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

September 2014

Typeset in Palatino by $\TeX$ and $\LaTeX\,2_{\varepsilon}$.

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

Arrian Purcell
16 September 2014

Acknowledgements

Abstract

The visualisation of "live coding" computer music performance was evaluated as a means to enhance programmer and observer communication. Following an exploratory field study conducted during a live coding performance at an arts festival, interaction-driven visualisation techniques were incorporated into a live coding system. Through a process of visualisation evaluation and refinement, two controlled laboratory studies were conducted to evaluate the visualisations' contributions to the audience experience, with emphasis on the (self-reported) experiential dimensions of *understanding* and *enjoyment*.

Discuss thesis results...[final user study pending]

Discuss contributions...[final user study pending]

Contents

List of Figures

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*

Introductions...
-introduce the topic...
-introduce visualisations...
-introduce live coding...
-problem statement and tie in quote...

Thesis statement...
This thesis proposes that "code visualisation improves observer understanding and enjoyment". More specifically, this thesis investigates the question "can the application of visualisation techniques to live coding enhance audience experience by increasing understanding and enjoyment?".

Definitions...
-define live coding, understanding and enjoyment
-define visualisation techniques
-define the programmer and observer relationship
-define audience experience

Introductory discussion/summary...
-this thesis will explore code visualisations
-specifically, it will investigate visuals within the combination of the domains of software and music
-will be using Iive coding as a platform and case study for this (will discuss later)
-will develop and test code visualisations on audiences with audiences of varied levels of experience with programming, addressing code comprehension

1.1 History

1.2 Background

Existing code visualisations and representations...
-gource [Caudwell 2010]
-code flower
-lighttable - inline feedback (developed due to the idea that code is unobservable, indirect and incidentally complex)
-code annotations [Swift et al. 2013]

1.3 Structure

The structure of this thesis consists of an initial field study (Chapter 3), discussion of the first iteration of the visualisation prototype (Chapter 4), a user study examining the prototype visualisation approach (Chapter 5), discussion of the refinement of the prototype visualisations (Chapter 6), a second user study examining the refined visualisations (Chapter 7), summary of the results and contributions (Chapter 8), and a conclusion (Chapter 9).

Chapter 2

Literature Review

Software visualisation is building momentum within the space of live coding. This section 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 within the space of 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 acheive 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 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]). However, 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 the relation to the source code with some representations presenting a highly simplified source code representation

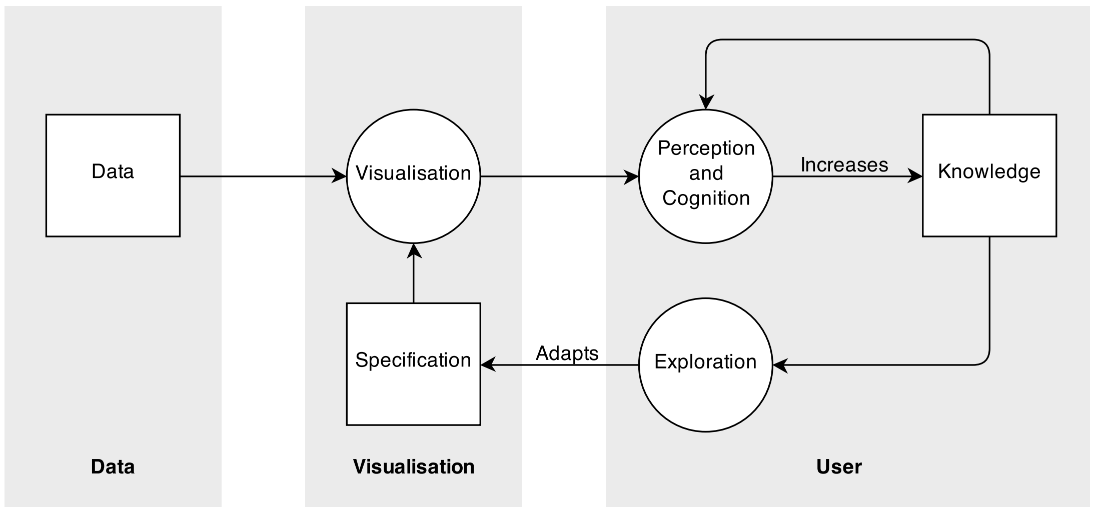


Figure 2.1: Generic model of visualisation [van Wijk 2005].

compared to finer grained specific representations [x]. 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.

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 the human visual information processing capability [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]. Following 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 [x]. However, the usefulness of these diagrams declines as they become more comprehensive and less abstract [x] and these static diagrams must be continuously updated or generated to keep up with the ever changing software [x].

Visual languages provide an alternative method of interaction with the software development process. However, visual languages are often not suited to general purpose programming [Myers 1990], providing only a subset of a full-featured text-based language. Studies based around visual languages show... [x].

Modern software development systems usually consist of an integrated development environment (IDE) to manipulate source code. These systems may feature source code annotations and allow the programmer to interact with the running program's source code [x]. These modern IDEs emphasise the relationship between the source code and the running programming but are still fundamentally tools for source code manupulation [x]. Despite the age of some of these tools, only the initial steps have been taken in order to implement and evaluate methods of communicating source code intent and mapping to the active program [x].

A number of studies hint at the limitations of static diagrams, visual languages and modern software development environments [x] and identify the need for alternative software reprentations and evaluation of their effectiveness [x].

Visualisations have the capacity to present information more effectively than traditional programming languages. Nevertheless, software visualisations still require significant development to benefit in the understanding of the complexity of software [Baecker 1995]. 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 1995].

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.
-use this paragraph to link to the next section

2.3 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].

The concept of "liveness" is covered comprehensively within the literature [Auslander 2008; Masura 2007].
-why is liveness important
-should it be used as a metric
-what are the gaps in the investigation of liveness

Live coding is in a unique position 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. These approaches are outlined below.

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]. Studies of live coding regarding the effects of visual and audible communication have indicated...

Live coding has a history exposing audiences to code with the goal of live coding to ensure 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 [x].

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 [x] 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] to effectively communicate intent [x].

Studies conducted into the communication of the live coding process indicate...

2.4 Programmer and 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.

-existing programmer/observer such as pair programming

-the artist and the audience within 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].

2.5 Enjoyment and Understanding

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]. Increasing enjoyment is one of the fundamental goals of the modern media entertainment industry and with increased enjoyment usually characterised by “pleasurable affective response to a stimulus” [Green et al. 2004].

Enjoyment results in increased attention to the stimulus and has been suggested to increase learning outcomes [x].
-how do these factors relate to understanding?
-how do these factors relate to visualistions?
-how can these two factors contribute to effective visualisations?
-the concept assignment problem [Biggerstaff 1994]

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 [x] 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]. Furthermore, understanding software while it is changing presents an additional challenge [Eisenbarth 2003].

Understanding and enjoyment are often considered opposite concepts when implementing strategies to increase one or the other.

2.6 Didacticism and Aestheticism

Enjoyment and understanding are often closely associate with the concepts of didacticism and aestheticism.

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]. Live coding has much potential and opportunity to evaluate and improve aesthetic effect [Bell 2013].

The artistic nature of the live coding field cannot be ignored when evaluating the effect of aesthetic and didactic approaches on the enjoyment and understanding of the audience.

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.

Questions raised within the literature suggest that it may be possible to combine or balance the goals of didacticism and aestheticism in order to manipulate understanding and enjoyment [x]. On the other hand, it may be that these two concepts cannot be completely separated.

The educational aspects of software visualisations have been examined in a number of studies [x] though few have applied these visualisations to areas outside of 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 [x] and within the space of software graph visualsations [x], 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.

Reviewing the model in Figure 2.1 from the didactic and aesthetic perspective identifies one major limitation. 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.
-has the literature identified a need to redefine the model in terms of the relationship between understanding and enjoyment, didacticism and aestheticism?

Chapter 3

Exploratory Field Study

"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*

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.

3.1 Rationale

The purpose of this interview 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.

3.2 Method

Audience members were asked to fill out a survey 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 in typing the code 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. Other survey questions addressed their sense of "liveness" of the performance (c.f. [Auslander 2008]) and whether the projected code was confusing.

3.3 Participants

A total of thirteen survey responses were received. Of these, 77% regularly listen to music and 54% perform regularly. 38% of the respondents have high exposure to programming through work, study or their hobbies, as opposed to 31% who have no

Figure 3.1: A live coder performs at the "You Are Here" arts festival in Canberra. An exploratory user study was conducted following the performance to examine audience perception of live coding.

experience with it. Of the respondents, 69% had never been to a live coding performance before.

3.4 Results

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 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 due to the small sample size. Nine of the thirteen respondents stated that the code projection provided a sense of liveness to the performance and the remainder stated that viewing the code had no effect on their sense of liveness. Four respondents felt that the code projections were confusing, five felt that they were not confusing, and four did not answer the question.

-more comprehensive results here + graphs

3.5 Discussion

Taken as a whole, the results of this small field study were salutatory towards the benefit of "seeing as well as hearing" code during a live coding performance, especially as far as the general public is concerned. The majority of the audience felt that they made the performance seem more "live". However, a minority stated that they found the projections confusing and only a very small number of respondents claimed to have actually understood what the programmer was doing. We were quite intrigued by the small cohort of respondents whose understanding increased through the performance and whose enjoyment remained high, and we wished to test whether augmenting code projections with additional visualisations might increase the understanding and enjoyment of the audience in live coding.

-more comprehensive discussion here

Chapter 4

Visualisation Design

Following the field study (see Chapter 3), a strategy for visualisation prototyping and evaluation was developed. This process and the result are outlined in the following sections.

[Ware 2013; Mclean et al. 2010; Purchase et al. 1996] will be useful here.

4.1 Rationale

4.2 Design

Visualisations developed employed dynamic analysis of the running code to generate the visuals. The intended knowledge flow from programmer to observer is shown in Figure 4.1.

Music visualisation is an extremely rich and open-ended task, so to guide the development of the visualisations for our lab study, we used the concepts of understanding and enjoyment from the initial survey to develop two new code visualisations: a *didactic* one and an *aesthetic* one.

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 (which is particularly relevant in a time-sensitive programming context such as music making). Bright colours and solid shapes were used to ensure constant visibility

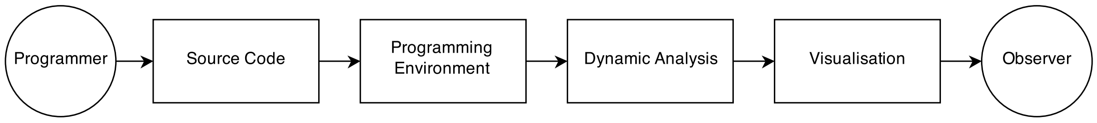
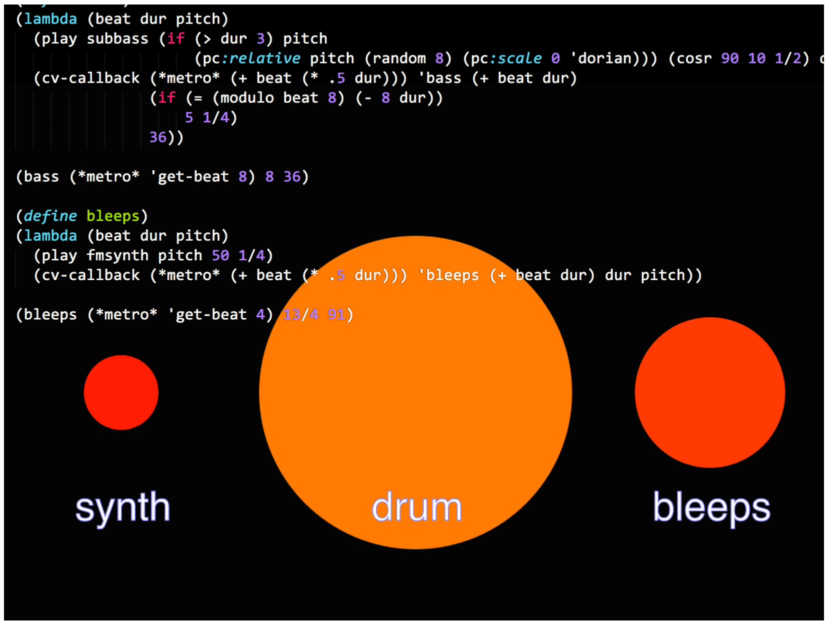


Figure 4.1: Knowledge flow from programmer to observer as directed by the visualisation technique employed.



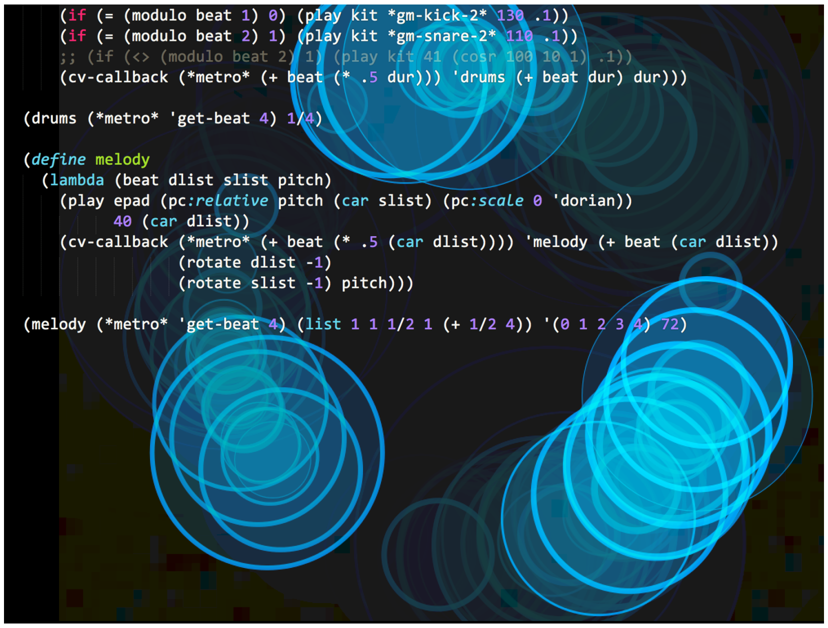
```
(lambda (beat dur pitch)
  (play subbass (if (> dur 3) pitch
                    (pc:relative pitch (random 8) (pc:scale 0 'dorian))) (cosr 90 10 1/2) c
  (cv-callback (*metro* (+ beat (* .5 dur))) 'bass (+ beat dur)
               (if (= (modulo beat 8) (- 8 dur))
                   5 1/4)
               36))

(bass (*metro* 'get-beat 8) 8 36)

(define bleeps)
(lambda (beat dur pitch)
  (play fmsynth pitch 50 1/4)
  (cv-callback (*metro* (+ beat (* .5 dur))) 'bleeps (+ beat dur) dur pitch))

(bleeps (*metro* 'get-beat 4) 13/4 91)
```

Figure 4.2: An example didactic visualisation.



```
    (if (= (modulo beat 1) 0) (play kit *gm-kick-2* 130 .1))
    (if (= (modulo beat 2) 1) (play kit *gm-snare-2* 110 .1))
    ;; (if (<> (modulo beat 2) 1) (play kit 41 (cosr 100 10 1) .1))
    (cv-callback (*metro* (+ beat (* .5 dur))) 'drums (+ beat dur) dur)))

(drums (*metro* 'get-beat 4) 1/4)

(define melody
  (lambda (beat dlist slist pitch)
    (play epad (pc:relative pitch (car slist) (pc:scale 0 'dorian))
          40 (car dlist))
    (cv-callback (*metro* (+ beat (* .5 (car dlist)))) 'melody (+ beat (car dlist))
                 (rotate dlist -1)
                 (rotate slist -1) pitch)))

(melody (*metro* 'get-beat 4) (list 1 1 1/2 1 (+ 1/2 4)) '(0 1 2 3 4) 72)
```

Figure 4.3: An example aesthetic visualisation.

and to communicate the intention of the underlying code. The didactic visualisations proceeded through four stages, with phase changes made depending on the number of active functions (instruments).

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 livecoder's edits, the generation of shapes was driven by instrument volume and synchronised with the musical beat. 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.

4.3 Mappings

Chapter 5

User Study

A second lab 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 approach would result in enhanced audience understanding, and a reduction in audience confusion through the performance. In contrast, we predicted that the aesthetic visualisations would positively influence audience enjoyment, both overall and over the course of the performance.

5.1 Rationale

5.2 Method

Two independent audiences ($N = 19 + 22 = 41$) recruited through an on-campus advertisement 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 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 questionnaire 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 lab study performance, a video-cued-recall [Suchman and Trigg 1992] interview was conducted with the live coder using a video of the performance.

5.3 Participants

A total of 41 participants 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 stated that they were male (see Figure E.2) and most participants were aged between 18 and 32 (76%, see Figure E.1). 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 (see Figure E.3). Nevertheless, only 15% of participants had previous experience with any of the Lisp style of languages (see Figure E.4), the style used within the performance.

Of the participants, 68% stated that they listened to a large amount of music (see Figure E.5) though only about 15% of participants stated that they played an instrument or sung regularly (see Figure E.6). Only 22% of participants had seen a live coding performance before.

5.4 Results

Of the 41 audience participants 66% were male, 76% were aged between 18 and 32 and 78% of the participants had never seen a live coding performance before.

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

5.4.1 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 ($\chi^2 = 3.7733, df = 2, p = 0.1516$).

Participants were asked to rate their enjoyment during the (self-determined) "beginning", "middle" and "end" of the performances (see Figure 5.2 and Figure 5.1). 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.

5.4.2 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

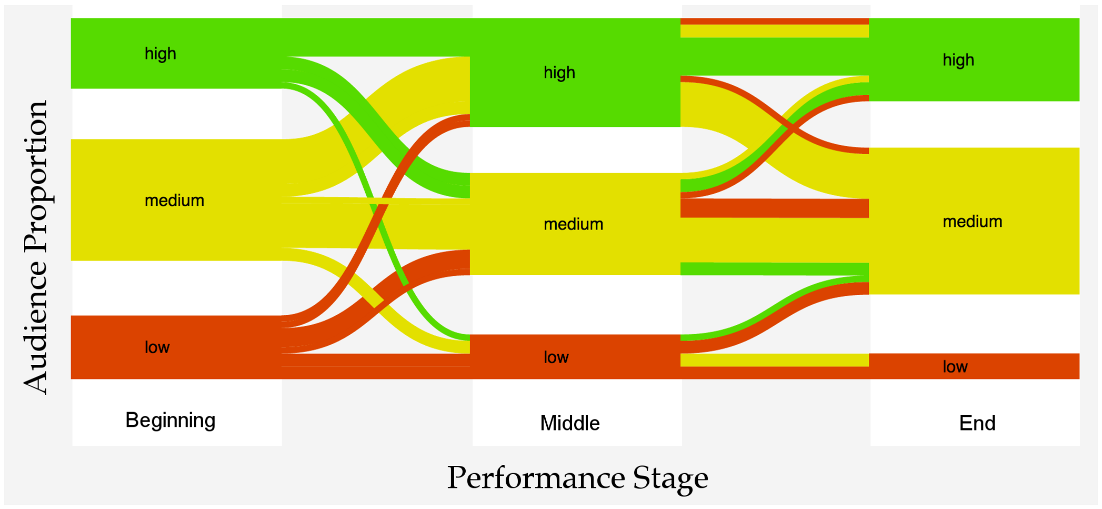


Figure 5.1: Audience reported enjoyment during the beginning, middle and end of the performance for the **didactic** condition. Line width at each stage indicates proportion of the audience reporting high, medium or low enjoyment, and line colour is determined by the enjoyment level at the *beginning* of the performance.

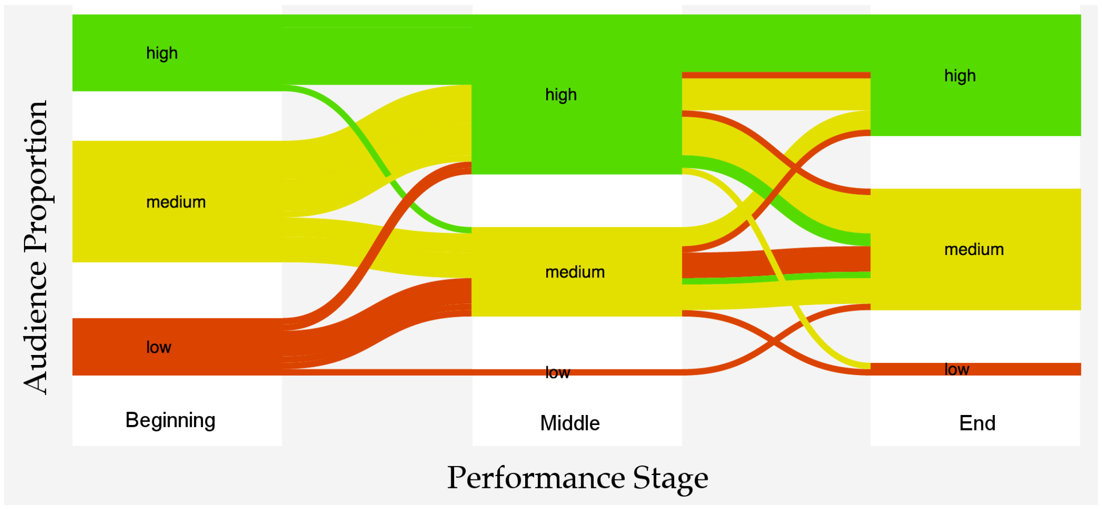


Figure 5.2: Audience-reported enjoyment level during the beginning, middle and end of the performance for the **aesthetic** condition.

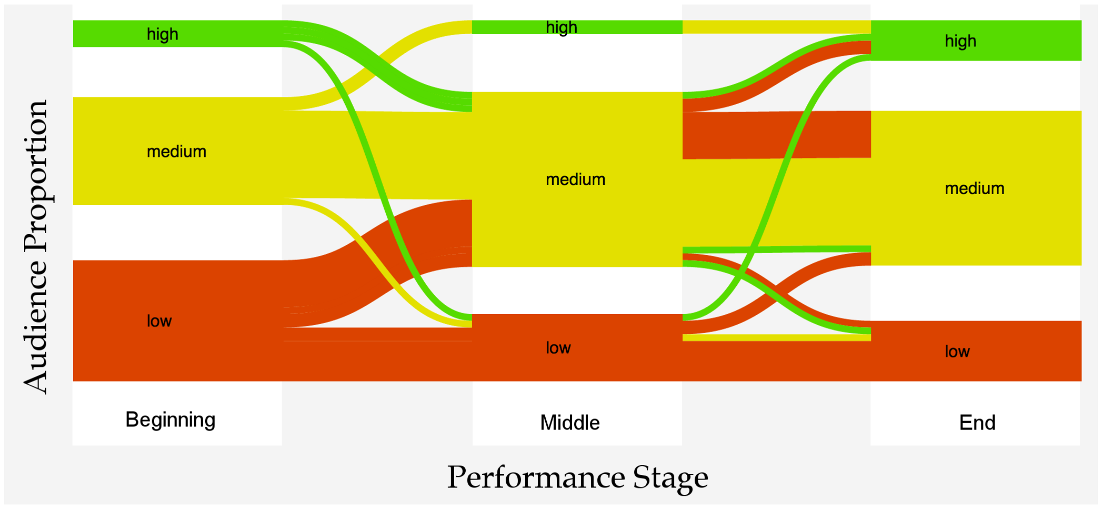


Figure 5.3: Audience reported understanding during the beginning, middle and end of the performance for the **didactic** condition.

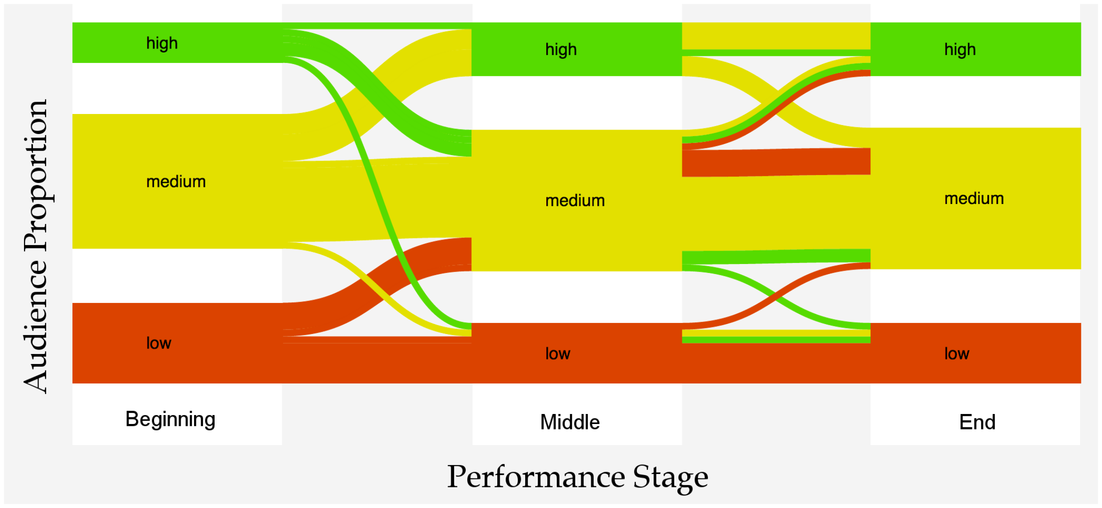


Figure 5.4: Audience reported understanding during the beginning, middle and end of the performance for the **aesthetic** condition.

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 and Figure 5.3). 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 questionnaire results for audience understanding are complex, and reported levels of understanding fluctuated during the performances. Dramatically, Figure 5.3 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 (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.

5.4.3 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.

within 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 referencing 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 its 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. A variety of the response 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”.

5.4.4 Improvements

The audience was asked to indicate possible improvements to the visualisations. 40% of the audience suggested improvements that indicate 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.

5.5 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 5.2 and 5.1 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 experience of the visualisations of the live coder and the audience was fundamentally different. 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 their 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".

Chapter 6

Visualisation Refinement

Following the first user study (Chapter 5), limitations with both the visualisations and the evaluation method were identified and steps were taken to correct the limitations. The following sections seek to identify the limitations and the steps taken.

6.1 Rationale

“the aesthetic of fixing nodes and edges to an underlying unit grid was prominent” [Purchase 2014] (also [Purchase et al. 2001; Purchase et al. 1996])

6.2 Design

The visualisation technique employed required three major components including an application manager, a code manager and an editor plugin (see the class diagram in Figure 6.2). The following sections detail the implementation of these components.

6.2.1 Application Manager

-polycode [Safrin 2013]
 -opengl

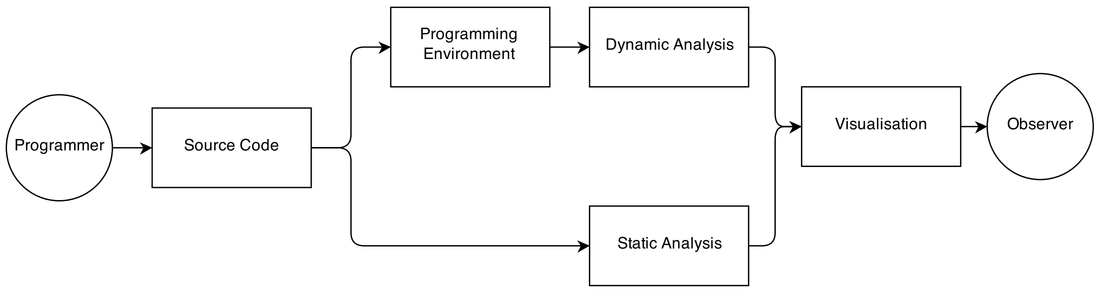


Figure 6.1: Knowledge flow from programmer to observer as directed by the visualisation technique employed.

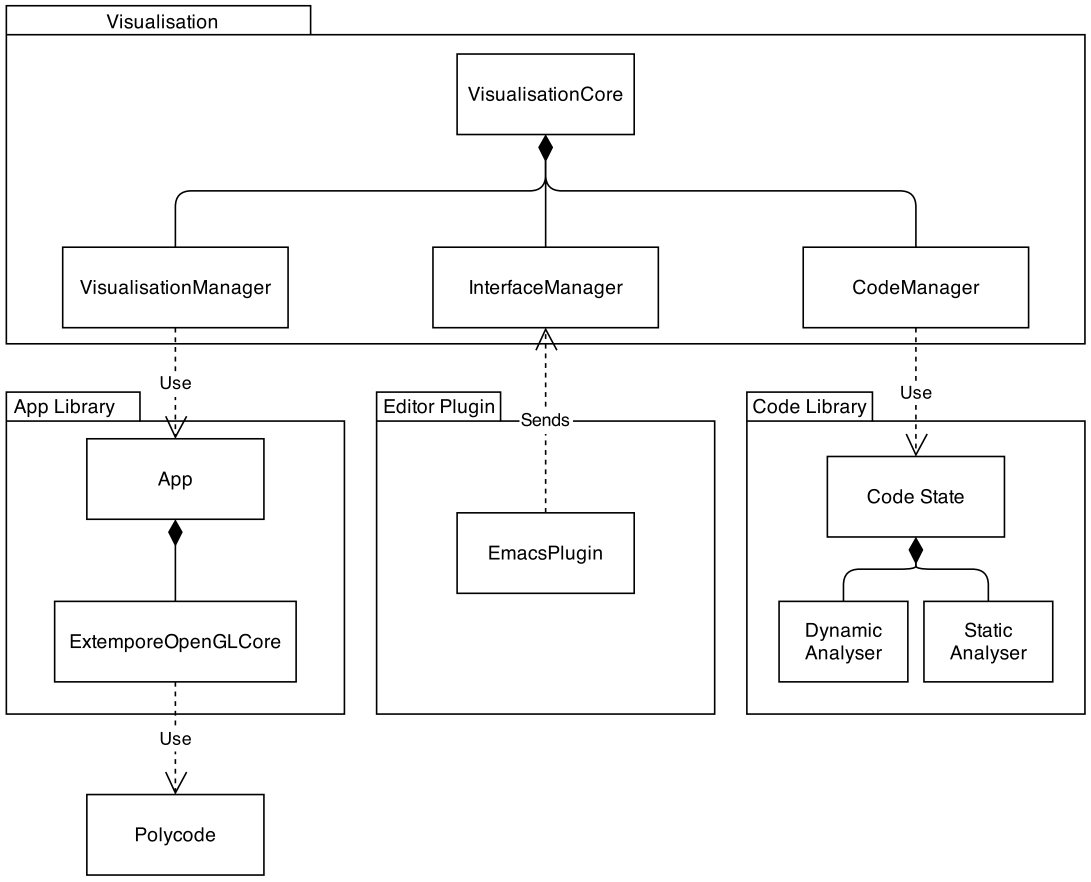


Figure 6.2: Class diagram of the visualisation technique employed.

6.2.2 Code Manager

-static analysis
-dynamic analysis

6.2.3 Editor Plugin

-sublime plugin
-emacs plugin

6.3 Analysis

Both dynamic program analysis and static source code analysis have been used to implement the visualisations (see combined static and dynamic approach taken in [Eisenbarth 2003])....

6.4 Mappings

A number of specific mappings were assigned to the visualisation. These visual mappings related directly to actions taken by the programmer.

Chapter 7

Follow-Up User Study

7.1 Rationale

7.2 Method

7.3 Participants

7.4 Results

7.5 Discussion

Chapter 8

Summary

8.1 Contributions

Updated software visualisation taxonomy built on the existing software visualisation frameworks...

Method of evaluating software visualisation...

Method of developing process driven software visualisation for live coding...

Process-driven software visualisation...

Evaluation of a process-driven software visualisation...

Contribution to live coding process and outcome... A multi-disciplinary approach, implementation and evaluation of a software application.

Discuss recommendations...

"...methodological issues have to be studied further. This concerns questions like how to design visualizations and how to measure and evaluate the effectiveness of various solutions." [van Wijk 2005] A methodology for visualisation design has been developed.

8.2 Limitations

-type of language may affect understanding. Imperitive vs functional etc.

-how wide can the results of this study be generalised outside of the field of live coding.

8.3 Future Work

Aesthetic elements of the software...

Didactic elements of the software...

Future of visualisations in live coding...

Application to software engineering practice...

Application to the arts...

Chapter 9

Conclusion

-restate research question
- -discuss how it has been examined
- -discuss how it has been answered
- -discuss wider implications of results
- -final words

Appendix A

Field Study Survey Results

Appendix B

Field Study Follow-Up Interviews

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

2) Would 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. Respondent 2

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.

2) Would 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. Respondent 3

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.

2) 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 Visualisations

-list all visualisations here

User Study Survey

Part A

Age?
18-22
23-27
28-32
33-37
38-42
43-52
53-62
63+

Gender? _______________

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

How much music do you regularly listen to?
Hardly any
A little
A large amount

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

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

Do you have much experience with the Lisp family of programming languages? (e.g. Scheme, Lisp, Clojure, Racket)
Yes
No
I dont know what youre talking about

Part B and C (repeated for the two performances)

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

Did the projected visualisations help with your enjoyment of the code?
Yes
No
No opinion

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

Did the projected visualisations help with your understanding of the code?
Yes
No
No opinion

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

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

User Study Survey Results

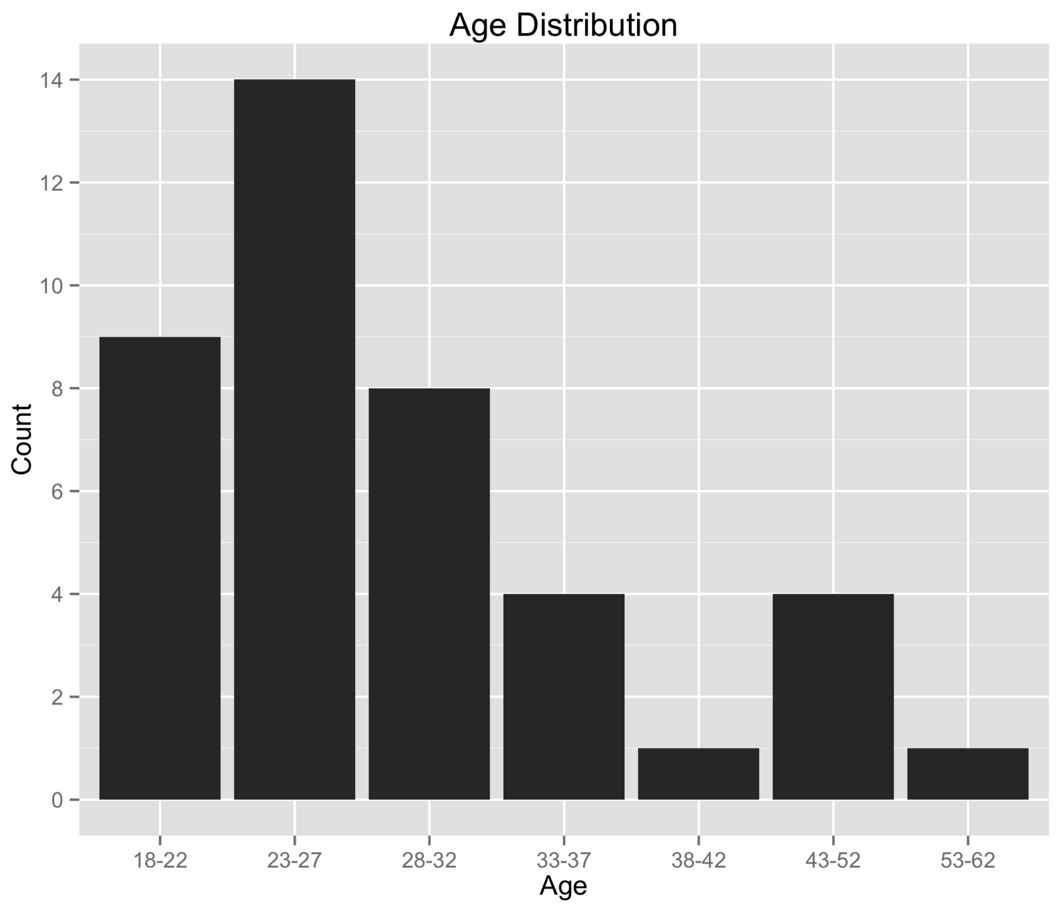


Figure E.1: Age Distribution

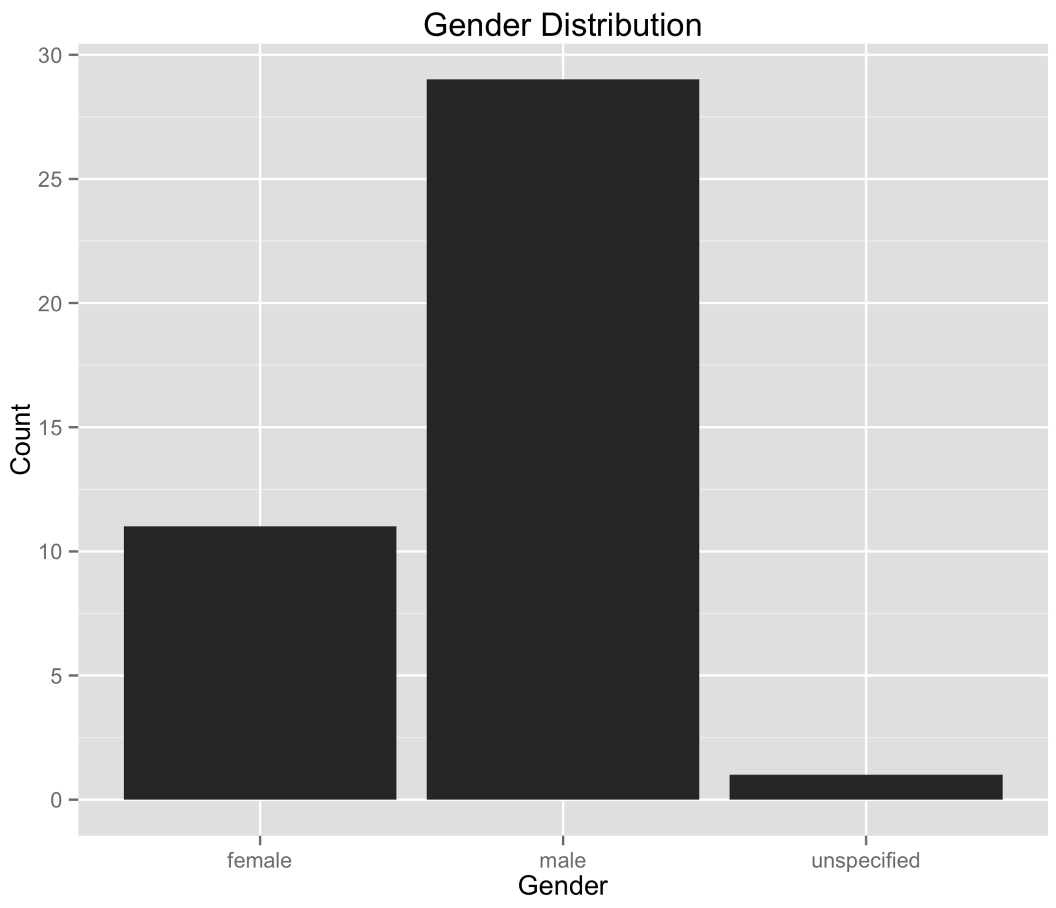


Figure E.2: Gender Distribution

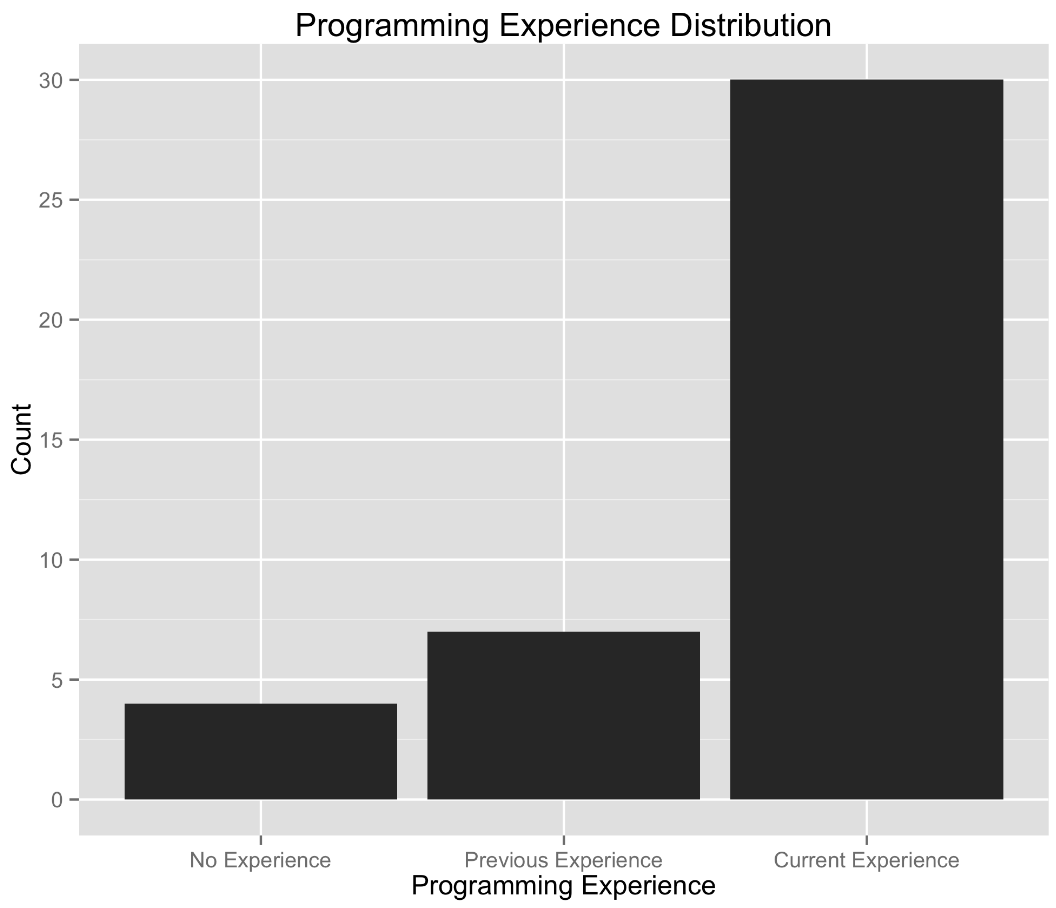


Figure E.3: Programming Experience Distribution

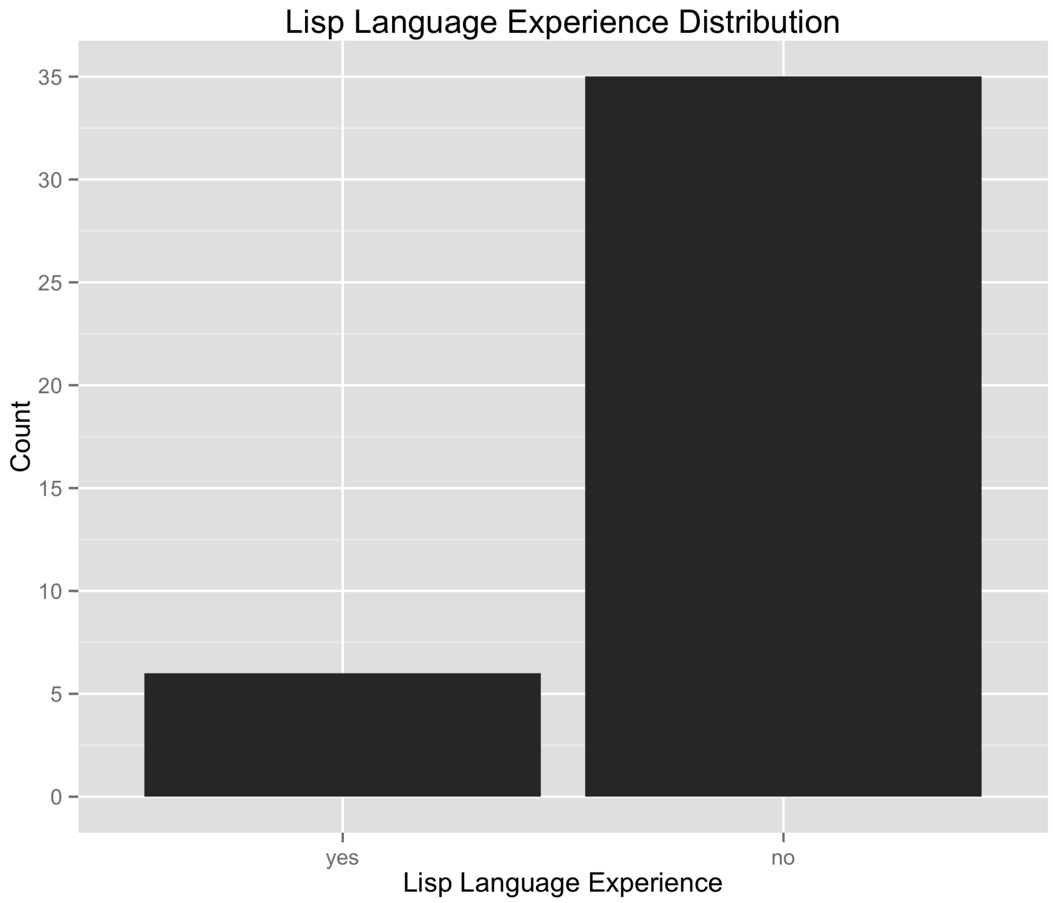


Figure E.4: Lisp Experience Distribution

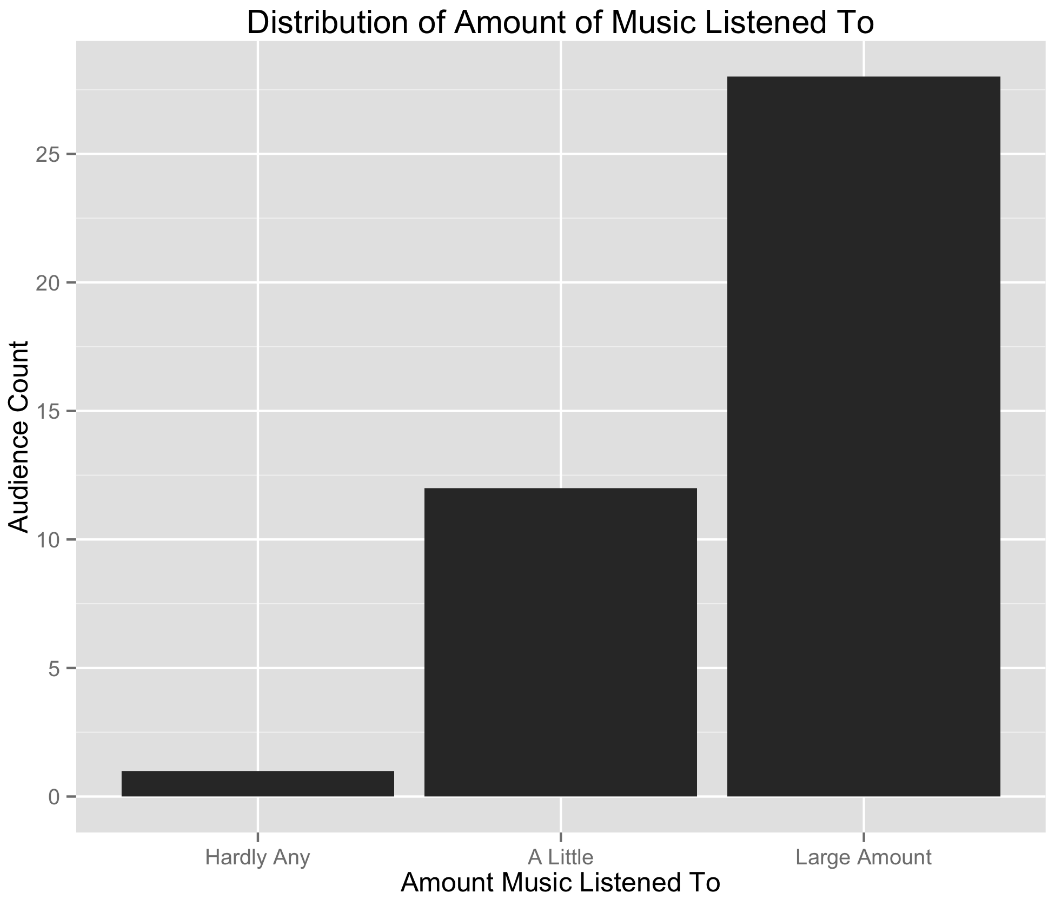


Figure E.5: Listen to Music Regularity Distribution

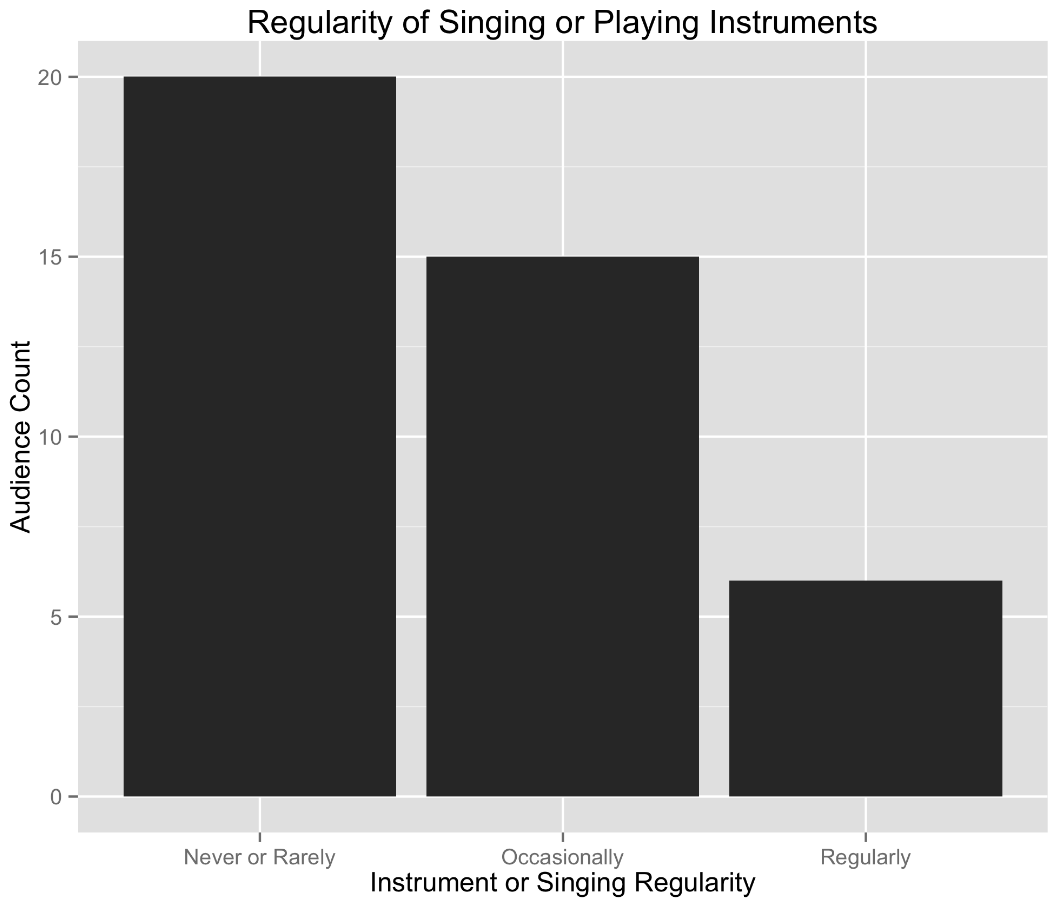


Figure E.6: Playing Instrument or Singing Regularity Distribution

Appendix F

User Study Live Coder Interview Transcript

This appendix is a transciption of the interview conducted with the live coder (Ben) following the first user study. The first two performances were discussed.

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 alot.

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

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 underneath 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 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. They both work fine with the visualisations. In fact the visualisation 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 the 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 G

Follow-Up User Study Visualisations

Appendix H

Follow-Up User Study Survey

Appendix I

Follow-Up User Study Survey Results

References

AUSLANDER, P. 2008. *Liveness: Performance in a Mediatized Culture* (Second ed.). Routledge. (pp. 5, 9)

BADROS, G. 2000. JavaML: a markup language for Java source code. *Computer Networks 33*, 159–177. (p. 3)

BAECKER, R. 1995. The Early History of Software Visualization. (p. 5)

BELL, R. 2013. Towards Useful Aesthetic Evaluations of Live Coding. *Proceedings of the International Computer Music . . . .* (p. 7)

BELL, R. 2014. Data Visualisation Tools for Enhancing Live Coding Usability and Audience Experience. (p. 7)

BIGGERSTAFF, T. 1994. Program understanding and the concept assignment problem. *Communications of the . . . .* (p. 7)

BROOKS, F. 1995. *The mythical man-month*. (p. 3)

BROOKS, R. 1983. Towards a theory of the comprehension of computer programs. pp. 543–554. (p. 3)

BURKHARD, R. 2004. Learning from architects: the difference between knowledge visualization and information visualization. *Information Visualisation, 2004. IV 2004. . . . .* (p. 4)

CARD, S., MACKINLAY, J., AND SHNEIDERMAN, B. 1999. *Readings in Information Visualization*. Morgan Kaufmann. (p. 4)

CAUDWELL, A. 2010. Gource: visualizing software version control history. *Proceedings of the ACM international conference . . .* , 4503. (p. 2)

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. 7, 8, 15)

COLLINS, N. 2011. Live Coding of Consequence. *Leonardo 44*, 3, 207–211. (p. 6)

COLLINS, N. AND MCLEAN, A. 2003. Live coding in laptop performance. *Organised . . . 8*, 3, 321–329. (p. 5)

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. 3)

DESMOND, M., STOREY, M., AND EXTON, C. 2006. Fluid source code views for just in-time comprehension. *Workshop on Software Engineering . . .* , 1–4. (p. 3)

DIEHL, S. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag. (p. 4)

EISENBARTH, T. 2003. Locating features in source code. ..., *IEEE Transactions on 29*, 3, 210–224. (pp. 7, 27)

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. 6)

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. 7)

LUCANIN, D. AND FABEK, I. 2011. A visual programming language for drawing and executing flowcharts. *MIPRO, 2011 Proceedings of the 34th ...*, 1679–1684. (p. 3)

MAGNUSSON, T. 2011. Algorithms as Scores: Coding Live Music. *Leonardo Music Journal 21*, 19–23. (p. 5)

MASURA, N. 2007. *Digital Theatre: A "Live" and Mediated Art Form Expanding Perceptions of Body, Place, and Community*. PhD thesis. (p. 5)

MCLEAN, A., GRIFFITHS, D., AND COLLINS, N. 2010. Visualisation of live code. *Visualisation and the Arts*, 1–5. (pp. 3, 5, 6, 7, 13)

MCLEAN, A. AND WIGGINS, G. 2010. Live coding towards computational creativity. ... *Conference on Computational Creativity*. (p. 6)

MCLEAN, C. 2011. *Artist-programmers and programming languages for the arts*. PhD thesis. (p. 6)

MYERS, B. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*. (pp. 4, 5)

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. 3)

PURCHASE, H. 2014. A healthy critical attitude: Revisiting the results of a graph drawing study. *J. Graph Algorithms Appl. 18*, 2, 281–311. (p. 25)

PURCHASE, H., COHEN, R., AND JAMES, M. 1996. Validating Graph Drawing Aesthetics. *Graph Drawing*. (pp. 8, 13, 25)

PURCHASE, H., MCGILL, M., COLPLOYS, L., AND CARRINGTON, D. 2001. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. *Proceedings of the 2001 ...*. (p. 25)

RAJLICH, V. AND WILDE, N. 2002. The role of concepts in program comprehension. *Proceedings 10th International Workshop on Program Comprehension*, 271–278. (p. 3)

RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. 2004. *The Unified Modeling Language Reference Manual*. (pp. 3, 4)

SAFRIN, I. 2013. Polycode. http://polycode.org/. (p. 25)

STASKO, J. AND PATTERSON, C. 1992. Understanding and characterizing software visualization systems. *Visual Languages, 1992. Proceedings ....* (p. 4)

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. 17)

SWIFT, B., SORENSEN, A., GARDNER, H., AND HOSKING, J. 2013. Visual Code Annotations for Cyberphysical Programming. pp. 27–30. (pp. 2, 6)

TAO, Y., DANG, Y., XIE, T., ZHANG, D., AND KIM, S. 2012. How Do Software Engineers Understand Code Changes ? - An Exploratory Study in Industry. (p. 3)

VAN WIJK, J. 2005. The Value of Visualization. *IEEE Visualization 2005 - (VIS'05)*, 11–11. (pp. xi, 4, 8, 31)

VORDERER, P., KLIMMT, C., AND RITTERFELD, U. 2004. Enjoyment : At the Heart of Media. pp. 388–408. (p. 6)

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 ....* (p. 5)

WARE, C. 2013. *Information Visualization: Perception for Design* (Third ed.). Morgan Kaufmann. (pp. 4, 13)