - I play or sing regularly
7. How much experience do you have with programming?
   - No experience
   - Some experience
   - I currently program for my study/hobby/work

Part B - Please complete after the first performance

8. How would you rate your levels of **enjoyment** during the beginning, middle and end phases of this performance?
   Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:
   - Beginning: Low Medium High
   - Middle: Low Medium High
   - End: Low Medium High
9. How would you rate your levels of **understanding** of what the code was doing during each phase? Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:
   - Beginning: Low Medium High
   - Middle: Low Medium High
   - End: Low Medium High
10. What do you think the performer was doing in the very early stages of the performance?

    _____

    _____

    _____

11. What do you think the performer was doing in the very last stages of the performance?

    _____

    _____

    _____

Part C - Please complete after the second performance

12. How would you rate your levels of **enjoyment** during the beginning, middle and end phases of this performance?
    Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:
    - Beginning: Low Medium High
    - Middle: Low Medium High
    - End: Low Medium High
13. How would you rate your **understanding** of what the code was doing during each phase? Circle one alternative for each phase - interpret "beginning", "middle" and "end" as you wish:
    - Beginning: Low Medium High
    - Middle: Low Medium High
    - End: Low Medium High
14. What do you think the performer was doing in the very early stages of the performance?

    _____

    _____

    _____
15. What do you think the performer was doing in the very last stages of the performance?

    _____

    _____

    _____

Part D - Please complete after both performances

16. Did the projected code help with your **understanding** of the performances?
    - Yes
    - No
    - No opinion

17. Did the projected code help with your **enjoyment** of the performances?
    - Yes
    - No
    - No opinion

18. In the performance with the additional visualisations, did the projected visualisations help with your **understanding** of the performance?
    - Yes
    - No
    - No opinion

19. In the performance with the additional visualisations, did the projected visualisations help with your **enjoyment** of the performance?
    - Yes
    - No
    - No opinion

20. These were live performances! Did the projected code or visualisations help communicate the feeling that the performances were *live*? If so, how?

References

ANGELI, A. D., SUTCLIFFE, A., AND HARTMANN, J. 2006. Interaction, usability and aesthetics: what influences users' preferences? *Proceedings of the 6th conference on Designing Interactive systems*, 271–280. (p. 12)

AUSLANDER, P. 2008. *Liveness: Performance in a Mediatized Culture* (2nd ed.). Routledge. (pp. 4, 15)

BADROS, G. J. 2000. JavaML: a markup language for Java source code. *Computer Networks 33*, 159–177. (p. 9)

BAECKER, R. 1998. Sorting out sorting: A case study of software visualization for teaching computer science. *Software Visualization: Programming as a Multimedia Experience 1*, 369–381. (pp. 11, 13)

BAECKER, R. AND PRICE, B. 1998. The Early History of Software Visualization. In J. STASKO Ed., *Software Visualization: Programming as a Multimedia Experience*, pp. 29–34. MIT Press. (p. 11)

BECK, F., HOLLERICH, F., DIEHL, S., AND WEISKOPF, D. 2013. Visual Monitoring of Numeric Variables Embedded in Source Code. (p. 6)

BELL, R. 2013. Towards Useful Aesthetic Evaluations of Live Coding. *Proceedings of the International Computer Music Conference*. (p. 13)

BELL, R. 2014. Data Visualisation Tools for Enhancing Live Coding Usability and Audience Experience. (p. 13)

BEVAN, N. 2006. International Standards for HCI. May, 1–15. (p. 22)

BIGGERSTAFF, T. 1994. Program understanding and the concept assignment problem. *Communications of the ACM 37*, 5, 72–82. (p. 12)

BORGO, R., KEHRER, J., CHUNG, D. H., MAGUIRE, E., LARAMEE, R. S., HAUSER, H., WARD, M., AND CHEN, M. 2013. Glyph-based visualization: Foundations, design guidelines, techniques and applications. *Eurographics State of the Art Reports*, 39–63. (p. 23)

BROOKS, F. P. 1995. *The mythical man-month* (2nd ed.). Addison-Wesley Professional. (pp. 1, 6, 9)

BROOKS, R. 1983. Towards a theory of the comprehension of computer programs. pp. 543–554. (p. 9)

BURKHARD, R. A. 2004. Learning from architects: the difference between knowledge visualization and information visualization. *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, 519–524. (p. 10)

CARD, S., MACKINLAY, J., AND SHNEIDERMAN, B. 1999. *Readings in Information Visualization*. Morgan Kaufmann. (p. 10)

CAUDWELL, A. 2010. Gource: visualizing software version control history. *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 73–74. (pp. 5, 6)

CAWTHON, N. AND MOERE, A. V. 2007. The Effect of Aesthetic on the Usability of Data Visualization. *2007 11th International Conference Information Visualization (IV '07)*, 637–648. (pp. 12, 13, 25)

CHEN, Z. AND MARX, D. 2005. Experiences with Eclipse IDE in programming courses. *Journal of Computing Sciences in Colleges*, 104–112. (p. 11)

COLLINS, N. 2011. Live Coding of Consequence. *Leonardo 44*, 3, 207–211. (p. 5)

COLLINS, N. AND MCLEAN, A. 2003. Live coding in laptop performance. *Organised Sound 8*, 3, 321–329. (p. 4)

COX, P. T. 2007. Visual Programming Languages. In *Wiley Encyclopedia of Computer Science and Engineering*, Volume 10, pp. 305–11. John Wiley & Sons, Inc. (p. 9)

DESMOND, M., STOREY, M., AND EXTON, C. 2006. Fluid source code views for just in-time comprehension. *Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT06)*, 1–4. (p. 9)

DIEHL, S. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag. (p. 10)

EISENBARTH, T., KOSCHKE, R., AND SIMON, D. 2003. Locating features in source code. *Software Engineering, IEEE Transactions on 29*, 3, 210–224. (p. 41)

GALL, H., JAZAYERI, M., AND RIVA, C. 1999. Visualizing software release histories: The use of color and third dimension. *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*. (p. 13)

GREEN, M. C., BROCK, T. C., AND KAUFMAN, G. F. 2004. Understanding Media Enjoyment : The Role of Transportation Into Narrative Worlds. *Communication Theory*, 311–327. (p. 12)

HENDRIX, T., II, J. C., AND BAROWSKI, L. 2004. An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. *ACM SIGCSE Bulletin*, 387–391. (p. 11)

HOLTEN, D. 2006. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5, 741–748. (pp. 2, 3)

HUNDHAUSEN, C., DOUGLAS, S., AND STASKO, J. 2002. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing 13*, 3, 259–290. (p. 12)

HUNDHAUSEN, C. D. 1996. A Meta-Study of Software Visualization Effectiveness. (p. 12)

HUNDHAUSEN, C. D. AND BROWN, J. L. 2007. What You See Is What You Code: A live algorithm development and visualization environment for novice learners. *Journal of Visual Languages & Computing 18*, 1 (Feb.), 22–47. (p. 13)

ISO. 2002. ISO 14915-1:2002 - Software ergonomics for multimedia user interfaces. *2002*. (p. 22)

JERDING, D. AND RUGABER, S. 1997. Using visualization for architectural localization and extraction. *Proceedings of the Fourth Working Conference on Reverse Engineering*, 56–65. (pp. 9, 41)

KODOWA. 2014. Light Table. http://www.lighttable.com/. (p. 6)

KOSARA, R. 2007. Visualization criticism - The missing link between information visualization and art. *Information Visualization, 2007. IV'07. 11th International Conference*, 631–636. (p. 12)

KOT, B. 2005. Information Visualisation utilising 3D Computer Game Engines - Case Study: A source code comprehension tool. Technical report, Department of Computer Science, University of Auckland, Auckland. (p. 12)

LUCANIN, D. AND FABEK, I. 2011. A visual programming language for drawing and executing flowcharts. *MIPRO, 2011 Proceedings of the 34th International Convention*, 1679–1684. (p. 9)

MAGNUSSON, T. 2011. Algorithms as Scores: Coding Live Music. *Leonardo Music Journal 21*, 19–23. (p. 11)

MASURA, N. 2007. *Digital Theatre: A "Live" and Mediated Art Form Expanding Perceptions of Body, Place, and Community*. PhD thesis. (p. 4)

MCCARTNEY, J. 2013. SuperCollider. http://supercollider.github.io/. (p. 3)

MCLEAN, A., GRIFFITHS, D., COLLINS, N., AND WIGGINS, G. 2010. Visualisation of live code. *Visualisation and the Arts*, 1–5. (pp. 4, 5, 6, 9, 11, 12, 13, 15)

MCLEAN, A. AND WIGGINS, G. 2010. Live coding towards computational creativity. *Proceedings of the International Conference on Computational Creativity*. (pp. 4, 5)

MCLEAN, C. 2011. *Artist-programmers and programming languages for the arts*. PhD thesis. (p. 12)

MYERS, B. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*. (p. 10)

NOVAIS, R. L., TORRES, A., MENDES, T. S., MENDONÇA, M., AND ZAZWORKA, N. 2013. Software evolution visualization: A systematic mapping study. *Information and Software Technology 55*, 11 (Nov.), 1860–1883. (p. 9)

OGAWA, M. 2012. code_swarm. http://www.michaelogawa.com/code_swarm/. (pp. 5, 6)

PURCHASE, H., COHEN, R., AND JAMES, M. 1996. Validating Graph Drawing Aesthetics. *Graph Drawing*. (p. 13)

PURCHASE, H. AND MCGILL, M. 2001. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. *In Proceedings of the 2001 Asia-Pacific symposium on Information visualisation 9*, 129–137. (p. 13)

RAJLICH, V. AND WILDE, N. 2002. The role of concepts in program comprehension. *Proceedings 10th International Workshop on Program Comprehension*, 271–278. (p. 9)

REED, P., HOLDAWAY, K., ISENSEE, S., BUIE, E., FOX, J., AND WILLIAMS, J. 1999. User interface guidelines and standards: progress, issues, and prospects. *Interacting with Computers 12*, 2, 119–142. (p. 12)

REIS, C. AND CARTWRIGHT, R. 2004. Taming a professional IDE for the classroom. *ACM SIGCSE Bulletin*. (p. 11)

ROBSON, D., BENNETT, K., CORNELIUS, B., AND MUNRO, M. 1991. Approaches to program comprehension. *Journal of Systems and Software 14*, 2, 79–84. (p. 11)

RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. 2004. *The Unified Modeling Language Reference Manual*. (pp. 9, 11)

SAFRIN, I. 2013. Polycode. http://polycode.org/. (p. 40)

SKINNER, J. 2013. Sublime Text. http://www.sublimetext.com/. (p. 41)

SORENSEN, A. 2013. Extempore. http://extempore.moso.com.au/. (p. 3)

STALLMAN, R. 1981. *EMACS the extensible, customizable self-documenting display editor*. MIT Artificial Intelligence Laboratory. (p. 41)

STASKO, J. AND PATTERSON, C. 1992. Understanding and characterizing software visualization systems. *Visual Languages, 1992. Proceedings., 1992 IEEE Workshop on*, 3–10. (p. 10)

SUCHMAN, L. A. AND TRIGG, R. H. 1992. Understanding practice: video as a medium for reflection and design. In *Design at work*, pp. 65–90. (p. 29)

SWIFT, B., SORENSEN, A., GARDNER, H., AND HOSKING, J. 2013. Visual Code Annotations for Cyberphysical Programming. pp. 27–30. (pp. 5, 6, 42)

TAO, Y., DANG, Y., XIE, T., ZHANG, D., AND KIM, S. 2012. How do software engineers understand code changes? - An exploratory study in industry. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 51. (p. 9)

TOPLAP. 2010. Toplap Manifesto. http://toplap.org/wiki/ManifestoDraft. (p. 4)

VAN WIJK, J. 2005. The Value of Visualization. *IEEE Visualization 2005 - (VIS'05)*, 11–11. (pp. 10, 13)

VORDERER, P., KLIMMT, C., AND RITTERFELD, U. 2004. Enjoyment: At the heart of media entertainment. *Communication theory 14*, 4, 388–408. (p. 12)

WANG, G. 2008. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality*. PhD thesis, Princeton University. (p. 3)

WARD, A., ROHRHUBER, J., OLOFSSON, F., MCLEAN, A., GRIFFITHS, D., COLLINS, N., AND ALEXANDER, A. 2004. Live algorithm programming and a temporary organisation for its promotion. *Proceedings of the README Software Art Conference*, 243–261. (p. 4)

WARE, C. 2013. *Information Visualization: Perception for Design* (Third ed.). Morgan Kaufmann. (pp. 10, 23, 40)

WORTH, C. AND ESFAHBOD, B. 2012. Cairo. http://cairographics.org/. (p. 27)

WÜRTHINGER, T., WIMMER, C., AND STADLER, L. 2011. Unrestricted and safe dynamic code evolution for Java. *Science of Computer Programming*, 1–23. (p. 3)

ZEROTURNAROUND. 2014. JRebel. http://zeroturnaround.com/software/jrebel/. (p. 3)

ZHOU, H., XU, P., YUAN, X., AND QU, H. 2013. Edge bundling in information visualization. *Tsinghua Science and Technology 18*, 2, 145–156. (p. 3)