Poster: Programming Practices Among Interactive Audio Software Developers

Andrew Thompson, Gyorgy Fazekas
School of Electrical Engineering and Computer Science
Queen Mary University of London
London, England
andrew.thompson@qmul.ac.uk

Geraint Wiggins
Department of Computer Science
Vrije Universiteit Brussel
Brussels, Belgium

***Abstract*—New domain-specific languages for creating music and audio applications have typically been created in response to some technological challenge. Recent research has begun looking at how these languages impact our creative and aesthetic choices in music-making but we have little understanding on their effect on our wider programming practice. We present a survey that seeks to uncover what programming practices exist among interactive audio software developers and discover it is highly multi-practice, with developers adopting both exploratory programming and software engineering practice. A Q methodological study reveals that this multi-practice development is supported by different combinations of language features.**

***Index Terms*—Practices of Programming; Sound and Music Computing; Q Methodology**

I. Introduction

Domain specific languages for creating music and audio applications have existed since 1957, with the MUSIC-N family of languages [1]. Historically, these languages have been motivated by some technical challenge as in the case of SuperCollider's efficient garbage collector [2] or ChucK's time-based operations [3]. What is now becoming common is the consideration for the human factors that affect both a language's design and its users. Much of this research has been focused on the impact to the creative process [4], [5], particularly in the context of live coding [6].

What is less common is a focus on developers from the wider sound and music community; those creating DAWs, plugins, software synthesisers, and software for other audio developers. This community includes programmers from a wide variety of backgrounds, educations, and interests. To that end we present a survey of these programmers designed to categorise their programming practice as either software engineering or exploratory programming, and to uncover what language features have the most impact on that practice.

II. Methods

In many ways, exploratory programming and software engineering are dichotomous. Exploratory programming is open-ended, focused on error correction, and suited to small and medium sized projects [7]. On the other hand, software engineering focuses on well defined, large-scale projects, often with multiple collaborators. As a means of understanding an individual's programming practice, we have devised a three-part survey.

First, we collect demographic data such as age, education, programming experience, and what programming languages they use for creating interactive audio software. Second, 20 statements are arranged into a Likert scale that describes three of the dimensions set out by [8]: how well defined the software and its purpose is before beginning development, how long development takes and the size of the resulting code-base, and the relation between the intended and subsequent function of the software. These statements were composed by looking at existing literature on the programming practices in question and include statements about the use of version control software, the idea that creating software is a creative feedback loop with the computer, and the use of code testing[1].

Finally, we employ a methodology commonly used in the social sciences known as Q methodology [9] to learn what programming language features are most impactful to an individual's practice. Crucially, Q factor analysis allows us to reduce the individual responses we collect into a handful of factors that demonstrate some shared thinking or point of view.

Participants were asked to sort a concourse of 36 language features in two phases. First, by simply rating each feature based on its impact to their programming practice as either "Negative", "Neutral", or "Positive". Then by arranging them into a quasi-normal distribution such that the majority of features are rated as neutral or little impact and a single most negative/positive feature exists at the extremes.

The 36 features were sourced in a variety of ways. Some were obvious, such as static and dynamic typing. Others were more niche features like dependent types or first-class testing. We also looked at how existing languages described or marketed themselves; SuperCollider's website draws attention to supporting both object-oriented and functional styles of programming and Csound notes that different parts of a program are written with different syntax, for example. We also include

[1]Likert and Q sort items are made available online: https://github.com/pd-andy/papers/tree/master/materials/thompson_practice_2020

978-1-7281-6901-9/20/$31.00 ©2020 IEEE

features not directly related to the programming language but those that have an impact on the development experience such as tooling, libraries, and debugging capabilities.

As the survey was conducted online, it was important to make sure all participants understood each feature equally. To help with this, each feature included a brief description and, where appropriate, an image, code snippet, or video that demonstrated its use. In cases where a participant was unfamiliar with a feature, we asked them to first consider how it *might* impact their practice if they had access to it and to rate it as "Neutral" if in doubt.

III. Findings

Participants were recruited from a number of online communities; some language specific and some more generally related to music and/or creative computing. A total of 46 participants completed the demographic and Likert sections of the survey. Our key findings are presented here.

Likert items were treated as ordinal data. For items related to software engineering, ratings were ordered from -2 (Strongly Disagree) to 2 (Strongly Agree). For items related to exploratory programming this ordering was reversed. Doing so allows us to sum the ratings from every Likert item to get an overall score indicating tendency towards software engineering (a positive total score) or exploratory programming (a negative total score). We find the results to be normally distributed with a mean score of 1.63, a mode of 2.00 and a standard deviation of 7.06. We also find that neither language choice, nor experience, nor education has any significant effect on this score.

Of the 46 participants, 17 completed the entire survey and did the Q sort exercise. Exploratory Q factor analysis reveals three factors categorised by those language features most impactful to their practice. Factor 1 is distinguished by a preference for static typing, official tooling such as linters and formatters, and a comprehensive standard library. Factor 2 is distinguished by a preference for meta-programming features, dynamic typing, and strong debugging tools. Finally, factor 3 is characterised by a preference for first-class functions, immutability, and partial function application.

It is worth noting that at least one participant loaded into each of the three factors indicated that a visual programming language was their preferred or best language, even when this is seemingly in contradiction to their Q sort. The languages most commonly indicated as the best or most preferred language were: Pure Data (N = 14), C++ (N = 10), Max/MSP (N = 9), and JavaScript (N = 8). Additionally, 91% of participants indicated they had used at least one text-based programming language (N = 42) and 76% had used at least one visual programming language (N = 35).

IV. Discussion

Our findings support the idea that programmers wear many "hats" while programming. The results from the Likert scale show programmers are adopting a multi-practice approach, and that this occurs regardless of language choice and experience. The results of our Q factor analysis show programmers choose different languages to support this multi-practice development, but the survey presented here is not equipped to explain why that might be the case; especially when considering why participants loaded into factors 1 and 3 might prefer a visual programming language.

Designers of new languages for audio software development should keep this in mind and consider how their language can support multi-practice development. It is clear from the Q factor analysis that different features can support this in different ways and further investigation is necessary to explain this further.

It is important to note the shortcomings of this study. In one discussion online, a would-be participant felt they did not know enough about "computer science" to complete the Q sort exercise, despite having many years of programming experience. This is an unfortunate consequence of ensuring each statement was precise enough to avoid misunderstanding. Another participant reached out to note that the survey was difficult to complete when their programming practice could mean different things in different contexts; they work differently in a professional setting versus a hobbyist or creative one.

A follow-up study is already planned to address some of these concerns and answer questions arising from this study. We plan to gain a deeper understanding of *why* these language features are impactful and what, specifically, that impact is through semi-structured interviews. Additionally, participants will be asked to perform the same Q sort exercise to potentially corroborate the findings presented here.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council and the Arts and Humanities Research Council (grant EP/L01632X/1), ESPRC and AHRC Centre for Doctoral Training in Media and Arts Technology at Queen Mary University of London, School of Electronic Engineering and Computer Science.

References

[1] Wang, Ge. "A history of programming and music." The Cambridge Companion to Electronic Music. 2007.
[2] McCartney, James. "Rethinking the computer music language: SuperCollider." Computer Music Journal 26.4. 2002.
[3] Wang, Ge, and Perry R. Cook. "ChucK: A Concurrent, On-the-Fly, Audio Programming Language." ICMC. 2003.
[4] McPherson, Andrew, and Koray Tahiroglu. "Idiomatic Patterns and Aesthetic Influence in Computer Music Languages." Organised Sound: an international journal of music and technology. 2019.
[5] McLean, Alex, and Geraint A. Wiggins. "Bricolage Programming in the Creative Arts." PPIG. 2010.
[6] Nilson, Click. "Live coding practice." Proceedings of the 7th international conference on New interfaces for musical expression. 2007.
[7] Kery, Mary Beth, and Brad A. Myers. "Exploring exploratory programming." 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2017.
[8] Bergstrom, Ilias, and Alan F. Blackwell. "The practices of programming." In 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2016.
[9] Stephenson, William. "The study of behaviour: Q-technique and its methodology." 1953.