Motivating Programming Language Design for Digital Lutherie

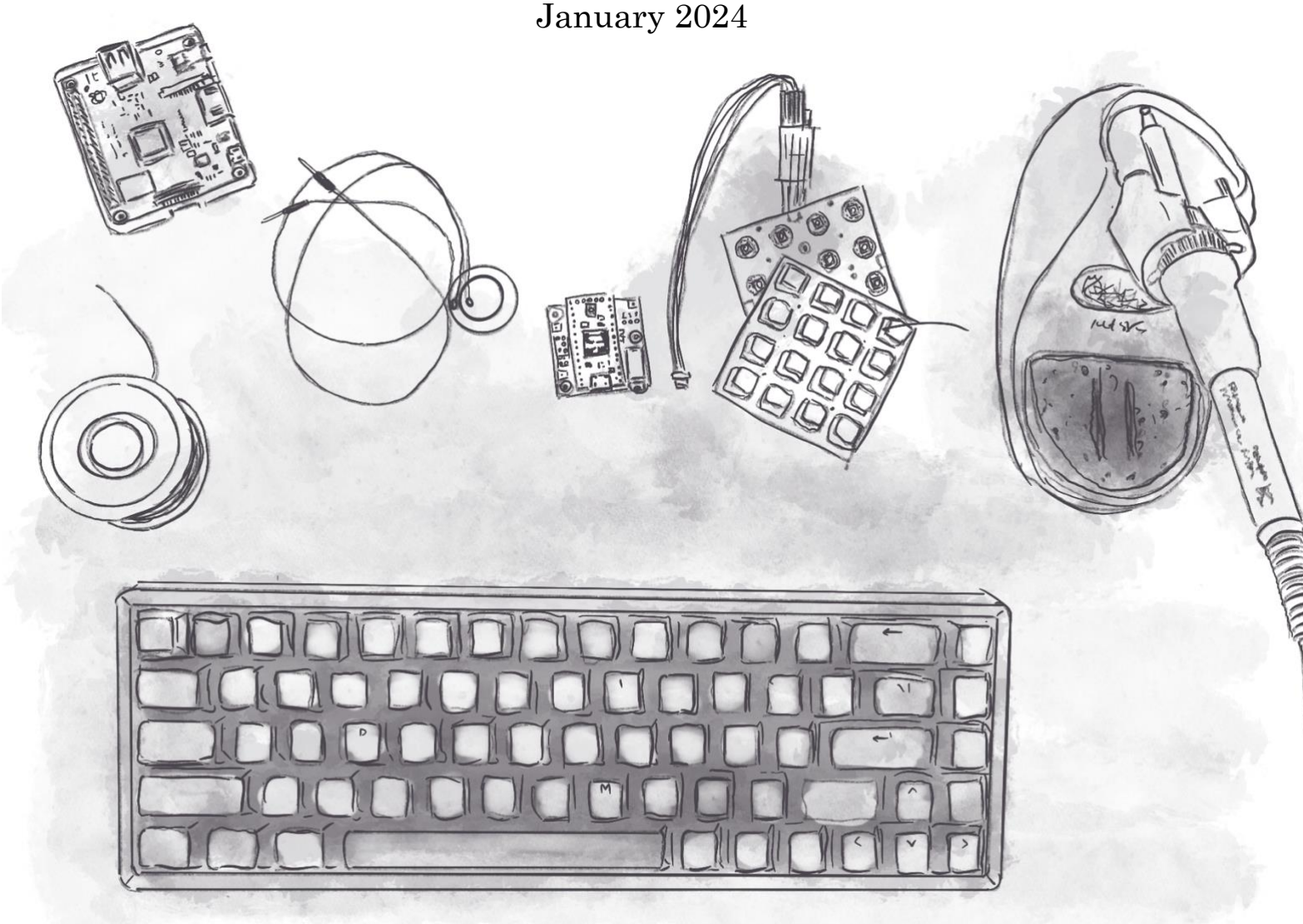
Word Count: 48557

Nathan Glyn Renney

A thesis submitted in partial fulfilment of the requirements of the University of the West of England, Bristol for the degree of Doctor of Philosophy in Computer Science

Faculty of Environment and Technology, University of the West of England, Bristol

January 2024



0.1 Abstract

Digital lutherie is a sub-domain of digital craft focused on creating digital musical instruments: high-performance devices for musical expression. It represents a nuanced and challenging area of human-computer interaction that is well established and mature, offering the opportunity to observe designers' work on highly demanding human-computer interfaces. Through the integration of instruments and computers, a new digital 'material' is introduced to the craft. And with a new medium comes new tools. Digital luthiers require expressive use of programming languages to draw together multiple different problem domains in creating new instruments. Motivated by initial explorations in programming language design, this thesis explores the motivations for tool choice in digital lutherie and inductively researches what characterises good programming language design for digital lutherie. Findings from 27 standardised open-ended interviews with prominent digital luthiers from commercial, research, independent and artistic backgrounds are analysed through reflexive thematic analysis. Our discussion explores their perspectives, generating a set of themes that are analysed and discussed. Through this process, a set of 'selective pressures' on language design is presented in order to help motivate and guide future language design in digital lutherie. We also present how challenges faced by digital luthiers relate to social creativity and meta-design, key components of end-user development. Some suggestions are also made to inspire strategies and approaches to programming language design.

Prior Publication

Parts of this thesis were previously published in the following papers:

- **Return to temperament (In digital systems)** (Renney, Gaster, and Mitchell 2018) N.Renney is the author of this work. The additional authors supervised the work.
- **Digital Expression and Representation of Rhythm** N.Renney and B.Gaster conceived the premise for this work. N.Renney designed the algorithms and structures and wrote the paper. B.Gaster and N.Renney jointly implemented the Haskell implementation. B.Gaster later added an additional implementation in C++. (Renney and Gaster 2019)
- **Studying How Digital Luthiers Choose Their Tools** (Renney et al. 2022) N.Renney and B.Gaster conceived the idea for an inductive study. N.Renney designed and conducted the study, led the analysis and wrote the paper. B.Gaster assisted in coding. All authors contributed to discussions in the iterative generation of themes.

Other previous works include:

- **Fun with Interfaces (SVG Interfaces for Musical Expression)** (Gaster, Nathan, and Carinna 2019)
- **Outside the block syndicate: Translating Faust's algebra of blocks to the arrows framework** (Gaster, Renney, and Mitchell 2018)

Contents

Chapter 1

Introduction

1.1 Introduction

For the traditional luthier, one might envisage a woodworking space where the luthier uses hand tools to sculpt the form of an instrument such as a violin or guitar, applying years of muscle memory and craft in order to create an instrument that empowers a musician to create music. This thesis concerns the digital luthier whose craft is extended into the digital space, resulting in the creation of digital musical instruments that can leverage computation to extend the possibilities of music making beyond the physical constraints of traditional instruments.

For the digital luthier, programming languages are as much a tool of their craft as a chisel or a saw. But as programming languages have become a ubiquitous tool for the creation of new technology, the design of programming languages has primarily relied on the tacit knowledge of communities or drawn upon features that develop from theoretical constructs that are more mathematically derived. This approach has led to a rich socio-techno-environment where programming languages develop through a means analogous to Darwinian evolution. The space between designing programming languages and the human-computer interaction of that process is waiting to be better understood and should be explored to inform the next generation of programming languages.

This thesis explores how we should approach designing the next generation of programming languages used by digital luthiers with a human-centred approach to programming language design.

1.2 Motivations (Prelude)

As this thesis focuses heavily on motivations for Digital luthier's approaches to creating new instruments, it would be appropriate to first relay how this work itself was motivated. My journey to this point began with the ambition of becoming a professional jazz musician, influenced heavily by the cross-over electronic jazz scene. This interest in how technology can be used, particularly in live and improvised contexts, led to a focus on music technology, digital musical instruments and ultimately, the world of computer science. In learning to program, a whole new world of creative expression was uncovered. Having taken a year after my degree to explore areas of programming real-time systems such as audio apps, games and embedded systems, I took the opportunity to work on a PhD which, inspired by Dr Bendict Gaster, began to look at programming language design through the lens on functional programming and programming language theory. And so we set out, exploring programming idioms and languages for musical expression, always with a keen interest in tangible ways to manipulate sound. Our mutual interest in controllers such as the MPC and its descendants led us to explore a strongly typed functional language for laying out and mapping

musical controllers, beginning with exploring two core components of music pitch and rhythm, each in the form of programming strategies for expressing the concepts. This work was influenced considerably by functional programming and the rise in popularity of functional techniques in general-purpose languages. As we worked on and considered the addition of a new domain-specific programming language, it struck me that from the many great programming concepts that I had been introduced to in the field of functional programming research (generalised algebraic data types, dependent types, even the ever-present and revered monad (Maguire 2018; Wadler 1992)) it was not easy to rationalise the inclusion of language features in a way that felt evidence-based. Ben's suggestion to take a step back and perform a study to motivate the design was the beginning of an unwinding stack of regression (nb. not recursion). Ultimately, exploring the literature and the lack of rigorous HCI around programming language design led me to a far more fundamental research question than whether adding pattern matching to a language improves the mapping problem. In 1966, Landin wrote the paper 'The Next 700 Programming Languages', which observed the direction and the language features that would define the next generation of general-purpose programming languages (Landin 1966). Whilst the number might be exaggerated for the more niche domain of music programming, this thesis sets out to provide a foundation for those developing the next '700 languages for digital lutherie' and to advocate for evaluation and design in a human-centred and democratised way. You may take a proof engine and provide a proof that a given program is 'correct', but of course, there is no way to prove that a programming language itself is the correct way for everyone to express themselves. However, scientifically rigorous and nuanced research has largely not been an informing factor for the design of programming languages. Instead, this has been left to the empirical learning, intuition and foresight of just a few. In the world of digital lutherie, there are many examples of language design with a number of domain-specific programming languages targeting audio and music programming. When writing their follow-up to Landin's seminal paper, Chately, Donalson, and Mycroft focused on capturing the evolutionary and human nature of programming languages rather than the adoption of specific features predicted by Landin originally, and this thesis takes a similar form (Chatley, Donaldson, and Mycroft 2019). Through investigating means for writing expressive programs for DMI creation, we realised that a more holistic and inductive approach was required to capture better the complex requirements of designing tools for digital lutherie. This work intends to provide future developers of programming languages that might be used to create music a nuanced discussion informed by the practices of digital luthiers, which might contribute ideas and inspiration for creating new tools, physical instruments and software instruments.

1.3 Research Questions

The primary research questions this thesis explores are:

How do digital luthiers select the tools they use?

What do digital luthiers require from their programming languages?

These research questions developed from the formative research questions used early in this work:

How do we design the next generation of programming languages used in Digital Lutherie?

How should we research the relationship between Digital luthiers and programmers?

Finally, through the course of this work, the reflective question was asked:

How can qualitative analysis be best applied in researching programming language design?

As such, some discussion is provided on this topic to share the experience from this thesis.

1.4 Overview

Following this introduction, this thesis is structured as follows.

Chapter 2 provides a literature review and background that aims to provide a high-level understanding of the fields drawn upon in this thesis. Due to the cross-disciplinary nature of this work, many ideas from Computer Science, mainly programming language design, must be drawn together with the rich field of digital lutherie. As such this chapter introduces and links together many different ideas that are then later used in discussion in chapters 5, 6, 7.

Chapter 3 introduces the early work that motivated the direction of this thesis. This chapter explores work on the design of new languages and language features that support the expression of musical ideas such as polyrhythm, polymeter and tuning systems. These ideas were used as a means to explore language design strategies, some of which offer solutions to the needs of digital lutherie, discussed in Chapter 7.

Chapter 4 describes the methodology for a study inductively analysing 27 standardised open-ended interviews with prominent digital luthiers using reflexive thematic analysis.

Chapter 5 presents the analysis and discussion of the inital reflexive thematic analysis for the study described in Chapter 4. This analysis produces three themes: 'The Pragmatist', 'A Product of Our Environment' and 'Intentions' .

Chapter 6 presents the findings from a secondary analysis following on from the findings and discussion of Chapter 5, further examining the data from the study outlined in Chapter 4. This analysis produces three more themes, titled, 'A Guiding Force', 'The Mutable Instrument' and 'Expressing My Ideas'.

Chapter 7 provides a discussion that contextualises the themes generated in both Chapters 5 and 6 within the broader literature, transferring existing theories from work in areas such as DMI design to the programming language theory of digital lutherie, as recommended by Chasins, Glassman, and Sunshine (2021). From this discussion, a series of 'selective pressures' are described to represent the needs of the digital luthier. Ideas from the literature and from the exploration of language design in Chapter 3 are discussed and related to these selective pressures and new directions for research on programming language design are presented.

Chapter 8 presents a peripheral contribution of this thesis, where through the process of conducting a significant reflexive thematic analysis, a discussion of strategies for improving the use and methods of qualitative research in HCI is presented. This chapter draws on various other fields and their methodologies, providing suggestions for improving transparency, rigour and developing future work.

Finally, Chapter 9 provides a brief conclusion, highlighting the contributions of this work.

1.5 Contributions

The contributions of this work are aimed at motivating and stimulating ideas for those designing new programming languages and tools for digital lutherie. Through an exploration of theoretical programming language design, HCI and reflexive thematic analysis, this thesis inductively creates hypotheses for exploration in future works, aiming to seed work that joins the fields of programming language design and HCI in the form described by Chasins, Glassman, and Sunshine (2021). This works analysis and discussion ties together ideas from the field of DMI research with the HCI of programming language design, corroborating ideas and extending their relationship to the tools used to create DMIs. This work also signposts DMI design as a rich example of meta-design as described by Fischer and Scharff (2000), establishing the role of programming languages in facilitating the continued development of an instrument after it is created and given to users.

This thesis contributes the following:

- Introduces an approach to expressing temperament through lists and list comprehensions (implying the potential to manipulate tunings with catamorphisms such as map, filter, reduce etc.), as well as a data model to represent musical rhythm as time-independent patterns that can be transformed into efficient representations that support polyrhythm and polytempic patterns.
- Presents a qualitative Analysis of interviews with prominent digital luthiers, inductively generating themes explaining the designer tool relationship for digital lutherie and signposting digital lutherie as a mature example of meta-design in practice.
- Continues the previous analysis to examine what features digital luthiers look for in their programming languages, resulting in themes that describe the idealised concepts that digital luthiers seek in their programming languages.
- Through discussion of these generated themes presents a set of design guidelines, framed as selective pressures for why digital luthiers choose their programming languages, then introduces some programming language design ideas that address these pressures.
- Produces a set of directions for future research of programming language design for digital lutherie, including the use of end-user development and in particular, meta-design in digital lutherie, how the influence of programming languages impact digital lutherie and how programming languages can avoid compromise in digital luthiers tool choices.

Additionally, this thesis shares insights and observations on the application of reflexive thematic analysis in order to provide rigorous and open research that promotes transparent research that can be built upon.

Chapter 2

Background

Whilst the term digital luthier is far from a standardised and far-reaching one, we see its evolution and popularisation in the field of research surrounding digital musical instruments (DMI). Popularised by Jordà (2004b), it has departed from the origins of lutherie, the building of stringed instruments, to replace the strung component with that of the digital domain. As such, we use digital luthier to refer to those who build digital instruments.

To introduce this thesis, we must explore the aspects that comprise the role of the digital luthier and, given its highly fluid definition, look to understand the developing wealth of understanding around how new instruments and interfaces for musical expression are designed. In order to contextualise the field, this background provides insight into the essential roles and relationships in the craft of digital lutherie and then considers how research is conducted in each of these areas, such that we might answer the research questions presented in the introduction. This chapter breaks down the archetype of the digital luthier to expose the underlying fields that provide the background required to contextualise this thesis.

This begins with an exploration of the programmer, the role that currently defines the creator within the digital domain, this section contextualises the environment of programming languages and those that use them. The digital luthier is concerned with creating instruments and tools for performance. The Second section explores how the digital luthier is not just a creator and how the need for performance and artistic expression drives this role.

Finally, Section 2.3 provides the overarching background that joins the world of technology, creation and performance into a single amalgamation of technologist, programmer, musician and composer.

2.1 The Programmer

This section aims to provide a high-level overview of the critical areas related to programmers, particularly how they approach problems and how the ecosystem of programming languages operates. Providing in-depth detail of all areas of programming languages alone would require an entire book. This background section aims to stitch together areas of programming in such a way that the relationships are clear enough to be further explored if required, and the foundations for following the discussion in this thesis can be followed. Studying the programmer requires understanding how many areas of computer science relate, and this section should help contextualise the discussion and comments from participants in the study described in Chapter 4.

2.1.1 The Programmer as an Individual

The role of a programmer is one that blends together a number of different application domains suggesting the need for a programmer to have a complementing range of domain-specific knowledge. Carey and Spelke state, 'Humans are endowed with domain-specific systems of knowledge such as knowledge of language, knowledge of physical objects and knowledge of number (Hirschfeld and Gelman 1994, page 169). They describe domains as systems of knowledge that directly apply to specific entities and phenomena. In the case of the programmer, we may immediately suggest these

are, at a minimum, the digital domain of computers and the domain for which they are developing an application. In 1979, Newell described a model of the 'problem space', a fundamental unit for reasoning and problem-solving that addresses goal-oriented cognition (Newell 1993). This describes problem-solving as a heuristic-based search of a given or novel problem space. Whilst this works for well-constrained problems such as puzzle-solving, Goel and Pirolli (1992) argue this oversimplifies design tasks. They observe that design tasks tend toward being underspecified from the outset, where the knowledge required for a solution draws on a near-limitless set of domains. They conclude that a set of invariant features define design-based problem-solving and present a framework they define as the 'design problem space'. As Goel and Pirolli present it, in the design context, we are required to consider a phase of four key points when observing problem spaces for design. Analysis of the problem through decomposition. The identification of how these components interact. Solving these components in isolation. The synthesis of the complete solution from the previously solved components where this final step must factor in the implications of how these components interact. Payne, Squibb and Howes examine the idea that working on a task on a computer (a device) requires mapping between two problem spaces: the device and the task domains goal (Payne, Squibb, and Andrew Howes 1990). Typically a programming language provides an abstraction over the device that is being programmed, meaning this mapping is more likely a mapping between the task domain's goal and the language's programming model. However, in many cases where performance is required, the interplay between the programming language and the device must be considered deeply, extending the domains in which a programmer works. While the nature of this thesis does not tackle pedagogy directly, it is implicitly related to programming languages as whether the programmer is a novice or an expert, new languages require some level of learning with perspectives in the literature typically focusing on psychology and education for novices and software engineering practises for experts (Robins, Rountree, and Rountree 2003). Due to the nature of digital lutherie, practitioners are likely to land all over the continuum of novice to expert, and so this background briefly introduces the current literature on each.

2.1.2 Novice Programmers

The literature on learning to program as a novice focuses on cognitive psychology and our understanding of knowledge representation, problem-solving and working memory (Robins, Rountree, and Rountree 2003). The constructivist viewpoint is currently the primary position on the philosophy of learning in computer science (Szabo and Sheard 2022). When applied to programming pedagogy, a leading tenet of constructivist views orient around programmers building a mental model of how the computer (or a programming language) works. These models represent 'shortcuts' that allow practical progress without fully understanding every detail of the device on which they work, which are refined over time (Ben-Ari 1998). But as Ben-Ari points out, it is critical that these models be explicitly taught and provide suitable stepping stones to more nuanced understanding rather than flawed models that later form obstacles. Winslow suggests that rather than specific language features, it is basic program planning that limits novice programmers meaning that rather than language syntax or nuances of the programming environment, it is the design and structuring of the program logic that presents a challenge Winslow (1996). In more recent work, following suggestions that mathematical aptitude was a predictor for programming success (Bennedsen and Caspersen 2005; Quille and Bergin 2018), Graafsma et al. present a study that explores five cognitive skills; pattern recognition, algebra, logical reasoning, grammar learning and vocabulary learning (Graafsma et al. 2023). They found that logical reasoning, algebra, and vocabulary skills were predictors of generalised programming performance and that logical reasoning was a further predictor of course-related programming performance. They concluded that algorithmic or, more broadly, mathematical skills were the most significant predictor of general

programming performance.

2.1.3 Experienced Programmers

For the most part, our understanding of the general psychology of expert programmers is provided by work done up until the mid 90's (Hoc 2014; Davies 1993), with trends in this field moving on to be far more focused on pedagogy and accessibility.

Adelson and Soloway studied the application design of expert programmers finding they applied breadth-first design when dealing with familiar domains and depth-first design for unfamiliar domains (Adelson and Soloway 1985). For domains that are familiar, the programmer would design an abstract solution and then decompose the problem into a series of subproblems in order to apply a known solution to each. In cases where the target domain was unfamiliar, however, programmers solved the problem one piece at a time, creating detailed solutions and composing those together to form a broader solution. Building upon this, Rist suggests that programmers design programs in a top-down fashion, with breadth or depth first design approaches dependent on expertise in the target domain of the application (Rist 1991).

However, in the limited research in this space, there is not a strong consensus that this reasoning holds. Davies argues that for the design of programs, strategies employed by experts vary considerably and are not indicative of the level of their level of expertise (Davies 1993).

As research focuses moved to the pedagogical side of programming and how novices solve problems, how more experienced programmers solve problems was left to remain as tacit knowledge (Thomas and Schneider 1984, Chapter 5), where the discourse largely resides in technical books and blog posts (Sprankle 2003; Gamma 1995). This tacit knowledge is often characterised as 'best practices' or in the context of particular languages or communities' idioms (discussed in Section 2.1.6). The formulation of these software engineering guidelines typically aims to capture understanding learnt through previous projects; however, this is prone to confirmation bias (Stacy and MacMillan 1995). For more formalised research, programming is explored in more specific scenarios, centred around programming features or concepts (Koeppe 2018).

2.1.4 Programmer Communities

In addition to considering the programmer as an individual, given the highly social nature of programming, it is also crucial to consider both the social context and cultures of programmers. Programming, and more generally creativity, are understood to be highly social endeavours (Fischer 2004). Fischer has extensively explored the importance of creativity in the context of socio-technical environments such as programmers, highlighting how 'communities of interest' form diverse, heterogenous collectives that share a common problem and solve them through sharing knowledge. This work draws attention to how creativity needs to extend beyond communities that focus on particular domains. For example, programmers form what Fischer defines as a community of practice, a community who are domain experts (for example, in computing) with a shared background and knowledge system. By extending their collaboration to a wider community of interest (such as designers, product testers and marketers for a software product), communities of practice are able to augment their creativity mediated through so-called boundary objects, which provide abstractions that can transfer meaning effectively between knowledge systems (Star 1989).

Further, in order to consider cultures of programming, an appropriate description of cultures is helpful. This, however, is not a simple topic to define, where culture has been criticised as too broad a definition to be helpful in a research context (Halabi and Zimmermann 2019). The idea of programmer culture is a familiar colloquialism that has been addressed in various forms around computing (Levy 1984; Selic 2008; Silver 2006). Much like the digital subculture 'the geeks' (McArthur 2009) or even punk culture (Lohman 2017), this notion of culture is based

around the shared context of its members. Though cultures as a component of social theory is still a heavily debated subject (Hesmondhalgh 2005; Bennett 2005; Bennett and Rogers 2016) and an in-depth description is beyond the scope of this work, culture is considered a critical dimension of HCI research (Sayago 2023). Much of the HCI literature depends on theories that were not formed to take into account the relationship between people and digital technology as we know it. In order to properly consider cultures, this thesis must look to incorporate cultures through inductive practice and through the lens of culture as a sensitising concept, forming and transforming theories of culture for each study rather than attempting to retrofit culture to existing generalised categories. Rather than using the taxonomic perspective, which categorises how culture is defined as a set of finite categories, research should aim to view culture as a fluid and constructed phenomenon of interaction between people, referred to in the literature as the contingent perspective (Halabi and Zimmermann 2019). With this approach, both the cultural backgrounds of the people who make up programming communities (Fleissner and Baniassad 2009), as well as the cultures that develop in communities of programmers, can be reconciled more holistically.

Ko et al. (2011) state, ' Most programs today are written not by professional software developers, but by people with expertise in other domains working towards goals for which they need computational support'. Examples of domain experts who program are particularly common in the sciences (Deardorff 2020; Eglen et al. 2017), integrating programming in the course of their day-to-day work. Notably, their code will likely not resemble that of more dedicated programmers (Sakulniwat et al. 2019) as their priorities do not align. Code may be written as a short-term test or to transform and handle data, therefore not requiring the diligence of a programmer who has end users to consider other than themselves. This principle may justify the focus of research shifting toward that of novice programmers described previously, as for many domain experts, learning to program is an ongoing process that is done in conjunction with their daily work whilst they simultaneously make progress on their goal. This, combined with the digitisation of people's lives, makes improving the learning process around programming important to multiple stakeholders.

2.1.5 The Evolution of Programming Languages

From punched paper tape to the programming languages we are familiar with today, programming has developed from the necessity of innovation to a tool to empower the expressive potential of computers. The design of programming languages has been developed through the duality of engineering practical solutions to real-world problems and through the development of theoretical constructs in fields of logic. Famously, javascript, one of today's most prolific programming languages, was developed in just ten days to add interactivity to Netscape's web browser and facilitate the transformation from static to dynamic web content. It would have been impossible to foresee that this language would grow to develop from what was considered a 'toy language' with many shortcomings into such a widespread language (Mikkonen and Taivalsaari 2007). With its use cases now extending beyond the web, many substantial innovations have enabled the language to thrive, such as its advanced interpreter (Dot, Martínez, and González 2015) and the addition of frameworks (Tilkov and Vinoski 2010) and supersets of the language (Bierman, Abadi, and Torgersen 2014; Burnham 2015) that make it more ergonomic to work with in teams or in larger code bases (Cherny 2019; Rastogi et al. 2015). We see this trend with many well-established long-standing programming languages, with criticisms of the C language's lack of safety (Akritidis et al. 2008) as another example. These languages are examples of how languages are developed in the short term to solve an immediate problem, then later rely on innovation and absorbing new theoretically informed features in order to thrive.

With this engineering-first, solutions-driven approach to programming language design, languages evolve to fill their niche. Chatley et al. describe the landscape of languages as existing in

a ‘Darwinian tree of life’ largely implying survival of the fittest, yet also including mechanisms in which languages with less fitness, in the evolutionary sense, can remain relevant long beyond what seems likely (Chatley, Donaldson, and Mycroft 2019). Examples of these factors include established communities, maturity in libraries and tools and the challenges of replacing large legacy codebases.

Though many features of programming languages are incorporated in an ad-hoc fashion in order to solve immediate problems, we still see the languages of today drawing from the theoretical work on programming languages design that still builds heavily upon the likes of Gödel, Church and Turing nearly 90 years ago. The seminal Landin paper titled ‘The next 700 programming languages’ (Landin 1966) makes a number of predictions about what programming languages concepts will define the future of programming languages. Landin describes the ISWIM family of languages imagining that this ancestral approach to language design may form a systematic way of thinking about programming language design. This is a reasonably apt prediction with the influence of language ‘families’ being commonplace in categorising programming languages, however much like in the evolutionary analogy constructed in the Chately et al.’s follow-up paper 50 years later (Chatley, Donaldson, and Mycroft 2019), language features act like genes in the evolutionary sense, where ideas such as lambdas, streams and futures (all concepts discussed in the context of ISWIM languages by Landin) are added into existing languages through selective pressures. For a real-world example, we can consider the use of futures to manage asynchronous control flow in programs as concurrent processing has become more critical.

Type theory is one of the most significant theoretical ideas to impact modern programming language design. As codebases have grown to projects with vast codebases and many contributors, it has been seen as more and more critical to have tools such as compilers, transpilers and linters that are able to enforce correctness about a program, effectively eliminating many runtime bugs statically, when the code is written. In addition, types also provide many opportunities for runtime performance optimisation. Dynamically typed languages describe languages that do not enforce types when written but instead allow the type of data to represent to change at runtime dynamically. As codebases using dynamically typed languages have grown to a size that no individual is familiar with an entire code base, pressures are applied to help prevent code from being misused as it changes. There are many examples of where this pressure has resulted in efforts to retrospectively add stronger types to languages through techniques such as transpilation (as with Typescript) and gradual typing in Python code bases (Jin et al. 2021), a strategy that allows types to be incrementally annotated in a code base, rather than requiring it to be added immediately.

2.1.6 Paradigms and Idioms

As programming languages have evolved, they have grown to address solutions in wildly different ways. In 1972 Dijkstra lamented that as computers had grown more powerful, so programming had become a greater problem (Dijkstra 1972). 25 years later, it was clear that there was no one solution to programming, with the prominent voices of programming language design expressing many different opinions on the state of programming languages (Trott 1997) and paradigms continuing to evolve (Floyd 1979). Since then, a range of different programming paradigms has continued to develop in order to address the ‘gigantic problem’ that is programming. Due likely to its more direct mapping to the mechanisms of computing, procedural and imperative programming largely dominate programming. However, declarative programming and the functional paradigm are growing in popularity (Gagniuc 2023). For general-purpose programming languages, this tends to support multiple paradigms (Steffen 2019), further demonstrating the adaptability of languages to stay relevant and evolve to the use cases that its community requires. Of course, languages may also reject certain premises as they fall out of favour, such as the Rust language’s lack of

support of object-oriented programming (Milanesi 2022). As communities develop approaches to solving problems, the notion of idiomatic code developed (Sivaraman et al. 2022) to support shared reasoning about the code, lowering cognitive load (Sakulniwat et al. 2019). This has also led to broader patterns in programming that are recognised as common design patterns (Gamma 1995), codifications of the expert knowledge that can be used to tackle problems.

Practically, programming languages must provide an abstraction over the finite resources of the computer that runs the program it outputs. Memory management is an example of a complex practical problem in computing that is typically an entirely different domain to reason about than the problem the programmer is solving. Programming languages attempt to abstract this problem space away through strategies like Garbarge collection (Grgic, Mihaljević, and Radovan 2018), RAII (Combette and Munch-Maccagnoni 2018), reference counting (Hudak 1986), and, more recently, borrow checking (Blackshear et al. 2022).

All of these strategies have different tradeoffs, for example, Java and its implementation of the Java virtual machine (JVM) provides a cross-platform way to develop applications without the overhead of managing memory allocation, making a good case for ease of development that suits many businesses. For performance-sensitive environments, however, this lack of control over when garbage collection occurs can lead to unacceptable delays in processing at critical times. As a result, alternative strategies such as C++'s RAII, Swift's reference counting and Rust's borrow checking have remain as solutions to managing memory more effectively, adding more varying degrees of safety over completely manually memory-managed languages like C.

As the mechanism for the development of programming languages, a Darwinian tree of life represents a good analogy for the current way in which programming languages develop. As research focused languages introduce new ideas, we see these features inherited and mutated by other languages.

2.1.7 Domain Specific Languages

Whilst general-purpose programming languages provide the backbone of software development, *Domain-Specific (Programming) Languages* (**DSL**), such as those described by Hudak (Hudak 1997), offer a number of benefits over general-purpose languages in particular problem spaces. The increased expressivity afforded through specialised and nuanced semantics and convenient syntax allows for the language to meet many requirements of the programmer whilst typically remaining easier to learn. Both Latex and SQL provide examples of prolific DSLs, for document layout and database query, respectively. Domains are a concept that features heavily throughout this thesis, in part due to its role in identifying problem spaces for programming but also because digital lutherie itself is composed of multiple domains which must be addressed. Hudak (1997) presents the advantages of DSLs as:

- being more concise
- quicker to write
- easier to maintain
- easier to reason about

When Kosar, Bohra, and Mernik (2016) reviewed the literature on DSLs, they found that research is focused on the technical methods of creating DSLs and that there is a lack of evaluation and study of integration. It is hard to deny the potential of a well-executed DSL in light of ubiquitous examples such as SQL[1]. Due to the design of a DSL requiring a deep understanding of both the target domain and programming language design and implementation, it is a complex topic that deserves research focused on technical methods. Music DSLs do provide a compelling

[1] https://en.wikipedia.org/wiki/SQL

perspective of DSL usage in fields such as live-coding (Blackwell and Collins 2005; Aaron and F. Blackwell 2013), with Max[2] and visual lanugages (Snape and Born 2022) and also languages such as Faust (Orlarey, Fober, and Letz 2009). We also see this area explored effectively in user interface design through examples such as that of Elm in the domain of web programming, where the Elm programing language heavily influenced the adoption of reactive programming (Wan, Taha, and Hudak 2002) in modern web frameworks (Czaplicki 2012).

2.1.8 Humans and Programming Languages

There was a time when using a computer necessitated being a programmer. Now the field of *Human-Computer Interaction* (HCI) is concerned with a nearly innumerable range of ways that a human may interact with a digital system. But for now, programming remains one of the most potent methods to interface with computers and so programming remains a significant facet of HCI research. Myers and Ko document the research landscape of early HCI study on programming as moving from areas such as pedagogy, graphical programming and program visualisation into a human-focused observation of software engineering, leading to the development of software engineering methodologies (Myers and Ko 2009). Glass describes the history of computing research as being divided into three categories; Computer Science, Software engineering and Information systems (Glass, Ramesh, and Vessey 2004). Research around programming languages fits into both computer science and software engineering fields both rich fields with many other components to consider. Early on, the software engineering field had been criticised for its immaturity as a research field and lack of rigour and for focusing on what was termed 'advocacy research' where researchers would use the broadly adopted term 'software crisis' as a crutch to advocate for their theoretical idea (Glass 1994). However, due to the worldwide relevance and importance of software engineering, this has largely improved (Glass, Vessey, and Ramesh 2002) though in such a complex environment, the vastness of the fields still leaves some areas under-researched.

Whilst Myers and Ko looked forward and advocated that the trends they describe all converge on a '... need for a better understanding of how to design and support programming', we see that even today, there is still a call to approach better how we research programming language design (Chasins, Glassman, and Sunshine 2021). Existing work on designing programming languages is fragmented and often fails to draw from the many related fields in order to provide a more holistic perspective. It therefore seems compelling that to effectively contribute to language design, both fields such as programming language theory and HCI need to be incorporated into existing research. We see examples in the ongoing evaluation of static type systems as a feature that is claimed to improve exploration of undocumented code (Mayer et al. 2012); however, studies find conflicting observations. Some document benefits of static typing (Hanenberg et al. 2014) and in some cases, even where dynamic typing should excel, they still suggest positive results (Okon and Hanenberg 2016). Meanwhile, other studies show no improvement from static typing when it would be expected (Harlin, Washizaki, and Fukazawa 2017). This is likely the result of a very nuanced and complex subject of study that is difficult to distil into self-contained and testable single experiments. Design patterns are a well-established idiom in programming in general, though studies have failed to demonstrate a compelling case for their claimed benefits (Zhang and Budgen 2012).

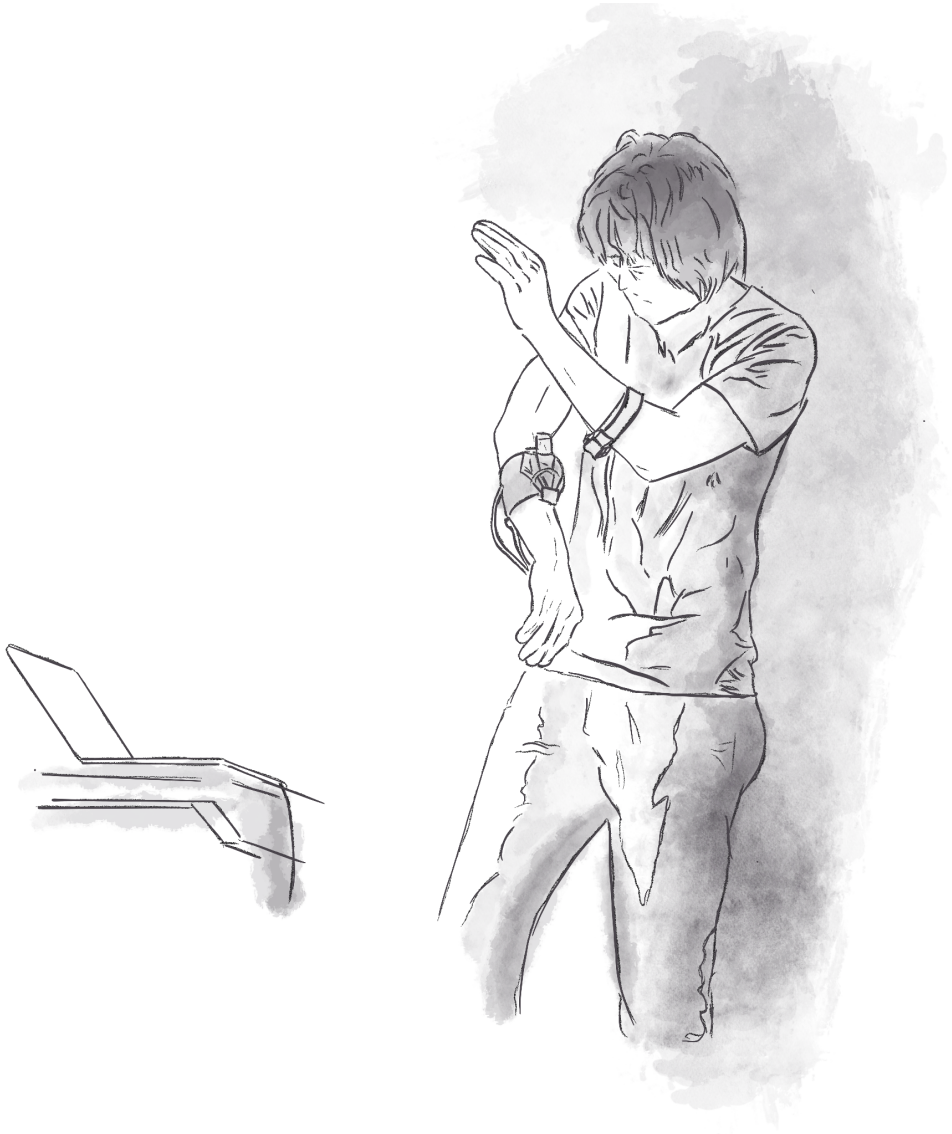
Empirical research through study and observation is an approach still being explored heavily, it is an approach that is criticised in Kaijanaho's thesis on evidence-based programming language design (Kaijanaho 2015) for not being rigorous enough and failing to isolate for control groups. Often it may be reasonable to consider this form of research a reiteration of 'advocacy research' as criticised by Glass until improved methodologies are introduced in empirical research for programming

[2]https://en.wikipedia.org/wiki/Max_(software)

languages. In their article 'PL and HCI: Better together', Chasins et al. discuss the importance of directly cross-pollinating ideas between the HCI research community and Programming languages (PL) community Chasins, Glassman, and Sunshine (2021). In particular, they describe how the design of programming languages cannot be a staged process where the PL researcher designs a language and then hands it off to a HCI researcher for a 'second pass'. They describe the need to meet the goals of both subfields by having '... HCI and PL expertise at the same table' which are faciliated by advances in both language engineering and both methodological and theoretical inovations in HCI.

2.1.9 Summary

To summarise, the programmer of today, no matter the field, requires first learning the domain-specific skills required to program before then being able to apply the many paradigms and idioms of programming for their use case. The relationship between programmer and programming language relationship has taken on the survival of the fittest strategy (Chatley, Donaldson, and Mycroft 2019), where languages develop to suit the needs of their community. While programming languages may have continually been developed with the intention of good human-centred design, it is only more recently that we have become equipped to better integrate the fields of HCI and PL design in order to empower the programmer in their application domain (Chasins, Glassman, and Sunshine 2021), improving the designer tool relationship.



2.2 The Performer

Despite the dominance of traditional instrument performance in mainstream music, digital musical instruments such as the Akai MPC and its derivatives have made a significant cultural impact and are perhaps the clearest case of DMI being used as a performance tool in widely consumed music. Of course, DMIs integrate with performance tools and music productions in many hybrid ways, through MIDI [3] controllers and samplers, for example, but also often as highly bespoke components of an artist's output. In light of Heidegger's philosophy of technology, Magnusson describes how musical instruments are technological tools that are tied to our development as humans Magnusson (2019) (page 8) where we encode knowledge in the technological artefacts we use and develop in conjunction with the technology we build. Magnusson describes how this tracks from the design of bone flutes that provide a functional memory of a discrete set of pitches through to that of computers situating the performance and creation of musical instruments at the forefront of our technological innovation and growth as a species. Performance remains a means of expression and communication of knowledge implicit in the instrument that is being performed. Having such a nuanced and interwoven relationship with music and technology, it is entirely appropriate that there is a well-developed research community that studies the relationship between performers and DMIs.

[3] https://en.wikipedia.org/wiki/MIDI

It is likely that the exploration of DMIs is most focused in the conference New Interfaces for Musical Expression (NIME), which initially started as a workshop at the ACM Conference on Human Factors in Computing Systems (CHI) but grew into an independent conference and community that incorporates researchers and notably also artists/musicians directly in its discourse (Morreale, McPherson, and Wanderley 2018). Of course, other collections such as the Computer Music Journal, Journal of the Audio Engineering Society and many other organisations also provide substantial contributions to the literature around DMIs. This surprisingly deep niche demonstrates the continued relevance of people's ability to express themselves using technological instruments.

Unlike traditional instruments, DMIs are unconstrained by their physical attributes. Sound generation and interaction with DMIs are typically considered separate concerns with a mapping relationship determining completing the dynamics of the system (Hunt, Wanderley, and Paradis 2002). For a DMI, form and function can be largely unrelated, and the performer of a digital musical instrument must develop sufficient control over an interface to express their ideas. This so-called 'gestural control' (Wanderley 2001), separated from the sound synthesis component of the DMI, means that the potential change in the mapping of parameters between gesture and the resultant sound complicates performers' mastery of a digital musical instrument. Also, the exponential explosion in possible gesture-to-sound generator mappings has developed a culture in which many DMIs are not deeply explored by performers to the point of reaching a high proficiency (Cannon and Favilla 2012). DMIs are often cited as requiring a low barrier to entry whilst maintaining the potential for virtuosity (Wessel and Wright 2002), a mastery over the gestures of their instrument and an innate intuition as to how this manifests as sound. However, some work on DMIs has considered virtuosity as a goal through a continuation of instruments as a method of storing knowledge in the instrument, where the DMI is described as a musically intelligent interactive system and incorporates machine-augmented instrumental technique. In Machover and Chung's early example, the Lisp programming language provides a means for encoding music theory and constraints into the performance of the instrument (Tod, Machover and Chung, J 1989). This included strategies such as grid quantisation and sequencing prerecorded sounds and patterns. This formed the notion of hyperinstruments, which explored the potential of augmenting traditional instruments through sensors to pursue new dimensions of virtuosity. This focus around virtuosity, however, is only a single dimension of DMI performance that, for many performers, is not the focus or potentially even possible. We see the concept of hyper instruments subverted by the description of infra-instruments, devices of 'restricted interactive potential' which allow performance to focus on other factors such as aesthetic, usability and other emergent properties (Bowers and Archer 2005). Given that, ultimately, performance is the most important evaluator of an instrument, it is critical to consider the role of the performer within the wider context of digital lutherie (O'Modhrain 2011).

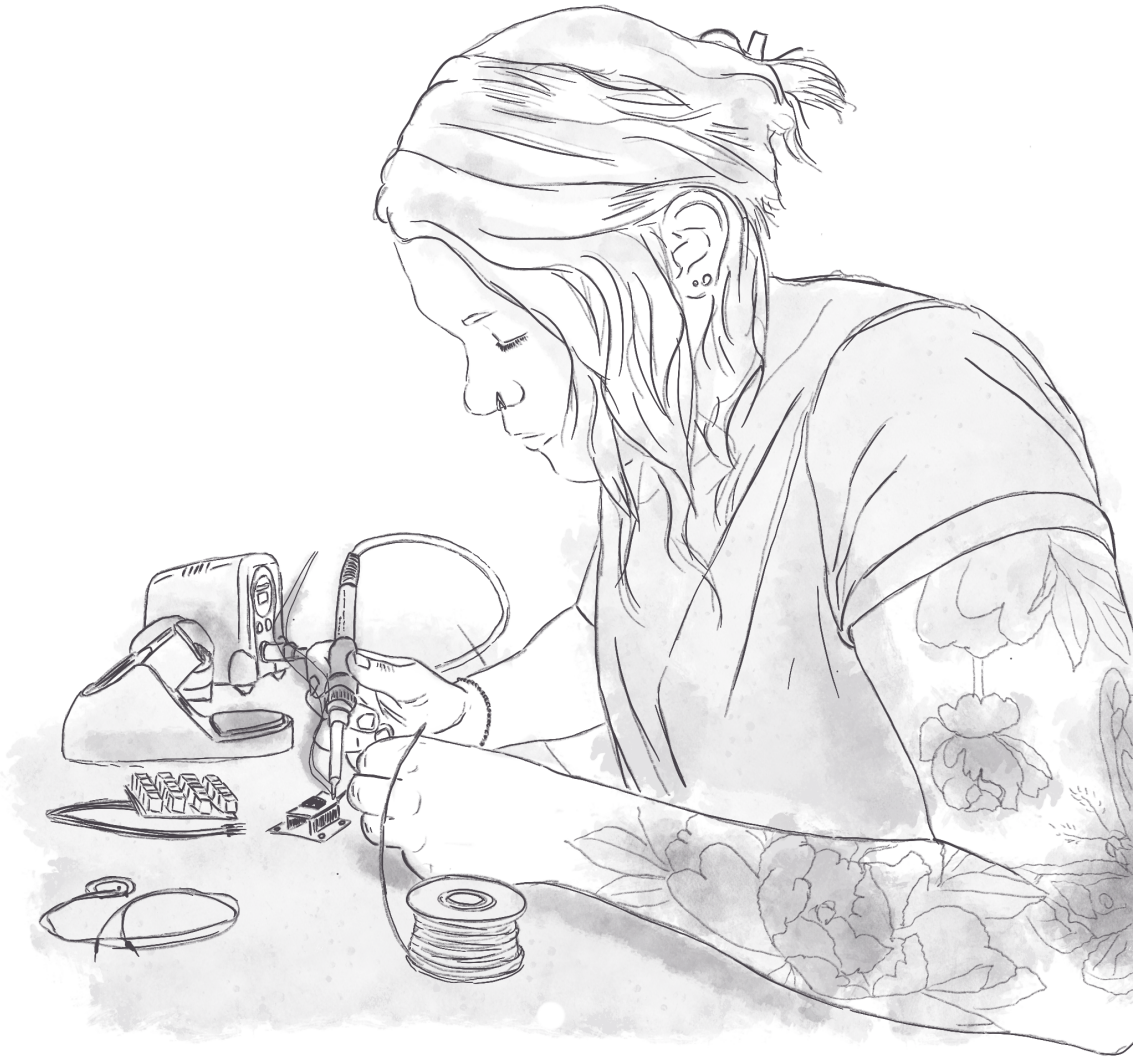
In contrast to traditional musical instruments is the capacity for both real-time and event-driven systems of performance, where the latter describes the control of the temporal evolution of the piece through discrete events that are triggered somewhat more analogous to the control a conductor has over instruments rather than the instrumentalist themselves (Wanderley 2001). This asynchronous approach to performance is further exemplified by instruments supporting fully deferred performance, where ideas are expressed before the performance to be integrated into live performance later. The sequential Drum provides an early example of this, allowing the drum-like interface to be performed with but also used to launch music expressed using the Music-N family of programming languages (Mathews and Abbott 1980). This kind of instrument has pushed the performance of DMIs into new territory beyond that of traditional music performance.

Given that music making and technology have always been deeply intertwined, it is no surprise to see that performing music using computers has a legacy that goes back to the formative years of computing, with the first of the MUSIC-N family of languages being used on the CSIRAC, one of

the earliest electronic stored-program computers ever developed (Annab 2021). Music DSLs have close ties with the very origins of computing, with the Music-N family of languages originated by Max Matthews at Bell Labs for the first mainframe computer to support floating point arithmetic (Roads and Mathews 1980). In the world of music technology, DSLs have a long history that tracks developments in computing. More contemporary music DSLs include the likes of Faust and Stride, but also an extensive range of others (McLean 2014; Magnusson 2010b), typically with ties to the live coding movement (Collins et al. 2003) where these languages are used for expression of music directly. Languages like Faust are particularly useful for digital lutherie as it is a DSL for signal processing, allowing the expressive description of efficient audio processing, which is far harder to achieve with low-level code. Magnusson says, "Anyone who speaks more than one language, in particular, if those languages are of different linguistic families, knows how differently each language portrays the world. A language is a world-view."

2.2.1 Summary

The role of digital lutherie is intrinsically tied to the role of performance, with digital luthiers uniquely different to performers of traditional instruments due to their duality of both designer and performer (Morreale, McPherson, and Wanderley 2018). Performance is considered the defining method of evaluation for a DMI; however, this can be measured along many dimensions. Virtuosity is a commonly sought-after goal in traditional musical performance, and whilst this pursuit continues in performance with DMIs, performers also take on many alternative directions of artist expression. This pursuit of alternative goals in DMI performance leads to the development of communities of practice that are highly specialised and drastically transform the notions of musical performance, even incorporating typically prepared processes such as programming and introducing it to a live context as a component of performance. This form of 'live coding'. It is apparent that the performer forms an essential facet of the archetype, the digital luthier. From here, the two key components of the digital luthier related to this work, the programmer and the performer, can be built upon to describe the digital luthier.



2.3 The Digital Luthier

Digital lutherie, a term coined by Jordà (2004b), refers to the specialised domain (Hirschfeld and Gelman 1994) and diverse community that is concerned with the creation of musical instruments featuring a digital component. Though the term digital musical instrument (DMI) is a common term used in music technology research, the term does not have a broadly accepted definition. Miranda and Wanderley (2006) suggest that a DMI is an instrument that uses 'computer-generated sound' and features a control surface to act on musical parameters in real-time. In this work, and informed our exploration of the idea in interviews featured in this work, we choose not to be prescriptive in defining the term. Within this work, we may use DMI to loosely refer to any entity that may be used for performance and that embraces the capabilities and attributes of digital systems in some way, in particular allowing for a capacity to be programmed whether ahead of time or in a continued fashion. However, throughout the study presented later in this work, participants have suggested many different definitions. While for some, defining DMIs may require some adherence to the use of discrete systems or of real-time performance capacity through discussion with digital luthiers and users of DMI, it is clear that exceptions are abundant and providing a concrete definition of the term actually has very little function in the wider sense of music creation. Creating digital instruments or interfaces capable of expressing musical intention is a process incorporating many disparate and specialist skills (Moro et al. 2016). In the pursuit of

this craft, the designer is required to use tools that extend beyond the traditional tools of a luthier (or any other traditional instrument builder), allowing the manipulation of digital technology as an additional medium (Lindell 2014).

At first glance, the motivation for creating an instrument suggests digital lutherie is concerned wholly with the artistic goal of the expression and making of music. There are, of course, many additional motivations for the design of DMIs, including as tools for research (Gurevich 2016), as a means to preserve and explore different mediums and also to support pedagogic efforts (Jack, Harrison, and McPherson 2020; Rossmy and Wiethoff 2019; Théberge 1997; McPherson, Morreale, and Harrison 2019). Given that its namesake is derived from the artisanal craft process of building stringed instruments, it is little surprise that digital lutherie, the process of building any form of DMI, has also been examined in the context of digital craft (Armitage and McPherson 2018). This perspective has extended to the study of digital practices more widely in the arts, where programming is also explored as a craft process (Blackwell 2018). This approach to viewing digital craft in processes such as digital lutherie opens the door for implicit, tacit and embodied experiences in a craft practice to contribute to a deeper exploration of these digital practices. Armitage, Morreale, and McPherson (2017) state that 'Though these ways of knowing are often personal, subjective and unverifiable, they enable the realisation of fine instruments.' Digital luthiers often embrace a multifaceted role in their craft process, blurring the boundary between designer, builder and player (Gurevich and Treviño 2017; Tahiroğlu 2021). The building and design process both involve a range of technical cross-disciplinary skills. Nevertheless, the challenges of digital lutherie do not end there. There are considerations in the design of new DMIs extending beyond the artefact itself and its use (Jordà 2004a). Numerous challenges related to the continued use and practice of the artefacts produced must also be considered (Morreale and McPherson 2017). Whilst this does raise the question concerning the importance of persistence in digital artefacts, all of these factors are important to consider in light of the intentions of the digital luthier (and further also the digital craftsperson more generally).

2.3.1 Situating Digital Lutherie as a Design Domain

Design is a field that naturally is called upon by many domains, from architecture to biology and beyond. Due to its application in such a vast range of fields, it is no surprise that research tends towards domain-specific studies of design. While efforts toward a generalized definition of domain-independent design have been attempted (Suh 2001; Suh 1998; Braha and Maimon 2011), these failed to effectively incorporate aspects such as the creative and innovative components of design; a characteristic attempted to be captured by C-K theory (Hatchuel and Weil 2003). Despite the attempts of C-K theory, much of the research design community have been more divided than unified by such theories, as highlighted by Dorst (University of Technology Sydney and Eindhoven University of Technology and Dorst 2016). Dorst, too, looks to strategies to bring together independent fields of design with an approach that seeks to reconcile design practice and research. Rather than a single generalized framework, Dorst advocates for recognizing that design research exists as a discussion between dynamically interrelated fields and suggests that they should build bridges between them when appropriate to create a richer discussion.

The design process of creative technologists has primarily been researched in the context of digital craft, which focuses on expressivity, allowing individual mastery over the medium with which they work (Jacobs et al. 2016). Much in line with Dorst's suggestions, combinations of craft and technology are being explored in considerable depth in areas such as DMI design (Armitage and McPherson 2018) and eTextiles (Posch and Fitzpatrick 2021). This introduces a focus on the capacity for the craftsperson to achieve an ever more comprehensive and demanding set of engineering challenges whilst retaining the critical component of design that is a capacity for cre-

ativity (Fischer et al. 2005). Fischer describes the requirement for social and individual creativity in design as a spectrum that ideally depends on cross-pollination varieties of social and individual ideas, facilitated through the environment in which they interact.

Frankjaer and Dalsgaard observe that craft-based practices can address many outputs and processes (Frankjær and Dalsgaard 2018). Examples given include the digitally assisted design of physical artefacts, computational physical artefacts and materials, digital artefacts such as code, merging digital and physical media and practices, and artefacts emerging from within Maker and DIY culture. They further observe that the ambiguity in defining a 'craft process' presents challenges in addressing knowledge creation that generalises to all craft processes. This ambiguity tends to also be prevalent within the specific domains in which we see digital craft, such as in providing a firm taxonomy of artefacts in the DMI community (Tanaka 2010). Ultimately, Frankjaer and Dalsgaard observe three approaches to the scientific inquiry of craft practices. They state these as:

1. "Combining, aligning, and integrating analogue and digital crafting techniques and processes"
2. "Creating highly refined artefacts, defined by attention to detail and aesthetics"
3. "Creating knowledge through deep, embodied engagement."

They define these to encourage researchers to engage in craft processes in a tacit and embodied manner, engaging with the knowledge in a practice-based manner. This emphasis is due to the narrowing of experiential knowledge as it is transferred into the written form that constitutes typical scientific literature, a perspective shared by University of Technology Sydney and Eindhoven University of Technology and Dorst (2016). As such, the developing approach to analyse the design process of practitioners incorporates strategies such as workshops (Posch and Fitzpatrick 2021; Lepri and McPherson 2019), interviews with expert practitioners (Stolterman and Pierce 2012) and qualitative analysis of data derived from the first-hand experience.

The maker movement has profoundly lowered technological barriers, democratising and opening up access to technology (Tanenbaum et al. 2013). Increased quality and availability of 3D modelling software and manufacturing methods drive the production and further development of more traditional instruments (Zoran 2011; Dabin et al. 2016; Kantaros and Diegel 2018). Entire hardware platforms dedicated to supporting the design intentions of digital luthiers also situate high performance embedded computing within the community (McPherson, Jack, and Moro 2016; Madgwick and Mitchell 2013; Turchet and Fischione 2021). This enables the realization of many forms of instrument design, from hybrid instruments (Tod, Machover and Chung, J 1989), to entirely novel instruments.

2.3.2 Design of Digital Musical Instruments

Designing a DMI, or indeed any human-used artefact, is not just a physical endeavour but an exercise in psychology. This is particularly relevant for DMIs with the goal of effectively facilitating musical expression. Drawing originally from ecological psychology and Gibson's term 'affordances' (Gibson 2014), Gaver Gaver (1991) introduces 'perceptual affordances' as a concept for describing human interaction with technology, suggesting that we explore interfaces through the interaction of the affordances they provide. By considering a door handle, a person may see that the shape affords the opportunity to grip the handle and through the extension of some tactile exploration, turned, pushed or pulled, somewhat intuitively leading to the action of opening the door through the affordances offered by the handle. DMI design through the affordances presented to the performer has been widely explored (Tanaka 2010; Marshall and Wanderley 2006; Silva

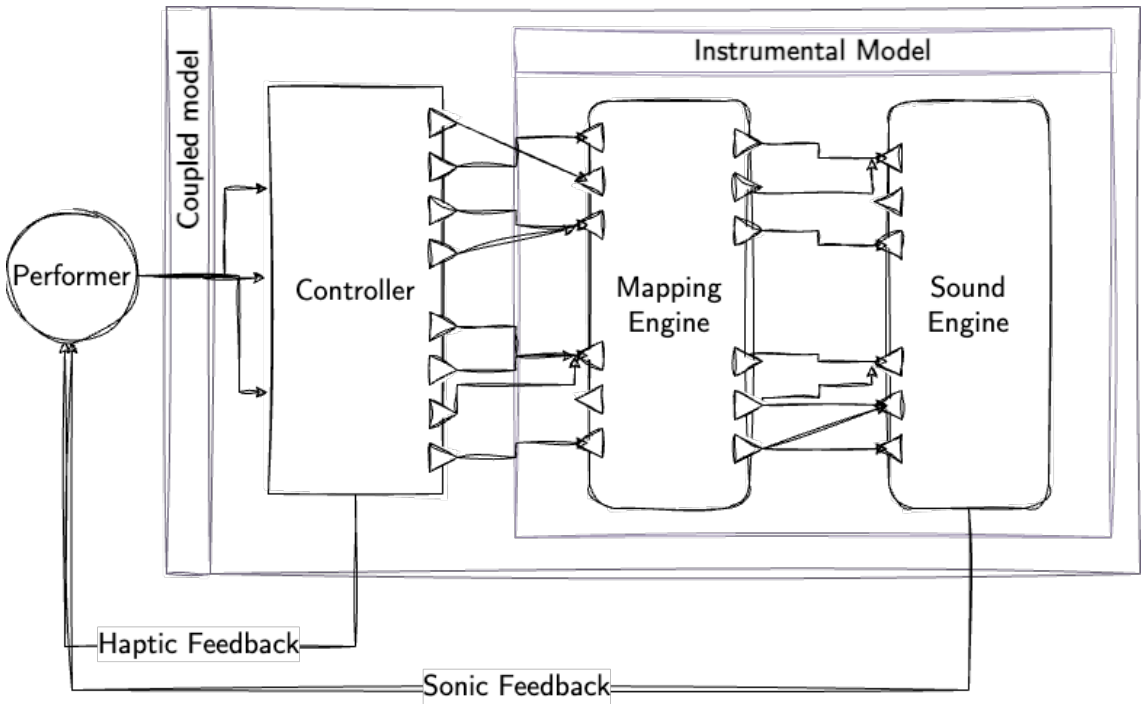


Figure 2.1: The widely described instrumental model (Magnusson 2010a; Wessel and Wright 2002; Wanderley 2001)

et al. 2013; Kaltenbrunner et al. 2006) and emerged as a common method for designing and evaluating DMIs. In DMI design, Wessel and Wright's description of a ' low entry fee with no ceiling on virtuosity' is commonly viewed as the ultimate goal in creating an expressive DMI (Wessel and Wright 2002). Cannon and Favilla examine the facilitation of virtuosity in DMIs through affordances offered to the performer. They introduce a concept they term 'investment of play' (Cannon and Favilla 2012). This process is initiated through the exploration of the affordances of the instrument and then develops until the performer's proficiency facilitates expression. The mechanism for this is described as a cognitive shift from the operation of the control interface to the conception of the abstract, performed sonic material. Their study couples the increase in subjective expressivity with the affordances offered by a system to provide considerations for frameworks that address DMI design. In contrast to evaluating DMIs through their affordances, Thor Magnusson proposes that constraints are the defining characteristics in composing and performing with digital musical instruments and that designing DMI can be viewed as designing these constraints (Magnusson 2010a). Magnusson suggests that in complex digital systems, affordances can be imperceptible. This is a factor acknowledged by Gaver as a facet of perceived affordances, potentially including hidden or false affordances (Gaver 1991). Having considered other models of constraints (Norman 1999; Pearce and Wiggins 2002), Magnusson draws a new model of constraints for the context of DMI design. He defines subjective, objective and cultural constraints. Subjective constraints are described as the limitations on the thinking, creative person's expressivity, formed as habituated traits of a musical tradition and its practices. Objective constraints are physical limits on the environment and tools. Finally, cultural constraints are described as "...conditions in which technology and ideas exist". With these constraints formed, Magnusson presents a variation on the model of a musical interface as described by Wanderley (2001), as well as Wessel and Wright (2002), the instrumental model is shown as a distinct sub-component of the overall coupled model, consisting of the mapping engine and sound engine. The controller is captured in the coupled model, denoting a distinction between the core sound generation of the instrument and how the instrument is controlled.

To Magnusson, the observation that the mapping engine and sound engine are captured as a singular perceived system (the instrumental model) matches his model of constraints. The constraints that a user interacts with are found in this instrumental model, which they interact with via the controller, which is perhaps more loosely coupled, allowing it to be changed for a different controller that realises the same mapping interface.

Having described some of the physical and practical implications of DMI design, the cultural and social implications on DMI design warrant an independent and introspective view of how it affects DMIs. It is widely observed that musical tools and software are fraught with cultural and social bias. Looking again to Gaver's perceptual affordances, Gaver recognises the social/cultural implications of affordances. Magnusson, in his model of constraints also recognises this stating 'The digital instrument is an artefact primarily based on rational foundations, and, as a tool yielding hermeneutic relations, it is characterised by its origins in a specific culture' (Magnusson 2009). Magnusson draws attention to the importance of instrument designers considering the effect these cultural assumptions have due to DMIs potentially having more symbolic and compositional influence than our existing physical tools, primarily because the software itself has agency and inherently imposes greater specifications than traditional physical instruments. To contextualise this further, Puckette acknowledges that despite the intentions to remove semblances of western musical notation from Max (staves, bars, time signatures and even notes), that even the blank page that greets the user bears the connotation of paper pages, a notion implicit in western musical culture (Puckette 2002). Puckette expresses interest in seeing how these fundamental assumptions can be further discovered and then 'peeled away'. Overall, the literature forms a view where initially interacting with a DMI is focused on exploring the affordances, but as this develops through invested play learning the instrument is more about forming an innate mental model of the constraints. This ultimately emphasises two well-researched areas of DMI design. The presentation and description of affordances and constraints, and the relationship between different implementation domains of a DMI; the controller, the mapping engine, and the sound engine.

2.3.3 Domains and the problem space of digital lutherie

Certainly, the domains required by the digital luthier are vast and at times disparate. Puckette observes, '. . . computer music software most often arises as a result of interactions between artists and software writers. (occasionally embodied in the same person. . .)'. Today, a better understanding of artist programmers has formed (McLean 2011) and it can be assumed that through the proliferation of technology, this trend will grow. Of course, digital lutherie incorporates elements of computer science, software, electronics and physical design, merging many independent domains, that stretch the capabilities of even the polymath that is an artist programmer.

In his talk at the 2017 Audio Developers Conference, Zicarelli describes the desire to work in the problem space rather than in code (David Zicarelli 2017). In essence, Zicarelli advocates for remaining in the problem space when designing sound. That is to say, work with a graph representing an audio signal and the parameters that are applied to it, rather than considering how to manipulate the computing constructs (audio buffers, pointers and memory) that are used to implement the desired signal processing. Rather than straddling multiple problem domains simultaneously, the designer can focus on the problem space they are approaching and a simplified mapping stage to the device they are using. We might consider this in the context of the design problem space as reducing the number of sub-components being incorporated into the problem, effectively simplifying the required solution. Modern programming language design and their implementations are responsible in a large part for removing the burden of implementation details from programmers as discussed previously.

Programming DMIs has many performance requirements and implications (McPherson, Jack,

and Moro 2016; Jack et al. 2018) and while we might not supose that DMIs are 'mission critical' systems in the way that a car safety system is (requiring formal verification of code for example), it is still unacceptable to have failures and glitches in a performance, and many of the approaches and innovations made in mission critical programming are being absorbed into the ecosystem of digital lutherie (Turchet and Fischione 2021).

While many typical hard realtime systems however are purely utilitarian, for DMIs the greatest capacity for expression and creativity are demanded, with users desiring that their instruments inspire them. These two sides alone begin to show the crossdisciplinary nature of DMI design with the role of the programmer covering the domain of design and HCI as well as necessitating the rigoured control of a well-seasoned engineer that can extract all the run-time performance their platform has to offer. This typically necessitates a vast range of specialism and naturally, through the leveraging of social creativity community-focused projects to bring the more challenging engineering tasks have introduced strong platforms to address the performance engineering challenges of the audio domain (Moro et al. 2016; Turchet and Fischione 2021; McPherson 2017; McPherson 2017). At the other end of the spectrum, to address the highly nuanced and expressive power, we also see the use of domain-specific programming languages used to limit the scope of the problem space for the designer. Faust demonstrates one of the most compelling examples of a DSL for audio processing that can be used to program DMIs (Orlarey, Fober, and Letz 2009; Michon et al. 2020a). Other relevant DSLs used in DMIs include Supercollider and to a lesser extent, newer languages such as Stride (Tilbian and Cabrera 2017; Tilbian et al. 2017) and Chuck (Wang, Cook, and Salazar 2015). These examples demonstrate a growing ecosystem and potential for DSLs to be used in digital lutherie however, it is only graphical programming languages Max and Pure Data that have truely taken a share of developers from using general purpose languages (Puckette 2002).

2.3.4 Tools For DMI design

Whilst the relationship between performer and their tools for performance is studied extensively, the relationship between the digital luthier and their tools is typically studied from the performer's perspective (Blackwell and Collins 2005; Magnusson and Mendieta 2007) or otherwise focuses on the luthier's processes and intended outcomes (McPherson and Lepri 2020; Cook 2001; Magnusson 2006; Armitage and McPherson 2019). This is perhaps owing to the multifaceted role of the digital luthier, often performing the tasks of a designer, builder and performer (Jordà 2005). It is then true that, as Cheatle and Jackson put it, artists "...act as creative and critical users of tools – both computational and otherwise – whose practice has the potential to reveal new insights and understandings about the world in which we live..." (Cheatle and Jackson 2015).

To design a DMI, an instrument designer can no longer consider only how they will form the physical instrument itself, but rather, they must consider both the physical hardware elements, along with the software and algorithms that take the place of physics for interaction and sound generation. The learned skills and intuitions of the traditional instrument builder that manipulates materials, gives way to the electrical engineer and the programmer. These skills appear further removed from musicality. As Magnusson phrases it "*Code as material is not musical; it does not vibrate; it is merely a set of instructions turned into binary information converted to an analogue electronic current in the computer's soundcard.*" Magnusson (2009). The materials of DMI design become silicone chips, circuitry and the more complex, technological artefacts built from them. The tools become programming languages, operating systems and design patterns and the suggestion is that this removes the intuitive sense of musicality from this design process. Software tools for DMI creation exist at a number of levels of abstraction. User-centred software such as digital audio workstations or VST plugins capture sound or instruments at a level that closely resembles the domain of music. Highly configurable music programming environments, such as Cycling74's Max,

look to provide a near-endless supply of programmatic potential, whilst remaining closely focused on the problem domain. At the very least, the Max family of languages decouple the engineering of the underlying implementation and the use of musical constructs they create, though there exist limitations in this approach. Software frameworks and libraries tend to offer a more powerful level of control for the creation of DMIs, but are closest to the implementation of digital systems and as such, present the same challenges typically associated with programming computers. Limitations in designing digital instruments quickly arise when considering the use of user-oriented software. Through the combination of standalone controllers and virtual musical instruments running on a computer (Mulder 1994), instruments that match the model initially seen in Figure 2.1 can be produced, offering a vast world of sound synthesis potential. These instruments tend to be a passive controller such as a MIDI keyboard tethered to a laptop, however, for many this is an undesirable and cumbersome characteristic. Whilst having a full personal computer constitute a component of a DMI does fit some models of a 'digital instrument', in reality, it is not ideal in a practical sense, particularly for practice or performance unhindered by a complex setup phase. Further, the reliance on a personal computer brings with it email, a browser and perhaps a semblance of work that performers do not want in their instruments. For this reason, DMIs are often based on the use of embedded computers and microprocessors which can be used to create a singular artefact. With the rise of the 'Maker' scene, a culture of 'do-it-yourself' (Vallgårda and Fernaeus 2015) and knowledge exchange, a number of technology platforms and support for them have emerged as the technology has developed, increasing accessibility and decentralisation (Kuznetsov and Paulos 2010). Perhaps the greatest example of this is the Linux OS (Torvalds 1999).In contrast to an engineering team with a complementary set of skills that solve DMI creation as a broader engineering task, open-source software must be considered the single digital luthiers greatest asset (Fitzgerald 2006). DMI design has largely taken the form of leveraging the Linux environment, providing the opportunity to use OS-dependent high-level programming environments such as Pure data (Puckette 1996) to describe audio processes. By adding appropriate audio I/O and general-purpose I/O, the Bela platform sets the bar for creating a comprehensive embedded Linux platform for creating DMIs (Moro et al. 2016). Bela meets the challenging real-time requirements of a DMI (McPherson, Jack, and Moro 2016) and reduces friction in the design of instruments through excellent support, tooling and integration with other platforms (Morreale et al. 2017). Despite these benefits, as Michon et al. (2020b) observe, there are also limitations associated with this OS-dependent approach. Namely the increased costs, complexity and reduced efficiency for both hardware and software. Whilst Bela provides an effective solution for many DMI design scenarios, there are cases when an embedded system is more appropriate (Chowdhury 2020). Due to resource constraints (memory and clock speed), embedded systems can be a challenging environment to work in. Michon et al. (2019) provide a compelling demonstration of how effective programming for microcontrollers can be achieved using Faust, where environments such as that of Faust (Orlarey, Fober, and Letz 2009), Stride (Tilbian et al. 2017) and the ubiquitous open-source microcontroller Arduino (Bianchi and Queiroz 2013), demonstrate the relevance of embedded systems in creating instruments.

The understanding and appreciation of musical context and culture and how they influence design are given due attention (Lepri and McPherson 2019) and the open-source community around DMI design provides a unique and notable observation of social influences on design (Morreale et al. 2017). This work demonstrates the complex landscape of the designer-tool relationship and suggests many contributing factors at play. However, less work has been done to look at these relationships from the perspective of contemporary practitioners of digital lutherie, despite support for their apparent influence (McPherson and Lepri 2020).

In the wider human computer interaction (HCI) community, we see Stolterman and Pierce examine the designer-tool relationship more directly (Stolterman and Pierce 2012). They suggest that, in the HCI community, there is a tendency to focus on end-users (user-centred design).

Similarly, in the digital lutherie community, considerable attention is given to the processes and use of tools (Inie and Dalsgaard 2017; Frich et al. 2019; Koch et al. 2020). Research on the motivations for choosing tools remains sparse. As such, this work looks to contribute to Stolterman and Pierce's suggestion that *"... there is a need for more developed understandings of how practicing designers use, understand, and interact with their tools"* by exploring this designer/tool relationship as a means to better understand the design practice (Goodman, Stolterman, and Wakkary 2011). This thesis examines a group of digital luthiers, with differing perspectives, backgrounds and motivations and seeks to understand the challenges they face and the tools they use to overcome these challenges. This includes meeting the performance demands of real-time systems (McPherson, Jack, and Moro 2016; Jack et al. 2018) and the complex interaction goals (Wessel and Wright 2002) amongst many other individual challenges.

Stolterman and Pierce study the relationship between tool and designer concerning interaction design (Stolterman and Pierce 2012), noting the nearly infinite combination of tools that the designer may pick to support their approach. While superficially, characteristics such as 'efficiency' or 'ease of use' may motivate selecting a specific tool, they contend that the reality is more complicated and involves the social, cultural and material contexts in which design occurs. They look to Argyris' theory of action (Argyris and Schön 1974) as an explanation whereby the idealized way the designer wishes to approach the problem contrasts reality.

In the context of digital lutherie, we see many interesting relationships between designers and their tools. If we look to the field of 'live coding', where performers use programming languages to manipulate audio and music in real-time (Collins et al. 2003), we can see a tendency of performers to write their own programming languages (McCartney 2002; Wang, Cook, and Salazar 2015; McLean 2014; Magnusson 2010b; Bovermann and Griffiths 2014). Within this community, this is recognised to the extent that work is actively exploring the facilitation of creating new languages (Bernardo, Kiefer, and Magnusson 2020). We also see a similar niche fulfilled using machine learning (Fiebrink and Cook 2010), with both cases deferring some design component to the performer. These all represent individual components of end-user development (Fischer 2021), which empowers users to continue the develop technology through a variety of means. The concept of meta-design is presented as a socio-technical framework to address the nature for digital systems to interface with the real world, facilitating users' needs in real-world conditions which are unpredictable and require improvisation, evolution, and innovation (Fischer and Scharff 2000). It is common to see strategies involved in end-user development and meta-design present and deliberately employed in DMIs. Through work in this thesis, this connection is both demonstrated and built upon to suggest that for the digital luthier, the capacity for end-user development may in fact be critical, as in many cases for expressive musical interfaces, some component of the the specification is deferred to the user. This deferred design component may be a controller mapping to synthesizer parameters, for example, which can be approached in a variety of ways (Laguna and Fiebrink 2014).

Ultimately, the role of a digital luthier provides a rich insight into digital craft, with well-established communities, research and technological ecosystem.

2.3.5 Summary

The notion that combining both instrument maker and performer is a key point of the definition of digital lutheir by Jordà (2004b), and is corroborated by studies which in a survey demonstrated exceedingly high percentages of performers of NIME's who were involved in instrument creation (Morreale, McPherson, and Wanderley 2018). McPherson and Tahiroğlu examine many of the languages and frameworks for the development of DMIs, observing this potential influence on the instruments they create (McPherson and Tahıroğlu 2020). Their study suggests a bi-directional relationship between the DMI designer and the instrument, with these influences impacting how

designers select their tools. As the set of tools and technology accessible to the digital luthier develops, the relationship between them and their tools is of growing relevance and interest to the research community. As programming remains a significant aspect of digital lutherie, there are many developments in the wider research on programming languages that stand to offer benefits to the digital luthier both in the expressivity of the code they write and ensuring that code is correct and less error-prone.

Chapter 3

Exploring Languages for Digital Lutherie

At the outset of this journey, the intention was to build a new programming language for digital lutherie. This led to a series of conceptual programming languages that targeted domain-specific problems and looked to pair programming features or paradigms to target the domain of each problem, with the potential to isolate and compose domains to build up a complete language that was domain-oriented and potentially visible more as a collection of miniature DSLs, rather than one langauge. The proposal for this design could be described as a system of small languages, with interfaces defined for composition of the languages depending on the target instrument.

In order to rationalise the design choices of a language such as this, it became a consideration that design choices needed to be well motivated and testable and as such, these explorations developed to prioritise asking the question of how one would go about designing a new language for digital lutherie. This chapter will document some of these explorations in programming language design, primarily to provide a context and motivation for answering these questions but also to demonstrate some of the novel ideas that developed in this line of thinking. What is presented here are the more relevant results of exploratory work over the period of 2 years, considering the design of musical programming languages. They are presented chronologically and demonstrate the development of structures for the modelling of musical systems in programming languages, first in the description of tuning systems and then expressing rhythmic structures. These ideas are then explored in the context of mapping to digital instruments, which remains an ongoing work but was deferred in order to better inform the design through a better understanding of the needs of the digital luthier.

3.1 Exploring Programming Idioms for Expressing Tuning Systems

Despite the many approaches to DMI design and music creation through programming langauges, general purpose (GP) languages still dominate the development of DMI. As discussed in Section 2.1.5, these GP languages tend to inherit programming language features as language communities converge on shared idioms for solving particular problems. This section explores the programming construct of 'list comprehensions', derrived from set builder notation [1], and demonstrates them as a highly expressive and idiomatic way to work with musical tuning systems, which could be useful in defining tunings, and by extension, scales for Digital Musical Instruments.

[1] https://en.wikipedia.org/wiki/Set-builder_notation

3.1.1 Introduction

Many musical traditions settled on the compromise of twelve-tone equal temperament (12-TET) and it has since become the de facto standard for composition and musical instrument design (Duffin 2008).

Many instruments are physically bound by an inability to tune or modulate between just intonations in a practical setting, and as such, the use of this form of harmony is constrained for most ensembles. The legacy of these physical restrictions means it is uncommon for extensive support for alternative temperaments with discrete-pitched instruments, digital controllers or software instruments. As such, composers typically meet many challenges in exploring this form of harmony in a manner that is intuitive or supports rapid, iterative experimentation. Furthermore, with 12-TET deeply ingrained in the harmony of many musical cultures, many theoretically acceptable harmonic relations no longer have the same effective perceived quality in an alternative temperament.

For the most part, the details of tuning and temperament remain a hidden layer upon which music systems sit. Given the mathematical foundations on which alternative systems of temperament are built and the ease these calculations can be performed on modern computational systems, we can expect that there is a way to describe different tunings in a way that is both clear and allows for accessible, creative exploration.

As the concept of a scale or tuning naturally fits the data structure of a list, a functional language such as Haskell (Marlow 2010) provides an excellent set of tools and concepts for programmatically manipulating these ideas.

This section aims to examine the potential ways for existing tunings to be expressed and, beyond that, how functional programming may present a foundation for exploring new tunings and, by extension, harmony. The results allow scales to be created from a base tuning and ultimately applied within digital instruments, laying the foundations for a reimagining of temperament in the context of digital instrument design and domain-specific languages.

3.1.2 Tuning and Temperament

There are a number of ways to describe the relationship between notes. Due to the ubiquity of twelve-tone equal temperament (12TET), the most common unit tends to be cents, a logarithmic unit adopted and developed by Alexander J. Ellis for his work comparing tunings from around the world (Ellis 1885) that uses 12 TET cornerstone. This unit sees an octave within the 12-TET system divided into a geometric sequence of 12 divisions, each equal to 100 cents, referred to as a semitone. This approach is widely adopted in music technology, such as by the MIDI specification. The MIDI specification and the majority of compatible synthesisers adopt this model to describe alternate temperaments as a deviation (in cents) from 12TET.

As this approach requires making adjustments to each note, tuning becomes a time-consuming exercise that raises the potential for mistakes and requires a precalculated target value of what frequencies to select, stifling the ability to make intuitive decisions about the tunings or to iterate and make changes in an explorative space quickly. Largely, the exploration of tunings on many modern instruments couples the domain of musical tuning with the technical domain of configuring the instrument. Due to the exponential nature and unfamiliar approach to discussing frequency in a musical context, the instrumentalist or composer is left to solve tuning as a technical problem that does not relate directly to their musical intentions.

An alternative approach defines tunings as a ratio to the first scale degree, as shown in table 3.1. Whilst this method lacks the anchor point of 12TET deviation, it gives a far more elegant representation of the scale. Just intonations, in particular, are represented clearly in this way due to their inherent use of natural numbers for ratios.

Table 3.1: Pythagorean tuning

Note	*G♭*	*D♭*	*A♭*	*E♭*	*B♭*	*F*	*C*	*G*	*D*	*A*	*E*	*B*	*F#*
Ratio	1024:729	256:243	128:81	32:27	16:9	4:3	1:1	3:2	9:8	27:16	81:64	243:128	729:512
Cents	588	90	792	294	996	498	0	702	204	906	408	1110	612

Equal Temperament

Equal temperament creates a perceptually consistent width interval [2] between each note. This creates a harmonic compromise with partials from each note close enough to integer multiples to be perceived as harmonic, without creating larger gaps that are perceived as inharmonic[3] at other points in the chromatic scale.

This tempering system works by dividing an interval range (typically an octave) into equal divisions. In Western traditional theory, this is typically twelve subdivisions following the formula:

$$\sqrt[12]{2} = 2^{(1/12)} \approx 1.05946309436$$

It is possible to divide the octave into more than twelve notes, and this is the most typical choice for exploring microtonal music in the 21st century. Again temperament, by its compromise, facilitates the use of the full scale and, therefore, key changes for physical instruments. The concept of tempering a scale is a compromise in the harmonic accuracy of all scales, such that they are all equally usable without having to retune the instrument. Whilst equal temperament has been the dominant and widely adopted tuning system for much of the music of the world, there is still debate around alternatives that DMI provide the perfect platform to explore (Hinrichsen 2016).

Just Intonation

Just intonation refers to tunings where the fundamental frequency of each note is related as an integer ratio to some common reference pitch. This concept creates a set of harmonics that align in an audibly consonant way and is, by strict definition, what is considered the most 'in tune'.

As this series of notes is built on integer ratios, this relationship only holds for the current key[4]. Just intonation is more accurately conceptualised as a collection of relationships rather than a single rigid tuning. There are variations such as *Pythagorean Tuning* (table 3.1) and *Five-limit tuning*, handling different scale degrees with different ratios, creating subtly different qualities.

3.1.3 Current interactions with instrument tuning

Currently, there are a number of synths that support the ability to create alternate tunings.

Whilst there is support for altered tunings within the MIDI standard (via SysEx message), it is the accessibility and quality of digital instruments that present the best way to overcome the physical tuning limitations of traditional instruments.

The major limitation in tuning arises from the limited ways to derive, experiment and configure tunings rapidly enough to fulfil creative needs. Tuning by deviation from 12TET has been used by the likes of Terry Riley in "Songs For The Ten Voices Of The Two Prophets" (Riley 1983) to explore tuning in a more contemporary context, using Prophet 5 Synthesisers. Riley's work is an experimental piece that goes to deliberate efforts to explore the harmony not typically associated with keyboard instruments. It is an excellent example of what can be achieved, but generally, the process is demanding on the composer's technical capacity in overcoming the conventions of typical keyboard controllers and in finding support for arbitrary tuning systems before they even

[2] Equal to 100 cents

[3] Referred to as wolf note - the limiting factor in key modulation when using just intonated tunings.

[4] Even then, for some intonations, more than 12 notes are required, meaning enharmonic notes that are not equivalent.

begin to explore the expanded potential space for composition beyond 12 TET tuning systems. We see work that aims to make this task more accessible, for example through digital fabrication methods and modelling microtonal tunings for 3D printed flutes (Dabin et al. 2016).

Further, Hayward (2015) presents an interface to describe just intonations as a lattice. This demonstrates the use of visualising abstract relationships in just tunings, thereby demonstrating an opportunity to explore them creatively. However, their work focuses exclusively on just intonations, which presents as more of an analysis tool than a tool that can be used directly for practical applications or integrations to physical DMIs. There is also work that explores the opportunity for dynamic tuning. For example, Milne presents an isomorphic controller that facilitates tuning adjustments during performance (Milne, Sethares, and Plamondon 2007). Milne's work certainly allows for tonal exploration however, the price for this power is a technically challenging controller, which may not meet the coveted 'low barrier to entry'.

3.1.4 Expressing Temperament

Many functional programming languages have an idiomatic and highly ergonomic approach to working with lists. Haskell, thanks to higher-order functions, has the ability to perform a number of generalised actions over lists, such as; mapping, zipping and filtering. These can allow a programmer to quickly but also arbitrarily describe and interact with lists representing tunings. This is highly beneficial as results can be computed and applied far faster than manually tuning each note, helping encourage the iterative, conductive workflow for creative and intuitive exploration.

Given the fact that there are a number of ways to describe a tuning, a generic tool such as a programming language actually offers the designer the opportunity to approach tuning an instrument from a number of perspectives. Whereas for many instruments (in particular keyboard instruments), tuning is bound to at least a variation on 12TET, generating lists through a series of expressions or application of a set of helper functions allows more compatible and unified access to other tuning systems that depend on other intonations or divisions of the octave, for example.

Formulaic Expression

For generating equal temperaments, creating an expression that divides the desired interval is a simple and effective method for creating a tuning.

Equal temperament takes the interval of an octave and splits into twelve perceptually equal parts.

This takes the form:

$$r = \sqrt[n]{p}$$

Where r is the ratio, p is the interval and n is the number of divisions.

This relationship can be used to create an expression that gives the frequency of a note given a scale degree d and a reference pitch R:

$$2^{(d/n)}R$$

To this extent, exploring equal tempered scales should be trivially simple as variations of this formula. Typical microtonal music can then be explored given any expression where $n > 12$.

Ratios and Lists

It is well recognised that ratios are a powerful tool for describing tunings, a good example being Wright's book on the mathematics of music (Wright 2009).

Whilst for equal temperament, an expression is an effective and simple way of expressing a scale, due to the variation in interval size, this is not so simple to work with for just intonations.

An alternative to formulaic expression is to describe a scale by its intervallic spelling, that is a list where each interval that constructs the scale is explicitly described as a ratio.

$$A = 1024/729, 256/243, 128/243...242/128, 729/512$$

If this list[5] is evaluated what remains is a set of coefficients that can be multiplied with a reference frequency R to calculate the tuning.

$$f = RA_n$$

This presents a very intuitive way of experimenting with very strong harmonic tunings in a way that suits the human inclination for seeing patterns.

12TET deviation

Whilst we consider deviation from 12TET to be a limiting method for tuning, as this approach is general, it is easy to include it, for both completeness and familiarity. Below n is equal to the number of cents up from the reference pitch.

$$f = R(2^1/1200)^n$$

3.1.5 Practical Examples

Given its light, mathematical syntax, these formulas translate conveniently into functional languages such as Haskell or languages that inherit these idioms (such as Python).

We present several very simple implementations here based upon Haskell's list comprehensions, where given a list of the desired scale degrees, a list of frequencies are returned.

A list comprehension is a powerful programming construct, found in languages such as NPL, Miranda, and Haskell, that enables the concise and expressive construction and transformation of lists (Turner 1986).

A list comprehension can replicate the formula for generating an equal tempered scale of twelve chromatic notes with the expression below. The named constants make it a simple change in order to recreate the scale from a different reference point, for example, to tune to the popular alternative reference pitch like $A = 432Hz$.

```
referencePitch = 432
freq_12tet = [2.0 ** (n/12.0) * referencePitch | n <- [0 .. 13]]
```

In the same form, a Pythagorean circle of fifths can also be generated using a list comprehension.

```
freq_PT = [referencePitch * (3/2) ** n | n <- [0..12]]
```

To create a scale using an intervallic spelling, a list of ratios can be used. This is then evaluated to be used in the list comprehension, giving a similar overall feel. The resulting list of coefficients can then be used either in another list comprehension or using a function such as filter, to select only notes from a scale.

To further demonstrate this strategy of list manipulation an auxiliary function is also demonstrated here that can be mapped over a scale to shift the scale up by an octave.

```
octaveUp x = x * 2
```

[5] Pythagorean temperament described as a list.

```
ratios_just = [1, 25/24, 9/8,6/5, 5/4, 4/3, 45/32, 3/2, 8/5, 5/3,
    ↪ 9/5, 15/8, 2]
freq_just = [ referencePitch * (ratios_just !! (n - 1)) | n <-
    ↪ [1..12]]
freq_just2 = map octaveUp freq_just
```

3.1.6 Applications

Scale Generation

Based on the presented examples, it is possible to interface with a number of existing Haskell harmony libraries and frameworks (such as Haskore (Hudak et al. 1996)), taking the lists generated and using higher-order functions such as a filter to select or remove notes according to diatonic patterns found in those libraries. Alternatively, the lists found in these libraries could also be used in the list comprehension itself to create predicate conditions or to select only values from the list of diatonic notes, such that scale notes are the only frequencies created.

Digital Instrument Tuning

Applying these concepts are intended as part of an ongoing work however it would be simple to apply the output of the implementation proposed here to an OSC system or even for use with MIDI. For example, the output could be formatted into a SysEx message that can be sent to configure a MIDI device that supports the MIDI standard's tuning.

3.1.7 Conclusion

This approach to programmatically describing tunings can empower the digital luthier to more deeply explore and support tuning systems in an instrument and potentially create user-facing approaches and tooling for configuring the instrument through this style of strategy. One might imagine a model for example, where the instrument tuning is configured through the evaluation of an expression in the form of a list comprehension which generates a static file residing on the DMI itself, configuring the tuning of the instrument. When language features such as this have such a natural fit to problems associated with the domain, there is a powerful potential to create idioms within the language that significantly reduce the cognitive burden and allow programmers to solve musical problems with ergonomic and ultimately expressive strategies that can develop into the kind of idioms that become a shared language in programming communities.

Conceptually, a DSL that utilises the ideas from this work would resemble the code fragment shown in Figure 3.1, where a 4 by 4 grid is created and mapped onto a generated 12TET diminished blues scale.

A related hardware controller could then be configured using the output from this program.

```
grid = Button 4 <+> Button 4
referencePitch = 440
freq_12tet_dimBlues = [ 2.0 ** (n / 12.0) * referencePitch | n <-
    ↪ diminishedBlues]
applyToGrid grid freq_12tet_dimBlues
```

Figure 3.1: A DSL to map tempered scales to interfaces

Given the basic premises outlined here, it is hoped that these ideas can be applied when considering the design of new digital musical instruments,to further facilitate the exploration of tonality. Exposing this level of functionality is highly beneficial given the number of people engaging

with digital instruments and computers. Composing and creating on an intuitive level need not be bounded by the conventional theory that has underpinned, but perhaps constrained, 100 years of music.

3.2 Exploring DSLs for Expressing Musical Patterns

Rhythm is the sequencing of musical events in time. These events are set apart by a time step dictated by our sensory threshold to synchronize with these events. Repp suggests this inter-onset interval is between 100-120ms (Repp 2003). Variations around this duration contribute ornaments and nuanced variation, such as grace notes and the fluctuation in meter commonly referred to as 'feel' (Polak 2010). Western music theory describes music at a rate of beats per minute (BPM) as an isochronous sequence with an equally spaced, periodic pulse (Fitch 2013)[6]. Beyond the periodic groupings referred to as meter, the emergence of pulse, the feeling of intense and weaker beats, is implied within the time signature of a musical piece and, as such, leans towards specific, almost idiomatic approaches to notating rhythm in traditional Western notation. Whilst traditional notation is expressive and information-dense, looking towards more complex manipulations of musical time, digital systems tend to struggle to manage and represent this well. Most notably, this is a challenge in a number of software notation programs, where expressing two-time signatures in parallel (polymeter) or working with concurrent time (polyrhythm) is not generally well supported, and so requires undesirable workarounds in order to express these ideas, hindering the creative process.

In this Section, we build upon the notion of cyclic time as described by Tagg (1997), looking at alternative ways cycles can be expressed, drawing inspiration from Tidal Cycles (McLean 2014), a DSL for Live Coding of musical patterns.

There are many examples of DSLs used in music creation and performance (Magnusson and McLean 2018) and for the processing of audio (Puckette 1997; Orlarey, Fober, and Letz 2009). However, these DSLs have not been widely applied to the design of Digital Musical Instruments, particularly in the context of the complete instrument, for example, Magnusson's aforementioned Instrument ModelMagnusson (2010a). Expanding on Magnusson's definition of a DMI, an instrument can be viewed through the lenses of different domains, each with its own nuances. Therefore, the design of these instruments can be composed over these domains, avoiding being tied down with or hindered by implementation details.

This section introduces a small functional Domain Specific Language for manipulating musical patterns and sequences. This use of a DSL for musical patterns provides the building blocks for working with sequence-based musical constructs encapsulated in cycles. We go on to provide abstractions that apply polyrhythmic and polymetric rhythms to our representation of time.

To demonstrate the practical benefits of our system, we describe a DMI designed for the exploration of polyrhythm, utilising tools from the Muses Project[7]. The DSL, pat, described in Section 3.2.2, is used to create a tangible, interactive instrument that applies an approach similar to Varney's 'Wheel Method' (John Varney 2014) to spread polyrhythms around a circle. This allows polyrhythms to be expressed using physical rings, as seen in Figure 3.2.

More details about the project, implementation, and examples are publicly available from `https://muses-dmi.github.io/`.

[6] Though there are examples of non-isochronous music, which can be identified by the clapping test, described by Arom (2004).

[7] https://muses-dmi.github.io/pat/overview/

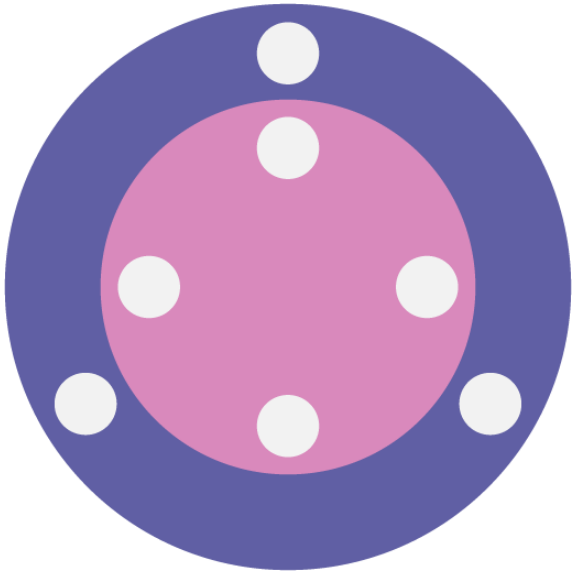


Figure 3.2: A 3 against 4 polyrhythm with cycles represented with circles.

3.2.1 Traditional Expression of Rhythm

Whilst comprehensively covering the music theory relating to rhythm is beyond the scope of this thesis, we briefly introduce and illustrate the topics we will focus on in order to demonstrate the incorporation of these concepts, which are often difficult to express in computer notation packages and digital instruments.

Musical notation is an information-dense medium that lends itself well to transferring musical information to a performer. Whilst superficially, a mathematical equivalence can be shown between two methods for notating a rhythm, the choice of notation contains inferred information concerning meter and pulse for a performer to act upon. As such, it is occasionally appropriate to notate work using concurrent staves with different timing to portray the intended pulse of the piece elegantly. This can be viewed from two perspectives; ideally in one respect; allowing a composer to effectively express how a rhythm should 'feel' [8] to the performers in order to realize the piece fully. Alternatively, in some performance situations, the composer's realization may be compromised in order to notate a piece in a way that reads more idiomatically, allowing players that may be sight reading to read a more familiar part.

The result of these differences are subtle, constituting minor fluctuations from the strict subdivision that is notated or articulations of a musical event based on the position within the bar, but we argue for the former, which facilitates the expression of intention.

In order to avoid overly quantised playback of sequences, a digital system requires an implementation that balances a comprehensible representation with a playback system capable of applying the more nuanced variations eluded to in this work. As an important facet of performance, an abstraction that allows expression over this is desirable and factors into the design considerations of the approach presented in this thesis, though deeper consideration of rhythmic feel and micro timing will be deferred to future work.

Beyond the musical feel embedded in a rhythm, there are also other concepts that are typically not well presented to users of digital systems. Both polyrhythm and polymetric rhythms are concepts that are incorporated into many specialized devices and applications. However, many mainstream digital systems do not typically facilitate their use in an expressive idiom.

In the following sections, these concepts are presented using short meters, allowing the concepts

[8] where the pulses/strong beats of different instrumental parts should be and how they line up against other parts

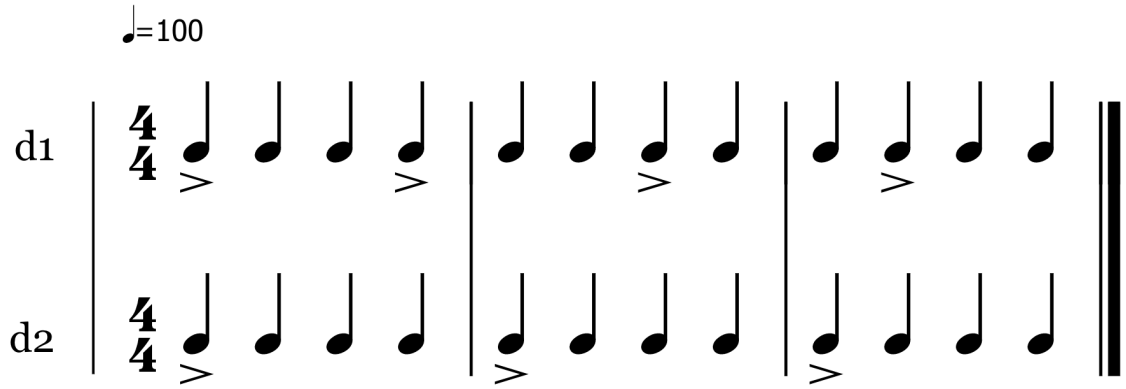


Figure 3.3: Polymetric passage of 3/4 over 4/4, notated in 4/4 with accents indicating the first beat.

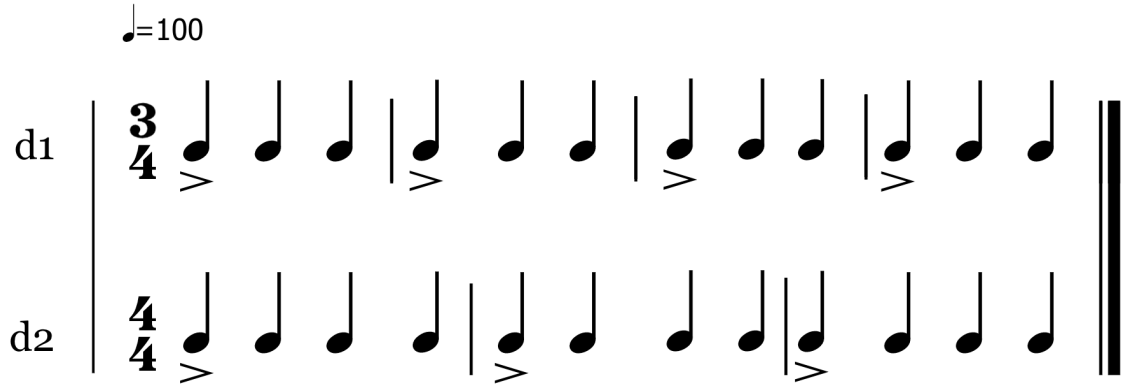


Figure 3.4: Polymetric passage of 3/4 over 4/4, notated in with dual time signature.

to be clearly displayed and fit well on the page, however, these principles extend for other values.

Polymetric Rhythms

Consider a simple example that notates a polymetric phrase. Phrases are constructed using note durations rooted in the same tempo, but using different rhythmic meters (time signatures). This concept is often expressed by notating the parts in a common time signature, perhaps providing articulations that are suggestive of the 'feel' of a different meter. This is demonstrated in Figure 3.3, where the first beat of a phrase is accented.

In some cases, a composer may wish to work directly in the time signatures that the work was intended to be in, producing a notation similar to that shown in Figure 3.4. This allows idiomatic writing for both parts and implies the feel of each part individually. Despite, fairly common use, this is not supported by many mainstream digital systems, and often when it is, there are restrictions that prevent a complete sense of expressibility.

Polyrhythmic Rhythms

Figure 3.5, shows two approaches to expressing a polyrhythm. In the first measure,a four over three polyrhythm is expressed as a 4/4 measure, at a tempo of 100BPM. This is demonstrated as being equivalent to the second measure, notated in a different meter and tempo.

Using traditional notation, these phrases are expressed quite well, indicating how these rhythms anchor against each other. In more complex examples, however, it may be beneficial to conceptually

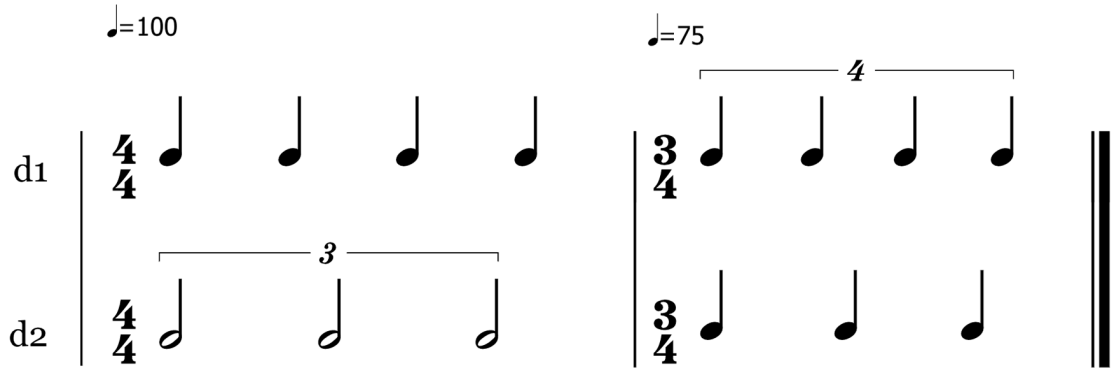


Figure 3.5: A polyrhythm of 4 against 3,demonstrating a notational equivalence

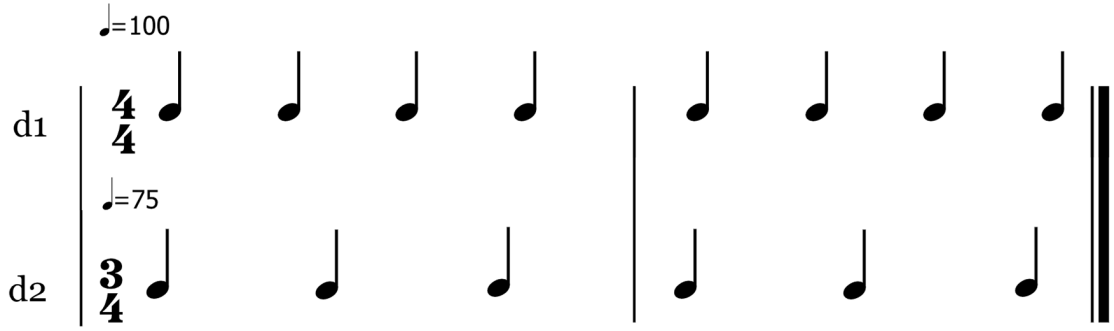


Figure 3.6: 4 against 3 polyrhythm simplified to quarter notes at related tempo and time signature.

work with tempo and time signatures applied against each other rather than having some parts entirely in tuplets[9]. This is mainly, the case where polyrhythms form the basis for an entire piece, and further, the intention may be to apply both different tempo and pulse. Figure 3.6 demonstrates how this may be notated, where the implied pulse from each time signature should be considered.

Rate

In Western notation, the rate at which a rhythm is performed is represented in a relative manner, with durations for notes given as subdivisions of a measure of time (a bar). This measurement is made absolute by providing the number of beats per minute (BPM). While BPM allows a performer to approximate tempo in relation to seconds, the approach described in Section 3.2.3 assumes the rate to be given as the duration of the sequence. It is, therefore, essential to consider this relationship where a measure's duration (in seconds) can be calculated from beats per minute (BPM) with the following equation, with B representing the number of beats in the measure.

$$\frac{60}{BPM} \times B$$

As mentioned previously, traditional notation leaves perturbations in tempo to be inferred based on the notation (time signature, articulation and style), but it is recognized that variation of the interonset interval between notes also significantly contributes to the 'feel' (Gouyon 2007). This is typically contextual, with proficient performers modulating the tempo based on the style of piece being performed.

Limitations in Expression

Many electronic instruments and software do facilitate the expression of these rhythmic ideas, however, it is observable that there are often hurdles in expressing complex rhythmic relationships that require working against the functionality of the application or device. Further, a common complaint levied against digital systems is the lack of 'feel'. This is due to many implementations of metre and pulse lacking the continuous modulation a human performer adds. While there are systems that aim to capture this, there is a lack of work that abstracts these ideas in a transferable and expressive way. This, therefore, motivates the approach laid out in this work.

These issues are beginning to be addressed in different systems, but a consolidated approach that affords expressivity to the composer or performer is missing.

3.2.2 Describing Time with Tidal influenced Patterns

Programming languages, particularly those from the live coding movement (Collins et al. 2003) offer another method for exploring sequences, with a focus on sequencing being a staple part of many live coding languages (Magnusson and McLean 2018; Aaron and F. Blackwell 2013). Within these languages, code is used to express a sequence during performance. The ideas presented in this work draw from and are influenced by these languages, later drawing on them within the context of Digital Musical Instrument design.

Whilst several programming languages have been built to express musical ideas, Tidal Cycles captures rhythmic expression in a way that is transparent when working with rhythmic sequences and is syntactically light.

Due to the density of musical information that traditional notation presents, it is difficult to provide a rich and expressive representation of music as a text based programming language. Focusing on music based on patterns in time, Tidal Cycles excels. Sequences are expressed as cycles, analogous to bars or measures, though they do not inherently suggest a meter. A single

[9] In practice this may require musicians to perform using individual metronomes in order to realize the intention effectively.

Figure 3.7: An example pattern featuring subdivisions down to 16th notes.

cycle is a length of time into which some number of events may be distributed. Musical events are distributed equally throughout a cycle and any single event may be further subdivided by providing subdivisions (described as nested lists) of musical events.

We derive a variation of Tidal cycles syntax for describing patterns, where a pattern is a string delimited by white space. Further subdivisions are expressed using a notation for nested lists, incorporating the most fundamental ideas of Green's cognitive dimensions for notating lists (Green 1989) [10]. This syntactically allows, for example, expressing the pattern of two sixteenth notes, one eighth note and three quarter notes as seen in Figure 3.7, using the following pattern[11]:

[[bd bd] bd] bd bd bd

Whilst Figure 3.7 notates the pattern above in 4/4 we should consider that the pattern, unlike the notation, has no implication of how the pulse of the part should feel. In the context of this work, pulse and meter may be considered functions that act on a pattern and as such are not represented in the pattern itself which will be built upon in future work.

3.2.3 Notions of Time

This section describes an abstraction for musical time, providing an underlying data structure that can be used by digital systems for performance and playback, termed sequences. Further, constructs for the manipulation of patterns are provided. The notation described in Section 3.2.2 is used for patterns, showing how they are translated to 'flattened' sequences.

The approach presented here is intended as a conceptual model rather than a strict format specification. As such, each sequence may be extended with front matter or meta data to encode the articulation and phrasing applied to the sequence and other implementation-specific requirements. A complete implementation based on the ideas presented in this section can be found on the project's Github page[12].

3.2.4 Representing Sequences

A sequence is a series of events, taking the form of a data structure that approximates Schaeffer's definition of a sound object (Schaeffer 2017), with a collection of functions which are able to manipulate them.

A sequence of events in time is represented as an ordered list[13], where each element represents the onset of a given set of events:

$$E = [e_1, e_2, \ldots, e_n]$$

[10] We use Haskell's list notation, [] for empty lists and [x1 ,..., xn] for lists containing n elements, where x_i could also be a list.

[11] Technically this should be written "[[bd bd] bd] bd bd bd", but quotes are ommited when clear from context.

[12] https://github.com/muses-dmi/pat/

[13] As already noted, we use Haskell's list notation, [x1 ,..., xn], to represent an ordered set.

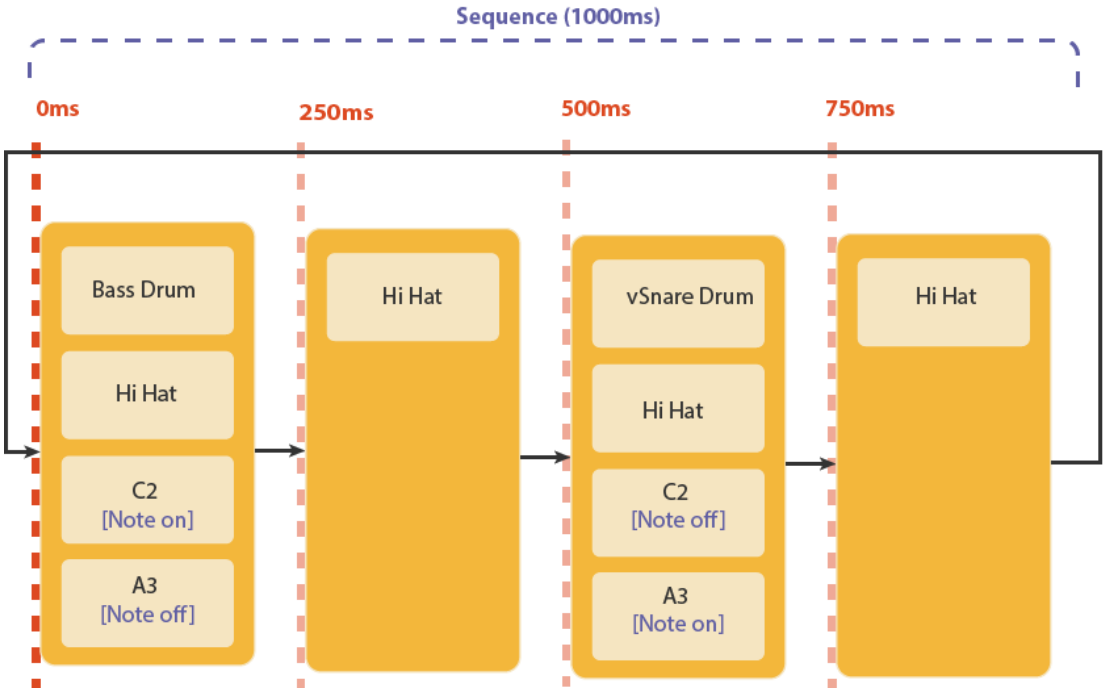


Figure 3.8: Sequence structure

Events at a given point in time happen simultaneously and, therefore, do not require ordering.

Specific representations of events are undefined and left to a particular implementation, examples include MIDI[14] or OSC[15] messages. This structure, called a **sequence**, is assigned the following type, where τ is the type for events and is supplied by an implementation[16]:

$$\textbf{sequence} \ : \ [[\tau]]$$

The position of events in the list represents events in time, with the gaps between considered the inter-onset interval, analogous to the interval explored by Madison, for the perception around inter-tap interval (Madison 2001). The sequence can be naively played by stepping between each element of the list with a fixed time interval. Modulation of this playback interval remains an exciting opportunity for future work.

In order for a sequence to be played, the inter-onset interval must be supplied as a function of the patterns used to generate it. Therefore, an implementation requires a sequence and an inter-onset interval derived from some notion of rate (cycle duration or tempo) in order to operate. Figure 3.8, provides a visual representation this[17].

Tempo calculations

Devices utilizing this representation are required to calculate and manage tempo. Separating the management of interonset intervals from the sequence of events, such that the tempo is free to be modulated without the need to operate on the sequence itself. This provides a opportunity to reflect real world variations in time such as those described by BartonBarton, Getz, and Kubovy (2017).

Given a target cycle duration, the interonset interval can be calculated as follows, where T is the length of time a cycle lasts and N is the number of cycles. These values are then divided by

[14] https://www.midi.org/
[15] http://opensoundcontrol.org/
[16] A expression of the form $x : \tau$, states that x has type τ.
[17] Observe that if MIDI or a similar mechanism is used and a sequence is being used as a loop, the note off will be required to be on the starting note of the 'next conceptual cycle', meaning an unpaired note off will be sent on the first beat of the first playback of the cycle, as seen in this case.

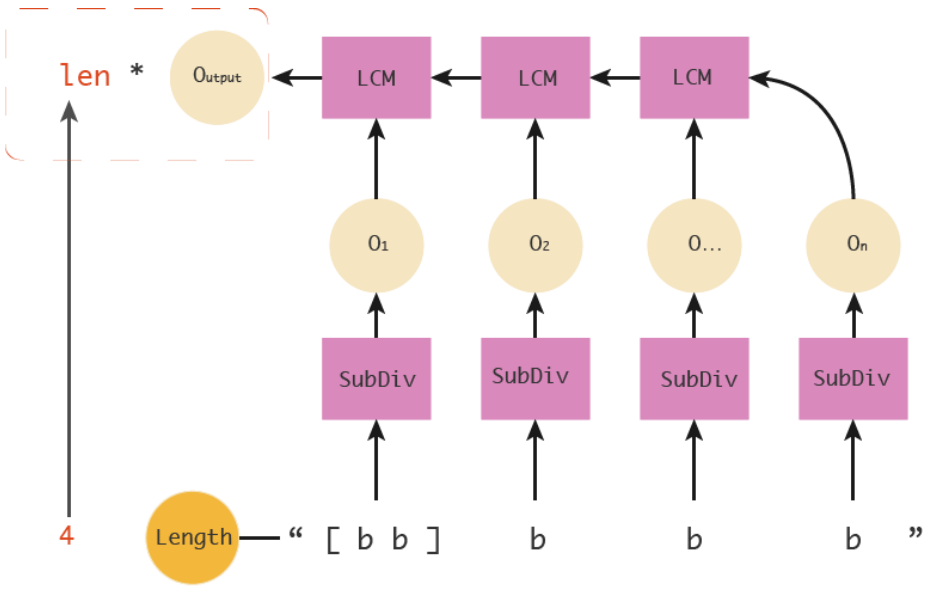


Figure 3.9: The process of expanding a pattern's subdivisions.

the total number of steps in the sequence to provide the interonset interval, defined as follows:

$$\mathrm{interval} = \frac{T \times N}{Steps}$$

Translating Patterns to Sequences

As described above patterns form an ordered list, where sub-lists represent nested subdivisions. Non list elements, i.e. elements that are not subdivisions, represent musical events in time.

Translating patterns to sequences, with explicit quantization, is the process of 'flattening' a pattern such that it contains no subdivisions and is correctly spaced with respect to time. This process is straightforward in the case of a single pattern, but requires a different approach for polyrhythmic and/or polymetric composition of multiple patterns. We first consider the case of 'flattening' a single pattern to a sequence and then use this to account for multiple patterns, including polyrhythmic and/or polymetric composition.

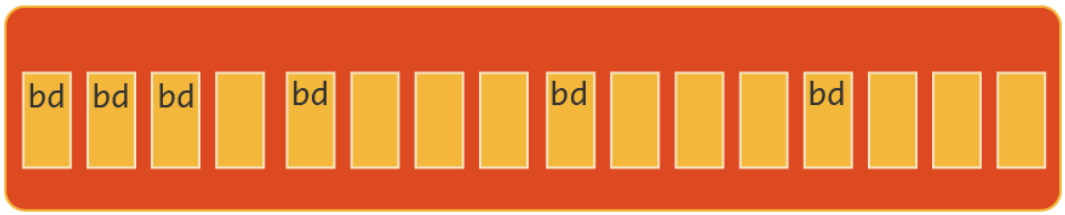
Flattening Subdivisions

In order to create the ordered list of events that represents a sequence the subdivisions of the input patterns must be 'flattened', such that each step between elements represents the smallest possible interonset interval that can represent the list.

Consider as an example the notation from Figure 3.7, which is described by the following pattern:

[[bd bd] bd] bd bd bd

The smallest subdivision used is a sixteenth note, which implies that the sequence must be flattened to the following:



The expansion of subdivisions is a function from a pattern to a sequence (i.e. pattern → ↪ **sequence**). This translation results in a sequence that has a length equal to the least common multiple of every subdivision in a pattern, multiplied by the length of the pattern itself. This operation is demonstrated over a pattern in Figure 3.9.

Handling Polyrhythmic and Polymetric

We now consider the creation of sequences that combine patterns in either a polyrhythmic or polymetric manner, utilizing the previously discussed method for expanding subdivisions.

3.2.5 Polyrhythmic Merge

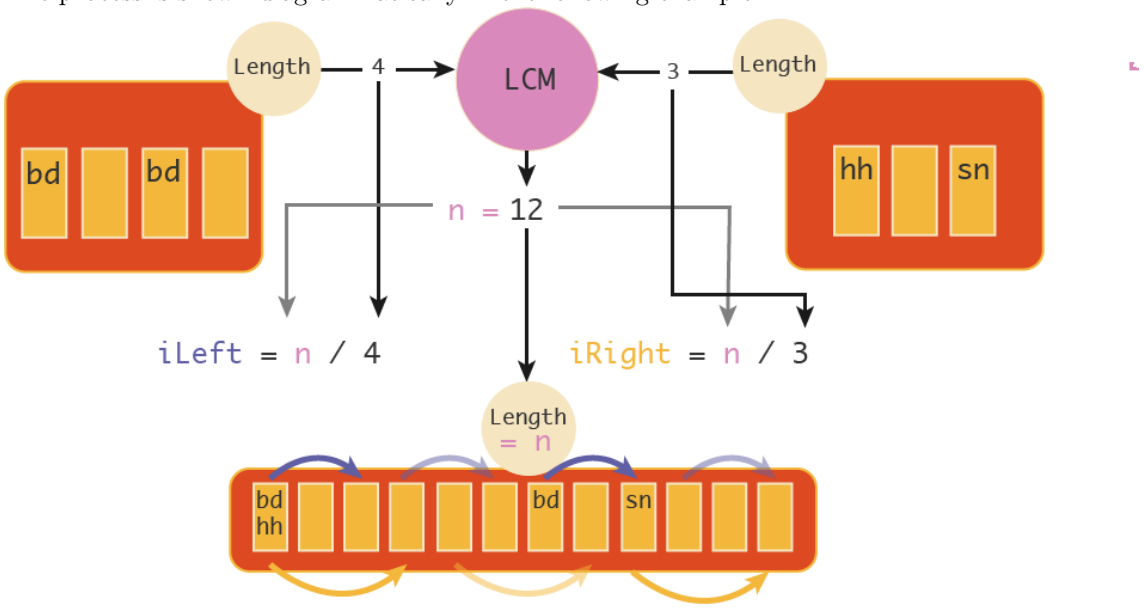
Polyrhythmic merge is the process of first expanding the subdivisions for both sequences and then creating a new list that is n elements in length, where n is the least common multiple of the 'flattened' pattern's length, e.g.:

```
lcm (flatten l) (flatten r)
```

where l and r are short for left and right, respectively. Each element is then inserted into the new list, at intervals of i, calculated for left and right independently as:

```
iLeft = n / (length (flatten l))
iRight = n / (length (flatten r))
```

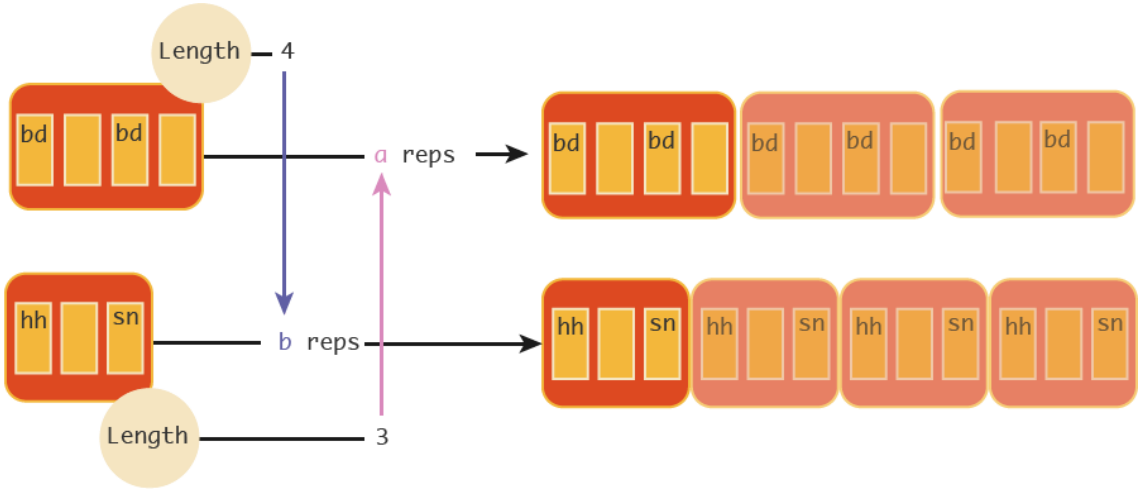
The process is shown diagrammatically in the following example:



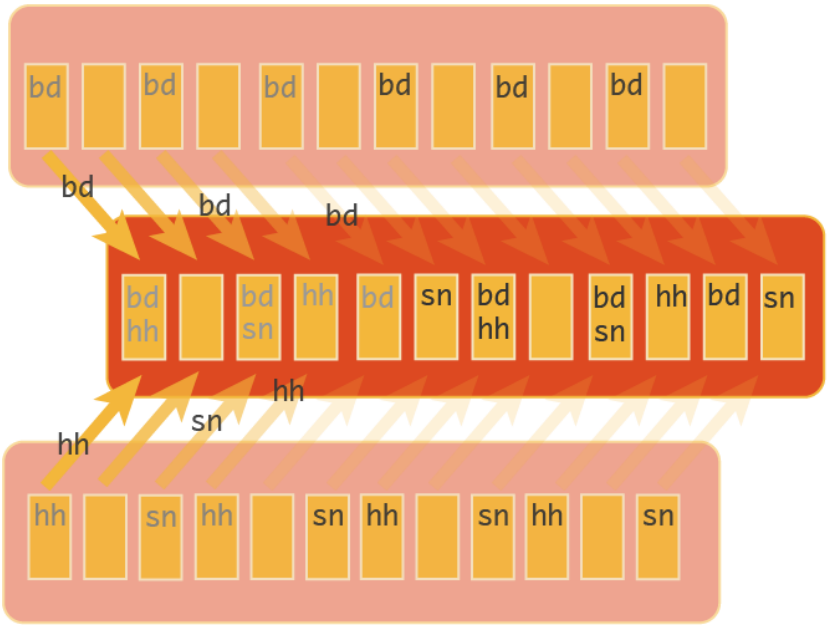
Polymetric Merge

The process of merging a polymetric phrase creates a new sequence extending to the point where the input sequences synchronize. This is calculated by multiplying the length of both sequences prior to flattening, which is achieved by repeating the list.

This is demonstrated in the following diagram:



The resulting sequences are then combined such that each element becomes a list of lists, where the nested list contains the events on that subdivision, as demonstrated in the following diagram:



Patterns can then be be subdivided as described previously.

3.2.6 An example DSL for expressing notions of time

To demonstrate the use of our representation this section explores using DSLs (as described by Hudak (1996)) in the context of Digital Musical Instrument Design. A tiny DSL, called pat, for creating patterns in time' with a syntax inspired by Tidal Cycles is presented, whose grammar is specified in Figure 3.10. We assume a symbolic representation for events, e.g. alpha numeric sequences such as bd and snare, etc. The binary operations |:| and —:— represent polyrhythmic and polymetric merge, respectively.

pat can express sequences with a light touch. For example, consider the following, terse, description producing a 3 : 4 : 7 polyrhythmic pattern:

```
[a c e] a a |:| a c a c|:| g g g g g g g
```

Further, these combinators can be used together to create complex, evolving patterns. The following expression generates twenty four steps of evolving musical material[18]:

[18]This example can be heard on https://muses-dmi.github.io/pat/listen

⟨*event*⟩ ::= identifier

⟨*pattern*⟩ ::= ⟨*event*⟩

| [⟨*pattern*⟩ { **,** ⟨*pattern*⟩ }]

| ⟨*pattern*⟩ |:| ⟨*pattern*⟩

| ⟨*pattern*⟩ -:- ⟨*pattern*⟩

Figure 3.10: Pattern grammar

```
[a c e] a a [a c e] a a
—|— a c a c
|:| g [g c] g [g c] g [g c] g
```

A practical realisation, as given on the project's Github and outlined in Section 3.3, must provide concrete representations for events and other implementation details omitted here, for ease of presentation.

3.3 Applications for DMI

Given the DSL pat, we now briefly discuss how it might be utilized in the design of a novel instrument that facilitates the ability to both describe constraints (in the form of rhythmic sequences) and also the ability to perform. We provide an example of two instruments, one virtual and one physical.

The Muses Synth and Pat

A combination of the Muses Synth[19] and pat can be used to describe a virtual instrument.

In order to operate with the Muses Synth we transpile from our representation into a JSON file representing a sequence. The Muses Synth implements the ability to parse and perform a sequence as described in Section 3.2.4. This provides the ability to define a rhythmically complex Virtual DMI using pat expressions. This instrument can be downloaded and explored at the project's website.

Polyrhythmic Ring Sequencer

The 'Polyrhythmic Ring Sequencer' is an instrument that provides an exploration of polyrhythm, inspired by the 'Beat Bearing' (Bennett and O'Modhrain 2008). Namely:

- Accessible - requiring little technical proficiency
- Playable - can be played in real time, constituting a performance that requires input from a performer.
- Visually represents polyrhythm in an intuitive manner.

Figure 3.12 shows the instrument produced for this purpose. This instrument is a circular step sequencer with three channels, represented by three concentric rings. It is implemented using a

[19] https://muses-dmi.github.io/pat/synth

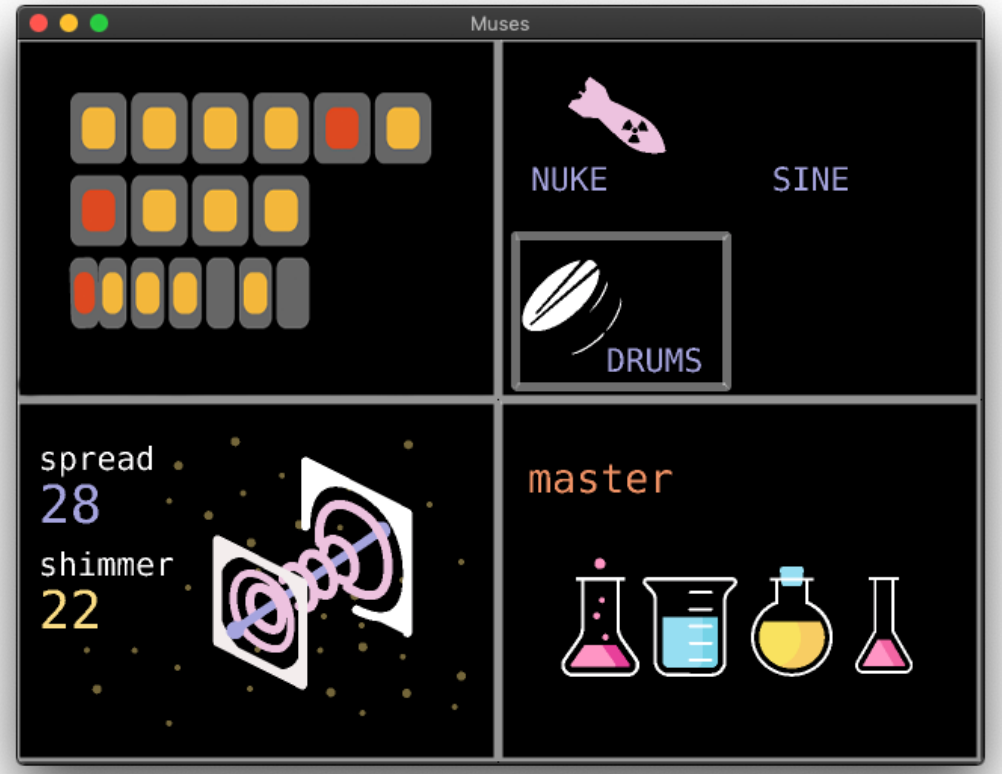


Figure 3.11: Screen shot of Muses Audio Application.

Figure 3.12: A novel instrument for polyrhythmic expression with marbles.

Sensel Morph[20], a touchpad capable of measuring force, and uses marbles to represent musical events. A cycle is represented from 12 o'clock, around 360°.

pat is used for the description of the rings with a set of functions that use ImplicitCAD[21] to produce an STL, a 3D model format for 3D printing. We allow the $|:|$ combinator to be used but restrict the use of polymetric merge $(-|-)$ due to limitations of the system. This implementation is provided on the project website, alongside tools to generate new rings based on pat.

3.3.1 Conclusion

This section presents a method for representing musical sequences as an ordered set containing musical events. An event may be considered analogous to Schaeffer's definition of a sound object (Schaeffer 2017), where some representation describing a musical event is provided.

This method is motivated by a desire to be able to manipulate the sequence to represent complex notions of time, such as polyrhythm and polymeter. A process for merging sequences inline with these concepts is provided, producing a sequence of events that represents merging the provided sequences.

A conceptual overview of how sequences may be merged both polyrhythmically and polymetrically is demonstrated as alongside how time is represented within this context. This is presented at a high level in anticipation of further work, where this method produces a suitable digital representation to explore more complex and expressive manipulations of time.

In order to explore this concept, a small language was presented to express patterns in time, inspired by the language Tidal Cycles (Magnusson and McLean 2018). This facilitates the expression of a polyrhythmic pattern of seven snare hits over a 'four on the floor' bass drum pattern, in the following manner:

```
bd bd bd bd |:| sn sn sn sn sn sn sn
```

[20] https://sensel.com/pages/the-sensel-morph
[21] http://www.implicitcad.org/

To provide real-world examples, and to position future work on the topic, two digital musical instruments were introduced, that incorporate the use of our pattern language in their design. These instruments are made available for further exploration of how this overall approach may be implemented in a digital system.

3.4 Exploring Embedded DSLs for Dynamic Grid Controller Layouts

This section provides an overview of the final exploratory work that prompted the decision to understand better how new programming languages for digital lutherie can be designed with a more informed strategy. This work began to introduce a functional programing language for describing grid-based control interfaces. The intention of this language was to improve the composability of components used to create musical interfaces, such that functional catamorphisms [22] could be applied to the interface to make mapping things like tunings and other controls more expressive.

3.4.1 Embedded Domain Specific Languages

Language design is a non-trivial problem that requires a considerable investment of time. In particular, whilst cited as having many benefits of general-purpose languages, DSLs lack strong development guidelines and underpinning research, meaning that it can be difficult to know how effective a DSL design will be until it is built and used. We have already discussed the nature of languages to need to evolve and adapt to its niche, meaning that it is unlikely for a design to be correct the first time. Hudak (1998) discusses a solution for avoiding this by embedding DSLs inside of a host language, inheriting the features and many of the semantics of the host language. Som languages are particularly good for this due to features such as algebraic data types (Maguire 2018), as they are able to effectively model and work with structures such as abstract syntax trees (ASTs) idiomatically.

To build a small embedded language for laying out common control widgets such as buttons and sliders, we explored the use of EDSLs in Haskell [23] through the use of pretty printing (Hughes 1995). This allowed for the embedded language to be used to express interfaces which could then generate outputs for different target architectures, where we target littlefoot for defining interfaces on the Roli Blocks, and openSCAD for defining physical widgets that could be 3dPrinted.

3.4.2 Transpilation Strategy

Transpilation is a process that involves the conversion of source code into the source code of a different target language. Part of the reason for selecting the Roli Block as an initial target for this language was its support for dynamic reprogramming of the controller's layout through an interpreted programming language called littlefoot. This C like programming language included a simple API for access to the main features of the Roli Block and could be compiled into a bytecode that is stored and run on a Roli Block.

To work with littlefoot in a functional style an Abstract Syntax Tree built around the following grammar is used and manipulated by a variant of the language termed 'functional-littlefoot'.

Given an AST in Haskell implementing this grammar, a small library was built up to work with the Roli Block API and a set of combinators for composing widgets together were implemented.

[22] operations such as map, filter, reduce and zip
[23] https://www.haskell.org/

⟨*id*⟩ ::= [a-z]+

⟨*Int32*⟩ ::= [0-9]+ ⟨*Float*⟩ ::= [0-9]+ '.' [0-9]+ ⟨*Bool*⟩ ::= 'true' | 'false' ⟨*Word8*⟩ ::= [0-9]+

⟨*Type*⟩ ::= '`int`' | '`float`' | '`bool`' | '`void`'

⟨*Literal*⟩ ::= Int32 | Float | Bool | Word8

⟨*Binary Operator*⟩ ::= '`+`' | '`-`' | '`%`' | '`/`' | '`*`' | '`=`' | '`<`' | '`<=`' | '`>`' | '`>=`' | '`!=`' | '`&&`' | '||' | '`&`' | '|'

⟨*Unary Operator*⟩ ::= '`!`' | '`~`'

⟨*Expression*⟩ ::= ⟨*Literal*⟩ | ⟨*Binary Operator*⟩ ⟨*Expression*⟩ ⟨*Expression*⟩ | ⟨*Unary Operator*⟩ ⟨*Expression*⟩ | ⟨*Type*⟩ ⟨*Expression*⟩ | ⟨*id*⟩ ⟨*Expression*⟩ | ⟨*id*⟩

⟨*Statement*⟩ ::= 'if' ⟨*Expression*⟩ ⟨*Statement*⟩ | 'for (' ⟨*Expression*⟩ ';' ⟨*Expression*⟩ ';' ⟨*Expression*⟩ ')' ⟨*Statement*⟩ | 'while (' ⟨*Expression*⟩ ')' ⟨*Statement*⟩ | 'return' ⟨*Expression*⟩ ';' | ⟨*Type*⟩ ⟨*id*⟩ ';' | ⟨*Type*⟩ ⟨*id*⟩ '=' ⟨*Expression*⟩ ';' | ⟨*id*⟩ '=' ⟨*Expression*⟩ | ⟨*Expression*⟩ ';' | '' ⟨*Statement*⟩ '' | ';'

⟨*Definition*⟩ ::= ⟨*Type*⟩ ⟨*id*⟩ [(⟨*Type*⟩, ⟨*id*⟩)] [⟨*Statement*⟩] | ⟨*Type*⟩ ⟨*id*⟩ ';'

Figure 3.13: Littlefoot grammar

⟨*id*⟩ ::= [a-z]+

⟨*Colour*⟩ ::= ⟨*Word8*⟩ ⟨*Word8*⟩ ⟨*Word8*⟩ ⟨*Word8*⟩ ⟨*Data*⟩ ::= ⟨*Word8*⟩ ⟨*Size*⟩ ::= ⟨*Word8*⟩

⟨*Midi Message*⟩ ::= ⟨*Status*⟩ ⟨*Word8*⟩ ⟨*Word8*⟩ | 'MessagePair' ⟨*Midi Message*⟩ ⟨*Midi Message*⟩

⟨*Widget*⟩ ::= 'Tile' ⟨*Size*⟩ | 'Button' ⟨*Size*⟩ ⟨*Midi Message*⟩ | 'Slider' ⟨*Size*⟩ ⟨*Midi Message*⟩

⟨*Interface*⟩ ::= ⟨*Widget*⟩ ⟨*Colour*⟩ | ⟨*Interface*⟩ '⟨*+*⟩' ⟨*Interface*⟩ | ⟨*Interface*⟩ '⟨^⟩' ⟨*Interface*⟩

Figure 3.14: Gridlang grammar

3.4.3 Modelling with Types

Through this exploration, we embed a small language referred to as GridLang in the functional language Haskell. This language provides a set of primitives, modelled in the Haskell type system, for describing widgets that might be used in a typical musical controller. To remain simple, this version of the language features blank tiles, buttons and sliders for the creation of interfaces.

A grammar for this system is provided below.

3.4.4 Horizontal and Vertical Composition

In a similar style to the toy language `pat`, described in Section 3.2 provides two new combinators for working with GridLang types. These combinators, <+> and <^> represent horizontal and vertical placement of widgets, respectively. This allowed for a means for expressing controllers through the composition of arranged widgets referred to in the grammar as interfaces. Through this recursive definition, these interfaces could be organised by the user and recombined to work with higher-level building blocks for defining grid-based interfaces. Figure 3.15 demonstrates the layout of a simple interface on the Roli Block using GridLang, the code for which follows.

```
buttonSize = Size 4 4
```

```
spacer =  (Size 12 12) none

sliderSize = Size 4 9

ccCh1full = (Message 176 39 127)

ccCh1nil = (Message 176 39 0)

b1 = Button buttonSize (MessagePair (Message 144 67 127) (Message 128
    ↪ 67 127)

b2 = Button buttonSize (MessagePair (Message 144 69 127) (Message 128
    ↪ 69 127)

b3 = Button buttonSize (MessagePair (Message 144 70 127) (Message 128
    ↪ 70 127)

sli1 = Slider sliderSize (MessagePair (Message 176 39 127) (Message 176
    ↪  39 0))

sli2 = Slider sliderSize (MessagePair (Message 176 42 127) (Message 176
    ↪  42 0))

sli3 = Slider sliderSize (MessagePair (Message 176 1 127) (Message 176
    ↪ 1 0))

buttons = (b1 red) <+> (b1 blue) <+> (b1 green)

sliders = (b1 green) <+> (b1 red) <+> (b1 blue)

layout = button <^> spacer <^> sliders
```

3.4.5 Motivating a Study on the Designer Tool Relationship

The exploration of GridLang was started as an investigation into using EDSLs to rapidly design and test language design ideas for digital lutherie. Through the use of embedding in Haskell, once again a language was rapidly produced to test a set of ideas. Further, these design patterns provided the option of targeting multiple different ecosystems to work with through the use of code generation techniques such as pretty printing. Evaluating the basic functionality of GridLang implied the need to use some strategy to constrain the definition of interfaces such that they could transfer between targets. As formulating a new set of programming language features was explored, it became apparent that it was difficult to imagine which features would be most desirable and how to prioritise them. This presents little issue in defining a small toy language to experiment with, however, developing new languages that are impactful and have the potential for real-world use is a significant undertaking and requires considerable investment. This suggests a clear benefit to understanding the nuanced nature of selecting programming language features and characteristics and goes on to motivate this thesis.

Digital lutherie where there is so much nuanced stuff to study and limited time and resources, there needs to be other effective tools. Qualitative methods offer a far broader invertigative tool

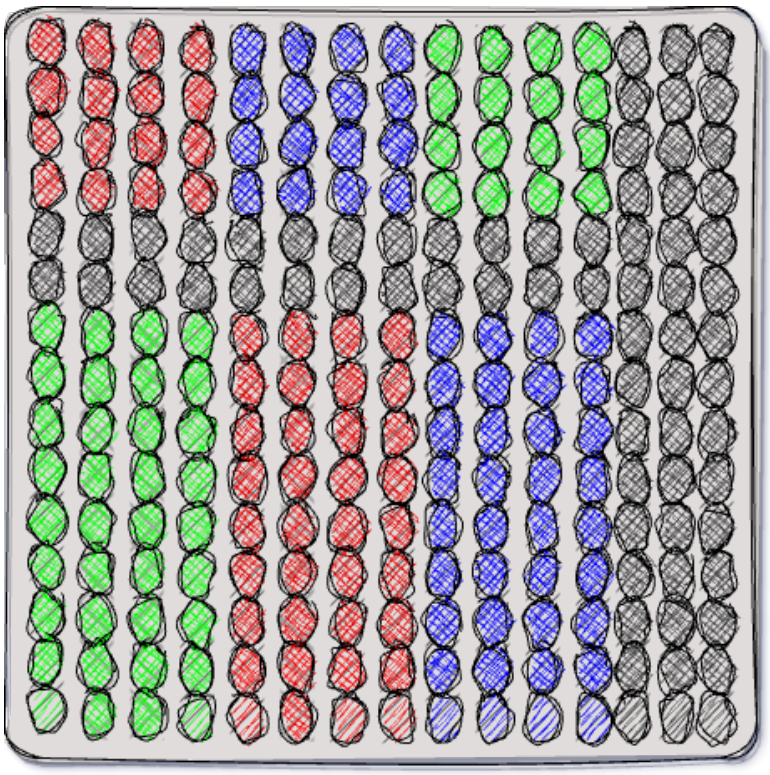


Figure 3.15: An interface on the Roli block defined using GridLang, showing three buttons and three sliders that can be interacted with. The code for this layout is shown and discussed in Section 3.4.4

.

capable of exploring multiple areas and angles at the same time that potentially offers a much more holistic picture capable of capturing a complex world. Having been introduced to the inductive approach of grounded theory, this appeared an effective research tool to contextualise more targeted exploration of our understanding of programming language use. As a study was designed, through examination of suitable qualitative methods, thematic analysis stood out as a more accessible tool for qualitative research. Through the design and implementation of this study, many methodological considerations and implications for this area of research were found and as such, in a later chapter this thesis also explores and makes recommendations for methodological approaches for this kind of research.

3.4.6 Summary

In summary, this chapter presents a series of ideas that explore programming language design as it relates to expressing musical ideas. Though this section is primarily provided to motivate the need to inform design through the inclusion of digital luthiers, this work is highlighted as it provides examples that are later somewhat supported and highlighted by the findings of the user study presented in the following chapters. These contributions include the following ideas, which are introduced here and stand as a basis for future work. Through the work on tuning systems, this chapter demonstrates the importance of programming constructs and abstractions that fit the paradigm of music programming. In Particular, Section 3.1 demonstrates the application of the functional paradigms ergonomics around list abstractions and the value this offers for working with tuning systems. In Section 3.2, this is further emphasised in the use of rhythmic sequences. These sections demonstrate the use of language features such as list or set comprehensions, catamorphisms and function composition through combinators as expressive tools for music programming.

In Sections 3.2 and 3.4, the use of DSLs is presented to programmatically specify static components of instruments that can then be used for performance. This includes the dynamic interface of a programmable surface controller but also the physical components of a ring-based step sequencer where the components are modelled programmatically and printed on a 3D printer. This work presents these ideas as an initial exploration and base for the idea. Importantly, this concept ties into the strategy of using an intermediate representation in the form of a programming language and typically a data model, such as an abstract syntax tree. This strategy is used throughout this section and provides an implementation-specific starting point for future work that may look to build upon this in relation to the design guidelines presented in Chapter 7. Throughout this chapter, the influence of functional programming is clear, and ideas from the functional programming paradigm appear to fit musical systems well. While the implications and potential of this are more specifically explored later in the thesis in the form of programming paradigms, this chapter and the projects it documents demonstrate some ideas and examples of functional programming concepts in digital lutherie. In addition to those already mentioned, these include strong typing and imply the use of dependent types as a system of constraint. It is important to add that the ideas presented here remain exploratory and primitive as a full realisation of such ideas would be a considerable task and, therefore, require a full understanding of their role and value to digital luthiers before they are pursued more fully.

Chapter 4

Study Methodology

This chapter introduces the methodolgy used for the study culminating in the publication 'Studying How Digital Luthiers Choose Their Tools'. [1]

This study explores the perspectives of digital luthiers with a range of different motivations using reflexive thematic analysis as introduced by Braun and Clarke (Braun and Clarke 2006). In order to best demonstrate rigour in our approach (Cockburn, Gutwin, and Dix 2018; L. Haven and Van Grootel 2019) and to encourage further development of this work, we refer to a prepublication of our study, presented ahead of undertaking analysis (Renney et al. 2021). Essential information is summarised below. Further details can be found in the related prepublication.

This methodology somewhat contrasts the approach typical of similar research based on grounded theory (Braun and Clarke 2019), omitting the need for peer-validated coding for example. In this study, two primary coders Familiarised themselves with and inductively coded the transcripts. Throughout the process, the research team met regularly to discuss and iterate around the analysis process (described below). This approach aimed to draw on the knowledge and experience of the research team to examine and generate themes (Braun and Clarke 2020), utilizing reflexive practice (Alvesson, Hardy, and Harley 2008).

Based on Braun and Clarke's process for reflexive thematic analysis, the approach followed these steps:

1. Data familiarization period for reviewers
2. Data coding
3. Generation of themes
4. Discussion between researchers on themes, reflection and development
5. (Iteration around steps 2-4)
6. Refining and naming themes and development of themes
7. Writing paper; discussion of themes

In line with Braun and Clarke's (Braun and Clarke 2019) description of reflexive thematic analysis, we recognise that as researchers we play a role in the generation of qualitative information (Gough and Madill 2012). Initial codes were generated by N. Renney along with Gaster who assisted in a first phase of code generation, creating around five percent of the codes. The research team primarily contributed to coding in providing iterative discussion, exploring and evolving the codes, and ultimately in forming and discussing the themes. This work was heavily driven and

[1] https://github.com/muses-dmi/dmi-design-study

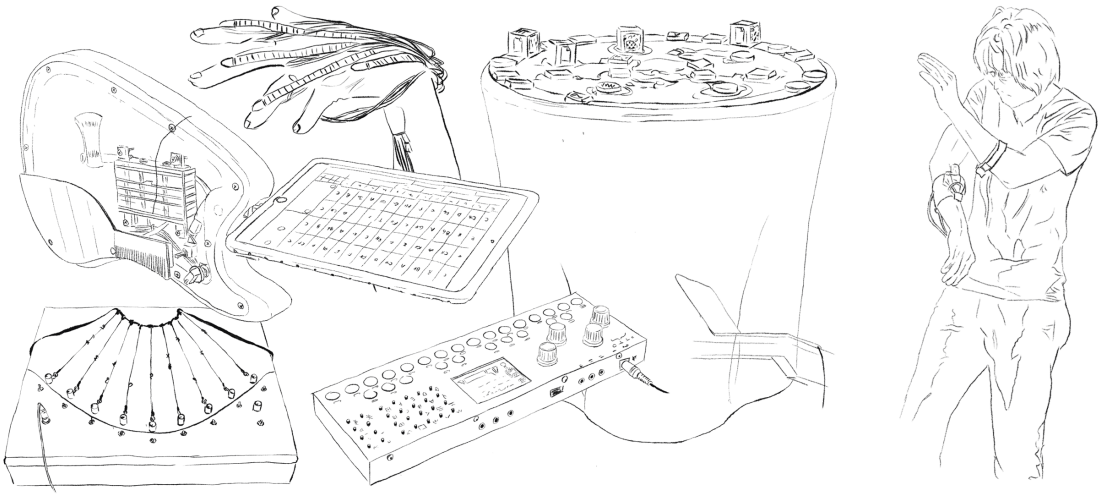


Figure 4.1: The Blade Axe, Electronic_Khipu_, The Ladies Glove, OTTO, Reactable, EMG based instrument

lead by N. Renney but drew on the experience and ideas of the whole team in interpreting and contextualising the themes. Researcher backgrounds and more information to contextualise our stance can be found in the prepublication (Renney et al. 2021). Ethical approval for this study was granted by the authors' Faculty Research Ethics Committee. Participants provided written consent for the information provided to be used in this work and provided in a raw format for future works. Prior to the publication of the data, participants were offered the opportunity to make amendments or redact any part of their transcripts.

4.1 Motivations

This work is motivated to explore the relationship between tools and the designers of high-performance devices for human-computer interaction. Digital lutherie represents a well-developed example of such a community, typically centred around the NIME conference (Marquez-Borbon and Stapleton 2015). In particular, we seek to explore how digital luthiers choose the tools that facilitate the digital components of their craft, with a primary interest in how the interactions of programming instruments occur. We recognise that digital lutherie has become a rich opportunity for the development of new programming languages (McPherson and Tahıroğlu 2020). In particular, domain-specific languages (Hudak 1997) such as those that underpin the live-coding community (Collins et al. 2003). However, this area of research is missing an extensive analysis of how people come to settle on the programming languages (and other tools) that allow them to build complex systems. We provide further context to our motivations in our prepublication (Renney et al. 2021).

4.2 Participants

Participants were directly invited according to a purposeful sampling strategy based on their contributions to a range of novel digital musical instruments or association with an organisation that produces novel DMIs. Participants were approached online and invited to participate or recommend a suitable participant. A subset of these instruments are listed in Section 4.3, and a selection is illustrated in Figures 4.1 4.2 and 4.3.

Categories of Commercial, Research, Community and Artist backgrounds were defined as a basis to select participants. Commercial and Research categories describe instruments for either

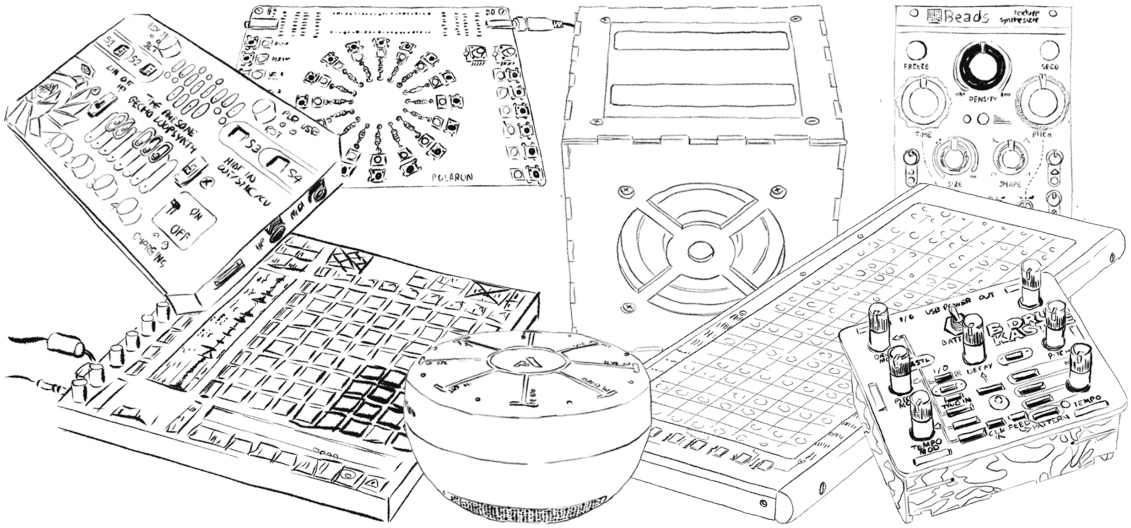


Figure 4.2: Gechologic Loopsynth, Push, Polaron, Orba, The D-Box, Linnstrument, Mutable Instruments Beads, Bastl Kastle Drum

commercial production or coupled to a research process, respectively. Community instruments broadly encompass open source projects, small teams or individuals, independently making instruments in low volumes. The Artist category represents instrument designers who build instruments to support their artistic endeavours. Of course, there is significant overlap between these definitions; however, drawing evenly from these groups helped to vary the sample of the community.

Purposeful sampling was also selected to more deliberately distribute perspectives across genders in search of a more gender diverse representation (Morreale et al. 2020; Xambó 2018; Mathew, Grossman, and Andreopoulou 2016). We acknowledge a lack of cultural diversity in this study, another important facet of diverse study populations that should be accounted for in future work (Williams 2014). These factors could be improved with a broader call in conjunction with the selection process used here, such that selection is not limited to the networking capacity of the researchers.

For this study, 27 participants were interviewed. A demographic of the population is provided in Table 4.1, and a selection of the participant's self-described roles can be seen in Section 4.2.1. Further, information gathered on the participants includes the programming languages and the tools they use and a rudimentary metric of experience in the form of years spent in the field and the number of instruments they have designed. We emphasise that this is a metric of limited insight that can poorly characterise the experience. However, attention was given to incorporating a range of experience levels when sampling participants in the selection process. For further details on the data set (including the published dataset), see the prepublication (Renney et al. 2021) (also included in Appendix B and refer to Appendix A to view participant data within this thesis. Appendix A includes tables displaying correlated participant information and a labelled diagram of the instruments presented which may be useful when reading through the thematic analysis in later chapters.

Gender		**Ethnicity**		**Age**	
Male	14	White	21	18 - 24	1
Female	8	Asian	1	25 - 34	12
Non Binary	1	Latinx	1	35 - 44	6
Prefer not to say	4	Brazilian	1	45 - 54	2
		Prefer not to say	3	55 - 64	4
				Prefer not to say	2

Table 4.1: Participant demographics (N = 27)

4.2.1 Participant Roles

- Music Technology Researcher and Professor.
- Digital Artist/Performer/Composer
- Artist
- Software Engineer
- Software Engineering Manager
- CEO
- Composer
- Founder
- Researcher and Lecturer
- Assistant Professor of Music Technology
- Composer & Instrument Builder
- Audio Developer
- Researcher, Designer, Performer
- Software Developer
- Professor
- Electronic musician
- DSP Engineer
- Professor of Media Computing
- Hardware and Software Engineer
- Lead Designer
- Composer and Interactive Hardware Developer
- Creative Director

4.3 Instruments

The instruments created by this group of designers represent a range of novel devices with a significant digital component. The sampling strategy aimed to incorporate many modes of interaction and motivations for instrument development. Instruments include open source and proprietary instruments, with some instruments representing a hybrid of the two (for example, open-source software only). Instruments also vary from bespoke instruments designed for a limited project to instruments intended for commercial mass-market production. This is often reflected in the designer's role; however, it is notable that many designers themselves work on multiple instruments that have very different use cases, decoupling the role of any one instrument and the designer. The instruments included in this study are intended to generally represent the work of the digital luthier and may include instruments that were developed with others or individually. It should be noted that the interviews were conducted around the holistic experience of the participants and not focused on the design of single instruments for the most part. The instruments listed are shared to help to represent the motivations and context in which the participants operate and therefore the kind of perspectives they may share.

Participant Designed Instruments

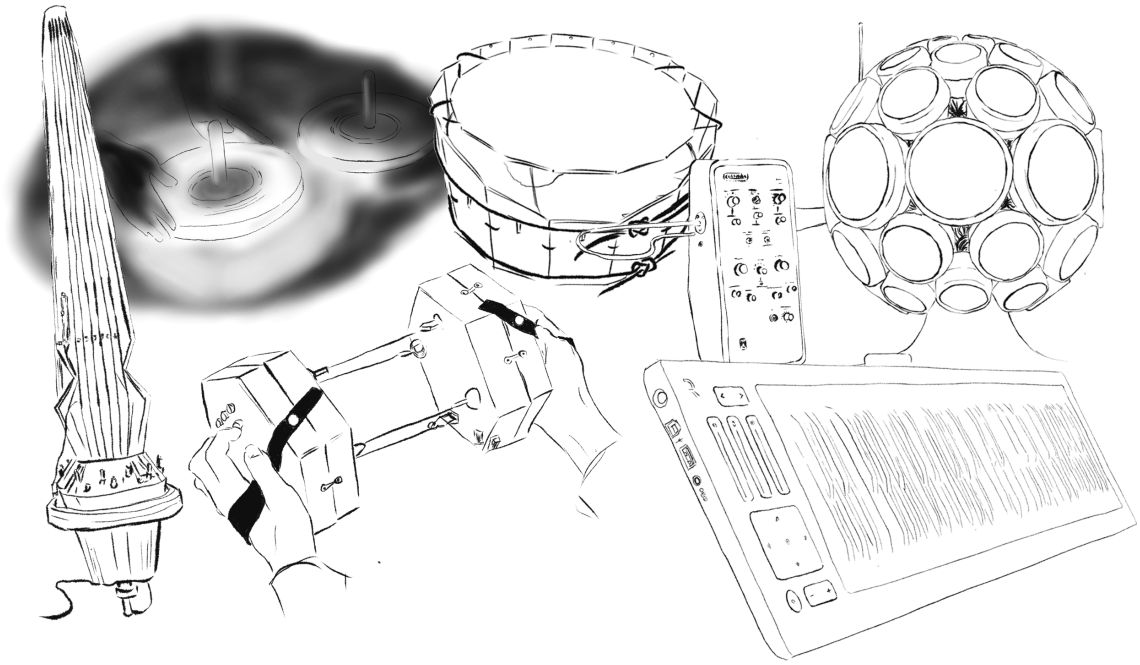


Figure 4.3: Knurl, Soft Revolvers, Concertronica, The Daïs, Claravox, Alpha Sphere, Roli Seaboard

1. Soft Revolvers
2. Alpha Sphere
3. Knurl
4. Artiphon Orba
5. Reactable
6. The Blade Axe
7. Mutable Instruments Beads
8. Claravox
9. Polaron
10. The Ladies Glove
11. Linnstrument
12. Concertronica
13. Abelton Push
14. Roli Seaboard Grand
15. OTTO
16. Electronic_Khipu_
17. The Daïs
18. EMG instruments
19. Gechologic Loopsynth
20. Bastl Kastle Drum
21. The D-Box

4.4 Interviews and Analysis

Following an internal pilot study with peers with DMI design experience, standardised open-ended interviews were conducted with 22 participants engaging with an interviewer via video call and five via email. Interviews had a duration of 20 - 60 minutes at the discretion of the participant. Interviews took place between 25th January 2021 and 1st April 2021. Participants were provided with a copy of the questions to use as a reference during the interview. The lead author carried out all interviews.

Interviews were recorded (audio only) and transcribed verbatim, then processed to ensure appropriate confidentiality and IP protection. In the case of email interviews, emails were formatted

to match transcripts.

Due to the overall scope and the open-ended nature of this study, it was clear whilst forming the themes for this work that in order to inductively construct a shared narrative of participants, the themes first needed to be focused around a more broad research question, before potentially being focused onto other questions later on. Motivated to explore the explicit semantics presented by participants it became apparent that the initial theme generation should focus on the designer-tool relationship and how digital luthiers select tools in a broader sense, creating a formative base to build from. This allowed for a second analysis to draw on coding in related areas that could build upon prior themes, further exploring programming languages more specifically. As a result Chapter 5 explores the initial analysis which is followed by a secondary analysis in Chapter 6 which narrows the research scope toward programming languages.

It is notable to suggest that following the second analysis there are still groupings from the coding that can be further explored in other contexts as they were largely not incorporated into themes in this work and these will likely be explored in a future publication focused on the design philosophies of digital luthiers.

Chapter 5

How do Digital Luthiers Choose Their Tools?

From the 27 participants interviewed, three themes were generated that provided narratives to perspectives of the digital luthiers that address the research questions of this study. To contextualise quotes, we reference the participant ID found in the transcripts (e.g P7 for participant 7). Where relevant, we will mention data provided by the participant that is accessible as metadata in the transcripts or within the data repository[1]. For more details on the published data and metadata, see the prepublication. Excerpts from transcripts may be edited from the verbatim transcripts for ease of reading (whilst maintaining meaning), with square brackets indicating notable edited words.

5.1 Theme 1: 'The Pragmatist'

'The Pragmatist', captures the prevalent tool selection approach conveyed by many participants. Participants tended to see themselves as having pragmatic motivations for tool selection, such as choosing the tool or programming language best supported on their target platform. In contrast, the pragmatic choice for others is using a language in which they are proficient, which may be less suitable for the platform but saves time overall. This theme portrays the shared experience that designers tend toward making decisions with a cost-benefit analysis based on their own experience and the technology they interact with.

The theme takes its title from P1, who referred to being pragmatic throughout the interview:

> "So I would say I'm very pragmatic with that. If the question is why, I mean, it depends [on] so many things, on what's available and what you can do." (P1)

P23 also shares this terminology, describing their pragmatic decision-making process.

> "No, I think, my choice of language at that moment, I must say, it's very pragmatic. And it's, it perhaps isn't for the elegance of the coding act, but for interoperability, platform independence. And so there's a natural tendency to go towards just tried and true languages like C or C++. It's been very interesting to see how a slightly higher level language like Python has really evolved in the past few years to become a kind of standard for signal processing." (P23)

P1 and P23 both use the term 'pragmatic' to identify their motivations but focus on different technical requirements demonstrating that each project requires the weighing of specific characteristics rather than a common path of least resistance for all instruments. For example, P23, a

[1] https://github.com/muses-dmi/dmi-design-study

researcher, notes that platform independence is valued in their tools. P1 is also a professor but notes that due to the time in which they worked on their instruments, the primary consideration was what technology was available and could meet the required goal at all. In their interview, P1 goes on to suggest that there is much more choice for tools now. P15 builds on this, also referencing interoperability, this time in the context of the now rich existing ecosystem of technologies that already exist. When asked why they chose their tools, p15 mentions:

> "But also because of how it integrates with other tools like Max, for instance, because you can write your own externals in C or C++, like, audio plugin API's and so on. Right. So this would be the main reason why I, or people I work with continue using C++, I believe." (P15)

Aside from performance reasons, P15 suggests that C and C++ are pervasive in digital musical technology in large part due to interoperability. They also provide more personal insight, suggesting that fluency is highly valued.

> "The other one would be Python. Because first of all, I think this is still the most relevant reason, I am fluent in it." (P15)

This suggests two sides to the pragmatic designer. One that accounts for technical requirements of the system they are working with, such as those introduced by P1 and P23, and one that considers the individual capabilities of the designer (or designers), such as existing skills and knowledge. We look at these as the ideas of technical and individual pragmatism, respectively, representing two sides to the pragmatic designer described in this theme. Participant 15 emphasises the point that this fluency is a powerful motivation in choosing a tool that speaks to the capacity of being able to express ideas using the tools available efficiently, a principle that resonates with many other participants:

> "...the properties of the language lead you to be able to be expressive and create an instrument of different expressivity more directly and so I think the choices are deeply related." (P19)

> "I think I've talked about my main goals in digital musical instrument design are to be immediately expressive and is the reason i choose to make digital musical instruments as opposed to analogue." (P26)

> "As I said, a language or something for C++ or something that is [performant], but still plug and play and very expressive for audio would be would be great." (P17)

> "...not much profound to say here, it is kind of familiarity causes productivity." (P16)

Across a number of participants, expressivity resonates as a strong motivation for their tool choice. P16 phrases this as a capacity to be more productive, whilst P19 emphasises that this capacity leads to a deeper intimacy with the instrument that is produced, even implying that the expressive power of these tools can directly impact the instrument's expressivity.

We see these two sides of pragmatic decision making weighed up by P26, who considers the investment learning a new system takes against the potential returns.

> "...but they call them programming languages for a reason and learning each new language or each new interface takes more time and so if the i guess the main thing is if the promise of the efficiency of the new device or platform is worth the time and energy it takes to learn it..." (P26)

Many participants couple their expressivity with efficiency, ultimately suggesting that they benefit from better productivity when familiar with their tools. This implies an apprehension to use tools that feel less familiar due to potential loss of productivity. We see P16 grapple with this when working with the programming language Faust, which offered value; however, due to less familiarity, it ultimately lost productivity when working with the predominantly C++ codebase of their instrument OTTO.

> "But like I mentioned Faust, like, yes, it would be it's great to prototype. prototype. But at the end of the day, when I then have to translate that to C++, and it's not just it can't be, it can never be a one to one translation, having to redo a bunch of things, makes it so that I don't really end up saving any time compared to just making it in C++ from the beginning." (P16)

Participants tendency to make pragmatic choices appears to be primarily driven by one major perceived constraint on their design process. Whether the limitation is a deadline for a commercial release or is the desire to be rewarded by a sense of rapid expressivity as described above, participants across backgrounds appear to make practical choices that improve their efficiency. Therefore, a significant component driving pragmatic decision-making is the limitations of time.

> "But ultimately, the biggest challenge of making an instrument is, is time. Everything else follows after that, like building something good enough that someone else wants to play it again is a function of time." (P16)

> "Another important point in the approach to a new project is the time. This is the first time I'm working without deadlines.
>
> In the past I'm just working by the deadline. " (P21)

> "...and therefore, in the biggest challenge is usually time you have a lot of ideas, but then when you need to, make both mechanics and electronics and software, all that stuff takes a long time..." (P22)

Interestingly, this core constraint is shared across the backgrounds of participants. It therefore becomes a pragmatic choice to employ tools that offer efficiency in use. Participants such as P16 demonstrate the relationship between the individual pragmatism of utilising their skills and time. They present a view on how the maturity of documentation and learning resources can actively be a barrier to tool selection:

> "...if Faust was as developed as it is now when I started doing my PhD, I probably would have like, played around more with domain specific languages ."(P16)

Expertise is a limitation relating to time mentioned by many participants in that it takes considerable time to learn new skills relevant to DMI design. Accessible and well-documented technology provides a well-represented solution for this to the participants in the study:

> "I use Bela and Arduino because they are very well documented" (P18)

> "[On documentation] Yeah, yeah. It has to be also very elementary, like, really basic level so that, because even when you understand stuff, oh, what's a sample rate. All right. I know that in another context that you are in a flow or, you know, it's it can be a bit too much sometimes." (P27)

> "Beyond that, well-typed and well-documented APIs are really important. That's one reason I love Rust, it has great documentation tooling and a culture of aggressive documentation..." (P20)

In summary, 'The Pragmatist' provides a narrative shared across backgrounds in which the participants pick their tools according to a cost-benefit analysis of what practically impacts the project. We describe two distinct sides to this: one that prioritises technical choices, typical in more commercial settings, and one that prioritises the individual's practical limitations. For all backgrounds, we see that both sides are factored in and considered, and the exact criteria for the pragmatic choice are motivated to address the perceived constraints. This theme indicates there is a clear appreciation that participants choose tools to support these practical considerations and address the predominant constraint of time.

5.2 Theme 2: 'A Product of our Environment'

This theme examines how environments such as schools, communities or industry practices influence designers' tool choices. Irrespective of background, participants were very aware of their environment's influence on their choices, and many attributed their educational institution to be a significant factor in the tools they use, particularly in the context of their education.

> "I've learned C and C++ at the university around the year 1996 and thankfully it's still the most widely used language in embedded systems." (P24, Hardware & Software Engineer)

> I'm finishing a second master program in Electronic Arts and we must learn about all these tools, so this is the handiest world I have right now. And of course, this is a discussion and influence in how I form my work (P21, Artist)

For P11, the influence of their educational institution leads to their early adoption of the programming language Faust. Whilst two other participants also mention Faust, P11 is the only one that mentioned Faust being taught at their educational institution. Faust is a language based on the functional programming paradigm and a domain-specific language for audio processing. The functional paradigm is a departure from many more common languages digital luthiers use, which are more general-purpose, imperative languages. Whilst this does not suggest much about the suitability of these paradigms, the use of a tool that contrasts the tools others use helps to make the influence of education quite distinct.

> "So I think I started using Faust for geographical reasons. Because I did my undergrad in France, at the birthplace where Faust was created." (P11)

P16 and P17 both mention using Faust. However, neither adopted it long term, despite speaking highly of it. Ultimately, P11 developed into a role as a language developer for Faust. As such, the environmental pressures for using that tool have led to them using it extensively.

A reluctance to change their tools from the ones they learned initially seems common. Having been introduced to tools through education, P26 found little motivation to change the tools as the current tools serve their needs, though they do allude to the curiosity of other technologies. We see across these participants an exploration of affordances and constraints similar to those described by Magnusson (Magnusson 2006), largely presented in this context by the environment.

> "...I haven't used Bela because the tools that I learned, particularly when I was an undergrad just continue to work for me, but I am curious about other platforms..." (P26)

Much like educational communities, the open-source community represents a highly influential environment that impacts participants. We see that open source tools represent an ideology with which participants identify. Both P11 and P27 share their preferences for open source tools.

"...And it's just like something that needs to be 3d printed, for example, then I will use [it] and, I am an open source person, you know, like so. So I prefer to to use open source tools. ..." (P11)

"I choose because it's open source mostly. It's also what's my education has given to me in the conservatoire." (P27)

We see that the open-source community's influence and surrounding ideology plays into the environmental influence to which the individual is exposed. P27 is very aware of how their particular environment influences their tool choice. They observe that their environment is made up of various influences, implying that it can be challenging to move from the initial tool introduced through education, even balanced against an ideology to use open source tools.

"Relating to hardware, you may [say] it was a consequence of environments mostly. I am aware [that] Fusion [360] is not an open source thing, although I said in the beginning I want to use open source. So it's more emotionally about my environments right now as I say..." (P27)

It is clear then that there is a complex interplay between many factors of an individual's environment that have a significant hold on the tools participants might use. We see that for participants such as P26, the ability of open-source culture to facilitate knowledge exchange is the most important aspect, further blurring the boundaries between educational communities and open source.

"...I think open source culture is really important and I have the career that I do because of google, like hands down, I wouldn't know anything that I know if it wasn't for the internet... ..." (P26)

Participant 3 describes two external influences on tool choice, community and University, respectively. This further demonstrates that the environment is, of course, not a homogenous entity, but a blend of the environments to which the designer is exposed.

"And also, I went to university in Ohio, and there were classes in Python and in pure data. So I guess, like, the fact that I was exposed to those things at school, like made it easy for me to use them in my projects as well...

...And yeah, like the Fab Labs, in Montreal, were sort of my way in. And then like, from getting into those communities. And also with like, people around me there, were also interested in like, doing 3d stuff and whatnot. I learned about different software." (P3)

Participants often make decisions based on the availability of technology. Particularly in the case of participants who were working early in the history of DMIs, the idea that *'it was what was available'* was a common sentiment, meaning designers used whatever the most suitable, available tool for the job was at the time.

"However, in the 80s I was using C because that's the only computer I had. And the only compiler I had was a floppy of the C language." (P1)

"In fact, in the early days of my early machines, I used only assembly language. But then C finally became barely fast enough around for embedded processors around the 1990s. So I would say that, pretty much everybody just uses C, with a lot of optimization of loops and things like that." (P9)

However, availability remains relevant in more recent examples too. Notably, P2 qualifies that they chose the 'only mature' language, combining their environmental challenges with those of pragmatism and a need for reliability that participants refer to in Section 5.1.

> "C/C++ was the only mature programming language available on STM32F when I started developing the Eurorack line" (P2)

Despite the developments and richer ecosystem now available, this suggests that even today, the range of tools is quite heavily constrained by the technology the designer works with, necessitating the selection of tools from a suitable range. For P19, this relates to their workflow; beginning their process by considering the technology available to them, they immediately impose a set of constraints on the available tools.

> "And so I look at the environment in which that instrument has to sit, you know, it's got to be able to deal with MIDI, and it's got to be able to do this, and it's got to have and the constructs of like, well, what position does it take in the musicians Arsenal? Do I have the luxury of building like a two by four box made out of steel, that's gonna be the centrepiece or is this, like, you know, something that's guitar pedal size, I mean, these things have an influence on your hardware. And then working back your hardware generally dictates a very limited choice of software environments in which to work in." (P19)

The sense of availability described by participants appears to relate to the environment of the digital luthier rather than to the global availability of technology. But for some participants the tools and technology available are more than adequate:

> "I think the tools available nowadays are very solid and well designed, both from the users' and developers' perspective." (P18, Professor)

> "I don't feel that anything is seriously lacking among the tools we have. For each task there are multiple choices / providers so it is easy to replace what is not working." (P24, Hardware & Software Engineer)

> "I'm just pretty satisfied with what I have. I mean, I could complain for hours with how are you using it?" (P4, Artist)

> " I think it's the best time so far in the whole world to do this stuff as an individual." (P5, Software Engineer (open source))

Artistically, P8 notes that this itself can be distracting. As they discuss setting out with a clear intention in mind, they see tools tending to cloud artistic intention, causing them to get lost due to the available options.

> "... there's all these new advances endlessly. And technology's fascinating. So you end up going down endless streams...
>
> ... I already think all in their nature, all the software tools probably lie on the negative side of the previous point that was made there. There's that whole idea of artistic limitations being a sort of thing that we should embrace." (P8)

Another substantial environmental influence on tool choice is industry. 'Industry standard' is a prevalent term indicative of the impact industry has on digital luthiers, particularly those with commercial backgrounds such as P7, P12 and P10. We see that these participants look to industry practices to support their decisions around tool choice.

> "It mostly comes down to accessibility, familiarity, and cost. We look for tools that would be considered the standard for that task, within our specific industry.
>
> ... Due to the above mentioned options, STM32 has become somewhat of an industry standard for DMI, further perpetuating itself as the ideal solution due to devs familiarity with the platform, and the plethora of example projects." (P7, CEO)

> "I've continued to use like, different IDE's, because, like, you know, the industry standards for the various use cases that we're looking at. Right." (P12, CEO & Founder)

> "In members of the team who were used to, you know, who are doing aerospace or, or other even industrial kind of firmware, and so C was a shared language there on the on the app side...
>
> ... it was honestly just about the personnel, the people in our network or on the team and their familiarity.(P10, CEO & Founder)"

> "C++ has also been the language I used the most previously in my career." (P2, Company Owner)

When specific features of technology such as performance is highly valued, there tends to be a convergence on tools within the industry. This has formed a consistent image of what tools are appropriate across participants. We suggest this common focus may contribute to a feedback loop where more adoption leads to more development and support, further entrenching the sense that a tool is industry standard.

This is suggested by P6, who indicates how only given considerable development can a new tool become viable when discussing their delayed adoption of the programming language Swift. For P6, a designer with a commercial background, reliability is critical in the tools used in a software engineering role at a large instrument company.

> "I've gravitated towards maintainability. Because whenever you create any type of product that is software based, you will spend way more time dealing with the life cycle of the product and with the initial creation spike. So even though I have a natural affinity to cool stuff, and like emerging languages and new new ideas, I've been bitten quite a bit in my early career by choosing tools that weren't completely safe, stabilised, and then having to spend a lot of time dealing with problems that are introduced just to to evolve in programming languages or programming platforms." (P6)

Clearly, support for tools also factors into this sense of industry-standard, whether that be from hardware vendors or the open-source community. This support directly influences the industry standards:

> "Software tools are usually supplied by chip manufacturer, in my case all necessary compilers and IDE were freely available from STMicroelectronics & Atollic, Espressif, or as a free open-source software from 3rd party (Eclipse)." (P24)

In summary, we see that participants have unique social and cultural environments in which they work. Some of the clear examples of these include education, open-source communities and industry. The environment presents a set of pressures that have a significant influence on tool selection and we suggest that these environments also act as a buffer that preserves an intuition of the affordances and constraints of a tool, giving rise to notions such as *industry standard*.

5.3 Theme 3: 'Intentions'

Inevitably, there are unique pressures on tool selection related to the intention of the digital instrument being designed. This theme, titled 'Intentions', reflects how each designer also makes special considerations in tool choice related to the purpose for which the instrument is being designed. Here we see the stories of designers converge on the requirements dictated by how they intend to use the instrument they are creating. In this theme, we see that some of the ideas presented are outside the design problem space and focus on more peripheral issues with creating digital instruments.

P12 describes their goal to have a "mastery of production", explaining how tool choice extends beyond the design space and facilitates more typical manufacturing and collaborative design processes through communication.

> "And I want to do is like to have mastery of like, production, right, like, but production, I mean, like mass production. So, some of the tools are even like that communication with the people that run the factories ... I consider that as part of the, that's part of the instrument, that you're ultimately putting into people's hands." (P12)

We see others from commercial backgrounds build on this using a large variety of tools to coordinate and collaborate even in remote settings, emphasising the importance of selecting tools that facilitate their specific requirements for long-distance collaboration that operate in real time.

> "we are pretty much always on video, and in collaborative documents as we're doing the design. So it is a very real time design and development process for the team. Compared to just handing over documents, you know, as explicit files, these are usually collaborative real time documents. I think it makes a big difference." (P10)

> "These challenges together comprise the project management aspect of designing something. Tools vary, but it is important to use some sort of task management system(Trello, Asana, etc.) and a calendar." (P7)

In team settings, to meet an instrument's goals, additional communication and collaboration tools supplement more specialised design tools to enable operation that allows more people to work together more quickly.

Typically, for individual digital luthiers, their role tends to be less clearly defined. As an artist, P4 describes the challenge of managing the time spent between designing and working on the instrument and performing with the instrument. This is particularly relevant to those designers who are both performers and inventors of the instrument, an attribute that is common in the digital lutherie community (Jordà 2004b).

> "So it's really hard because you have to really focus on what you want to do. You know, I cannot, you know, I have to really be very careful. Do I spend more time in design and writing software? Or do I spend more time in actually applications of the software for performance?" (P4)

In particular, we see that the boundaries between the design tools they choose and the instrument itself are often blurred. For artists, tools are often part of the process, as P3 notes, where the idiosyncrasies of tools can help to guide the design.

> "So and I like that process of going through the difficulties of and also like, each platform has his idiosyncrasies that direct a bit, how are you going to use it, and I like that aspect of like, working with the code or working with a programming platform, and then like, it co constitutes to kind of work that you're going to be doing on those platforms." (P3)

This relationship can make it challenging to provide a taxonomy of roles for the tool designer, instrument designer and performer. However, it does imply interesting considerations of meta-design in this relationship whereby design choices are deferred between roles, allowing more multidisciplinary roles as discussed in the context of Armitage and McPherson's work (Armitage and McPherson 2018). This is well exemplified in the context of P23's research and instrument design, who works to expose low-level features at a high level, such as "signal processing, feature extraction, [and] some machine learning".

> "...we're working with people who create tools, and we try to design those tools so that they're available to the musician in the high level environments." (P23)

In particular, we see those with artistic motivations to be leveraging this deferred capacity for design the most, utilising visual patching languages that act like musical ecosystems (McPherson and Tahıroğlu 2020) and also machine learning such as in the work of P3. These tools allow the performer to take charge of design elements of the instrument, blurring the capacity of where the tools stop and the instrument begins.

> "I think that the neural net thing is, is a great improvement for me, I can feel. I feel like I react to what is there." (P3)

P11's research work directly involves the design of tools for creating DMI, in their case, working on the audio digital signal processing language Faust. This supports the image of the multifaceted roles digital luthiers have, and P11's deep adoption of the tool they work on demonstrates how many digital luthiers develop a deep understanding of their tools and the musical intentions of their work, but also how other designers might work.

> "But like, mostly I write things and Faust because, it's quicker for me to write them in Faust than in C++" (P11)

We see many other participants reflect this component of tool design and instrument design as they work on their software libraries. For many, the intent of their work becomes not only producing instruments for performance but also as a means of developing and sharing tools. This likely relates to the way participants identified regarding open-source culture.

> "So I have this, this library that I use, which is actually my own prototyping library that I made in C++. And [...] gives you like, real time audio, in one line of code pretty much, and then a super simple UI library, so that I can quickly prototype my audio algorithms." (P22)

While for some artists or researchers, the instrument they designed served a singular or personal role, the broader adoption of an instrument was the intention for many instrument designers. Participants describe this as a significant challenge for digital instrument designers.

> "The reception by the public of anything that is not directly
> recognizable as a Moog/Buchla adaptation. The weight of the tradition,
> and 'groupthink' embedded in clichés and concepts such as 'menu
> diving', 'digital coldness', 'presets'." (P2)

As such, participants emphasize incorporating user feedback into the design process.

> "Third step is to make an early demo, share it around, and get feedback from potential users, which often makes us to rethink it - add or remove a thing or two. They often ask questions that show us what is exciting, what nobody cares about and what is missing - at this stage there us usually enough time to improve and fix most of these things." (P24)

P15 describes how this requirement factors into tool choice and describes how a means for easily exporting their work into a common and widely supported format would be desirable in their tools.

> "Could be like, I don't know, like, maybe even something that quickly enables me to export my prototypes to like a plugin in a known format is already useful, right? So because also, for me, it's like, then it can be distributed to many users quickly for testing with users. Getting frequent user feedback is super important to me." (P15)

This is motivated by a desire to be able to "drop the idea when it's not good early". This user-focused design is an important motivation recognised by many participants who intend to create an instrument to be used and adopted more widely. In order to get feedback on their open-source development, P17 has leveraged social media platforms as a tool to help with both feedback and the future adoption of their project.

> "That that the the open source has been a joy. Yeah, for sure. Yeah, yeah. It's also the like, we've been mentioned a bunch of times on Reddit, like on the synthesisers subreddits and stuff and which has gotten us some attention." (P17)

For a researcher such as P16, instrument design can be a process for explorations in the instruments' craft. They describe their tools as facilitating different capacities to explore the design space from a more abstract perspective.

> " What does that mean in a, in a digital lutherie context. So you know, you need, you really need tools that can access that level of resolution that you're interested in. So and the majority of research and platforms have all been at a higher level, at a lower level of fidelity than that, until recently, I would say in the main, you know, Arduino, and Teensy, and whatever they are low resolution platforms, but they're cheap and quick and dirty, and they allow you to explore like, the breadth of the design space as opposed to the depth of the design space" (P16)

As a researcher, participant 23 also reflects on an intention for their work that relates to the digital component of their work. As with other digital technologies, digital instruments are susceptible to so-called *bit rot* (Król and Zdonek 2019). P23 recognises that to achieve their goals, their work should be reproducible "in another era by other people." This provides an essential motivation for tool choice that relates to making pragmatic decisions for interoperability described in Section 5.1.

> "...that's why the tools. [...] Why is that? That they're nice, I can describe the kind of interoperability over time, in that I have to perform a concert programme, sometimes with works that are over a decade old, alongside a piece that I've just written the other day, and they and they have to load up and don't have accounts to work with more than one system." (P23)

This theme describes the participants' unique tool choice requirements that serve the long term goals of the instrument they are designing. For commercial development, extra attention is given to more generally used tools that aid the realisation of 'products' capable of production at scale. These strategies are also employed to a lesser extent in their group work scenarios, where the skills of a team can be effectively employed and integrated through the use of tools that support collaboration. Other goals such as exploration of instruments in research and realisation of artistic intentions can lead to influences on tool choice that runs counter to productive outcomes, where limitations can become part of the workflow. Motivations for the instrument a participant is working on range from the development of digital lutherie tools themselves to many other forms of music-making (Small 1998). These narratives represent motivations for tool choice that track directly with the motivation for creating the instrument.

5.4 Discussion

Through an iterative and reflexive approach, we took transcripts from 27 interviews and provide a discussion and interpretation of the perspectives shared by the participants. Working inductively, we found many compelling observations that generalised from different participant backgrounds. We focused on the perspectives shared across participants with different motivations for design, aiming to address our research questions.

In summary, we see in Section 5.1 'The Pragmatist', a narrative is described whereby designers make their decisions based on analysis and their experience with a problem space (Goel and Pirolli 1992), selecting tools with attributes that solve the most significant challenges to them. These challenges are often problem-specific, so the value of attributes such as performance or maintainability is skewed according to the designer's requirements. For example, we see that those in commercial settings place explicit value on tools that support easily maintainable products, particularly when discussing code. Alternatively, those with limited technical expertise, such as an artist exploring instrument design, may prioritise well-documented tools as this facilitates an accessible form of support to help them achieve their goals. Across the study, we see that these pragmatic choices hinge around some of the following points:

- Performance
- Interoperability
- Ease of use
- Accessibility
- Availability
- Familiarity/efficiency of use

We also present the theme 'A Product of our Environment', where we interpret that external cultural, societal and institutional influences all impact on a designer's choice of tool. We see how notions such as 'industry standard' impact tool choice for those managing a team in an industry setting, but also in other contexts. There are examples where industry standards influence the choice of programming language at educational establishments, and ultimately, this popularity influences the available support from a mature community. We also see how open source as a community and shared ideology (although complex and open to interpretation) provide an environmental influence that is drawn on and used across design motivations. All of these environmental factors provide a push and pull influence that substantially impacts the tools used by the participants in this study. Ultimately, it is apparent that the tools provided by a university or endorsed by a community tend to affect tool selection significantly, and we suggest that this may be the overruling factor.

Our final theme, 'Intention' accounts for the salient challenges presented in the domain of digital musical instrument design. Despite the focus on addressing technical challenges through pragmatic tool choice, very few suggested these domain-specific problems constituted the most relevant challenge in DMI design. While considering the latent meaning of some interviews suggests there are prevalent challenges and demands of DMI design related to the interaction of controlling the instrument, we focus on the more explicit semantics presented in this study. This suggests that the most significant challenges digital luthiers face are related to how the instrument is intended to be used. This can motivate tool selection in a project-specific capacity. For those working in teams, tools may needed to either support collaborative features or be supplemented by other tools that do. For those looking for widespread adoption of their instrument, tools that allow for integrated user testing become critical, and intentions for artistic output can require tools that can facilitate

the reproducibility of technology in the distant future. For many, the capacity for some component of the instrument's design to be deferred to the user are also desirable. 'Intention' describes the pressures on tool selection that are unique to each project and driven by the motivations of the digital luthier to build an instrument.

5.4.1 Why and how do Instrument designers pick their tools?

Our themes show that across different contexts, the motivation for tool selection can be described with three shared narratives. Much like Stolterman and Pierce (2012), our study finds that a primary factor reported for tool selection is a rationalised selection process. They describe this as following a model "*in which one selects an appropriate tool based on a clear understanding of the design situation, the desired outcome of the situation, the types of activity needed to reach that outcome, and the types of tools that can satisfy the desired outcomes of the situation.*"

Our narrative around pragmatism very much corroborates this finding, where terms such as *interoperability*, *performance*, *accessibility* and *efficiency* highlight the rationalised decisions made by the designer concerning their problem. In addition to this, we find a second narrative comparable to Stolterman and Pierces findings. Stolterman and Pierce discuss environmental factors in a more specific capacity, considering community, culture and branding in relation to the designer's identity. Our interpretation frames the environment as external influences that incorporate the cultural and personal identity, but also includes more overt external influences. A strong example found in this study is the impact that educational institutions have on tool choice. Whether due to simple availability or increased accessibility due to pedagogical support, educational institutions' presentation of tools accounted for the initial use of tools and continued use of the tool in many cases. We see some scenarios where the tool is not expressive enough and is therefore outgrown (for example, using graphical programming environments for programming). However, the consensus demonstrated by participants is that designers tend to stick with a tool offered up by environmental factors as long as it meets their pragmatic requirements.

Together we see these pressures form a set of both affordances and constraints, a concept well established in HCI literature, based on work by Gibson (Gibson 2014) but reformed for use in design by Norman (Chong and Proctor 2020). Whilst in HCI, this term has mostly been used to explore the affordances objects offer to users; considering affordances as it evolved from ecological psychology, we see that the theory of affordances has been discussed in the context of properties and environments before. Chemoro's definition of affordances demonstrates the relationship between the features offered by the environment and a person (Chemero 2003). In light of our themes, we suggest that affordances are a valid way to analyse the tool choice of participants. The work of Magnusson has extended this potential and examined DMI design in the context of its limitations (Magnusson 2010a), which was a more prevalent topic in our study. Magnusson suggests that whilst learning a new DMI, people explore affordances. However, the majority of the time, learning the instrument "involves building a habituated mental model of its constraints." We suggest that whilst most digital luthiers in this study were likely experienced beyond the stage of initially exploring their tools affordances, the constraints of tools are preserved and shared by communities in the form of tacit knowledge. As such, we suggest that how digital luthiers (designers of DMIs) select their tools can also be considered in a similar capacity to how performers select their instruments.

A digital luthier may explore the affordances and constraints of their tools, evaluating them in order to meet the practical criteria that the instrument requires. Where the tool is widely understood to have a set of affordances or constraints, the community preserves and shares this knowledge, influencing tool choice for other digital luthiers in turn. For example, the idea that C++ is good for performance. It is not new to see this kind of application of the affordances moved to a

different level of the design process as DeNora examines it in the case of music sociology, considering what is afforded to the listener (DeNora and Adorno 2003). We see that both affordances and constraints should also be relevant for considering the designer-tool relationship of digital luthiers.

Stolterman and Pierce interpret designers' tool choices using the concepts of espoused theory and theory in use (Beck and Stolterman 2016). The manner in which we see environmental factors 'complicating' designers' espoused theory of 'pragmatism' may add support to this interpretation. The self-reporting nature of this study allows participants to inaccurately reflect their motivations, offering the potential for developing espoused theories of the designer-tool relationship, that do not fully reflect the reality of why they choose them. This study does indicate that digital luthiers do reflect on their tool choices and, due to their relation to projects they currently work on, do describe scenarios that suggest theory in use:

> "And because I'm working in an embedded environment for my instruments, it's their C/C++. Not because I really like them or not, not even that I'm really strong at them. But I think it's just more or less the only way I can get the results that I want." (P5)

Our observations show that digital luthiers tend to be particularly aware of the environmental influences that affect their tool selection, implying less disparity between their espoused theory and theory-in-use. We see an example of this when P27 reflects that they currently use a tool that goes against their ideology of using only open-source software (as discussed in Section 5.2). This self-awareness and reflection only complicates the relationship between designer and tool and is essential to consider in future work.

5.4.2 What distinct problem spaces do instrument designers consider to be involved in instrument design?

In answer to this research question, we have focused on the challenges faced by participants and how they related to their tool choices, a first step in building a model of the problem spaces that digital luthiers face (Goel and Pirolli 1992) when designing DMIs. Despite the differences between participants and the instruments they design, we have primarily presented shared narratives that are developed from this study. Of course, each instrument and the context in which it is developed has many unique factors affecting tool choice. We found that where these focuses differ, they are mostly tailored by the major challenges they face outside the domain or designing the instrument itself. Many participants use tools for support outside of collocated settings, typically finding additional tools to fill this role. For example, commercial teams and research teams discuss needing tools for collaboration. We see this observation reflected in wider HCI literature across disciplines, showing that the importance of social design should be reflected in a designer's tools (Jung, Lim, and Kim 2017; Alcántara, Markopoulos, and Funk 2015) Tools could therefore benefit by better supporting this workflow or integrating with other tools that do.

This need for remote collaboration extends beyond structured 'team members', however. Due to the importance of meeting the demands of the performer, user-focused design is vital for any instrument, a perspective clearly shared by participants. Of course, co-design, participatory design, and user experience are well-explored facets of DMI literature (Fyans et al. 2012; Brown, Nash, and Mitchell 2017). The importance of these considerations leads digital luthiers to call on their tools to facilitate the interaction between the user and themselves. Much like working in an industry team, those looking to engage with their instruments end-user require a seamless collaborative process. For participants designing instruments intended to be played by others, they clearly indicate a need to support social creativity (Fischer et al. 2005) in their tools in order to realise those instruments effectively. Fischer et al. recognise that individual and social creativity represents a continuum that should be integrated for solving complex design problems. The combination of challenges in

user experience and facilitation of social creativity culminates in digital lutherie situating itself as an ideal candidate for meta-design (Fischer and Scharff 2000). Due to the way digital luthiers tend to take many roles as designer and performer, Fischer's description of meta-design as a "coadaptive process between users and a system" seems an ideal fit. Meta-design allows for components of a systems design to be deferred to the user. P9 approaches this by making parts of their instrument open source to support this relationship.

> "One of the problems is it's such a new idea and so flexible, but a lot of people want to make it into the ideal instrument. So early on, I decided to make the software open source. And I wanted to make open source development as easy as possible." (P9)

DMI tools are often developed to encourage and facilitate meta-design (Calegario et al. 2020). For some participants, graphical patching tools such as Max and Pure Data can be considered a part of the instrument rather than just a tool, exposing the capacity for the instrument to be redesigned continually. This is particularly common for artists, such as P4 and P26, who are challenged by continually finding ways to express their artistic intent for new ideas and works. It is common in DMI literature to represent a DMI abstractly as a controller and sound generator with a mapping between the two. Many projects look to expose this relationship, constituting a meta-design relationship between the tool and digital luthier, who, as Jorda describes, typically embodies the instrument maker and the performer (Jordà 2004b). We consider that the field of digital lutherie might be one of the most developed examples of meta-design and that it naturally addresses some of the most significant issues faced by digital luthiers. Fischer's work on meta-design sees it as a tenet of end-user development which is described as a "society-changing invention" (Fischer 2021). This is clearly paralleled in the DMI community, and we suggest these two areas of research support each other and would do well to cross-pollinate.

Participants also indicate that limitations of time, resources and expertise are also some of the most limiting factors they face. We see that these challenges are largely coupled to the practical motivations in tool choice set out in Section 5.1. Participants search for tools that offer efficiency and accessibility in response to these limitations.

Finally, for some participants, the greatest challenge represents the steps beyond the creation of the instrument. The adoption of new DMI is something already considered in the literature (Marquez-Borbon and Martinez-Avila 2018; Jack, Harrison, and McPherson 2020; Morreale and McPherson 2017). In commercial settings, this can involve marketing, communication with a community, and even breaking down preconceptions of the user base (as in the case of P2). As this study focuses on the designer-tool relationship grounded in digital lutherie, these aspects are difficult to build upon. However, participants introduced these as prominent issues, which could be further investigated in future work.

5.4.3 How do instrument designers define a digital musical instrument?

Of course, the challenges in defining a digital musical instrument are well considered in the literature (Morreale, McPherson, and Wanderley 2018), and this ambiguity is well reflected in the participants of this study. Participants tended to create an ad hoc definition that they often recognised as contrived or quickly contradicted. There was a strong tendency to note that providing a definition does not matter. Any boundaries are quickly blurred by the continuum that represents controllers, interfaces, and instruments. A deep dive into this topic is beyond this work; however, drawing on the analysis from this study, we note some interesting and related points. Many participants value the observation that DMIs are not bound by physical properties, with P6 suggesting there is no limit to their potential.

Some participants view the digital component of DMI to be just another material used to create an instrument. Participants also considered aspects such as *idiomaticity*, *expressivity*, *feedback* and *virtuosity*, of importance in DMIs, which are well explored in literature (Moro and McPherson 2021; McPherson and Kim 2013; Nash and Blackwell 2011; Tahiroglu, Gurevich, and Knapp 2018; Berger 2010; Brown, Nash, and Mitchell 2018).

We also see many participants make a comparison of DMIs to a mathematical function. P2 provides us with the notion of $y = f(x)$:

"So in the end, it is just about evaluating a big outputs = f (inputs) function. (P2)".

This is possibly expanded by comments that suggest that DMIs are more than simply the combination of controller and sound generator (P1, P6), suggesting there is importance to the process of mapping between input and output. In this chapter we have focused on the designer-tool relationship where this question was formed to help understand what digital luthiers were motivated to create. We see through this analysis that not only is the taxonomy of DMIs difficult to define but also, the taxonomy of digital luthier itself is not clearly seperated between tool designer, instrument designer and performer, with some participants spanning all of these roles.

Ultimately, this question does little to provide a taxonomy of digital musical artefacts, but we suggest these transcripts offer useful perspectives that would be well explored in future works that explore this question more deeply.

5.5 Conclusion and Future Work

The relationship between digital designers and their tool choice is complex and continues to call for more attention to be better understood. Despite the different settings in which digital instruments are developed, we find that digital luthiers share many motivations for choosing tools. Through our inductive approach, we come to a set of narratives that supports the work of Stolterman and Pierce who observe similar themes of 'rationalised reasoning' and 'the social, material, and cultural context in which the design process takes place'. We see these as pragmatic decision making and the influence of the environment, respectively. Stolerman and Pierce consider Argyris' theory of action (Argyris and Schön 1974) and relates the pragmatic process to espoused theories and the environmental aspects to relate to theories in use. We suggest this is a good opportunity for future work to explore through observational study as our findings further complicate this relationship. We find the digital luthiers in this study to demonstrate an awareness of how their espoused theories and theories in use interact, understanding that the environmental influences often override their espoused theories of tools. We also consider these influences in the context of affordances offered by both the tools and the environment in which they exist (Gibson 2014). Much as Magnusson has found in the use of DMIs, we find that participants focused on the constraints of the tools they use to design DMIs (Magnusson 2010a). Therefore, we find that in analysing the designer-tool relationship, designers engage with a tool's affordances and constraints. In light of the environmental context in which tools exist, our study suggests that the community preserves and shares knowledge where a tool is widely understood to have a particular set of affordances or constraints.

Where we do see specialisation in tools, our findings imply these relate most notably to the instrument's intentions. For a majority, DMI design is socially collaborative. This may be in the case of a team of digital luthiers, but often this is directly the users (performers) providing feedback or further developing a design themselves. For those instruments, Fischer's recommendation of integrating social creativity and individual creativity is demonstrated in the tool choices digital luthiers make (Fischer et al. 2005). This included using domain-specific tools that supported

social creativity as well as supplementing them with more general-purpose collaborative tools. Those focused on individual creativity specialised their tools towards efficiency and accessibility. This, too, engages social creativity less directly, engaging with quality documentation and mature tools provided by communities such as industry or the open-source community.

Finally, through understanding the challenges designers face and how this influences their tool choice, we recognise digital lutherie as a rich example of meta-design (Fischer and Scharff 2000). The literature surrounding digital lutherie and DMIs describes the fluid nature of the roles digital luthiers and DMIs have (Jordà 2004b; Morreale, McPherson, and Wanderley 2018). For many, the ability to defer an instrument's design components to the performer is a defining feature of a DMI. This has seen the use of many of the principles that Fischer (Fischer 2021) describes as EUD (End User Design) being applied in this craft (Fiebrink and Cook 2010; Calegario et al. 2020). This study suggests that these features are important considerations for designing new tools for digital lutherie.

Much like Stolterman and Pierce, we continue to show the complexity of the designer-tool relationship and continue the trend of prompting more questions. However, we see that the perspectives shared by digital luthiers present many directions for enquiry relating to how we interact and design digital systems. We emphasise that this work presents a foundation for further enquiry into the designer-tool relationship for digital luthiers and encourage others to utilise the transcripts available to supplement other methods that explore these ideas.

Chapter 6

What do Digital Luthiers value from their programming languages?

This chapter further analyses the study outlined in Chapter 4 pertaining more specifically to programming languages. These themes are yet to be featured in a publication. This analysis uses the same approach as previously detailed, primarily considering the explicit semantic meaning of the participant's comments; this time, focusing on coding around programming languages that did not feature so heavily in the previous analysis. This analysis inductively explores the question, ***'What do Digital Luthiers value from their programming languages?'***

This question looks to encapsulate the qualities or characteristics of programming languages that are desirable for digital luthiers and, as such, may be valuable to consider when designing new languages. Three themes are presented, which describe the role that digital luthiers expect the programming language to play or the requirements they facilitate. These themes are then interpreted in the context of existing literature and begin to develop ideas around how programming language constructs, features, and paradigms fit these desirable language qualities.

6.1 Theme 4: 'A Guiding Force'

This theme titled 'A Guiding Force' emphasises how the influential qualities of a programming language and its associated ecosystem is a highly valued language component. Through enforcing practices, educating the programmer or inspiring specific approaches to problem-solving, users shared the notion that the language should guide and support the development of their instruments. Often the language can be viewed as a collaborator that assists the digital luthier. Further, the shared idioms and community knowledge embedded in the language further influences and guides the programmer. The capacity of the language to direct the programming work of the digital luthier is often directly related to choosing a language.

P19 describes this influence and guiding force of the language as rigour that encourages writing more resilient code:

> "... i think it's really really important to be deeply rigorous about the code you write for musical instruments because you're going to ask that code to be stretched, that musician is going to ask that code to stretch.
>
> ... in other kinds of programming, there is often a very practical engineering middle ground of, like I know this task needs to only do this so the fact that my code can

> only handle you know 10,000 rows and not 10 million rows... in music, you do have physical constraints, but they're often really ... wide, like I want this low-frequency oscillator to be either like five hertz but you know the user might want it to be five kilohertz and that's three orders of magnitude which is a lot of difference in a piece of code. And so rigour in your programming language helps you get that right. If you do stuff in less rigorous programming languages, you tend to end up with sloppy timing and crappy stuff." (P19)

Often utilising languages that provide this stricter sense of guidance requires changes in the problem-solving approach. P20 describes the Rust type system as an example of this, which contrasts their experience with Python, ultimately leading to code that has more desirable outcomes:
Despite the fact that a stricter programming language is described as

> "Rust, on the other hand, is all about correctness. It has a robust and very strict type system - where in Python it's easy to get code to run, but might require a lot of testing to get it right, in Rust the type system constrains which programs are valid to a large extent. This means that one must put in a lot more thought up front, but the program that comes out the other side of the design and development process is much less likely to be buggy, and is almost certainly really fast." (P20)

Later in their interview, we see that P20 also incorporates the Rust environment as an extension of the language, describing how programming language features such as strong typing reflect the Rust community's attitude. Where strong typing represents a strict set of conditions being enforced in code, P20 suggests this rigour is mirrored in the culture of supporting the programmer's use of the language through tooling and knowledge exchange.

> "Beyond that, well-typed and well-documented APIs are really important. That's one reason I love Rust; it has great documentation tooling and a culture of aggressive documentation, though sometimes to the exclusion of good tests." (P20)

Where characteristics such as performance or correctness are considered most important, some participants suggest the languages capacity to guide the developer does not have to be the path of least resistance:

> "So I think we have definitely chosen the performance, as opposed to the ease of development. So that's the, that's the main reason we have, we haven't used other more domain-specific languages before." (P17)

P17 demonstrates that they have no aversion to managing a more complex language to meet their project's needs, in this example, meaning they avoid using a potentially more expressive environment for development. For other participants, this is demonstrated by their use of more complicated language features in order to defer some programming processes to the language. P2 provides an example of using C++ over C to benefit from RAII to assist in preventing memory clean-up mistakes or the need to cover all code paths explicitly:

> "I favour C++ over C for several features:
>
> ...Scoping. There are a few cases in the codebase where cleanup code(especially code that puts back hardware into a default state) isexecuted when a variable goes out of scope. This makes sure that, forexample, the CS line of a SPI device goes back to high no matterthrough which path we leave the block of code." (P2)

Many participants desire features that help guide their programming by constraining their potential to make errors, checking assumptions or implicitly handling program elements. But participants also shared mechanisms for programming to guide their way of thinking. In the purest example, we see that the influence of learning to program allows for transferable, programmatic thinking that can be reapplied.

> "yes, know I did put Java, but in that very little level. This is my basic level to help me ... yes, the basis has helped me to do something but at the end of the day, I must research again and look how can I meet its logic. Okay, I know a little bit but it's helpful. It helped me to me, for example, if I try to do something in supercollider also, of course, is not the same language but all these structures helped me to create or to build something." (P21)

> "Yeah, I think any kind of language not to just programming but also talking about language in general, they, they are translation already of something they they kind of belong to certain mindset. So in these when you're also doing music, you kind of filter your ideas" (P27)

This can also develop to the point where the language constraints can be instructive, acting as an educational steering force. For P17, this is particularly apparent as they describe their use of the Faust programming language as having heavily influenced their approach to and understanding of digital signal processing (DSP) for audio, despite not using the language directly for their instrument.

> "I will definitely say that Faust was a tool that helped me even though we're not using it anymore. So I am the one doing the most most of the DSP programming ... I have a lot of like, numerical programming experience, but like audio programming and DSP programming... I've never really gotten into that. So as a tool for understanding, like filters and different effects, how they work and that sort of thing, something like Faust where I could, where you can even make, like a practical view of how a reverb effect looks, and you see the see the delay lines feeding back on each other. Right. That's that, that that was perfect for me. And it's that really kick started my understanding of this field." (P17)

This educative effect of programming languages is also described by P16, who describes how the functional live coding language tidal cycles have influenced their approach across programming in general.

> "Tidal has really influenced my understanding of programming and other things in a way that definitely bleeds over to like how I approach other languages like c++" (P16)

P3 appears to exploit the educating capacity of languages and the idioms that are present in different ecosystems, even embracing the limitations and shortcomings of a language and its environment for positive outcomes:

> "Yeah, I saw that question. And I, I don't have an answer for you... There's nothing like really that I'm just like, Oh, this is really frustrating. I wish this could do that. Because most of the time, like you find ways to bridge it, and through finding ways to bridge those gaps, you learn new techniques... each platform has its idiosyncrasies that direct a bit, how are you going to use it... it constitutes the kind of work that you're going to be doing on those platforms. And like I like using for that reason kind of like many platforms and renewing myself." (P3)

We also see that for some participants, the graphical paradigm acts as an introduction to programmatic thinking, guiding participants to be able to create highly customised instruments or leading to the use of text-based programming languages.

> "I guess I started more from like pure data. I mean, I should have written that also in the list, but I don't use it anymore. But yeah, I started by pure data, I guess. Because like, it's very accessible when you're not like a programming when you're like starting to try to programme or do things that are not afforded by like your regular DAW. So yeah, and I guess, like the visual programming at first." (P3)

> "There was a module in that called Live electronics that that looks at max MSP. So that's that sort of, as far as any kind of formal education I've had, I've had in it. But I really loved the flexibility of Max and, and how it sort of, you know, allowed me to try and build something that didn't already exist." (P14)

Interestingly, both P17 and P16 refer to functional programming languages when describing the languages that have significantly affected their understanding of programming. This could suggest the introduction of new paradigms and, therefore, ways of thinking about problems as being a strong influence that allows languages to direct the users toward solutions. In some sense, this may be best presented as paradigms being a good fit for a particular domain (as is the case with the functional paradigm and the domain of DSP for which function composition is an excellent fit for representing audio graphs), but also may suggest that the idiomatic uses of languages also help inform the approach of programmers.

To finish this theme, we look at one final dimension of the guiding influence of programming languages. P1, who chooses between languages to suit different stages of their design process, mentions needing the programming language to inspire them. Despite many constraining factors on what features they require from their languages, P1 leverages tools which guide them toward their goals, and as these goals are different throughout the design lifecycle, so too are the desired characteristics of the language they choose.

> "I would say that with three tools I could do anything. I would choose PD because I prefer pd towards max for prototyping audio. I would choose Python for dealing with more with with data, and I would choose c++ for going beyond prototypes.
>
> I like PD. ... [I] just have an idea and I can make it in five minutes. And so I think that's an inspiring tool. I wouldn't, I would never call C++ an inspiring tool." (P1)

In this theme, we see two ways in which the language and the environment it exists in support and guides the programmer. Some participants seek their language to be a collaborator that can take charge of some tasks and guide them to a more robust solution through constraints. Participants also seek guidance from their programming languages through the idioms embedded in the language, shaping their code.

6.2 Theme 5: 'The Mutable Instrument'

This theme is based on the common account of participants seeking languages and means to support an ongoing design process where an instrument is often never considered truly 'finished'. Participants described their instrument design process as integrating continuous feedback and requiring a capacity for easy sharing and distribution of the instrument, often extending to using decoupled and distributed components of an instrument. This is demonstrated by P25 discussing the 'V2' or 'version 2' of their instruments, suggesting a natural and uncapped iteration of the

instrument. Participants highlight a process to share an instrument once an acceptable quality is satisfied, then further develop it based on 'real' user interactions.

> "Recently, we've been ... doing a lot of v2, or like, you know, second iteration of the design idea, basically taking an existing design the v1, let's say it was successful in some ways, and then like seeing like, Okay, what would the does the user base usually complain about what we don't like about it? So then it's like, very, like market informed. And you have, you have a lot of data from the user base." (P25)

> "And then choosing the most convincing path, building an MVP, and constantly refining it. And then I think was also very important is to leave some leave it open ended. So making sure that it's structured in a way that allows extensibility that it can get better over time. Like, can be refined after being released to the public." (P15)

P20 builds upon the ongoing evolution of an instrument through a desire to distribute control of its continued development and introduces the term mutable to describe DMI. They emphasise their desire for DMI to be left unrestricted by the creator, allowing for the users to have a role in mutating the instrument, a sentiment shared by P19:

> "So, to me, a"digital" musical instrument is one that is, for lack of an unused term, mutable, without interference from its creator. Once you bake the software in, with closed source applications or chips that are under NDA, you've left the realm of what's interesting to me in a digital instrument." (P20)

> " you can build your own little musical instrument with this thing so do it ... it's unprecedented that digital music instruments can open that up so much. So I wish people would do it ... explore it see what it's like to build your own little sequence here, don't rely on you know a pile of stuff coming out." (P19)

This is also reflected in the comments of P9, who describes the open-source approach to Linnstument making full programmability available for the end user to continue extending the instrument, despite being a commercial product. This depends on accessible tooling and language support such as the Arduino IDE and is suggested as a significant contributor to the instrument's success.

> "So early on, I decided to make the software open source. And I wanted to make open source development as easy as possible. So I use the Arduino development system. My software developer **** says that, in his opinion, this is the modern day equivalent of taking your guitar or violin to a craftsman to adjust the action or adjust the the wolf tones ... And so these days, because there is such a large DIY culture, people found it very attractive. And in fact, I would estimate that ... my sales for linnstrument have increased 20% because the software is open source" (P9)

Whilst the ongoing evolution of the instrument is considered a part of the digital essence of DMIs, participants also require their programming languages to facilitate engaging the user interactively and facilitating fast iterations. P15 explains a need to validate ideas quickly by engaging users directly:

> "... being able to validated the idea, as soon as possible and move on, And but then what's equally important is to be able not to only evaluate the idea myself, but also be able to distribute the prototype to other people. And it's, this is tricky, because like... I have to bundle it in a plugin, ... have a working max or max for live device, which is already somewhat polished. But like, if this process could be sped up, then I think this would be the biggest one for me." (P15)

Participants such as P19 also discuss the need to be able to test ideas out quickly.

> Like, you can come up with dream aspects, like, you know, seven days a week of things that would be cool, without really knowing what it is you're dreaming about. Right? And the problem is that when you start diving into half of those, they're just they make no sense at all. (P19)

Exploring ideas in the context of highly interactive systems like DMIs present challenges to simplified simulations for testing.

> "...I would not really work with a simulator and clicking buttons with the mouse. So for me, like being able to create the physical product, try it out improve it. tools that are needed for for this they helped me most in my process." (P5)

As mentioned by P15 previously, a certain level of polish and tangibility are required, meaning that prototyping tools must focus on supporting the final target hardware for implementation. P15 comments on their need to be able to distribute components of their system for users to use, with enough 'polish' to not detract from the experience. They later go on to describe how good support of interoperability and modularity are required to achieve this.

> "So if that if like, the design and implementations is created in a way that allows actually quickly like exploring different ideas, different corners, and quickly changing these things, I think the overall product ...benefits from it a lot. ... I could call modularity also for that, like, you know, that this, like, totally having this, like, loosely coupled components is what makes things way easier.'' (P15)

> "But yes, that really that ability to take that raw data and then convert that into something that's like standardised for music producers, I Max for live object inside Ableton." (P8)

MIDI is presented as a critical component of decoupling systems by P10, who describes it as a universal musical language that facilitates interaction with other systems and highlights the importance of robust, modularised abstractions.

> "Well, I haven't said MIDI yet somehow have I. So that would, that is definitely the biggest tool that has enabled this... so that the mobile revolution was one thing but none of that would have been possible without MIDI tying all this stuff together. And that's, that's really the foundation that we've used this whole time is MIDI as this pretty universal musical language means that we can design new ways to translate human gestures into MIDI, and then allow for the 1000s of applications, you know, that people can do with that." (P10)

This is a common theme for other participants and demonstrates the way in which participants expect their tools to support this decoupled architecture that allows components to be swapped, changed and easily shared.

> But for me, it's not the language that's important. So much is but the manner in which it's exposed at a higher level environment. (P23)

> I was able to, for example, lend one of my instruments to a friend of mine, for them to use in their Ableton set, and they didn't have, you know, they can literally just plug it in, and it just appears straight as a MIDI device in Ableton that was actually quite groundbreaking in my mind when that when that happened, because it always felt like to be able to get this stuff to work. (P14)

As P23 points out, this modularity is expected to be a design decision in the languages they use, allowing them to work within their environment at a high level, suggesting deep ties between the programming languages and the technological ecosystem it exists in. Here MIDI provides a prime example of the intermediary technology that supports this.

The problems involved in digital lutherie are substantial, and this modularised set of abstractions is presented as a means for participants to focus on the important components of their DMI, as well as test and develop components in isolation.

> "A craftsperson is someone who understands the entire process from beginning to end, from growing the tree, to tensioning, the violin, they understand it all ... it's not possible for one person to understand the whole thing ... it's not possible for one person to understand all the things that matter for the instrument ... But the other way to do that is to create like, new representations of complex things that are much more efficient for people to parse." (P16)

> "Norns for instance is a platform. I didn't know any lua going into programming for norns but the platform is used by so many people now and is so robust it's supported by supercollider so I can also create my own engines for that ... I started learning because i saw all this potential you know it's a lot of shortcuts it takes care of a lot of the work that I had to do whenever I would try to make an embedded instrument it's like they they take care of a lot of that stuff already..." (P26)

Given the nature of DMI and, more generally, software, participants note the tendency of software to 'rot' if left unattended or become obsolete as platforms develop and themselves improve.

> "...there's something that I've been calling software rot, where it just just letting it sit there the rot sets in, and you it it creeps into everything and all of a sudden, like after a few years, it becomes dysfunctional, even though the code is exactly the same." (P6)

> "because digital instruments can, you know, they can become obsolete, because the technology changes..." (P25)

This further emphasises the need for DMIs to be continually evolving. P6 suggests that languages and ecosystems that support the maintenance of the instrument are critical attributes as more effort is spent maintaining an instrument than initially creating it.

> "There's various reasons I've gravitated towards maintainability. Because whenever you create any type of product that is software based, you will spend way more time dealing with the lifecycle of the product and with the initial creation spike" (P6)

For more collaborative participants the capacity for a language to quickly allow new programmers to be productive is an important side of maintenance. P7 includes familiarity and ease of reading as reasons for choosing C++, features which make maintaining code more futureproof as teams grow or change.

> "...Most programmers are already familiar with C++, which allows for quick onboarding of new programmers" (P7)

P23 views the challenge of maintenance in connection with the art the DMI produces, seeking a means to sustain the reproducibility of their art, despite the tendency of software to degrade. They focus on preserving their programming tools as part of the community, where the systems being used are maintained through popularity and a need for backward compatibility.

> "It is something, as a musician, I've, I've sought to have a long standing relationship, or a kind of longevity in my work, that means that old pieces need to be performed decades later. That means creating new pieces that will have potential to be reproduced later, in another era by other people ... I don't think I had that any vision in mind when I set out 35 years ago, but the tools that I did choose did stick in the community. But I was thinking about longevity already back then. So if then today, it's a computer based system that's doing signal processing, taking live gestural input, the actual device may have changed quite a bit, but surprisingly little about the way the signal is acquired, processed and mapped to sound." (P23)

Identifying DMI as highly mutable and evolving systems, participants expect software to support means for the continuous adaptation and development of both their own and other instruments. Participants appear to converge on heavily decoupled, modular approaches to programming and expect their programming languages to support this and deeply integrate with their environments. Practises such as open source and communities of sharing are integral to developing software for DMIs, both in the evolutionary development of instruments and in maintaining them. This culture even permeates commercial settings, with users pressuring for more control over the instrument's lifecycle. Even less permissive, proprietary devices supporting open standards, such as MIDI, to decouple their components and support the mutable nature of instruments contribute to this expectation of participants.

6.3 Theme 6: 'Expressing My Ideas'

Whilst we have seen in the analysis in Chapter 5 (Theme 1) that pragmatism is a defining factor in tool choice, this theme captures the more optimistic and idealised narrative around what programming characteristics participants pursue. Participants share the mutual position of seeking a low-friction environment that grants expressive control over the instrument they are building. In many respects, this theme avoids focussing on the decisions forced through pragmatism, where participants aim to have their tools most effectively serve their use case and capture as many requirements as possible whilst minimising the complexity with which they must work. Instead, 'Expressing my ideas' explores the narrative of participants who seek tools as an extension and component of their thinking. The theme takes its title from P22, who describes their programming language of choice as 'my main way of expressing my ideas'.

> "So most of my instruments are which in C++. And the reason why I use that is because it's the language I have always used. Very, very comfortable with C++. It's like, my main way of expressing my ideas" (P22)

Participants appear to seek languages that match their thinking and approach to problem-solving.

> "But it's also because I consider supercollider for example one of the between worlds of between programming and music production so and I feel quite liberating to use code instead of Max MSP or pure data" (P27)

> "I'm getting into Rust now because of how efficient and tiny it can be. And also, it's just, you know, as I'm learning to code it, it feels it just feels really nice to write in." (P26)

Using languages and programming paradigms that support a participant's mental models and problem domain provides a more ergonomic fit for them to express their ideas. In their interview,

P19 presents their preference for functional languages, suggesting that they are more expressive paradigms for instrument behaviour, though they notably find that this is practically limiting due to pragmatic constraints.

> "...because these functional languages i hesitate to call them modern since most of them have been around for 20 years or more really allow great expressivity... on that conceptual side of being able to express what you want the instrument to do, but on the other hand a lot of my most recent work is in something as dull as c++ because ultimately if you're going to run it on an arduino or an arduino like processor like that's your choice you know the other choices is micro python which just isn't practical... (P19)

A core component of expressivity, as implied by P22's initial quote, introduces a common factor among participants aiming to achieve low-friction programming: proficiency. This emphasis on familiarity and experience is a common trend, often rooted in the pragmatic rationale for time-saving and not needing to invest time in learning new concepts.

> "The other one would be Python. Because first of all, I think this is still the most relevant reason, I am fluent in it ... if you're comfortable in it, and I'm fast in it, and this is important to me to be able to prototype fast..." (P15)

> "So as a computer scientist, I felt more comfortable with c++ and probably more in control of what I was what I was doing." (P11)

> "There's no time to learn things and do things really well in dedicate a lot of time so of course, you're going to try to approach one language that fits into your lifestyle and your way that you thinK" (P27)

But this familiarity can also be reapplied to new contexts, where language families can make new tooling feel more immediately approachable, even when crossing problem domains. We see this example from P22, who demonstrates how familiarity can be exploited to cross into other problem domains that are perhaps less natural to some digital luthiers, in this case, transferring programming constructs to the application of 3D modelling the physical instrument. This is interestingly also an experience shared by P11.

> "...whenever I've been doing my mechanical drawings and 3d modelling, I have been using something called OpenSCAD. And which is Do you know it? Yeah, so that is kind of, I think it comes from my love of, I'm really comfortable with writing code as text. And it has kind of looks kind of like Python or C. So it's like a very familiar syntax and stuff. So rather than burning some like really advanced 3d CAD programme, I can just write code. And I can use all of the concepts and all of my like, my like patterns or thought patterns while doing my 3d design, just the same as when I write my code." (P22)

The experience of P17 provides insight into how participants juxtapose the desire for expressivity with the requirements of their project. In their interview, P17 discusses the challenges of translating between prototyping and production code, stating that there is little time saving working this way compared to working in the final intended programming language, which they find ultimately less expressive. They state that:

> ...if there was a way to get around that which, incidentally,the language Julia has sort of been for the scientific community... where I could write something that in a

> language, which is super expressive, and just easy to get something up and running really quickly. But it's also performant. Enough to like being able to run on our embedded system, and also can have the interoperability that we lacked from Faust. Like, if there was a thing that could give me all of that, like that would be like, that would be the Holy Grail." (P17)

In an ideal scenario, P17 describes their aim to program with maximum expression while working with a language suitable for their embedded performance requirements. They describe a clear set of limitations that prevent them from working with other languages and ultimately settle on using the language that offers the most benefits with the least friction. They describe the 'Holy Grail' in language as the combination of expressivity, performance and interoperability. Ultimately, however, most prioritise the pragmatic need for performance and interoperability. When offered to propose a speculative language that would suit their needs, P17 described the possibility of a domain-specific language built with first-class C++ support, coupling the goal of solid domain-specific support with the broader capabilities of a mature general-purpose programming language.

> ***Interviewer:*** Feel free to propose something more hypothetical...
>
> ***P17:*** Well, yeah. As I, as I said, a language or something for c++ or something that is performance, but still plug and play and very expressive for for audio would be would be great... If it was a DSL implemented in C plus? If it was a DSL for c++, but not as that transparent language?

For many, this level of expression is suitably captured in a language's ecosystem through libraries and frameworks with a deep domain focus.

> "...there's a lot of DSP libraries, which make it easy to kind of plug together something which you can use... mutable instruments git repository, which is like full of brilliant stuff, which you can just take and stick together in new ways and have a filter after something." (P5)

> "...or set up kind of an abstraction or pattern based on something that I'm doing, then I can replicate,again, at a higher level. And in doing so, still allow musical components, musical modules, and elements that are created to berecombined in different ways." (P23)

> "It forms the perfect compromise between processing efficiency and high level design." (P7)

> "it was like really quickly, developing a lot of interesting libraries to get things done quickly. So that was a platform of choice." (P25)

> "And I also would say I use Max. Also, because how it because it gives me a lot of tools out of the box and modules that I can freely connect with each other in the way I want." (P15)

Participants seek high-level abstractions in libraries allowing flexible and highly composable programming. In some cases, we see this composability between languages where for example, P11 can be more domain focused by using Faust for DSP and then integrating this within the broader environment with more general-purpose languages like C++.

> "So usually what I do is that I use a combination of Faust and c++. I write my DSP in Faust. And, and then I use c++ to sort of put it all together, basically... ...it's quicker for me to write them in Faust than in C++." (P11)

The narrative shared by participants tends to focus on the practical need to integrate with their platforms environment, and this leads to a desire to find both deep domain-oriented and inspiring tools that integrate with the project's environment.

> "I would, I would prefer it if Faust was a library for c++. So there was no, no transpiling step." (P17)

> "On the software side, I think having more projects like CircuitPython, and having those projects explicitly focus on audio DSP, would be wonderful." (P20)

> I like PD. Because it, it's very good not only for prototyping, but for brainstorming, or for just trying something I mean for Yeah, just have an idea. And I can make it in five minutes. And so I think that's an inspiring tool.

Some participants gravitate towards other tools to increase their expressivity and break free of languages that cause them friction. The most extreme example of this is found by participants aiming to remove the need for programming almost completely, aiming to express their intentions primarily through domain-specific tools:

> "Well, I'm in a quest, and I think in in various aspects of life to have the tools disappear. For example, software development, I'm a fan of the no code movement." (P9)

> "We, you know, not not just for instrument design, but app design in general, I think we're there's there's a, there's a moment that feels like it's about to happen, where these tools are gonna get much simpler. And it's it's like it's almost been there every year where people who aren't hardcore coders can get in and do even like Adobe XD, like doing UX and UI kind of workflows.And, and I've seen a few things pop up that are going to make that easier, but we haven't yet. We haven't yet use them. We've kind of still been doing it the hard way. I do think that that's gonna really change." (P10)

> "And a website is should be about design. So I would say in in so many aspects of software development, coding, you shouldn't have to do, you shouldn't be able to focus on design, particularly websites. You know, you can have objects, you just load in pre made objects, and then you just do your design. And I think that day is coming but it's not here, yet." (P9)

For certain digital luthiers with less technical backgrounds, programming in the traditional sense is also a barrier which they seek tools to remove.

> "I can only spend that much time going to the root of it, I'd love to do my own programming and learn C, you know, or whatever I need to learn or, but, you know, I I'm still an artist that needs to use this to, you know, to make music and performance." (P4)

Participants show that techniques such as machine learning can be leveraged to express design intention, with less programming, exemplified by P4's use of the machine learning tool Wekinator, which plays a role in programming their instruments.

> "But I did not write software from scratch. I use max MSP and I use that for creating interfacing the hardware to the music to the software. So people's idea of like writing software, yeah, I do write my own programmes, but they're all in max MSP, except in the case of like Wekinator, which I started using two years ago, no more six years

> ago, what connaitre also was not written by me, obviously, but Rebecca Fiebrink. And we did work together for like user interface and how to develop it for herself and for myself. So yeah, so I programmed myself but I did not write the software, what I call writing the software, I do not write objects, and I do not write C plus plus Java" (P4)

Participant 23 further expands on the use of machine learning to describe their extensive exploration and development of using machine learning to create intuitive strategies to map interfaces to sound generation.

> And if we take the task of mapping, as an example, and nothing is presented, in an interface as a bunch of sources, and a bunch of destinations, that reduces it to this rather straightforward and limited linear task. On the other end of the spectrum, by using things like machine learning, we might make the process more intuitive, creating what we call mapping by demonstration. (P23)

This narrative presents a more idealised outlook on what participants desire from programming languages. Overall, this theme hints toward the preferred modes of expression for participants, who are ultimately constrained by the practicality and challenges of digital lutherie. Currently, languages are mainly chosen as a compromise and do not reflect the full intent of what the luthier is looking for:

> " We chose C++ for a number of reasons:- It forms the perfect compromise between processing efficiency and high level design." (P7)

> "... You know, I use things like Haskell when I can I use C++ when it makes sense." (P19)

We see that participants seek means to support their expression, work in abstract composable blocks, and use paradigms that match their approaches to problem-solving and thinking, resulting in ergonomic and familiar patterns for them to create DMIs. We also see the expectation that programming can be further streamlined by using more specialised domain-focused tools and integrating machine learning to help address the challenges of digital lutherie.

6.4 Discussion

Building upon the initial analysis that explored the question *'How and why digital luthiers pick their tools?'* a secondary inductive and reflexive analysis has created three more shared narratives which add nuance to this initial question and explore the desirable attributes of the programming languages that digital luthiers search for. The first theme introduces the participant's search for languages that play the role of 'A Guiding Force'. This again fits into the model of affordances in design (Chong and Proctor 2020), where the language provides a set of affordances to work with. Theme 6 also complements the constructivist view of creating programming knowledge through constructed models that adequately abstract the problem space, allowing the participant to work with this model to solve problems (Ben-Ari 1998). The Digital luthiers examined in this analysis represent diverse experience levels, with experience creating DMIs ranging from one to 40 years and participants having worked on between one and approximately 100 instruments. With such a diverse population, this theme captures the guiding forces of languages on both novice and experienced programmers. Where programming pedagogy examines the construction of knowledge, it focuses on forming functional models of the computer or programming language. This study discusses constructing these models in the context of a range of experience levels in digital luthiers. This suggests that smaller, more focused, so-called 'toy' programming languages

can have a pedagogic role in digital lutherie, acting as creative playgrounds for learning new concepts and constructing new models. This also implies the need for mature, general-purpose languages to offer flexible support for programming styles and paradigms, both in language design and through the libraries and ecosystem it represents, to ultimately lead to the most expressive approach to the problem (Filippidis, Murray, and Holzmann 2016). While this is true for many mainstream languages, there can be implications in this space that are incompatible with other requirements of digital lutherie, such as performance challenges which are a longstanding and ongoing challenge for certain programming paradigms (Page 2001; Arora, Westrick, and Acar 2023).

A novice looks to and utilises programming languages, especially in exploring new paradigms, to form new models for thinking. More experienced programmers look to form models that can abstract away entire categories of errors, exchanging a potentially more complex model of the language for assurances around errors such as memory management. We see the example of this discussed in the context of Rust, which integrates a strict type system with a memory management system called the borrow checker. This is widely cited as a challenging model to adopt; however, once this model is internalised, it prevents many potential bugs entirely with minimal cost to runtime performance.

Theme 5 pertains to a technical and social solution that responds to the architectural nature of digital musical instruments. By the definition provided by participants and supported by existing literature (Magnusson 2010a; Wessel and Wright 2002; Wanderley 2001), DMIs are fragmented, continuously evolving and highly modular entities that link across domains. P8, a composer who builds tools for personal use describes a DMI as.

> "Input and input-conversion and output. Yeah, gestural input." (P8)

This gives us a minimal description of a DMI that can be described by the function $Y = f(X)$ where X is the domain of gestural input, Y is the domain of audio, and the function f is a mapping between domains [1]. This theme demonstrates the need for the technology to mirror this decoupled model, creating distributed solutions that can be built across the community (Fischer 2004) and that digital luthiers can compose together to create complex systems. The clearly defined boundaries between domains X and Y also support the capacity for DMIs to remain highly extensible to the users. Participants describe this extensibility as an expectation and inherent component of a DMI. This community factor is discussed in the context of the 'second performer problem' where an instrument is not used beyond the digital luthier who created it (McPherson and E. Kim 2012). For those concerned by the wider adoption of their instrument, participants suggest that the capacity to adapt and utilise subcomponents of an instrument are also critical dimensions of a community of use. Fukuda et al. (2021) provide a supporting project for this interpretation, where a group of composers work with their instrument 'The T-Stick', developing new modes of interaction through evaluation and modification.

The final theme in this analysis portrays the goal to achieve expressivity through the digital luthiers programming language(s) as an extension of their thinking, minimising the ways in which the tool limits them. We see that, in reality, this is an aspiration that is often compromised by the requirements captured in the previous theme 'The Pragmatist'.

Relating to findings in Chapter 5, we can further contextualise the digital luthiers espoused theory vs theory in use here, framing this theme as the participants espoused theory. The notable point in this interpretation is that largely, digital luthiers demonstrate awareness of where their espoused theory starts and ends. The compromise that participants make relates to the project management concept of the triple constraint, typically defined with variables such as time, quality

[1] A function also described by P2 and mentioned in the previous chapter

and cost (Van Wyngaard, Pretorius, and Pretorius 2012). While the value of the triple constraint, in particular in relation to these variables, is debatable (Baratta 2006), the triple constraint is recognised as a practical framework for describing tradeoffs to stakeholders. It may therefore be possible to devise such triple constraints for digital luthiers.

6.4.1 What do Digital Luthiers value from their programming languages?

6.4.2 Constructivist Models

When we begin to observe the relationship between these themes, we see that, in particular, Themes 4 and 6 both relate to participants' mental models of the problems they are solving. Participants seem to look for programming languages to fit their models of thinking and provide support in their problem-solving, in particular, focusing on abstracting problems such as memory management into different models that can be learnt and then utilised. In some examples, we see this presented as using less abstraction in the design in order to gain more control and more closely work in the domain of the computer hardware (Hoc and Nguyen-Xuan 1990) in search of coveted runtime performance characteristics (Wessel and Wright 2002; McPherson, Jack, and Moro 2016).

> So the programming languages I use on an everyday basis are c++ for performance reasons. ... And yeah, you can take control over your own memory layout. (P15)
>
> I felt more comfortable with c++ and probably more in control of what I was what I was doing. (P11)

In others, we see that more abstraction with a more complicated model that abstracts the computer is permitted:

> In Rust the type system constrains which programs are valid to a large extent. This means that one must put in a lot more thought up front... (P20)
>
> And so it's just more time and energy but rust for its quickness, efficiency and also ability to run an embedded devices. (P26)

In the example of using the Rust programming language, participants express their appreciation for the mechanisms of concurrent memory safety embedded in the langue through its ownership and lifetime model (Zhu et al. 2022).

Suspending the need for pragmatism, we see that the programming language's embedded models of knowledge are some of the most desirable attributes for language selection. Participants look to utilise the embedding of knowledge within a language as a cognitive store, where knowledge is part of its semantics, libraries, or even its ecosystem of tools and resources [2]. Magnusson (2009) makes the observation that musical instruments are epistemic tools that implicitly store musical and cultural knowledge. This perspective is also observed by McPherson and Tahıroğlu (2020) to be true of programming languages. Work by McPherson and Lepri (2020) contextualise this work on the influence on the DMIs that digital luthiers create. These ideas are reflected in many ways by participants (as seen below in a quote from P1), once again suggesting a high level of self-reflection and introspection being present in the culture of digital lutherie, although given that much of the digital lutherie community centres around the NIME (New Interfaces for Musical Expression) conference, where such reflections are disseminated, it should still be asked to what extent this influence is understood by digital luthiers not engaged with the likes of NIME.

> "On the other side, I have the convenient, I mean, I'm totally convinced that the tools are never natural. They are natural in the sense that you can do terrible and terribly good, but they condition very much what you can do with them." (P1)

[2] such as documentation and community forums

Alongside this embedded knowledge, participants describe a desire for cognitive harmony, where the model matches or easily extends their existing mental models. Existing work supports the observation of this approach in novice programmers and further suggests their use of analogy and reapplication of previous models to construct further models based on language semantics (Hoc and Nguyen-Xuan 1990; Abtahi and Dietz 2020). Participants search for cognitive shortcuts, such as reapplying familiar concepts to facilitate faster adoption of new languages and applications in new domains, which further supports this previous work. This approach to learning and working with mental models to solve problems relates heavily to expressivity with the language. We find that participants appear to be advocating for languages that complement their capacity to express their ideas through their models of understanding, implying the use of fragments of known solutions to problem solve (Spohrer and Soloway 1989; McCauley et al. 2015).

As discussed above, participants looked for their language to direct them through a system of constraints. In some cases, participants seek the language to constrain their design using language semantics around areas established as error-prone and challenging, such as the control flow of memory cleanup. Others also express their desire to actively express constraints by modelling their problem space using mechanisms such as strong static types, which provide a programmable system capable of enforcing and extending the knowledge embedded in a programming language.

Lubin and Chasins (2021) explores the use of static types in the context of the functional programming paradigm. They describe the scenario where participants in their study use the compiler as an assistant, providing corrective and directive support. Statically typed functional programmers were observed to utilise the compiler to provoke feedback on their current code, using error messages to examine their progress on a problem. This was also observed in the process of code refactoring, where the compiler was used as a checklist of things to update. This suggests an approach to development that is primarily suited to statically typed languages, as tooling such as this is difficult without static types. Future work might explore the role of the compiler as a tool to support the guidance within digital lutherie given the overwhelming use of statically typed languages[3] discussed by participants from the study in this thesis.

Through this discussion of Themes 4 and 6, the desire for languages as vehicles for embedding knowledge is established. We see that the semantics and tooling of a language form an epistemic tool that a digital luthier can select in order to match their mental models of a problem space and rely on to enforce some correctness within their work, using the language and tools as a collaborator. Digital luthiers also require mechanisms to further embed knowledge in their tools, such as static type systems, which allow further collaboration, and the development of paradigms and idioms which can be shared between users, typically that subscribe to the same mental models for problem-solving, suggesting design through constraints. Through comparison to related literature, we also see the nature of these guiding factors in digital lutherie as influencing the final DMI. As such, work should be done to make explicit the influence of tools, as it seems likely impossible to reconcile the luthiers hope for guidance through constraint without converging towards certain outcomes.

6.4.3 The Pluggable Architecture

Theme 2 provides a description of the need for the implementation of a DMI to match the DMIs nature, a modular abstraction, providing the highest level building blocks with which to make instruments and separate the domains that constitute a DMI. For digital luthiers, the DMI represents a highly collaborative and social artefact with a distributed architecture. Considering the role that extensible tools play in digital lutherie, the previous analysis observes the role of meta-design in this field, identifying digital lutherie as a prime example of meta-design in use (Section 5.4.2). When revisiting the meta-design framework ten years after its introduction, Fischer,

[3] C++ (24 participants) and C (12participants) in particular

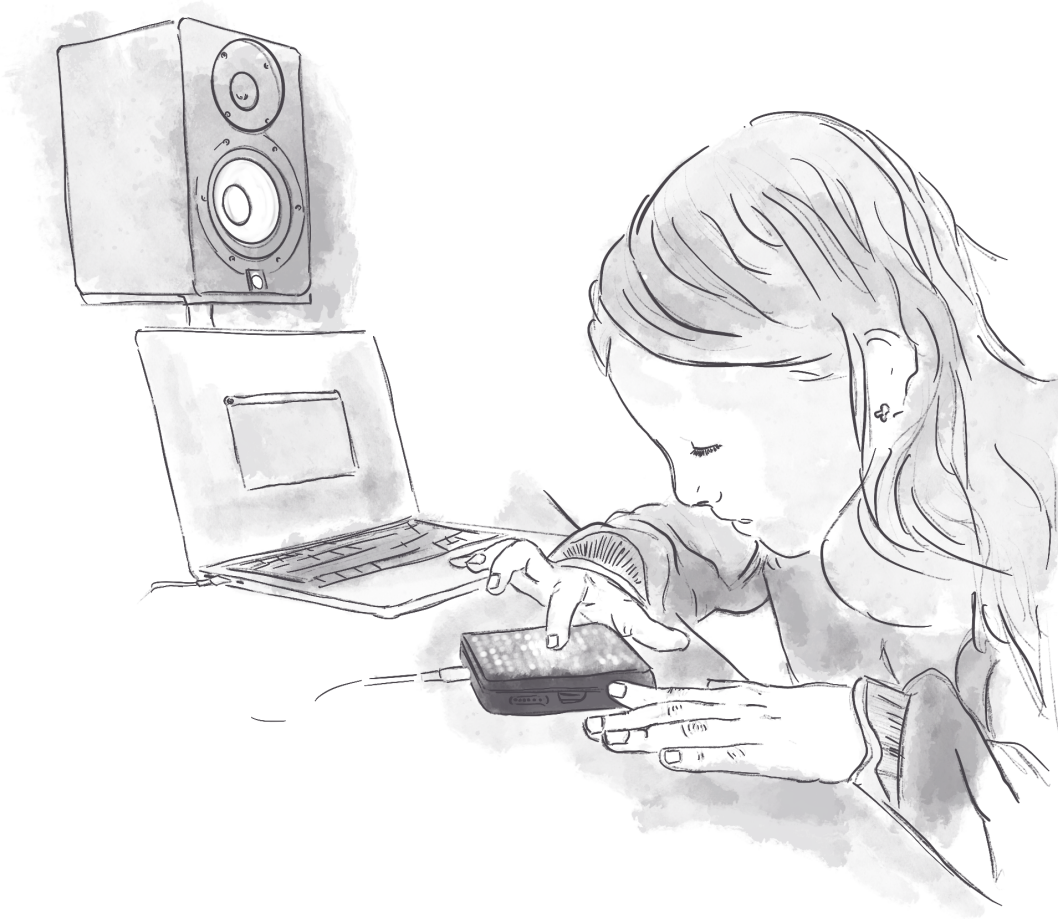
Fogli, and Piccinno (2017) sets the perspective of meta-design, stating 'design and use mutually shape one another in iterative, social processes.' This view aligns well with perspectives around DMI design, where for example, we see that evaluation through performance is considered a core component of the design process (O'Modhrain 2011; Cannon and Favilla 2012). Motivated by this approach to design, participants indicated a need for programming languages to support this decoupled architecture, through sharing, often in the form of subcomponents of an instrument. Through our analysis we see that participants describe processes that can be interpreted through the model of 'Interaction and Co-Evolution (ICE)' (Costabile et al. 2006). This model proposes a cyclic process 'in which system usage induces an evolution in the user knowledge, culture and socio-organizational contexts, which in turn induces the evolution of the system functionalities and possibly some changes in the technology on which the system is based.' It emphasises the importance of communication channels between the designer and the user. Also, it advocates for interactive systems to be built from networks of software environments to allow the evolution to happen with the user. Digital luthiers have an expectation to be able to explore and adapt the instrument (McPherson et al. 2016), and participants highlight this as perhaps one of the most important motivations around their tooling. Fischer, Fogli, and Piccinno (2017) describes integrating meta-design as a transformation of culture from closed to open systems, facilitated through what Lessig (2009) describes as 'read/write' cultures as opposed to 'read only' cultures, where an economy of evolution and 'remixing' is permitted and encouraged. With technologies such as MIDI and OSC[4], digital lutherie strikes a good balance between standardisation and freely modifiable source code which Fischer and Giaccardi (2006) one of the challenges in supporting meta-design, framed as a juxtaposition between standardisation vs improvisation. In digital lutherie, standards such as MIDI are typically used to provide connections between subsystems (Loy 1985) that divide the problem space along domain boundaries. This makes it easy for proprietary and open-source systems to interoperate.

Digital luthiers are describing a need to support meta-design in their languages through technologies and practices supporting their artefacts' co-evolution (DMI). A rich set of examples exists for this, but the support for these attributes is identified here as factors in programing language adoption. Along with the need for the embedding and expressing of knowledge constructs through their language, we can see that overall, digital luthiers place a heavy emphasis on the socio-techno environment in which a language exists (Fischer et al. 2005). These interpretations are supported by the observations of Morreale et al. (2017), who also derive the term 'pluggable communities' to describe the interactive and decoupled system. Not only does this facilitate the connection of technologies, but also the connection of different sub-communities that themselves contain embedded knowledge in their communities and tools, which matches Fischer's notion of social creativity, a precursor to the support of meta-design (Fischer 2004).

[4]https://en.wikipedia.org/wiki/Open_Sound_Control

Chapter 7

Selective Pressures: Toward Design Guidelines for Programming Languages in Digital Lutherie



This Chapter takes the analyses presented in Chapters 5 and 6 and derives a set of design guidelines, through a discussion of exisiting literature and the experience developed from work outlined in Chapter 3. Influenced by the work of Chatley, Donaldson, and Mycroft (2019), this work frames these design guidelines through the lens of the evolutionary analogy that they use to describe the development of programming languages. As the findings of this work lead toward a similar take this approach serves to emphasise the need to consider the wider ecosystem, social, and human factors which influence language requirements and adoption. This chapter aims to be thought-provoking and idea-rich reading for the future implementers of the next '700 programming languages' for digital lutherie. In Chapters 5 and 6, the results from the analysis of the study are discussed and, through reflexive thematic analysis, used to present a total of six themes. The first analysis creates themes about what motivates tool selection for digital luthiers, finding that participants select tools for a combination of pragmatic reasoning, environmental influences, and to meet the intentions of their use. As digital lutherie is a multifaceted discipline, this study incorporates digital tools such as CAD design software, graphics packages, programming languages and even more generalised tools such as business management software and web tools. On the other hand, this also incorporates physical tools such as soldering irons and maker staples such as laser cutters, CNC machines and 3D printers. The second analysis narrows this focus to what digital luthiers look for in the programming languages they use, finding that participants prioritise languages that fit the mental models they construct for problem-solving, allowing them to be expressive and rely on the broader knowledge and approaches enforced by the language. Participants also indicated preferring languages that support the capacity for the co-evolution of DMI with users.

As is discussed in Chapter 6, when designing programming languages is critical to consider them in light of the broader environment of the tools and ecosystems that are used. Together these analyses provide a foundation for stimulating further research around this topic and creating an informative and potentially influential discussion of programming design that can be used in motivating approaches to designing new programming languages and related tools for digital lutherie. This research's inductive nature means that these themes and developed ideas present hypotheses that can be developed beyond this work and explored more specifically.

This chapter draws together the discussion from the analysis of the study presented in Chapter 4 with the ideas developed exploring language design in order to stimulate a series of ideas that may be relevant in addressing the concerns of digital luthiers and provide hypotheses for future research and programming language design. Continuing the evolutionary analogy of Chatley, Donaldson, and Mycroft (2019), this chapter begins by deriving a set of selective pressures which we interpret as impacting programming language choice in digital lutherie. This chapter then presents a series of contemporary programming language ideas in a similar vein to Landin (1966) addressing the identified selective pressures. Finally this chapter presents how we can continue to join the worlds of programming language design and HCI (Chasins, Glassman, and Sunshine 2021), further developing research that empowers digital luthiers with the ability to continue to evolve digital musical instruments.

7.1 Selective Pressures for Programming DMI

Chatley, Donaldson, and Mycroft (2019) observes the evolution of programming languages, constructing an analogy to Darwinian evolution. Not only does this provide a reasonable and effective structure for understanding how languages develop, but many mechanisms they suggest for the longevity of programming languages correlate well with findings from the inductive exploration of digital luthier's language choices. By exploring the themes generated in this thesis, a set of 'selective pressures' have been created to describe the needs defined by participants in the study. In this

section, these selective pressures are summarised and mapped to the themes from which they were derived to allow for mapping between this chapter and the previous discussions that add nuance and context to their relationship to digital lutherie. This set of selective pressures stands alone as a set of requirements for programming languages in digital lutherie, derived from an inductive qualitative study of digital luthiers that supports and adds connections to a rich body of existing literature. Programming language designers may use these selective pressures as motivation and a framework for exploring their language offerings.

To provide an exploration of future language design theory that may be influential to the next generation of programming languages (for digital lutherie), this chapter then introduces contemporary language design ideas in the style of Chatley, Donaldson, and Mycroft (2019) and Landin (1966) in order to present potential motivating and inspiring ideas that may signpost and inspire ideas on how we might address these selective pressures.

7.1.1 Themes

1. The Pragmatist
2. A Product of our Environment
3. Intentions
4. A Guiding Force
5. The Mutable Instrument
6. Expressing My Ideas

7.1.2 Overview of Selective Pressures

Selective Pressure	Related Themes
Essential Requirements	1 ,6
Social Influence	2,4,6
Cognitive Fit	4,6
Epistemic Languages	2,4,5,6
Domain Ergonomics	3,5,6
Open to Entry	2,4,6
Propagation	3,5
Stability	1,2,3
Composability	3,5

Essential Requirements describe the pressure explored through the theme 'The Pragmatist'. This is discussed as the overruling characteristic of a digital luthier's decision-making, where given a set of more idealistic desires for a language (such as those described in Chapter 6) which may form an example of the triple constraint, a luthier is compelled to priotise these functions over any other desirable language attributes. This may manifest as a need for real-time performance on an embedded system or capability with an existing platform or other software for example.

Social Influence is introduced in the theme 'A product in our environment' where the social and cultural influences of a languge motivate the use of a given language. Enviroments such as academic instituions and online communities represent a strong selective force for programming languages and this influence on language adoption is reinforced by the tendency for models of thinking to be built up as a novice and condition the digital luthiers ergonomics of the language - where the lagnuage fits the mental models they use for problem solving.

The ***Cognitive Fit*** describes the reflection of a digital luthiers constructed understanding of their problem-space in their programming language. This pressure is prevalent in a number of themes for example, representing the pragmatic needs for familiarity and speed, the expressive power that this affords and availability of the technology. Manifested as familiarity and a sense of proficiency with a language, however this also incorporates the dissonance and friction associated with trying to use languages which do not fit the digital luthiers constructed models.

Epistemic Languages represents the embedding of knowledge within the programming language and how digital luthiers are supported, harmonise with and are influenced by this knowledge. Languages also need to support the extension of embedding knowledge.

Domain Ergonomics outlines the digital luthiers requirements for tools to effectively model the DMI, a decoupled system of composable elements, each representing a different problem-space. Whilst not all luthiers need to address all of the components that make up a DMI, luthiers depict specific attibutes that suit the different problem spaces and prefer libraries, languages and approaches that fit the domain effectively.

Open to Entry incorporates the accessibility and educational appropriateness of a programming language, along with the capacity for incremental learning. Digital luthiers continually learn through their craft and represent a wide range of experiences. The potential for an easy initial uptake and adoption of a language is a crucial selective pressure for programming languages.

Propagation is the capacity for the language to support sharing. The iterative and feedback-oriented nature of digital lutherie emphasises a need for streamlined sharing of high-fidelity prototypes and DMI.

Stability describes the resilience of a language to time. Digital Luthiers note the importance of maintenance and preservation of the DMI and depend on technological communities to support these needs. As such, more mature and well-adopted languages are biased for this pressure as they represent stable and more easily maintainable languages.

Composability is the capacity for the language to support decomposition and composition with other systems and sub-systems, relfecting the dynamic and modular nature of DMI.

7.1.3 Signposting Ideas to Address the Selective Pressures on Langauges for Digital Lutherie

This discussion aims not to make predictions but to highlight ideas in language design that offer the most to support the digital luthier's requirements. Clearly, languages such as C and C++, like Max and Pure Data, have well-established niches within digital lutherie. However, digital luthiers make decisions on languages using some system of compromise. As such, there is an opportunity to reduce these tradeoffs, and this section explores programming language design in the hope it may stimulate ideas for use in designing programming languages that support digital lutherie. These strategies acknowledge that creating entirely new general-purpose programming languages may be challenging, though this should not be ruled out completely. There are many possibilities for new languages to integrate into existing language ecosystems, including transpilation, targeting shared run-times, or embedding in an existing language

This section signposts ideas that support the Selective Pressures (SPs) derived from the thematic analysis and resultant discussion from the previous two chapters.

This section then closes with some observations on the potential of Rust to play a substantial role in these considerations and suggests its role in digital lutherie may be something we see develop in the future.

Essential Requirements

The essential requirements SP is built upon the largely prevalent theme of 'The Pragmatist', where instruments must meet a specific performance constraint or work on a particular platform. This results in a compromise in language choice, often leading to a language that fits their theory-in-use rather than their idealised espoused theory. Language designers should minimise this tradeoff by using techniques that allow digital luthiers to continue utilising the languages and ecosystems that solve these requirements. One such strategy is transpilation, which has become very popular in web development (Japikse, Grossnicklaus, and Dewey 2017).

Transpilation is a source-to-source strategy of compilation, where source code in one language is compiled and transformed into another. This has many advantages, such as benefiting from another language's features, ecosystem and tooling. We provide an exploration of this approach to language design in Chapter 3, where we form a simple language for describing grid-based layouts and transpile it to the C-like language littlefoot (Jules Storer 2016), to access the functionality of the Roli Blocks[1]. This strategy was also employed by Max (David Zicarelli 2017) and represented an approach to meeting the essential requirements of the Blocks platform, where Littlefoot provides the only native way to interact with the Roli Block in standalone mode. With this transpilation strategy, alternative targets can also be made, meaning that a language can support multiple platforms through a relationship demonstrated in Figure 7.1.

[1] https://en.wikipedia.org/wiki/ROLI

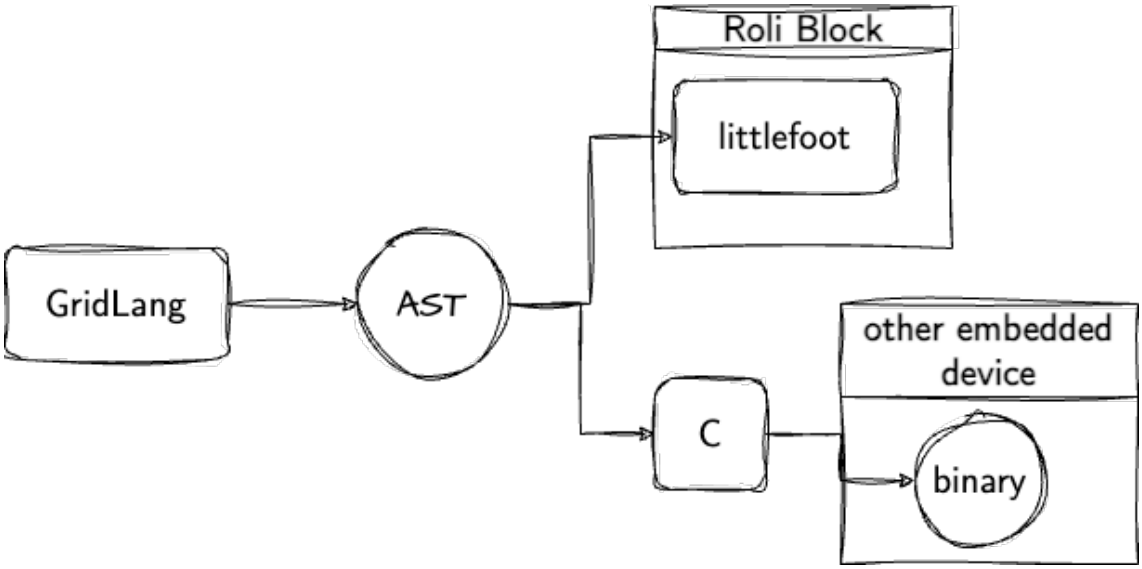


Figure 7.1: Using an abstract syntax tree (AST) to provide different backends to support different platforms.

Faust provides a more mature example of this strategy where, through transpilation to C++, Faust has a compelling means to integrate in a performant way with other technologies, providing a technical solution to the selective pressures of *essential requirements* and *composability* due to its excellent performance and interoperability (Michon et al. 2019; Michon et al. 2020b). It should be noted that in our study, Participant 17 did not find this approach to meet their requirements for *composability*. This can likely be attributed to their lack of familiarity with this transpilation strategy and the non-standardisation of C++ build systems (Miranda and Pimentel 2018). This suggests future work for better integration and education on this workflow, which is demonstrably effective in web development. This is likely due to the far more modern and widely adopted build tools within the web community.

An alternative strategy to integrate with well-adopted and supported general-purpose languages is through **embedded domain-specific languages** (EDSLs) (Hudak 1996). This strategy allows for a DSL to be more integrated with an existing programming language and its ecosystem, potentially avoiding some of the adoption and tooling challenges of transpilation. However, in this case, the tradeoff is that embedding within a language depends on the support for language embedding available in that language. We explore the use of EDSLs in Chapter 3, Section 3.2, where we define a Tidal-like language for expressing complex rhythmic structures. This EDSL uses Haskell due to its flexible overloading and powerful Algebraic type system, making it an excellent host language for a DSL (Gill 2014).

As the prominent language for digital luthiers, EDSL implementation in C++ is generally fairly limiting, essentially being highly idiomatic and opinionated libraries that utilise operator overloading. In some contexts, upon reaching a high enough level of abstraction, a library maybe seen as equivalent to a shallow EDSL (Zhang and Oliveira 2019), where new semantics are given to express domain-specific ideas. This currently appears to be supported well in the context of DSP libraries, where digital luthiers have access to or build libraries in this manner.

Social Influence

The social influence on tool selection is demonstrated in theme 5.2, where community recommendations and shared opinions influence a digital luthier's choice of tools. This selective pressure is less related to languages directly and more to the communities that form around them. Partici-

```
def recursive_factorial(n):
    if (n == 0):
        return 1
    return n *
        ↪ recursive_factorial(n-1)
```

```
def iterative_factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

Figure 7.2: Alternative solutions showing an iterative and a recursive idiom for calculating factorial in a Python-styled language.

pants found the community around education institutions, open-source communities, and industry heavily influenced their tools, including programming languages. When learning to program, knowledge is understood to be constructed through models of either the computer or programming language being taught (Ben-Ari 1998), tightly linking programmers' understanding and capacity to problem-solve with a programming language. This presents a significant barrier to new languages in digital lutherie. Communities need to be formed around the language to be introduced to new digital luthiers and for an ecosystem of support to develop. This area is under-explored, though Morreale et al. (2017) demonstrates its importance as a community begins to develop around the Bela platform. Much like our theme finds that social factors supersede technical ones, Morreale et al. also find that social factors such as community needs and accountability 'in all parts of design, development and delivery' stand out over 'technical merit, user experience and release of open-source'. To understand this relationship better, we advocate for exploring mature examples of languages and tools in digital lutherie in this manner, considering the social and pedagogical factors in adopting languages and tools. We recommend language designers consider **community building** as a language design priority.

The Cognitive Fit

The cognitive fit of a language ultimately creates the sense of an expressive language that helps the programmer to reason about their code and express their intentions through models of their understanding. Through analysing Themes 6.1 and 6.3, the discussion in this work finds the constructivist description of knowledge creation to explain this factor of digital luthier's needs effectively. If a digital luthier understood how to solve a problem using loops, however, a language provided only recursion, we may describe the language as having a poor cognitive fit because the knowledge constructed by the luthier cannot be effectively applied - despite a solution being possible using either approach.

This selective pressure is often best fulfilled when the embedded knowledge within a language either aligns with a participant's existing understanding or allows them to build on their knowledge constructs in order to learn new models and solve the problem they are working on, often in the form of building or having familiar idioms for writing code. This is most often present in a language through libraries, where, as mentioned previously, a library may be the equivalent of an EDSL (Zhang and Oliveira 2019). At this point, a strong idiom or programming paradigm is established within the library, and the programmer can think in a highly abstract way, using more abstract models of reasoning. This characteristic is shown to be highly desirable from participants in our study and has interesting implications when considering the influence of these highly idiomatic approaches on their output (McPherson and Tahıroğlu 2020).

This cognitive fit is best achieved through domain specialisation, such as DSLs and EDSLs. Participants often describe moving between an expressive DSL such as Max/MSP to prototype, where the cognitive fit allows for freely expressing ideas before implementing their ideas using C++, where they can focus on their essential requirements and the computing domain more.

An interesting topic in support of this is the concept of **Bi-directional code transformation** (Chugh 2016). This concept, currently explored in the context of vector graphics design through programmatic and direct manipulation (Hempel, Lubin, and Chugh 2019) presents many potential benefits to supporting digital luthiers. In the case of this selective pressure, previous work with Faust demonstrates the value of such a strategy, allowing different idioms to be used to express problems through translation between paradigms (Gaster, Renney, and Mitchell 2018).

Epistemic Languages

The epistemic nature of programming languages is discussed in relation to many of the themes in this work, where through different perspectives, the knowledge can be embedded in languages to support digital luthiers (Section 6.1), it can work in harmony to improve the expressivity of the digital luthier (Section 6.3), it can also be shared through artefacts and embedded in the DMI itself, as described by (Magnusson 2009) and relating to Section 6.2. The community can also act as a store of knowledge (Section 5.2).

Programming languages are examples of social creativity, co-evolving with the ideas of multiple stakeholders through communities of shared interest (Fischer 2001). The very nature of developing new languages to create DMI itself implies the embedding of knowledge into tools for digital luthiers. Through efforts such as (Bernardo, Kiefer, and Magnusson 2020), we see that empowering the digital luthier with the capacity to specify new languages furthers this approach. What appears to be an essential consideration in the design of new epistemological tools is to consider how the paradigms and idioms developed impact the output (McPherson and Tahıroğlu 2020). As McPherson and Tahıroğlu points out, 'The main difference with low-level languages is not the presence or absence of hidden scripts, but the extent to which they are concealed from the designer on first encounter.' In designing new languages, it is important to consider in what sense these influences are made explicit. We suggest that the notions of influence from a programming language extend beyond aesthetic influences on the musical output and impact all forms of problem-solving, where idioms form the grounds for the solutions that a language guides its user towards. Participants in our study suggest that these effects are primarily positive, supporting them in their process. Ultimately, the need for digital luthiers to have epistemic languages are complex and call for further research. While many of the technical characteristics we have discussed help support throughout this chapter support the embedding of knowledge, in response to this selective pressure, language designers should instead look to contextualise how they are embedding domain knowledge in there language, and to what extent should that knowledge be made explicit, to provide digital luthiers paradigms and idioms that work toward their goals. We suggest that this requirement warrants more research to support the development of new tools and to support meta-design in digital lutherie further (Fischer and Giaccardi 2006).

Domain Ergonomics

Domain ergonomics describes the selective pressure of a language's capacity to express specific problems. The capacity for a language to suit the domain through ergonomic and idiomatic expression underlies a number of the selective pressures in table 7.1.2. Throughout the study, participants discussed the need for languages to model the domain sufficiently. This was possible for general-purpose languages as well as Domain Specific Languages (DSLs), where well-designed libraries could provide idiomatic means for expressing domains such as the C and C++ DSP libraries mentioned by participants. While the importance of good library design should not be understated, ultimately, libraries are limited by the paradigms, semantics and syntax of the language they are written in.

For example, the control of memory and its impact on performance cannot be specified in

Python. On the other hand, participants describe Python as being better for working with data, over C and C++. Of course, for DMI design, the value of DSLs is once again apparent, where a DSL can have semantics designed around concepts important to digital lutherie. Chuck, for example, provides tight integration with expressing timing as it relates to sample rate, a semantic that is easy to express and functions in a sample-accurate way that suits the processing of audio (Wang, Cook, and Salazar 2015). Given the value of building these highly nuanced and concise languages, deep EDSLs (Gill 2014) or full standalone DSLs are typically required to achieve such significant features.

Programming languages also offer features to allow the programmer to model domains and create a system of constraints with which they can work: **type systems**. There is currently no strong suggestion that cognitive load is impacted by statically typed languages when compared to dynamically typed languages (Koeppe 2018). The development of code using types takes on an approach where a programmer alternates between modelling the domain of their problem through types and then using expressions to solve the problem (Lubin and Chasins 2021). How this differs from programming with dynamic programming is not yet examined; however, participants expressed through Theme 6.1 that the capacity for the language to take an active role in guiding and preventing errors is one that Lubin and Chasins describe the statically typed functional programmers as using the compiler for, often invoking the compiler to use the type system to guide them and test assumptions. Type systems seem to present a feature to support domain modelling that the programmer can use interactively and should be explored further.

Open to Entry

DSLs are often cited as being both easier to learn and more expressive in the domain Hudak (1997). Typically, as it needs to cover less use cases (and has not evolved over many years as a general purpose language, accruing legacy design decisions), a new DSL can have a much simpler grammar. This improves the capacity to learn the language due to less complexity in expressing ideas, and the semantics of the language are typically well suited to the domain, making the idiomatic way to achieve something more obvious. Simple grammars allow for the easier use of parser generators, which make tokenisation of the source language far simpler and positively affect language development and maintenance (Parr and Quong 1995; Parr and Fisher 2011).

Where digital luthiers may use a general-purpose language, the support for exploration of libraries and code is an important one that **type systems** may also provide benefits for. While it is still debated, through communities such as the Type Script community (Fischer and Hanenberg 2015) and functional programming communities (Lubin and Chasins 2021), there is a strong advocation for well-typed languages providing better documentation and tooling (Mayer et al. 2012), lowering barriers to entry.

Much like in the case of the social influence SP, community plays a major role and **community building** once again presents a critical design consideration for lower barriers to entry. Morreale et al. (2017) describes two factors to lowering barriers to entry, initial setup time and learning curve. They suggest streamlining setup time through frictionless development environments such as browser-based programming tools, a strategy used by Faust, lowering the set-up time enough for use in a school setting (Michon et al. 2021). Morreale et al. also suggests that the learning curve of their platform is addressed by using familiar programming languages. For the language designer, this implies that, as discussed throughout our themes, familiarity is a powerful component in language design, where familiar syntax and idioms help improve the learning curve. This also suggests a dependency on the selective pressure of social influence.

Digital lutherie also represents a design community with an established relationship to social creativity. In considering the ways in which barriers can be turned into opportunities, Fischer

(2004) present examples for facilitating social creativity, which they provide in Table 7.1.3. Language designers can facilitate community formation through the same mechanisms that support social creativity, in particular facilitating the interconnection of sub-communities that Morreale et al. refer to as pluggable communities, accelerating and diversifying community formation.

	Barriers	**Opportunities**
Spatial	Face-to-face supports maximal bandwidth; face-to face limits number of participants	Involving larger communities ("the talent pool of the whole world"); exploiting local knowledge
Temporal	Communication through artifacts; inherent difficulty of collaboration between people who do not know each other	Building on the work of the giants before us
Conceptual	Focus solely on communication; group-think	Making all voices heard; integrating diversity
Technological	Focus on what is technologically doable; requires formalization	Things are available all the time; computer-interpretable structures enable support mechanisms

Propagation

In new languages, the challenge of generating outputs that can be easily propagated from the designer to users (in support of feedback and sharing) creates a challenge in supporting and maintaining multiple integrations with different technologies. Whether this is the target operating systems, architectures or even other runtime environments such as plugin formats, this barrier often forces new languages into smaller niches, such as supporting only one desktop environment.

As such, it is recommended that language designers aim to separate their language into a front end and a back end, a common strategy in compiler design (Aho, Sethi, and Ullman 2003). Typically this involves creating some intermediate representation that can be transformed into multiple outputs supporting a range of target architectures.

This idea can also be realised through both **transpilation** and **EDSL** design, where the language is transformed into, or embedded in, a language that already targets the desired platforms or ecosystems. Once again, Faust provides a good example of this, however, other relevant examples in digital lutherie include the use of OSC as an intermediate representation in both Tidal cycles and Norns to use supercollider as a runtime environment for sound synthesis.

Propagation of technology largely orients around the technical requirements of targeting various devices, a challenging goal for which solutions are continuously evolving. This is best supported through the developing practises of separation of concerns and tools that separate the language from the device on which it runs, where examples such as WASM[2] and provide a compelling technology that may be useful in realising these needs (Gaster and Cole 2020; Gaster and Challinor 2021).

Stability

The effects of time on software are hard to predict and a challenge for digital culture on many levels.

Information storage and file reading which are relevant to the configuration and storage of many tools in digital lutherie are affected by the bitrot phenomenon (Król and Zdonek 2019). Participants presented the need for their instruments to remain functional years into the future and are motivated to use more popular tools, often grounded in other disciplines, to ensure that

[2]https://webassembly.org/

the community continues to maintain the technologies they rely on. This motivates two things in the design of new languages for digital lutherie. Support for the continued co-evolution of the instrument, where factors like opensource code reassure users that all stakeholders maintain the project, rather than depending on a single 'owner'.

Because technology is a continually evolving medium of interconnected technologies, the root cause of bit rot in the first place, the social role in digital lutherie is not only in creating but also preserving the DMI.

This necessitates a maintenance culture supported by documentation and tooling that can support the ongoing maintenance and onboarding of new maintainers. While approaches such as **type systems**, as discussed previously, may encourage and support this, we also see the value of translation, facilitated through **semantic first design**, as having value here. This would present the possibility of automatically translating source code for instruments for which there is no longer a functional runtime or target (Mariano et al. 2022). The design of languages through more formal language design allows for many benefits for digital lutherie. As Bartha, Cheney, and Belle (2021) describe, rarely are programming languages designed with a formalised semantic model in place. As such, many tools must first construct a model of the given language before they can offer the intended benefits whereas, drawing on a semantics-first approach to language design allows for deeper and more rapid integration with tools, as well as improving paths to translations that can provide analysis of software correctness or translate code into alternative paradigms or languages altogether.

Composability

Digital luthiers emphasise a need for their tools to support the decoupled and composable nature of DMI, offer them domain-specific expression, and support sharing and performance on different platforms. In software design, loosely coupled and modular design is considered a common goal, however, it also tends to increase complexity. Where languages interact using runtimes, protocols such as MIDI or OSC provide the current standard for interoperation, decoupling through a messaging-based model (Figure 7.3). This typical model tends to require the tighter coupling of the mapping engine and sound engine as described in the instrumental model (Magnusson 2010a).

For deeper integration, foreign function interfaces provide an option. In particular, the use of the hourglass model (Beck 2019), where an interface written in C [3] separates the underlying implementation and the calling system, creating a common, minimal interface. This is an approach explored in Gaster and Cole (2020) and Gaster and Challinor (2021)'s AudioAnywhere work, as well as an established model for implementing foreign function interfaces in programming languages (Stefanus DuToit 2014).

Cross-language composition may also be a desirable strategy to mirror the composability of DMI, and there are a number of possible ideas that may help with this challenge. This approach may be described at a high level through the diagram shown in Figure 7.5. Cross-language composition describes the potential of different languages through some form of interface, allowing the high-level 'puzzle-like' construction, where given the correct interface, languages can be joined. While this design is challenging and requires considerable planning, some of the strategies introduced here may help to realise this goal, where future work may consider this approach through the use of extra 'adapter pieces', that act as converters to allow the interfaces to connect. This opens up further experimental ideas such as type propagation between languages (Patterson and Ahmed 2017). Approaches such as this further support the need for defined semantics for languages and tie in the previously discussed expressive benefits of type systems with the compositional properties discussed here.

[3] Using the C Application Binary Interface (ABI)

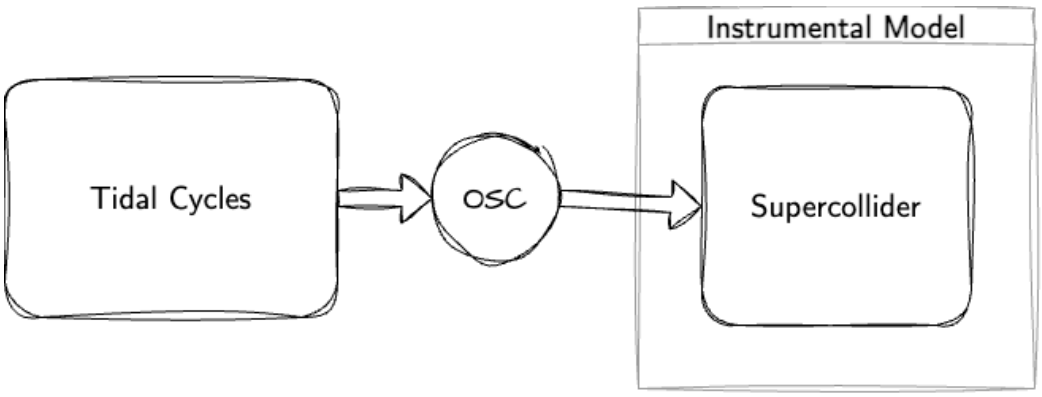


Figure 7.3: Decoupling through a protocol such as OSC or MIDI.

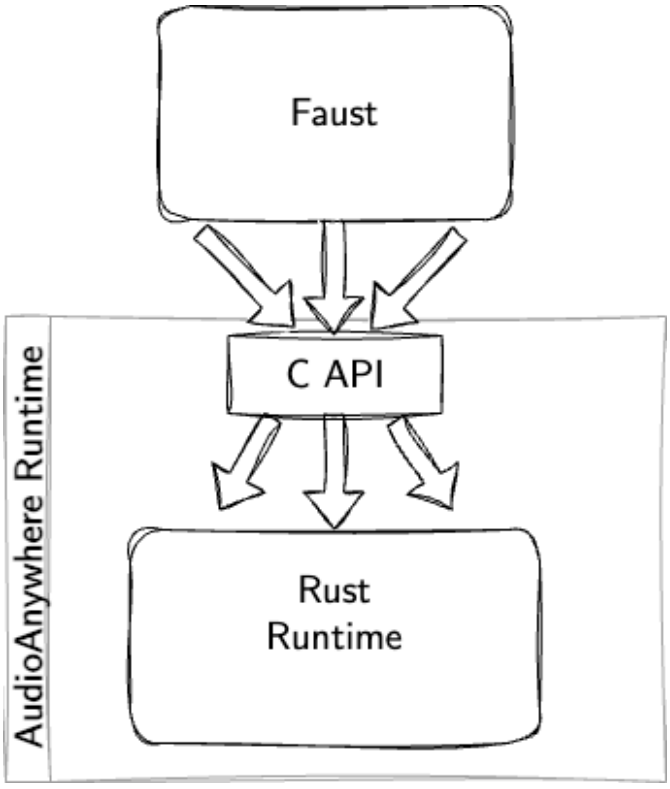


Figure 7.4: The hourglass model, using the Application Binary Interface (ABI) for foreign function interface.

The previously mentioned strategies of Transpilation and EDSLs (Dinkelaker, Eichberg, and Mezini 2010), discussed in Chapter 3, can provide a solution to composing domains through unification in a single host language. These approaches provide highly integrated and achievable approaches to improving composability between domains for new DSLs. Transpilation allows for

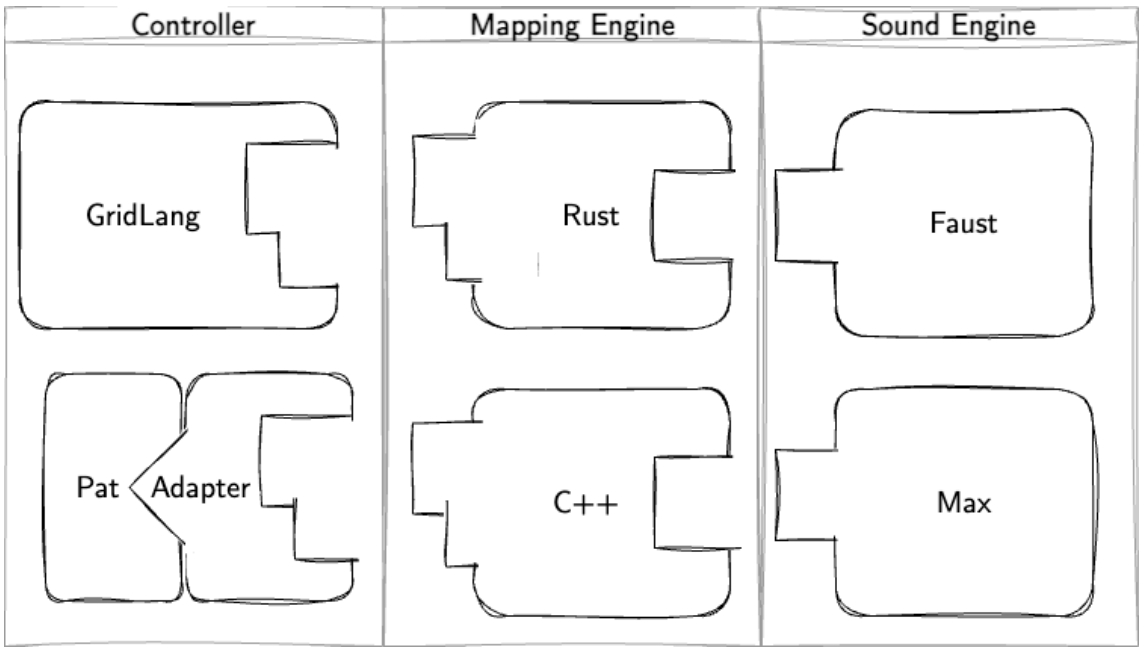


Figure 7.5: A model of the composition of languages per domain, where languages are composed together to each manage one domain of the instrumental model.

the generation of libraries for use in a wider codebase (Figure 7.6, where languages are unrestricted. We see this model already in use by Faust. The pattern using EDSLs is currently less common, but one we have explored previously and suggest is a powerful pattern to consider in future. Where languages support it, EDSLs provide the most seamless experience, acting as highly custom libraries and integrating with a codebase in the most idiomatic way (Figure 7.7). Languages such as OCaml, Haskell and Rust provide exceptional support for this approach.

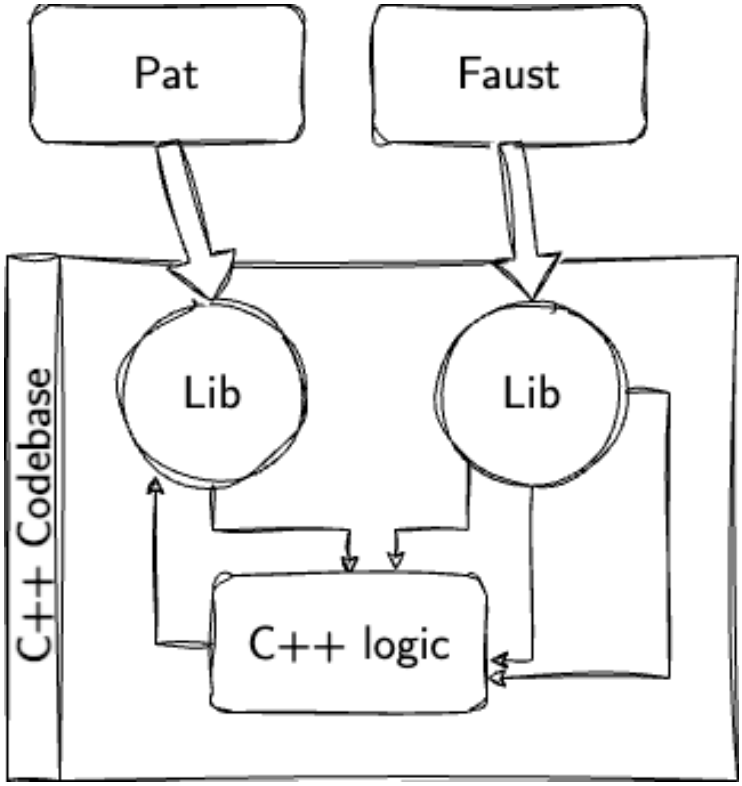


Figure 7.6: A model for composing domains through a shared transpilation target language.

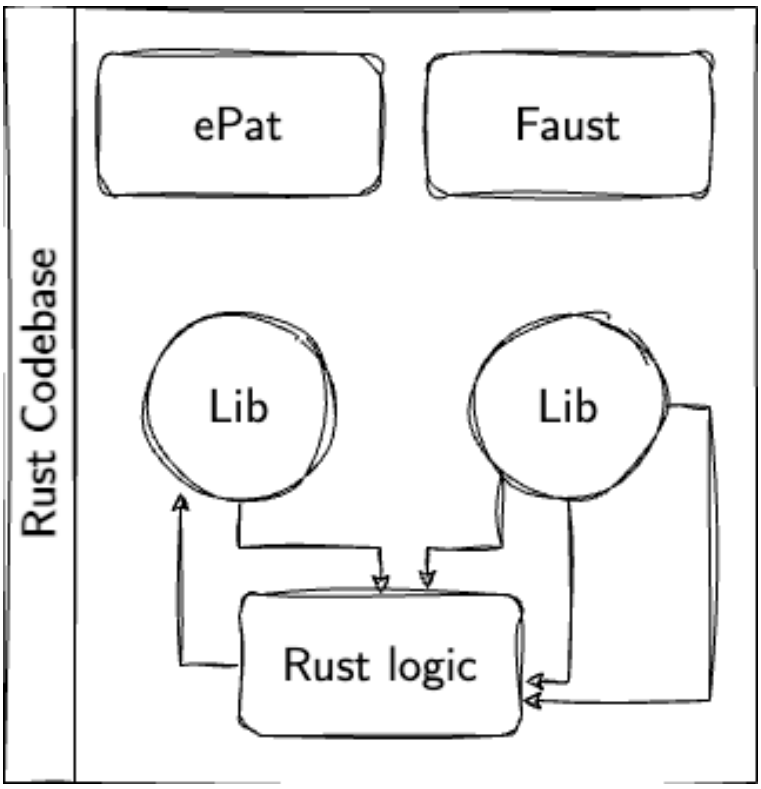


Figure 7.7: A model for composing domains through embedding in a single target language.

7.1.4 Summary

In response to the selective pressures presented in this chapter, this section introduces a selection of programming language development strategies that support the needs of digital luthiers. The implementation of domain-specific languages (DSLs) are discussed through transpilers, a common and growing trend, and embedded domain-specific languages (EDSLs), both of which provide patterns that address multiple selective pressures. The value of designing languages with a semantic first approach is introduced to provide a more robust and stable foundation for language design, supporting better integration and tool support in a new language. This feeds into the need for community building for new languages, where tools and features such as static types act as a bridge between the language and community, aiding in maintenance and education. This semantics first design also facilitates strategies such as bi-directional code translation, where source code can be moved between paradigms and languages, supporting new ways of reasoning and potentially a strategy for futureproofing work through translation to contemporary languages. Additionally, a series of patterns for supporting the composition of DSLs was described. Given the mutable nature of DMIs, discussed in Section 6.2, a system for supporting the domain-specific expression of instruments in this manner offers many advantages, mainly where high levels of optimisation or the target environment is resource-constrained yet still represents the entire coupled model of the instrument [4].

Many of these ideas can be related to work in Chapter 3 where, given the ideas developed through this study and analyses, there is a clearer direction for these explorations to build upon. Whilst they were initially explored based on intuition, certain approaches used in exploring language design clearly relate to the findings in this work. Transpilation techniques and the use of DSLs are well supported in the pursuit of composable and expressive tools for example. However, the implementation of ideas in Chapter 3 may be misaligned with other selective pressures identified in this section. Whilst the embedding of a DSL is a powerful concept that aligns with many

[4] for example, in fully stand-alone embedded instruments such as Artiphons Orba or the open source OTTO

selective pressures, social influence, cognitive fit and epistemic languages all present a challenge when leveraging unfamiliar ecosystems of technology, which, in the case of this work, relates to the use of Haskell for these implementations. We see that the use of new paradigms and unfamiliar languages can create friction in language adoption. This suggests the contributions in Chapter 3 could be developed to further align with the selective pressures outlined in this chapter in order to further explore tools for digital lutherie. In particular, as is outlined in Section 3.4.6, many of the ideas presented in this early work apply ideas from the functional paradigm to digital lutherie. This implies a different form of epistemic knowledge within the language and may therefore, suggest developing such knowledge within the digital lutherie community. Future work developing from Chapter 3 will therefore build upon a better understanding and develop the role of paradigms and idioms within programming languages for digital lutherie.

Rust as a Candidate for Digital Lutherie

Much like in digital lutherie, the C/C++ combination of languages dominates the niche of performance-oriented programming, such as systems programming and embedded systems. (Chatley, Donaldson, and Mycroft 2019) discuss the potential of new languages to replace C and C++, given their 'unsafe features'. They concede that there is unlikely to be enough inertia in new languages to remove such as dominant presence. As discussed in Chapter 5, digital luthiers show preferences toward familiarity and pragmatism and languages with a well-established niche, such as C and C++, represent a presence that even potentially 'fitter' languages in the evolutionary sense would struggle to replace. Perhaps as a reflection of the wider programming community, we see that the Rust programming language has become a newer language to warrant some exploration from digital luthiers. By design, DSLs are poorly placed to provide the digital luthier access to the systems-level programming that is often required. Rust presents one of the most contemporary options in this space that combines enough maturity in the ecosystem to be of value to the digital luthier. Rust provides effective and contemporary solutions to many selective pressures for languages in digital lutherie.

Rust meets many of the pragmatic considerations, being highly performant and modular both in its language design and through tooling such as its powerful package manager, Cargo. These features alone address many pressures, such as stability, propagation and composability, whilst lowering barriers to entry. Rust also has a powerful macro system, suitable for building EDSLs and has a powerful type system. Interestingly, whilst the language is multiparadigm, allowing for the expression of ideas in a number of ways, in light of the discussion on epistemic languages, Rust also represents both a fairly opinionated language and community. Along with many technical ideas, this makes it an exciting candidate to further explore the idiomaticity of programming languages further in the context of digital lutherie.

7.2 The Next Steps: Understanding the needs for Digital Luthiers

In inductively exploring the craft of digital lutherie and the relationship of digital luthiers and their tools, we have contextualised our interpretation in light of a number of key pieces of literature. We suggest that future studies may look to investigate some of the following ideas, testing the hypotheses formed through the thematic analysis presented here.

7.2.1 What are the Implications of Influential Programming Languages?

Gaver (1991)'s models of affordances are a longstanding approach to understanding interactions with technology. In context of the use of DMIs Magnusson (2010a) presents the users' exploration of DMIs through constraint, where after an intial exploration of affordances, users spend their time internalising the constraints of a system. Similarly, we see that participants described an active interest in systems of constraint. Features such as type systems and highly idiomatic, domain-specific libraries provide excellent examples of the pursuit of constraint in programming, with Magnusson (2010b)'s observations as an example, partially being contextualised in their live coding language.

Digital lutherie spans many different practices of programming (Bergström and Blackwell 2016), from software engineering to tinkering and hacking. Digital lutherie is predominantly a form of programming tightly related to creativity in much the same way its output (the DMI) is. Analysing digital lutherie in the context of constraints has clear implications on the DMI that are created (McPherson and Tahıroğlu 2020), where languages impart a bias and influence that may be difficult to avoid. While participants in this study showed awareness of this influence of their tools, for the most part, this impact was framed as a positive guide to better solutions. McPherson and Tahıroğlu suggests that it is the cultural and creative influence that may be implicit in the tool that needs consideration, where even the western notion of a 'page' presented by an empty MAX/MSP window can suggest some bias (Puckette 2002). This motivates developing a better understanding of what the impact of these influences is, and how they can be made explicit such that the digital luthier can factor them into their use.

7.2.2 How can PLs Support EUD In Digital Lutherie?

The analyses in the previous two chapters have drawn particular attention to the need to support End User Development (EUD) in digital lutherie. Participants identified DMIs as inherently evolving artefacts constructed through loosely coupled components. The co-evolution of DMIs is a largely social endeavour, and language design can address and support EUD in many ways. Fischer (2021) describes EUD as a 'suitcase word', stating that they are 'words carrying many meanings so researchers and practitioners can talk about complex issues in shorthand'. They describe many domains of EUD, including many relevant to the field of digital lutherie including meta-design, remixing and read/write cultures, social productions and cultures of participation. Primarily, this research identified two key components of EUD that require reflection in programming languages for digital lutherie. First, and perhaps most obviously, is its relationship to social creativity. In providing the early definition of digital lutherie Jordà (2005) described a role alternating between 'edit-mode' and 'run-time-mode' where the musician is a luthier-composer focused on the design of the instrument, then changes to the improviser-composer who need find answers to the 'questions posed by the luthier-composer'. As the relationships in digital lutherie have led to a more distributed network of roles, these questions are presented through

The language designer must consider this process as a core component of their language design, finding means to facilitate the dialogue of social creativity through their tools. Cultures of reuse and remixing can be seen to be developed through communities, in particular, through allowing so-called 'pluggable communities' where the ability to support well-established existing technologies facilitate faster and more diverse forms of community (Morreale et al. 2017). Further, we can see that cultures of sharing and remixing develop in ecosystems where the tools imply open-source and sharing in their design. The javascript ecosystem, in particular, package managers such as NPM [5] implicitly encourages a 'read/write culture', where sharing source code, reuse and remixing is the primary mode for the tool. Distributing closed or hidden code is a far higher friction way to

[5] https://en.wikipedia.org/wiki/Npm_(software)

work in these environments; therefore, it is simply not the default (Lertwittayatrai et al. 2017). Our findings support this view emphasised through themes in both analyses and ultimately in the selective pressures derived from them, where community formation should be a primary and first-class focus for language designers. To address this issue, we suggest that language designers aim to build their own 'pluggable community' and engage in the ecosystem that mirrors the composable nature of DMI.

The second key tenet of EUD that we find in digital lutherie is meta-design. Fischer, Fogli, and Piccinno (2017) calls for: 'new theoretical frameworks, new discourses and shared languages about concepts, assumptions, values, stories, metaphors, design approaches, and new learning theories, such as those aimed at promoting computational thinking.' We suggest that the literature on digital lutherie provides an excellent foundation for this already, where work such as that of the D-Box explores ideas around meta-design (McPherson et al. 2016; Zappi and McPherson 2018).

Artificial intelligence (AI) plays a crucial and developing role in meta-design (Fischer 2021). With the recent implications in the field of AI, in particular, that of large language models, there is the potential for new paradigms within programming and EUD to emerge, extending the tools of the programmer with powerful new modes of expression through natural language. We already see examples of this with the AI programming tool CoPilot(Imai 2022). This field is a very emerging side of HCI that needs attention to understand the implications for HCI but also the ethics of such tools.

From the study discussed throughout this work, and the selective pressures we define, we find that a number of design guidelines provided by Fischer, Fogli, and Piccinno (2017) are pertinent to the support of meta-design in digital lutherie. Fischer, Fogli, and Piccinno advocate for supporting domain specialists, who are interested in solving their own problems, rather than the problems of computing. Throughout this chapter we have presented some technical solutions that may support this. In particular, we note how the use of embedded DSLs in powerful languages such as Rust or transpilation to common targets like C++ may allow for strong integration between domain-specific problem-solving and the *essential requirements* of a system, engaging the more focused domain specialist being directly engaged in the meta-design process without the need to consider implementation details. Beyond this and considering EUD more generally, supporting this practice within digital lutherie focuses more on social aspects. As in the example above, technical ideas should be introduced to serve a role within the community, addressing many of the other design guidelines set out by Fischer, Fogli, and Piccinno (2017) and reflected in work in the digital lutherie community (Morreale et al. 2017) by facilitating accessibility and engaging communities with different perspectives and knowledge bases.

To language designers, we point to the selective pressures in Section 7.1, which, when combined, support end-user development. To researchers, we advocate for the value in further understanding how all stakeholders in digital lutherie can benefit from the languages used to create DMIs.

7.2.3 How can we Avoid Compromise in Digital Luthiers Tool Choices?

Through our first analysis, we found the digital luthier's pragmatic requirements to significantly influence tool choice, aligning with the work of (Stolterman and Pierce 2012). Where Stolerman and Pierce consider Argyris' theory of action (Argyris and Schön 1974) to suggest that pragmatism represents espoused theories and the environmental aspects to relate to theories in use. We found this did not hold for digital lutherie. Digital luthiers in this study demonstrate an awareness of how their espoused theories and theories in use interact, where we later develop the espoused theories as a set of themes around idealised languages they might use. Digital luthiers largely describe their theories in use as compromises that often require pragmatic choices, or follow familiarity and, therefore, convenience, prioritising working toward an outcome. For the adoption of future

languages, it will be important to understand this process further in digital lutherie, such that languages can become closer to the espoused theory and meet more of the requirements of the digital luthier. Much like DMIs, new languages for digital lutherie are faced with challenges in adoption, where more familiar or lower friction languages are preferred. In developing a better understanding of these tradeoffs, we believe it may be possible to move towards languages that are more enjoyable to work with and offer more idealised features that participants in our study discussed, converging on what may otherwise remain espoused theories.

7.2.4 Contributing to HCI Research on DSLs

As highlighted in Section 2.1.7, the field of research into DSLs lacks research on their evaluation and use. Given the varied and engaging ecosystem, we suggest that digital lutherie can contribute significantly to this. The range of DSLs used in digital lutherie includes many different paradigms and different use cases; Faust for generating DSP processing that can be used in other contexts from on platforms such as Bela or even microcontrollers. As Chasins, Glassman, and Sunshine (2021) suggests, the world of programming language theory would benefit significantly from cross-disciplinary collaboration with HCI, of which the broader DMI and digital lutherie community clearly are. While more nuanced research relating to the musical and artist craft is nurtured in conferences such as NIME, researchers should also look to contextualise the study and evaluation of DSLs in the wider HCI context and incorporate these findings in venues that help make the wider HCI community aware of these developments.

7.2.5 Mapping Problem

While this was not touched on directly in this thesis, inspired by early language explorations and codes from the interviews that did not feature so heavily in these analyses, a number of participants highlighted the so called 'mapping problem' in digital lutherie (Hunt, Wanderley, and Paradis 2002; Tanaka 2010; Van Nort, Wanderley, and Depalle 2014). There exist many domain-specific solutions to DSP in the world of computer music, however the mapping problem continues to be a challenge. There are of course, many approaches to this problem (Fiebrink and Cook 2010; Brown, Nash, and Mitchell 2018), however through exploration of paradigms such as function programming (Wadler 1992), and in particular functional reactive programming (Elliott and Hudak 1997; Helbling and Guyer 2016) there is a lot that programming language design can offer in this space, potentially leading to new idioms for understanding the mapping problem. In pursuit of providing good domain modelling for every domain of digital lutherie, new expressive paradigms to map interactions to sound suggest a compelling area for future work and one that can be explored in harmony with the other ideas introduced in this work.

Chapter 8

Methodologies

8.1 Designing Studies to better understand PL and HCI

Qualitative research is a key methodology in researching complex human interactions that reflect the real world. There are many popular methodologies for qualitative analysis including thematic analysis (TA), grounded theory, ethnography and interpretative phenomenological analysis often referred to as IPA.

Qualitative methods offer the potential to examine people's experiences, an ideal tool for HCI research, however, conducting qualitative research requires an extensive background in understanding the chosen methodology, with many implications relating to philosophies such as positivism, constructivism and critical theory. Whilst the popularity of these methods are growing in fields outside of sociology and psychology, it is not obvious that these methodologies are being effectively used in disciplines such as Computer Science and HCI. This has motivated work on tools and frameworks to assist researchers (Gauthier and Wallace 2022), and for approaches to the practice and publications of researchers in this field to be examined (Hoda, Noble, and Marshall 2011; Bowman et al. 2023).

On examining methodologies suitable for this work, it became apparent that, quite rightly, a considerable range of strategies and methodologies are used in HCI. In particular, and likely due to the highly cross-disciplinary nature of the field, many papers present both technical work and studies in a single paper. By combining qualitative analysis with the evaluation of technology in short conference papers minimal space is reserved for the clear description and transparency of the studies that are performed, which raises considerable methodological questions about the rigour of the work. This is particularly typical in examples of HCI work where technical development is followed by evaluation, most often through a user study. This condensing of work is common in science, where the pressure to turn over ideas and maintain a high output of papers that demonstrate positive results leads to the potential for poorer quality research (Fanelli 2010). This has resulted in a counter-movement, aiming to promote rigour and transparency in research (Robson et al. 2021).

As a field of Computer Science, HCI work requires a highly varied skillset incorporating elements of psychology and sociology. These skills need to be built through cross-disciplinary teams or taught programs, however, HCI practices are still a developing field of computer science in this respect (Ramirez V et al. 2021), and while there is, of course, good progress and research being done in the field, there is still significant room for improvement in the application of qualitative analysis (Matavire and Brown 2008; Gauthier and Wallace 2022; Braun and Clarke 2019; Bowman et al. 2023), particularly in programming languages research (Chasins, Glassman, and Sunshine 2021).

Having attempted to apply a rigorous methodology whilst remaining faithful to the reflexive

TA defined by (Braun and Clarke 2020), this section documents ideas that can combine reflexive TA with other approaches inspired by other fields of science and even open-source software culture to provide a call to action for HCI research to develop and be open to rigorous qualitative analysis. As such we present a brief overview of our practice and experience to help stimulate and propagate best practices for the field of HCI.

8.2 Promoting Rigour in Qualitative Research

Much like ongoing efforts across science to improve methodological approaches, this section shares strategies for improving rigour in qualitative work. There are often criticisms of the value of qualitative methods in science due to the challenges of interpretation and the nature of being inseparable from human tendencies. Of course, science as a collective endeavour is in the hands of humans, and quantitative methods depend on rigorous application just as much as qualitative methods should.

Through the course of working with reflexive thematic analysis, a number of ideas to aid in ensuring rigorous and accountable TA were identified, largely borrowing ideas from the emerging best practices which address methodological issues in quantitative methods. In particular, problems such as P-hacking (Bruns and Ioannidis 2016) and 'HARKing' (Cockburn, Gutwin, and Dix 2018) are primarily approached through transparency and easily audited documentation of their method. Not only do these approaches benefit the validity of research, but they also improve the capacity for replication. We suggest these as opportunities which also benefit qualitative analysis, where strategies such as data sharing and prepublication can improve the validity of interpretation and stimulate secondary interpretations and, therefore, a deeper understanding. This is particularly important in a qualitative study, as inherently, researchers impart their own biases on research, which must be factored into any results interpreted in a study.

8.2.1 Transparency

Transparency in methodology is crucial in science, allowing readers to understand the biases and overall context of the work, empowering them to make gauge validity. We argue that this transparency must also be reflected in the qualitative analysis process. This reasoning is twofold; this allows for further exploration by peers who may challenge the conclusions of the authors, a process that is particularly important in qualitative research where interpretation is fundamental to the process. Further, this also opens the door for peers to extend and reevaluate work. This allows for accessible analysis of the information, uncoloured by the initial analysis and further allows for the development of alternative interpretations, creating a richer and more nuanced discussion of the source material. Other's have observed the need for transparency in related fields such as design (Meyer and Dykes 2020)

Borrowing a strategy from other areas of science, in undertaking this study we chose to pre-register the methodology for the study before undertaking analysis (Yamada 2018; Bastian 2014; Lauer, Krumholz, and Topol 2015). We chose to use this strategy of pre-registering the method to increase transparency and provide an extra point of contact that could provide more in-depth detail on the method, which can be easily referenced in the final publication, saving space without sacrificing transparency (Bowman et al. 2023). This method is also seeing adoption in fields conducting empirical research, such as psychology, with the journal 'Psychological Science', even going as far as promoting this method by presenting 'Open Science Badges' for pre-registered studies (Pham and Oh 2021).

Though there is some push-back against the pre-registering of studies, we believe these issues can be resolved as the practice becomes better understood and more widespread. Pham and Oh

(2021) suggest that, amongst other things, preregistration can lead to a false sense of transparency, with authors 'running long calibration tests' or suggesting they 'specify the hypotheses to be tested and analyses to be performed in loose terms'. We argue that the potential for abuse doesn't completely undermine the system and that the benefits far outweigh the costs. They go on to suggest that 'open data access, self- and independent replications, and multiverse analyses' would all be more useful. We found the use of pre-registration to aid with these tasks when performing qualitative analysis such as TA and therefore can only suggest they present additional benefits in such a system.

8.2.2 Reproducibility & Replication

In quantitative studies, the reproducibility of results is an important step in establishing a valid result. These replication studies are uncommon in qualitative work and arguably qualitative studies cannot be reproduced (Meyer and Dykes 2020). However, when performed correctly, sharing data from qualitative studies can be a valuable contribution. This is not without its risk, where the decontextualisation of the data can impact the interpretation (van den Berg 2008). But in the correct settings, there is growing recognition of the potential for better data sharing around in qualitative research (Alexander et al. 2020). A considerable amount of work already goes into collecting and preparing data for qualitative analysis, potentially suggesting that adding to this will make research harder. But with the technological improvements both in the handling of data and in the storing of social bandwidth, there are many ways in which workflows can be made easier and sufficient context can be stored. While we see the sharing of data extracting further value from the effort of data collection, it also has the benefit of enforcing better practices and preventing shortcuts and bad habits as data becomes auditable (Nosek et al. 2015). It also encourages a better post-publication culture, improving the capacity for peer review beyond just the publishers (Bastian 2014). While Pham and Oh (2021) do not believe that pre-registration is an effective means for improving this culture of review, they do advocate for it in the context of data-sharing.

In order to share data, there are, of course, significant ethical considerations that need to be factored in, which likely have been a limiting factor for data sharing in many cases (Feldman and Shaw 2019). In particular, Feldman and Shaw discuss the implication of sharing data from vulnerable groups or even being granted consent for use with future unknown researchers. We certainly cannot advocate for data-sharing practices lightly and without due consideration for these factors. However, with good methodological design, the development of supporting technologies and considering this practice early, data sharing practises can be a highly beneficial addition to qualitative research in HCI in many cases.

8.3 Reflexive Thematic Analysis as a tool for Programming Language HCI

Thematic analysis is a popular research method for its unrestrictive relationship to philosophies and other theoretical commitments (Clarke and Braun 2017). TA can be applied with respect to a number of these theoretical frameworks, research paradigms and fields. As a research method, TA is an accessible means for analysing a range of qualitative data through the generation of codes which are used to form themes that interpret the shared narrative of people's experiences, allowing for reporting, analysis and observation. While a range of qualitative methods are used in HCI, our research and experience suggests that TA, and in particular, reflexive TA (Braun and Clarke 2006; Braun and Clarke 2012; Braun and Clarke 2019), provides an exceptionally flexible and powerful tool for conducting qualitative research in the field of HCI.

Many areas of HCI can benefit from the more holistic approach to the observation that reflexive TA offers. As an example, the topic of this thesis, the HCI of programming languages, has traditionally been focused on evaluating usability through small user studies, often isolating small groups or language features to observe. Often, this is not reflective of the natural setting and use of programming languages but observing the work of hundreds of programmers remains practically challenging, and surveying using course questionnaires yields expectedly unconvincing results.

Studies, therefore, often turn to methods such as grounded theory and TA. As these methods become more popular, the value of such methods becomes apparent, producing far more nuanced observations of populations of programmers (Lubin and Chasins 2021).

However, these methods are often lacking in rigour, where often the practices used are under-reported, and the role of reflexivity is under-emphasised (Bowman et al. 2023). As the practice of using TA develops (and other qualitative methods), we suggest a need for a better presentation of the method and analysis in the pursuit of transparency and in order to support open science (Nosek et al. 2015). We suggest that practices such as preregistration and data-sharing, in particular, would considerably improve the quality of research in the field of HCI and further suggest that a review of existing analysis, in light of Braun and Clarke (2019)'s more recent work is warranted, to discuss and share best practices in the use of this method.

Chapter 9

Conclusion

In 1966, Landin (1966) published the paper 'the next 700 programming languages'. Then 53 years later Chatley, Donaldson, and Mycroft (2019) followed up with the paper 'The next 7000 programming languages'. Much like these papers, this thesis set out to consider the human-computer interaction and evolution of programming languages. Through an inductive qualitative study and reflexive thematic analysis, this thesis begins to provide an understanding of the relationship between digital luthiers and their programming languages. Building upon the evolutionary analogy that Chatley, Donaldson, and Mycroft construct, we contextualise the discussion from this study

as a set of *selective pressures* that influence the digital luthier's motivations for the selection and use of programming languages. These selective pressures provide a set of ideas to further explore in future research and to provide a partial answer to the primary research question of this thesis:

How do we design the next generation of programming languages used in Digital Lutherie?

Of course, due to the inductive nature of this work, this research question is not answered in as much as it is used to motivate a discussion and exploration of ideas that provide a foundation for us to build and research new languages for digital lutherie.

This chapter summarises the contributions of this work in the following sections.

9.1 Explorations of New Music DSLs

In the early stages of this thesis, the design of a number of DSLs for use in digital lutherie were explored, presenting a number of techniques that were later offered as provocative ideas for language design that relate to the selective pressures derived from our themes.

Some of these ideas are presented in Chapter 3 and were featured in the following papers:

- Return to temperament (In digital systems) (Renney, Gaster, and Mitchell 2018)
- Digital Expression and Representation of Rhythm (Renney and Gaster 2019)

Further, the ideas developed through this time went on to influence and contribute to the following related works:

- Outside the block syndicate: Translating Faust's algebra of blocks to the arrows framework (Gaster, Renney, and Mitchell 2018)
- Fun with Interfaces (SVG Interfaces for Musical Expression) (Gaster, Nathan, and Carinna 2019)

Alongside exploring technical approaches to DSL design, these explorations also contributed novel approaches to encoding musical ideas that lie largely outside (or at least peripheral to) the current Western classical tradition of music. As such, they may present a starting point for exploring new idioms in programming languages for digital lutherie and, more broadly, musical expression.

9.2 Study Analysis: How do Digital Luthiers Choose Their Tools?

Following interviews with 30 prominent digital luthiers, described in Chapter 4, a reflexive thematic analysis was conducted generating three themes in response to the research question 'How do Digital Luthiers Choose Their Tools?' Themes titled 'The Pragmatist', 'A Product of Our Environment' and 'Intentions' where presented, contextualising the designer tool relationship of digital luthiers. These themes indicated that digital luthiers are aware of their 'theories in use' (Argyris and Schön 1974) and largely make choices through systems of compromise, rather than for idealistic qualities in their tools. These themes also introduced the challenges of meta-design in digital lutherie, the design of tools that are used by designers (Fischer and Scharff 2000).

9.3 Study Analysis: What do Digital Luthiers Value from their Programming Languages?

A secondary analysis using the same approach and focusing on programming languages was conducted following the previous analysis. The themes 'A Guiding Force', 'The Mutable Instrument' and 'Expressing My Ideas' were generated. These themes related their selection and use of programming languages to the constructed knowledge used to solve problems

(Ben-Ari 1998; Allen, Donham, and Bernhardt 2011) and how languages mirror these models through paradigms, idioms and domain-oriented design. These themes also point to the design of epistemic tools, where domain knowledge is embedded in the programming language to be leveraged by the user (Magnusson 2009). The theme 'The Mutable Instrument' also clarifies the roles of social creativity and meta-design in the programming languages digital luthiers use, demonstrating a need to support an ongoing co-evolution (Costabile et al. 2006) of the artefact (DMI) that is created, where programming languages play a critical role (Fukuda et al. 2021).

9.4 Selective Pressures on Programming Languages for Digital Lutherie

Through the analysis and discussion of the thematic analyses alongside existing literature, a set of selective pressures has been produced to explain the needs of digital luthiers. These selective pressures are described in Section 7.1.2, and Figure 9.1 visualises how the themes produced through the previous thematic analysis relate to these selective pressures. These selective pressures are presented to motivate design considerations for meeting the requirements of digital luthiers when creating new languages. They incorporate both technical and social requirements, and through Chapter 7, a number of ideas to address these pressures are presented. Transpilation strategies and the use of embedded DSLs are described to support the design of new DSLs, alongside some architectural patterns that allow for the composition of languages across the domains of digital lutherie. The use of well-formed semantics is also discussed as a cornerstone of language design that offers better support for tooling, learning and the potential for translating languages between paradigms, supporting alternative approaches to problem-solving.

9.5 Directions for Future Works

Following the inductive exploration of digital lutherie, four significant directions for future work have been created, where the ideas established in this thesis may be further and more deeply understood. In conclusion, we present three new research questions:

What are the implications of influential programming languages?

How can PLs Support EUD In Digital Lutherie?

How can we avoid compromise in digital luthiers tool choices?

In addition, we call the designers of new domain-specific languages (DSLs) for digital lutherie to consider contributing to the deficit in HCI research studying DSLs. Likely due to the inherent domain focus of DSLs, much of our understanding remains siloed in the niche that the DSL targets. However, much like we suggest that digital lutherie is a rich, real and well-researched field of digital lutherie, so too is the work on DSLs. We suggest that through collaboration with the wider PL/HCI community (Chasins, Glassman, and Sunshine 2021), music DSLs can contribute to the broader understanding and value of DSLs.

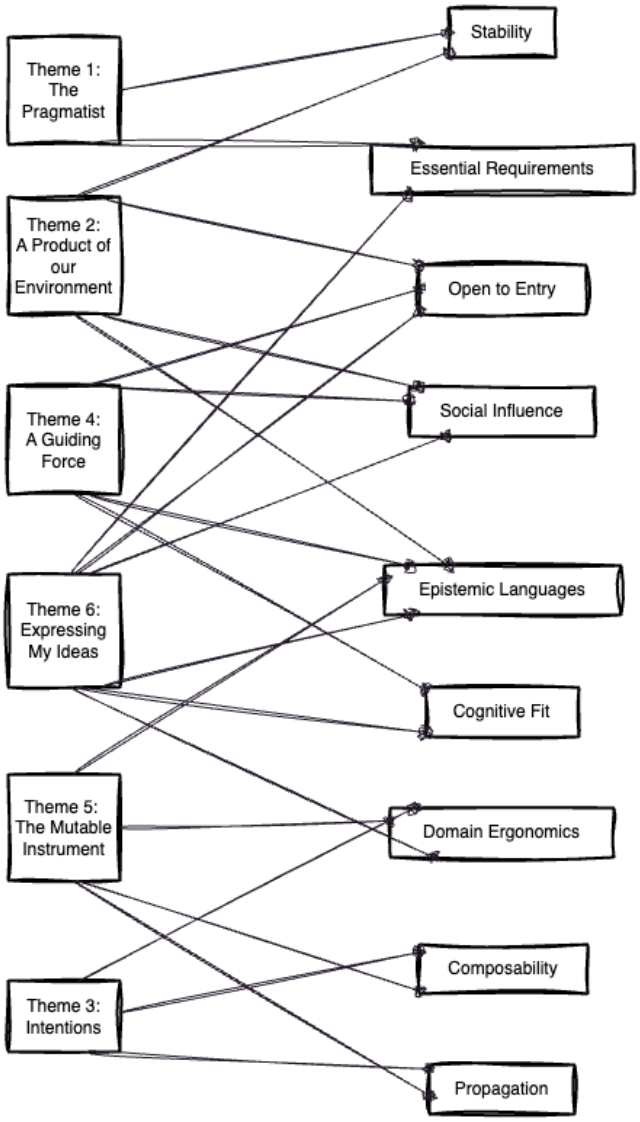


Figure 9.1: The mapping of themes to selective pressures, established in Chapter 7.

9.6 Reflexive Thematic Analysis

Alongside contributions to digital lutherie, this thesis aims to demonstrate the value of reflexive thematic analysis and attempts to provide a rigorous and transparent demonstration of the methodology. Currently, a number of qualitative analysis methods are used in the study of DMIs and in the wider HCI community. We suggest, however, that these methods often tend towards a 'lightweight version', which does not extract the most value from the methodology being used. In Chapter 8, a discussion around how the method may be developed within the HCI research community is presented. Through the experience of this work, it is suggested that further work is required in effectively using these methodologies and, in particular, highlights the suitability of reflexive thematic analysis (TA) as an appropriate methodology for HCI researchers. Chapter 8 discusses the practice of reflexive TA along with drawing together a number of practices that contribute to improving methodology in the space of HCI and potentially even further afield.

9.7 Final Words

The thesis changed from a journey to build a programming language to seeking to understand and motivate building better programming languages. In doing so, I found that building new languages is as much about people as it is about technical ideas. In much of language design, despite the construction of impressive and inspiring ideas that allow us to express our intentions in technology, too often, the human is forgotten or not given due attention. Digital lutherie represents an exceptional field. Music has always been a deeply creative endeavour, supportive of many technological cultures, art, composers and performers, never been constrained by existing music, nor has it shied away from adopting and experimenting with new technology, arguably being on the forefront and even potentially a motivator of technology. Having had the opportunity to speak to and work with such exceptional digital luthiers I am more motivated and equipped to be a part of the evolution of technology that it perfectly embodies. To those people, thank you. And to those looking to add to the collective knowledge and ecosystem of technology that is digital lutherie, I hope some of these ideas provide inspiration and guidance.

Appendix A

Participant Data

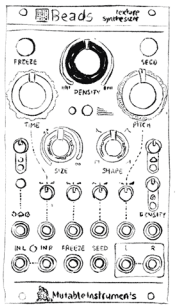
ID	Role	Experience	Instruments Created	Example Instruments	Languages
1	Music Technology researcher and professor.	30	10	FMOL, reactable (and many more)	C++, Pd
2	Owner of Mutable Instruments	11	35	Mutable Instruments Shruthi, Anushri, Ambika, MIDIpal, Eurorack line	C++, Python
3	Digital artist/performer/composer	10	4	Soft Revolvers, Ballistics, autopsy.glass	Supercollider, Max, Python, OpenScad, Touchdesigner, Processing, Javascript
4	Artist	30	3	Personal ones - the lady's glove, the Spring spare	Max MSP
5	Software Engineer	6	3	Polaron and other prototypes	C, C++, (Java / Javascript for non DMI work)
6	Software Engineering Manager	11	8+	Eigenharp, LinnStrument, GECO, Animoog, Model 15 App, Minimoog Model D App, Moog One, Claravox	Objective-C, C++, Swift, C, Bash, Python
7	CEO	8	100	Eurorack Modular, Desktop Synth, Effects Pedals, Audio Dev board	C++, Python, Csound, Arduino
8	Composer	4	2	No published products, tools for personal use	None in relation to this
9				Linnstrument	C, C++
10	CEO & Founder	10	5	Artiphon INSTRUMENT 1; Artiphon Orba; projects in development	C, C++, Apple Metal
11	Researcher and Lecturer	10	20	The BladeAxe, the PlateAxe, the Chanforgnophone, the Gramophone, Nuance, and many more.	Faust, PureData, C++, Objective-C, WebAssembly
12	CEO & Founder	13	3	AlphaSphere, BetaLoop, NUSIC	C++, Kotlin, Python
13	Assistant Professor of Music Technology	11	8	DMIs based on audio/visual physical modeling, hackable DMIs combinig digital and analog elctronics, immersive virtual DMIs, collaborative networked DMIs	C/C++, Assembly, GLSL, Pure Data; more rarely Super Collider, Max/Max For Live, C# [Unity], misc scripting languages
14	Composer & Instrument Builder	10	20	Lightdome, concertronica, eggiophone, the chromatic, the sonic bonnet and more	Arduino, Max

ID	Role	Experience	Instruments Created	Example Instruments	Languages
15	Software engineer / Audio developer	6	5	Ableton Live, Ableton Push, Ableton Simpler / Sampler / Wavetable	C, C++, Python
16	Researcher, designer, performer	12	8	ROLI Seaboard GRAND, Seaboard RISE, and Blocks, Stenophone, a series of "Unfinished Instruments" for my PhD. If you are counting DMI platforms: Bela	C/C++, JavaScript, Python, Haskell, SuperCollider, Pure Data, Bash
17	Software Developer	4	1	Mainly OTTO - a groovebox with synths, midi fx and audio fx	C++
18	Professor	20	15	mainly physical modelled based	C, C++
19	electronic musician	40	12+	a sample: 6502 based 8-bit instrument, Apple][based digital control for Serge Modular, Mac audio + MIDI software, RPi based effects unit + MIDI tools, MCU based metrical clock, MCU based looper + CV converter	Assembly, C, C++, Python, Haskell, Elm, Javascript, SuperCollider SCLang
20	Software Engineer	1	2-3	"SeqPal" and "DrumBud" for the Winterbloom Sol, and the Winterbloom Sol itself.	Python, Rust
21	Artist	3	2	Electronic_Khipu_	Java
22	DSP Engineer	4	5	The Daïs, Magrittophone, several unpublished	C++, C, Python, Matlab, Faust
23	Professor of Media Computing	35	5	EMG instruments like the BioMuse and the EAVI EMG, and have made musical instruments out of consumer devices like the Thalmic Labs Myo	C, C++, Python
24	HW and SW engineer	3	6	Gechologic Loopsynth, Phonicbloom's Glo Polyphonic Whale, MMXX T-ape, Wingdrum, Loopstyler, Don Iguano	C/C++, Python, php, Javascript
25	Lead designer	12	30+	the whole Bastl Instruments product line	C++, Arduino
26	Composer and interactive hardware developer	15	10+	Custom interfaces for live performance	Max/MSP, Supercollider, Arduino, Python, Lua, Rust
27	Creative director	2	1	Knurl	C++, Supecollider, Javascript

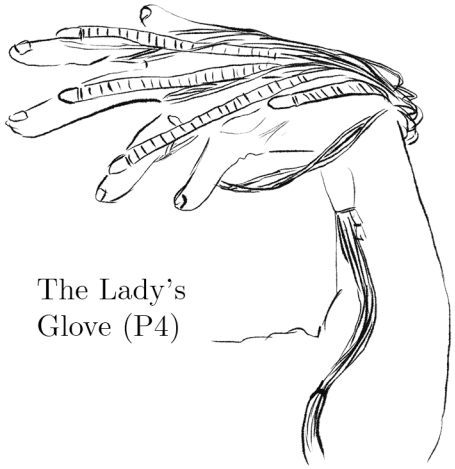
ID	Tools (Software)	Tools (Hardware)	Platforms
1		Sensors, cameras, projectors... It really depends on the instrument. I have also designed many mouse-based or touch-screen based instruments.	Linux, OSX, Windows, Android, IOS
2	vim, make, gcc, OpenOCD	STMicroelectronics Discovery boards, Olimex ARM-USB-OCD	STM32F
3	Fusion 360, Blender, Ableton, Eagle	modular synths, CNC, laser cutter, 3D printer, x-io, Arduino	Supercollider
4	max msp	MAC power book, Motu ultra lite	max map
5	Visual Studio Code / Teensyduino	Mainly Teensy / tried others like Bela / Raspberry / Axoloti	Teensy
6	Xcode, Electroacoustic Toolbox, MIO Console 3D, Grapher, CurveExpert Pro	Computers, Mobile IO ULN-8, various MIDI Controllers, headphones of various qualities, speakers of various qualities, acoustically treated room	macOS, Windows, Linux
7	Max/MSP, EAGLE CAD, Ableton Live, Audacity	oscilloscope, bench meter, audio mixer, soldering iron, spectrum analyzer	Arduino, Daisy, Raspberry Pi
8	For this project: Max Reaktor Variety of sample based synthesisers, Omnisphere for example It uses Ableton as host/mixer/midi router	For current project: Sensel Morph Variety of conventional midi controllers, Faderfox etc Octatrack	Mac Windows
9	Rhino	Not provided	Arm

ID	Tools (Software)	Tools (Hardware)	Platforms
10	JUCE; Solidworks; Various DAWs (Logic Pro, Ableton, etc.)	3D printing; laser cutters; CNC machines; soft-tooling; mass manufacturing; screwdrivers	Compatibility: MacOS; Windows; iOS; Android
11	SolidWorks and OpenSCAD	RPI, the Teensy, ESP32 boards, Zybo Z7, Android and iOS devices	Android, iOS, Linux, Free RTOS, Bare metal
12	VScode, Xcode, Android Studio	Injection Moulding	cross platform
13	Mainly IDEs and Pure Data; more rarely Max/Max For Live, Unity	GPUs, chip synths, custom electronics	Bela and custom platforms running on regular computers and sometimes on mobile phones
14	ableton, max, teensy loader, arduino	teensy, arduino, various breakout boards + sensors	arduino, teensy
15	Max MSP		OS X
16	TidalCycles, SuperCollider, Reaktor, JupyterLab, Svelte.js	Bela, Teensy	Bela
17	Visual Studio Code GCC/Clang and various libraries	Soldering iron, oscilloscope, multimeter	HW: A Raspberry Pi 3A+, SW is completely homemade
18	Matlab, Juce, Unity	Arduino, bela	Max, Juce

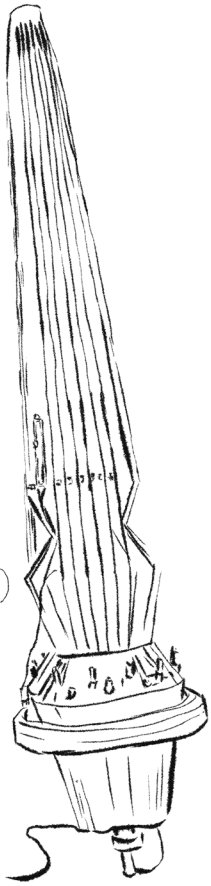
ID	Tools (Software)	Tools (Hardware)	Platforms
19	beside compilers and interpreters: SuperCollider IDE, Visual Studio Code, spreadsheets, Eagle, OpenJSCad	soldering iron, bread board, multimeter, oscilliscope	Arduino, SAMD21 MCU, Linux, Mac OS; in the past Apple][, 6502 systems
20	Vim CircuitPython	Microchip SAMD programming tools (but honestly Stargirl's work on the Winterbloom modules means I only have to use the native tools if I somehow break the module, which I've done once. Otherwise it's just plug in, edit in vim, go.) Various Eurorack modules, in an Arturia 6U case.	Linux; Microchip SAMD 50-series and SAMD 20-series chips
21	Arduino - processing - Pure Data - Max/MSP - Super collider - vvvv	teensy boards, different electronic sensors, Arduino boards, Bela sensors.	My own physical creations, MIDI controllers applied - Ableton
22	JUCE, Arduino, Processing, Bela IDE	The regular electronics tools (soldering iron, snips etc), laser cutter, CNC machine, 3D printer, various wood working machines.	Bela, Teensy, Arduino
23	Max, Pure Data	Myo, Beagle Board, LEAP Motion, Arduino, Huzzah	Max, TI Biosignal signal processing chipset, OpenBCI
24	Software: Atollic True Studio, CubeMX, Eclipse, ESP-IDF, Oracle VM Virtualbox, MinGW, Borland C++ Builder; PCBs: CadSoft Eagle, EasyEDA, Gerbv; enclosure & mechanical elements design: Onshape	3D printer, CNC milling machine, solder station & hot air for SMD rework, binocular microscope	STM32F4, ESP32 (ESP-IDF & FreeRTOS), embedded linux (mostly Allwinner)
25	Eclipse, Arduino, AVRdude, github desktop, eagle, illustrator, ableton live	digital oscilloscope, soldering tools, complete music studio for real-worlds testing	analog, avr chipsets, arm chipsets, arduino
26	Arduino IDE, SublimeText, Linux JACK	Norns, various midi keyboards, various sensors for arduino (most recently over I2C), wifi networks	Arduino, Teensy, ESP32, Max/MSP, Supercollider on Raspberry Pi, Norns
27	Fusion 360 Supercollider	Beagle bone,	Bela



Beads (P2)

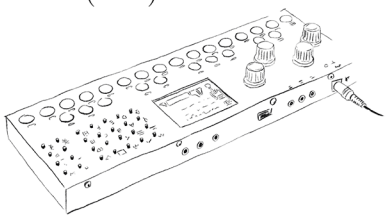


The Lady's Glove (P4)

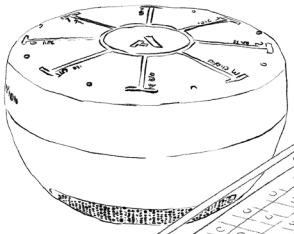


Knurl (P27)

Otto (P17)



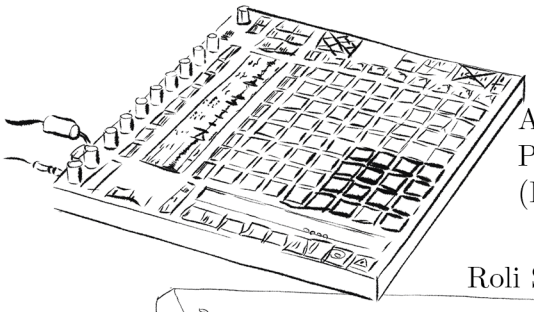
Artiphon Orba (P)



Linnstrument (P9)

Soft Revolvers (P3)

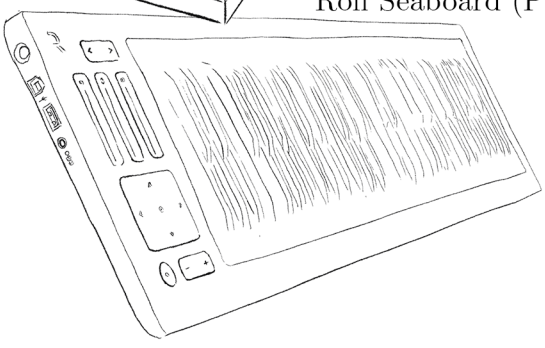
Polaron (P5)

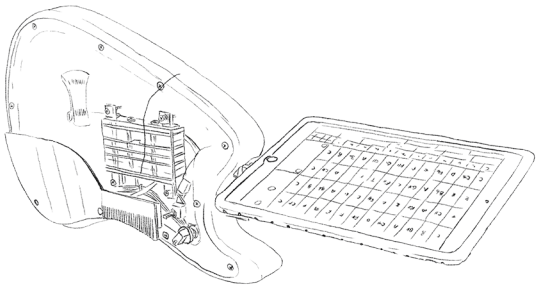


Ableton Push (P15)

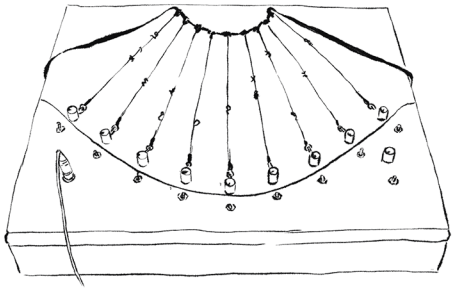
Reactable (P1)

Roli Seaboard (P16)

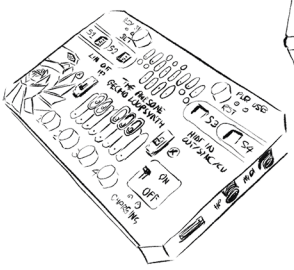




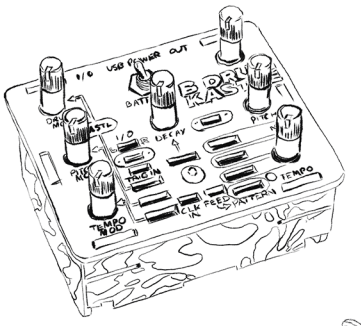
The BladeAxe (P11)



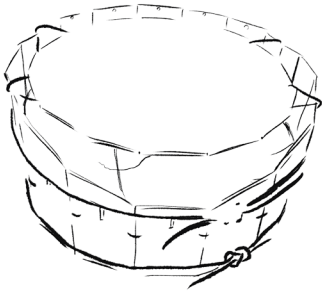
Electronic_Khipu_ (P21)



Gecho Loopsynth (P24)

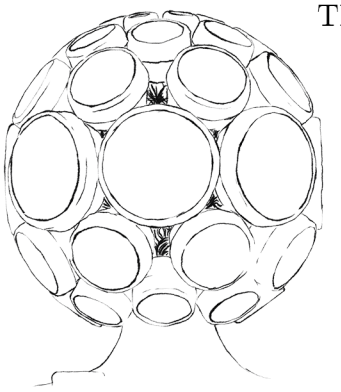


B Drum Kastle (P25)

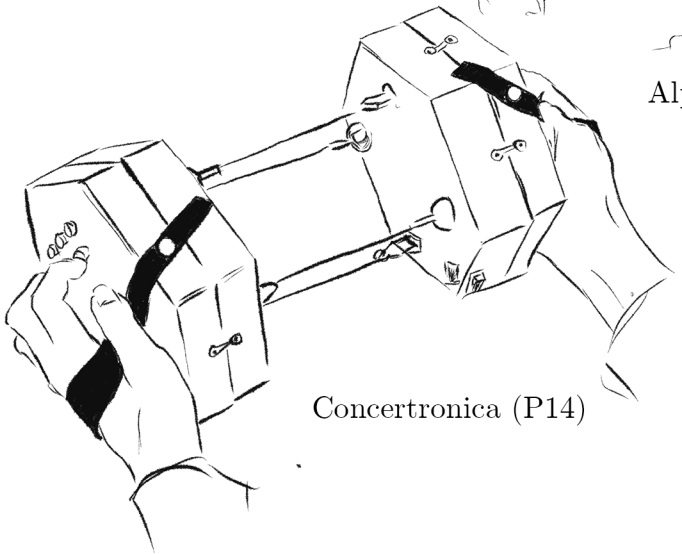


The Daïs (P22)

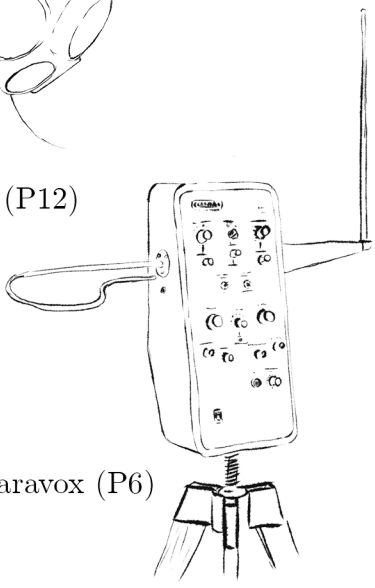
EMG Instrumnets (P23)



Alphasphere (P12)



Concertronica (P14)



Claravox (P6)

Appendix B

Preregistration of Study

Data and Methodology : Exploring How Digital Luthiers Choose Their Tools

NATHAN RENNEY, University of West England, UK
BENEDICT R. GASTER, University of West England, UK
THOMAS J. MITCHELL, University of West England, UK
HARRI RENNEY, University of West England, UK

Digital luthiers (designers of digital musical instruments) work with a range of both software and hardware tools in order to realise the instruments they build. A digital musical instrument is a highly performant piece of technology with which a performer may interact. This leads to these devices being multifaceted artefacts that combine fields of design, user interaction, audio synthesis and more. In this paper, we present the methodology and data for a study that explores how these digital luthiers choose their tools for creating digital musical instruments.

Additional Key Words and Phrases: digital musical instruments, human computer interaction, programming languages

1 INTRODUCTION

This paper is published as a supplement to a study that explores how digital luthiers[8] (designers of digital musical instruments) choose their tools, both software and hardware. It was intended to be pre-published ahead of the data collection phase, however, due to time challenges related to the COVID-19 pandemic it is published following the initial interviews and ahead of undertaking analysis. In this publication, we present the data that was collected and the methodology that will be used to analyse this data. The data is formatted and presented to provide an element of confirmability to the resulting study and further, to encourage research and analysis that builds upon this work.

This study was designed to collect perspectives from a range of prominent instrument designers from a diverse set of backgrounds.

This study was ethically reviewed by the University of West England's Computer Science and Creative Technology Faculty Research Ethics Committee. Following approval, participants were invited to share their thoughts on digital musical instrument (DMI) design in the form of a standardised, open-ended, online interview. Each participant was issued with a comprehensive information pack describing the study and was able to withdraw from the study at any time while it was active. Participants provided written consent and were able to review their transcripts following the interview and before publication.

2 MOTIVATION

As our role as researchers inherently impacts qualitative data analysis we present our motivations. This study seeks to explore ideas based on three research questions:

Why and how do Instrument designers pick their tools?

What distinct problem spaces do instrument designers consider to be involved in instrument design?

How do instrument designers define a digital musical instrument?

Authors' addresses: University of West England, Bristol, UK.

These questions stem from the larger context of our work, aiming to explore and develop more expressive tools for the creation of digital musical instruments, particularly programming languages. The study was initially designed to address the motivations for a PhD thesis, exploring topics that apply to the human-computer interaction (HCI) around developers of specialised technology, how they choose the technologies they use and what challenges they face. This work currently contextualises challenges faced by these developers in the notion of problem spaces as described by Goel and Pirolli [5] and further with regards to domains [7]. An inductive research approach was chosen to help broaden perspectives beyond the immediate research interests of the researchers, to identify the most salient issues identified by digital luthiers.

The final question addresses the notion of what constitutes a digital musical instrument - a surprisingly contentious topic that is exemplified by the debate as to whether the 'I' in 'NIME' (New Interfaces for Musical Expression) should be Interface or Instrument [10]. While this question less directly contributes to the immediate research aims, the study presents an opportunity to explore contemporary concepts, terms and attitudes that are present across the field. Whilst the value in providing a specific definition for a DMI remains debatable, it provides a thought-provoking question to explore in the context of an individuals perspective.

We view a qualitative research approach as the most effective method for understanding the very subjective aspects of HCI research that relates to the selection and use of tools and programming languages relevant to the domain.

3 PARTICIPANTS

Participants were directly invited based on their contributions to a range of novel digital musical instruments or association with an organisation that produces instruments. A subset of these instruments is listed in Section 3.2 and a selection of them is illustrated in Figure 1. The selection process follows a purposeful sampling strategy, selecting participants involved in designing novel digital instruments to capture a range of perspectives that are representative of the DMI design community and the various motivations they incorporate.

A loose set of categories were defined as a basis to draw participants from, these were; Commercial, Research, Community and Artist. Commercial and Research categories describe instruments whose motivations are for either commercial 'mass-market' production or coupled to a research process respectively. Community instruments largely encompass open source projects and small teams (or individuals) independently making instruments in low volumes. The Artist category represents instrument designers who build instruments to support their artistic endeavours. Of course, there is significant overlap with these definitions, however, drawing evenly from these groups helped to provide a balanced view of the community and the intentions for instrument design they represent.

Purposeful sampling was also selected to better distribute perspectives across genders, resulting in a more representative set of perspectives of the actual digital luthiers themselves, an approach that is supported by an ever-growing body of literature [9, 12]. This invite based selection process assisted in managing a more gender-balanced selection across participants and a variety of experience levels, however, we acknowledge a lack of cultural diversity in this study which is another important facet of this issue that should be accounted for in future work [11]. This could be likely be improved with a broader call in conjunction with the selection process used here, such that selection is not limited to the networking capacity of the researchers.

Participants were approached online and invited to participate or recommend a suitable participant.The sample size was intended to be between 24 and 32 participants, drawn equally from our previously defined categories. During this study, 27 participants were interviewed. A demographic of the population is provided in Table 1 and a selection of the participant's roles can be seen below (Section 3.1).

Gender			Ehtnicity			Age	
Male	14		white	21		18 - 24	1
Female	8		Asian	1		25 - 34	12
Non Binary	1		Lantinx	1		35 - 44	6
Prefer not to say	4		Brazillian	1		45 - 54	2
			Prefer not to say	3		55 - 64	4
						Prefer not to say	2

Table 1. Demographic of participant population.

3.1 Participant Roles

- Music Technology researcher and professor.
- Digital artist/performer/composer
- Artist
- Software Engineer
- Software Engineering Manager
- CEO
- Composer
- Founder
- Researcher and Lecturer
- Assistant Professor of Music Technology
- Composer & Instrument Builder
- Audio developer
- Researcher, designer, performer
- Software Developer
- Professor
- Electronic musician
- Software Engineer
- DSP Engineer
- Professor of Media Computing
- Hardware and software engineer
- Lead designer
- Composer and interactive hardware developer
- Creative director

3.2 Instrument List

- Soft Revolvers
- Alpha Sphere
- Knurl
- Artiphon Orba
- Reactable
- The Blade Axe
- Mutable Instruments Shruthi
- Claravox
- Polaron
- The Ladies Glove
- Linnstrument
- Cv
- Abelton Push
- Roli Seaboard Grand
- OTTO
- Winterbloom Sol DrumBud
- Electronic_Khipu_
- The daïs
- EMG instruments
- Gechologic Loopsynth
- Bastl Kastle Drum

Further, information gathered on the participants includes their familiarity with HCI literature and/or the NIME community and lists of their significantly used programming languages, hardware and software tools. For further details on the data set used (including the published data-set), see Section 6.

We also capture a rudimentary metric of experience in the form of years spent in the field and the number of instruments they have designed. We emphasise that this is a metric of limited insight that can poorly characterise a persons experience, however, in the selection process, attention was given to incorporating a range of experience levels when sampling participants.

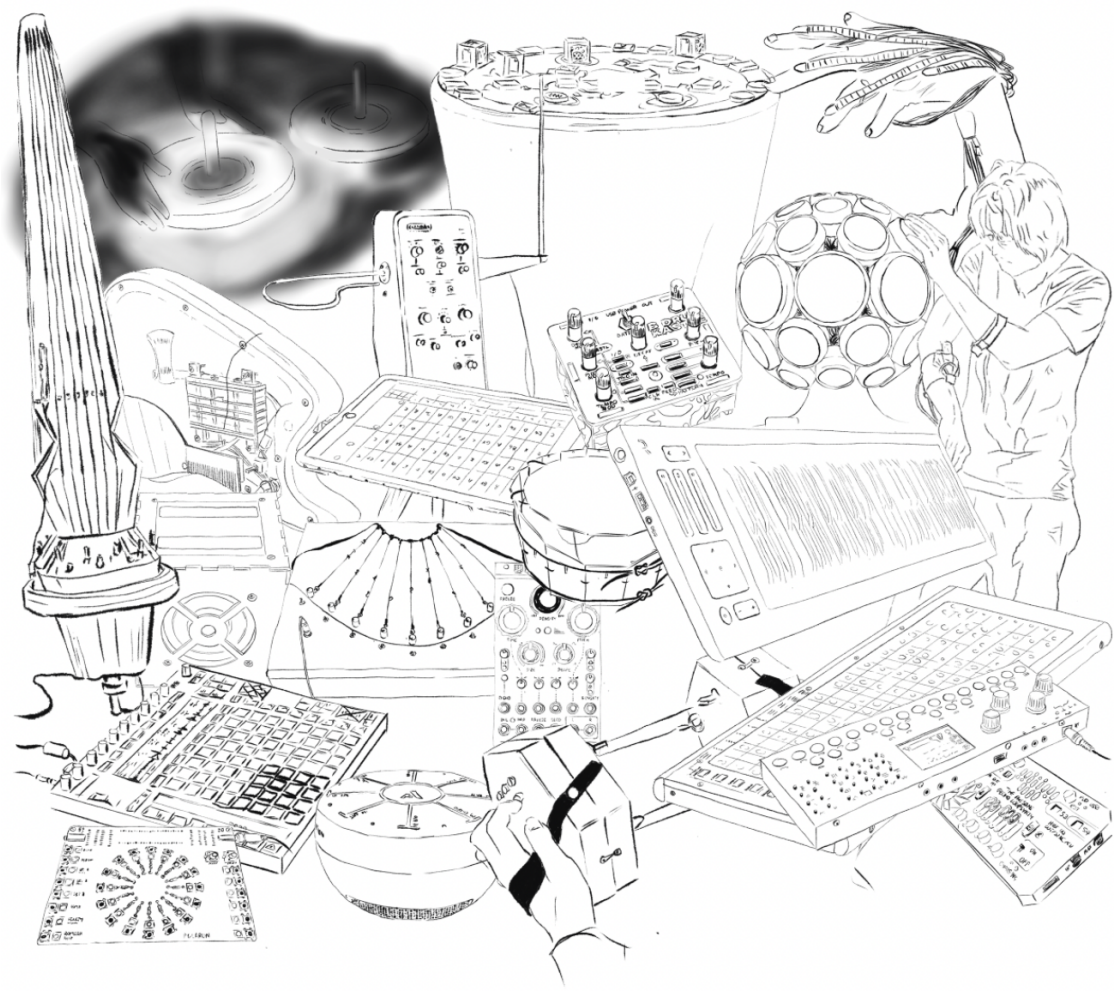


Fig. 1. A selection of instruments from the study.

4 INTERVIEWS

Interviews were conducted as standardised open-ended interviews, with 22 participants engaging with an interviewer via video call and five via email, following an internal pilot study with peers that had DMI design experience. Interviews had a duration of 20 - 60 minutes at the discretion of the participant. Interviews took place between 25th January 2021 and 1st April 2021. Participants were provided with a copy of the questions to use as a reference during the interview. All interviews were carried out by the lead author.

Standardised open-ended interviews were selected to allow extensive exploration of a small set of topics, with a direction lead by the interviewee. The interviewing style focused on allowing the participant to explore the questions with minimal interaction from the interviewer and questions were largely left open to interpretation by the participant.

If required, clarifications were made and included in transcripts. This approach was taken to allow for the data collected to be reviewed in other styles whilst still working for our reflexive analysis approach, described in Section 5

Interviews were recorded (audio only) and transcribed verbatim, then processed to ensure appropriate confidentiality and IP protection. In the case of email interviews, emails were formatted to match transcripts.

5 ANALYSIS AND METHODOLOGY

In this study analysis is based on a phenomenological perspective, exploring the personal perspective of digital luthiers. Because of its fit with an inductive approach and its capacity to create a rich and nuanced view of these interviews, reflexive thematic analysis based on Braun and Clarke's framework [2] was selected as the methodology for analysis. This study focuses on using a reflexive approach for analysis [1], exploring the perspectives of the many digital luthiers that have introduced their ideas to this study in an approach Alvesson, Hardy and Harley describe as multi-perspective practices. Notably, a varied set of perspectives and motivations have been captured in this data set and this study looks to draw out ideas from these different perspectives. With quite broad questions presented in the interviews, this analysis explores the topic largely inductively intending to explore the explicit semantic concepts introduced by participants, though consideration will also be given to latent, underlying meaning where relevant.

In line with Braun and Clarke's [3] description of reflexive thematic analysis, we recognise that as researchers we play a role in the generation of qualitative information [6]. To support this we also provide a descriptive background of the researchers taking part in this study at the end of this publication, aiming to make clear some of our biases. The wealth of introspective resources and writing from Braun and Clarke help us as computer scientists better leverage qualitative analysis [2].

5.1 Methodology

This methodology somewhat contrasts the peer validated approach typical of research based on grounded theory [3]. In this study, two coders will analyse all transcripts. Coders will regularly meet to discuss codes throughout the coding process and iteratively generate themes from the coded transcripts using the Quirkos software tool. Coders will also periodically present and discuss these themes and narratives with the rest of the research team. This approach aims to ensure we are generating codes and themes that draw on the knowledge and experience of the research team [4].

In line with Clarke and Braun's process for reflexive thematic analysis, the approach for analysis follows the following steps:

- Data familiarization period for reviewers
- Data coding using Quirkos software
- Generation of themes
- Discussion between researchers on themes, reflection and development
- Iteration around steps 2-4
- Refining and naming themes and developing of ideas around themes
- Writing paper; discussion of themes

6 DATA

Interview transcripts and corresponding data are available from the following repository:

https://github.com/muses-dmi/dmi-design-study

This repository contains the transcripts as individual documents, a comma-separated value (.csv) file of quantified data and any resources (scripts or guides) that are developed in the course of this study and may be of wider use. Transcripts are formatted as Markdown, with a metadata section at the start formatted as YAML. An example can be seen below that demonstrates the data related to each transcript. Participant numbers are randomly assigned numbers suitable for referencing transcripts.

```
---
# Note: This is a metadata section formatted in YAML.

participant: 1
role: Music Technology researcher and professor.
experience(years): 30
instrument_count: 10

instruments:
 - FMOL
 - Reactable
 - Many more

langugaes:
 - c++
 - puredata

...
```

In the body of the transcript, questions are denoted with a single '#' at the start of the line. These were read aloud to the participant (but also provided in written form for reference).

Any communication from the interviewer is denoted with a single '>' character at the start of the line. (When rendered as markdown this will render the interviewers comments as a blockquote).

7 NOTES ON CONTRIBUTORS

Nathan Renney is a PhD student at the University of West England, Bristol, UK. His thesis explores how the development of modern programming languages can be used to influence digital lutherie, the process of designing digital musical instruments. With a background as a musician, his research interests include using programming languages to be more expressive, both musically and beyond. As part of the Physical Computing Research Group at UWE, Nathan primarily works with embedded and real-time systems with an interest in the application of functional programming and type systems.

Dr Benedict R. Gaster is an Associate Professor at the University of West of England, he is the co-director of the Computer Science Research Centre, within which he also leads the Physical Computing group. His research focuses

on the design of embedded platforms for musical expression and more generally IoT. He is the co-founder of Bristol LoRaWAN a low power wide area network for Bristol city and is the technical lead for a city-wide project on pollution monitoring for communities, having developed UWE Sense (a hardware platform for cheap sensing). Along with his PhD students and in collaboration with UWE's music tech department, he is developing a new audio platform based on ARM micro-controllers using the Rust programming language to build faster and more robust sound! Previously Benedict worked at Qualcomm and AMD where he was a co-designer on the programming language OpenCL, including being the lead developer on AMD's OpenCL compiler. He has a PhD in computer science for his work on type systems for extensible records and variants.

Dr Thomas Mitchell is an Associate Professor in Creative Technologies, at UWE Bristol where he leads the Creative Technologies Laboratory. He is also an honorary scholar at the University of Bristol, resident at the Pervasive Media Studio and a Member of the Computer Science Research Centre and Bristol Robotics Laboratory. He is a co-founder of MiMU Gloves Limited, a technology start-up that he co-founded to enable musicians to perform with gestures. Connecting physicality and sound, Tom's work exploits new affordances of motion capture and extended reality technologies to inspire questions about music, art and science.

Harri Renney is a PhD student studying computer science at the University of the West of England. Specialising in the optimisation of digital audio processes for heterogeneous systems, he focuses on designing software tools for mapping processes to Graphics Processing Units, a now nearly ubiquitous hardware acceleration device that uses massively parallel processing architectures. Harri believes that with the advancing state of technology, powerful physical modelling audio processes will become increasingly viable, opening up a whole new range of possibilities in the field of digital audio.

8 CITATIONS

REFERENCES

[1] Mats Alvesson, Cynthia Hardy, and Bill Harley. 2008. Reflecting on Reflexivity: Reflexive Textual Practices in Organization and Management Theory. *Journal of Management Studies* 45, 3 (May 2008), 480–501. https://doi.org/10.1111/j.1467-6486.2007.00765.x

[2] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[3] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (Aug. 2019), 589–597. https://doi.org/10.1080/2159676X.2019.1628806

[4] Virginia Braun and Victoria Clarke. 2020. One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology* (Aug. 2020), 1–25. https://doi.org/10.1080/14780887.2020.1769238

[5] Vinod Goel and Peter Pirolli. 1992. The structure of Design Problem Spaces. *Cognitive Science* 16, 3 (July 1992), 395–429. https://doi.org/10.1207/s15516709cog1603_3

[6] Brendan Gough and Anna Madill. 2012. Subjectivity in psychological science: From problem to prospect. *Psychological Methods* 17, 3 (Sept. 2012), 374–384. https://doi.org/10.1037/a0029313

[7] Lawrence A. Hirschfeld and Susan A. Gelman (Eds.). 1994. *Mapping the mind: domain specificity in cognition and culture.* Cambridge University Press, Cambridge ; New York.

[8] Sergi Jordà. 2005. *Digital Lutherie Crafting musical computers for new musics' performance and improvisation.* Ph.D. Dissertation. Universitat Pompeu Fabra. https://doi.org/10803/575372

[9] Marlene Mathew, Jennifer Grossman, and Areti Andreopoulou. 2016. Women in Audio: Contributions and Challenges in Music Technology and Production. Audio Engineering Society. https://www.aes.org/e-lib/online/browse.cfm?elib=18477

[10] Atau Tanaka. 2010. Mapping out instruments, affordances, and mobiles. In *Proceedings of the international conference on new interfaces for musical expression.* Sydney, Australia, 88–93. https://doi.org/10.5281/zenodo.1177903 ISSN: 2220-4806.

[11] Joan C. Williams. 2014. Double jeopardy? An empirical study with implications for the debates over implicit bias and intersectionality,. Harvard Journal of Law & Gender.

[12] Anna Xambó. 2018. Who Are the Women Authors in NIME?–Improving Gender Balance in NIME Research. In *Proceedings of the International Conference on New Interfaces for Musical Expression.* Virginia Tech, Blacksburg, Virginia, USA, 174–177. https://doi.org/10.5281/zenodo.1302535

Bibliography

Aaron, Samuel and Alan F. Blackwell (2013). "From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages". In: *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP*. Volume 12. Taylor & Francis, pages 35–46.

Abtahi, Parastoo and Griffin Dietz (2020). "Learning Rust: How Experienced Programmers Leverage Resources to Learn a New Programming Language". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI EA '20. New York, NY, USA: Association for Computing Machinery, pages 1–8.

Adelson, B. and E. Soloway (1985). "The Role of Domain Expenence in Software Design". In: *IEEE Transactions on Software Engineering* SE-11.11, pages 1351–1360.

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman (2003). *Compilers: Principles, Techniques, and Tools*. [Nachdr.], international ed. Addison-Wesley Series in Computer Science. Upper Saddle River, NJ: Prentice-Hall.

Akritidis, Periklis, Cristian Cadar, Costin Raiciu, Manuel Costa, and Miguel Castro (2008). "Preventing Memory Error Exploits with WIT". In: *2008 IEEE Symposium on Security and Privacy (Sp 2008)*. IEEE, pages 263–277.

Alcántara, Jesús Muñoz, Panos Markopoulos, and Mathias Funk (2015). "Social Media as Ad Hoc Design Collaboration Tools". In: *Proceedings of the European Conference on Cognitive Ergonomics 2015*. Warsaw Poland: ACM, pages 1–8.

Alexander, Steven M. et al. (2020). "Qualitative Data Sharing and Synthesis for Sustainability Science". In: *Nature Sustainability* 3.2, pages 81–88.

Allen, Deborah E., Richard S. Donham, and Stephen A. Bernhardt (2011). "Problem-Based Learning". In: *New Directions for Teaching and Learning* 2011.128, pages 21–29.

Alvesson, Mats, Cynthia Hardy, and Bill Harley (2008). "Reflecting on Reflexivity: Reflexive Textual Practices in Organization and Management Theory". In: *Journal of Management Studies* 45.3, pages 480–501.

Annab, Rachid (2021). *CSIRAC: Our First Computer*. https://cis.unimelb.edu.au/about/csirac.

Argyris, Chris and Donald A. Schön (1974). *Theory in Practice: Increasing Professional Effectiveness*. 1st ed. San Francisco: Jossey-Bass Publishers.

Armitage, Jack and Andrew McPherson (2019). "Bricolage in a Hybrid Digital Lutherie Context: A Workshop Study". In: *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound*. Nottingham United Kingdom: ACM, pages 82–89.

Armitage, Jack and Andrew P. McPherson (2018). "Crafting Digital Musical Instruments: An Exploratory Workshop Study". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Thomas Martin Luke Dahl Douglas Bowman. Blacksburg, Virginia, USA: Virginia Tech, pages 281–286.

Armitage, Jack, Fabio Morreale, and Andrew McPherson (2017). "The Finer the Musician, the Smaller the Details: NIMEcraft under the Microscope". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Aalborg University Copenhagen, pages 393–398.

Arom, Simha (2004). *African Polyphony and Polyrhythm: Musical Structure and Methodology*. Cambridge university press.

Arora, Jatin, Sam Westrick, and Umut A. Acar (2023). "Efficient Parallel Functional Programming with Effects". In: *Proceedings of the ACM on Programming Languages* 7.PLDI, 170:1558–170:1583.

Baratta, Angelo (2006). "The Triple Constraint, a Triple Illusion". In: *PMI® Global Congress 2006*.

Bartha, Sándor, James Cheney, and Vaishak Belle (2021). "One down, 699 to Go: Or, Synthesising Compositional Desugarings". In: *Proceedings of the ACM on Programming Languages* 5.OOPSLA, pages 1–29.

Barton, Scott, Laura Getz, and Michael Kubovy (2017). "Systematic Variation in Rhythm Production as Tempo Changes". In: *Music Perception: An Interdisciplinary Journal* 34.3, pages 303–312.

Bastian, Hilda (2014). "A Stronger Post-Publication Culture Is Needed for Better Science". In: *PLoS medicine* 11.12, e1001772.

Beck, Jordan and Erik Stolterman (2016). "Examining Practical, Everyday Theory Use in Design Research". In: *She Ji: The Journal of Design, Economics, and Innovation* 2.2, pages 125–140.

Beck, Micah (2019). "On the Hourglass Model". In: *Communications of the ACM* 62.7, pages 48–57.

Ben-Ari, Mordechai (1998). "Constructivism in Computer Science Education". In: *ACM SIGCSE Bulletin* 30.1, pages 257–261.

Bennedsen, Jens and Michael E. Caspersen (2005). "An Investigation of Potential Success Factors for an Introductory Model-Driven Programming Course". In: *Proceedings of the First International Workshop on Computing Education Research*. ICER '05. New York, NY, USA: Association for Computing Machinery, pages 155–163.

Bennett, Andy (2005). "In Defence of Neo-tribes: A Response to Blackman and Hesmondhalgh". In: *Journal of Youth Studies* 8.2, pages 255–259.

Bennett, Andy and Ian Rogers (2016). "Scene 'Theory': History, Usage and Influence". In: *Popular Music Scenes and Cultural Memory*. Edited by Andy Bennett and Ian Rogers. Pop Music, Culture and Identity. London: Palgrave Macmillan UK, pages 11–35.

Bennett, Peter and Sile O'Modhrain (2008). "The BeatBearing: A Tangible Rhythm Sequencer". In: *Proc. of NordiCHI*. Volume 2008.

Berger, Michael (2010). "The GRIP MAESTRO : Idiomatic Mappings of Emotive Gestures for Control of Live Electroacoustic Music". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Sydney, Australia, pages 419–422.

Bergström, Ilias and Alan F. Blackwell (2016). "The Practices of Programming". In: *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 190–198.

Bernardo, Francisco, Chris Kiefer, and Thor Magnusson (2020). "A Signal Engine for a Live Coding Language Ecosystem". In: *Journal of the Audio Engineering Society* 68.10, pages 756–766.

Bianchi, André Jucovsky and Marcelo Queiroz (2013). "Real Time Digital Audio Processing Using Arduino". In: *Proceedings of the Sound and Music Computing Conference, Stockholm, Sweden*, pages 538–545.

Bierman, Gavin, Martín Abadi, and Mads Torgersen (2014). "Understanding Typescript". In: *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*. Springer, pages 257–281.

Blackshear, Sam, John Mitchell, Todd Nowacki, and Shaz Qadeer (2022). "The Move Borrow Checker". In: *arXiv preprint arXiv:2205.05181*. arXiv: 2205.05181.

Blackwell, Alan and Nick Collins (2005). "The Programming Language as a Musical Instrument". In: *PPIG*, page 11.

Blackwell, Alan F (2018). "A Craft Practice of Programming Language Research." In: *PPIG 2018 - the 29th Annual Workshop of the Psychology of Programming Interest Group*. London.

Bovermann, Till and Dave Griffiths (2014). "Computation as Material in Live Coding". In: *Computer Music Journal* 38.1, pages 40–53.

Bowers, John and Phil Archer (2005). "Not Hyper, Not Meta, Not Cyber but Infra-Instruments". In: *Proceedings of the 2005 Conference on New Interfaces for Musical Expression*. NIME '05. SGP: National University of Singapore, pages 5–10.

Bowman, Robert, Camille Nadal, Kellie Morrissey, Anja Thieme, and Gavin Doherty (2023). "Using Thematic Analysis in Healthcare HCI at CHI: A Scoping Review". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. New York, NY, USA: Association for Computing Machinery, pages 1–18.

Braha, Dan and Oded Maimon (2011). *A Mathematical Theory of Design: Foundations, Algorithms and Applications*. Applied Optimization 17. Dordrecht: Springer.

Braun, Virginia and Victoria Clarke (2006). "Using Thematic Analysis in Psychology". In: *Qualitative Research in Psychology* 3.2, pages 77–101.

— (2012). "Thematic Analysis." In: *APA Handbook of Research Methods in Psychology, Vol 2: Research Designs: Quantitative, Qualitative, Neuropsychological, and Biological*. Edited by Harris Cooper, Paul M. Camic, Debra L. Long, A. T. Panter, David Rindskopf, and Kenneth J. Sher. Washington: American Psychological Association, pages 57–71.

Braun, Virginia and Victoria Clarke (2019). "Reflecting on Reflexive Thematic Analysis". In: *Qualitative Research in Sport, Exercise and Health* 11.4, pages 589–597.
— (2020). "One Size Fits All? What Counts as Quality Practice in (Reflexive) Thematic Analysis?" In: *Qualitative Research in Psychology*, pages 1–25.
Brown, Dom, Chris Nash, and Tom Mitchell (2017). "A User Experience Review of Music Interaction Evaluations". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Aalborg University Copenhagen, pages 370–375.
— (2018). "Simple Mappings, Expressive Movement: A Qualitative Investigation into the End-User Mapping Design of Experienced Mid-Air Musicians". In: *Digital Creativity* 29.2-3, pages 129–148.
Bruns, Stephan B. and John P. A. Ioannidis (2016). "P-Curve and p-Hacking in Observational Research". In: *PLOS ONE* 11.2, pages 1–13.
Burnham, Trevor (2015). "Coffeescript: Accelerated Javascript Development". In: *CoffeeScript*, pages 1–124.
Calegario, Filipe et al. (2020). "Probatio 1.0: Collaborative Development of a Toolkit for Functional DMI Prototypes". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Romain Michon and Franziska Schroeder. Birmingham, UK: Birmingham City University, pages 285–290.
Cannon, Joanne and Stuart Favilla (2012). "The Investment of Play: Expression and Affordances in Digital Musical Instrument Design." In: *ICMC*.
Chasins, Sarah E., Elena L. Glassman, and Joshua Sunshine (2021). "PL and HCI: Better Together". In: *Communications of the ACM* 64.8, pages 98–106.
Chatley, Robert, Alastair Donaldson, and Alan Mycroft (2019). "The Next 7000 Programming Languages". In: *Computing and Software Science*. Edited by Bernhard Steffen and Gerhard Woeginger. Volume 10000. Cham: Springer International Publishing, pages 250–282.
Cheatle, Amy and Steven J. Jackson (2015). "Digital Entanglements: Craft, Computation and Collaboration in Fine Art Furniture Production". In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. Vancouver BC Canada: ACM, pages 958–968.
Chemero, Anthony (2003). "An Outline of a Theory of Affordances". In: *Ecological Psychology* 15.2, pages 181–195.
Cherny, Boris (2019). *Programming TypeScript: Making Your JavaScript Applications Scale*. O'Reilly Media.
Chong, Isis and Robert W. Proctor (2020). "On the Evolution of a Radical Concept: Affordances According to Gibson and Their Subsequent Use and Development". In: *Perspectives on Psychological Science* 15.1, pages 117–132.
Chowdhury, Jatin (2020). "A Comparison of Virtual Analog Modelling Techniques for Desktop and Embedded Implementations". In: *arXiv preprint arXiv:2009.02833*. arXiv: 2009.02833.
Chugh, Ravi (2016). "Prodirect Manipulation: Bidirectional Programming for the Masses". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. New York, NY, USA: Association for Computing Machinery, pages 781–784.
Clarke, Victoria and Virginia Braun (2017). "Thematic Analysis". In: *The Journal of Positive Psychology* 12.3, pages 297–298.
Cockburn, Andy, Carl Gutwin, and Alan Dix (2018). "HARK No More: On the Preregistration of CHI Experiments". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. Montreal QC, Canada: ACM Press, pages 1–12.
Collins, Nick, Alex McLean, Julian Rohrhuber, and Adrian Ward (2003). "Live Coding in Laptop Performance". In: *Organised Sound* 8.3, pages 321–330.
Combette, Guillaume and Guillaume Munch-Maccagnoni (2018). "A Resource Modality for RAII". In: *LOLA 2018: Workshop on Syntax and Semantics of Low-Level Languages*, pages 1–4.
Cook, Perry R. (2001). "Principles for Designing Computer Music Controllers". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Seattle, WA, pages 3–6.
Costabile, Maria Francesca, Antonio Piccinno, Daniela Fogli, and Andrea Marcante (2006). "Supporting Interaction and Co-Evolution of Users and Systems". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '06. New York, NY, USA: Association for Computing Machinery, pages 143–150.
Czaplicki, Evan (2012). "Elm: Concurrent FRP for Functional GUIs". PhD thesis. Harvard University.

Dabin, Matthew, Terumi Narushima, Stephen Beirne, Christian Ritz, and Kraig Grady (2016). "3D Modelling and Printing of Microtonal Flutes". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Zenodo, pages 286–290.

David Zicarelli (2017). *Code Generating Littlefoot (ADC'17)*. Audio Developers Conference 2017.

Davies, Simon P. (1993). "Models and Theories of Programming Strategy". In: *International Journal of Man-Machine Studies* 39.2, pages 237–267.

Deardorff, Ariel (2020). "Why Do Biomedical Researchers Learn to Program? An Exploratory Investigation". In: *Journal of the Medical Library Association* 108.1.

DeNora, Tia and Theodor W. Adorno (2003). *After Adorno: Rethinking Music Sociology*. Cambridge, New York: Cambridge University Press.

Dijkstra, Edsger W. (1972). "The Humble Programmer". In: *Communications of the ACM* 15.10, pages 859–866.

Dinkelaker, Tom, Michael Eichberg, and Mira Mezini (2010). "An Architecture for Composing Embedded Domain-Specific Languages". In: *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*. AOSD '10. New York, NY, USA: Association for Computing Machinery, pages 49–60.

Dot, Gem, Alejandro Martínez, and Antonio González (2015). "Analysis and Optimization of JavaScript Engines". In: *1st Workshop on High Performance Scripting Languages, in Conjunction with 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP), San Francisco (USA)*.

Duffin, R.W. (2008). *How Equal Temperament Ruined Harmony (and Why You Should Care)*. W. W. Norton.

Eglen, Stephen J. et al. (2017). "Toward Standard Practices for Sharing Computer Code and Programs in Neuroscience". In: *Nature Neuroscience* 20.6, pages 770–773.

Elliott, Conal and Paul Hudak (1997). "Functional Reactive Animation". In: *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming - ICFP '97*. Amsterdam, The Netherlands: ACM Press, pages 263–273.

Ellis, Alexander John (1885). "On the Musical Scales of Various Nations". In: *From The journal of the society of arts* 33.1688, [485]–527 p.

Fanelli, Daniele (2010). "Do Pressures to Publish Increase Scientists' Bias? An Empirical Support from US States Data". In: *PloS one* 5.4, e10271.

Feldman, Shelley and Linda Shaw (2019). "The Epistemological and Ethical Challenges of Archiving and Sharing Qualitative Data". In: *American Behavioral Scientist* 63.6, pages 699–721.

Fiebrink, Rebecca and Perry R Cook (2010). "The Wekinator: A System for Real-Time, Interactive Machine Learning in Music". In: *Proceedings of the Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)(Utrecht)*. Volume 3.

Filippidis, Ioannis, Richard M. Murray, and Gerard J. Holzmann (2016). "A Multi-Paradigm Language for Reactive Synthesis". In: *SYNT*.

Fischer, Gerhard (2001). "Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems". In: *Proceedings of the 24th IRIS Conference*. Volume 1. Department of Information Science, Bergen, pages 1–13.

— (2004). "Social Creativity: Turning Barriers into Opportunities for Collaborative Design". In: *Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices - Volume 1*. PDC 04. New York, NY, USA: Association for Computing Machinery, pages 152–161.

— (2021). "End-User Development: Empowering Stakeholders with Artificial Intelligence, Meta-Design, and Cultures of Participation". In: *End-User Development*. Edited by Daniela Fogli, Daniel Tetteroo, Barbara Rita Barricelli, Simone Borsci, Panos Markopoulos, and George A. Papadopoulos. Volume 12724. Cham: Springer International Publishing, pages 3–16.

Fischer, Gerhard, Daniela Fogli, and Antonio Piccinno (2017). "Revisiting and Broadening the Meta-Design Framework for End-User Development". In: *New Perspectives in End-User Development*. Edited by Fabio Paternò and Volker Wulf. Cham: Springer International Publishing, pages 61–97.

Fischer, Gerhard and Elisa Giaccardi (2006). "Meta-Design: A Framework for the Future of End-User Development". In: *End User Development*. Edited by Henry Lieberman, Fabio Paternò, and Volker Wulf. Volume 9. Dordrecht: Springer Netherlands, pages 427–457.

Fischer, Gerhard, Elisa Giaccardi, Hal Eden, Masanori Sugimoto, and Yunwen Ye (2005). "Beyond Binary Choices: Integrating Individual and Social Creativity". In: *International Journal of Human-Computer Studies* 63.4-5, pages 482–512.

Fischer, Gerhard and Eric Scharff (2000). "Meta-Design: Design for Designers". In: *Proceedings of the Conference on Designing Interactive Systems Processes, Practices, Methods, and Techniques - DIS '00*. New York City, New York, United States: ACM Press, pages 396–405.

Fischer, Lars and Stefan Hanenberg (2015). "An Empirical Investigation of the Effects of Type Systems and Code Completion on API Usability Using TypeScript and JavaScript in MS Visual Studio". In: *Proceedings of the 11th Symposium on Dynamic Languages*. DLS 2015. New York, NY, USA: Association for Computing Machinery, pages 154–167.

Fitch, W. Tecumseh (2013). "Rhythmic Cognition in Humans and Animals: Distinguishing Meter and Pulse Perception". In: *Frontiers in Systems Neuroscience* 7, page 68.

Fitzgerald, Brian (2006). "The Transformation of Open Source Software". In: *MIS quarterly*, pages 587–598.

Fleissner, Sebastian and Elisa Baniassad (2009). "The Culture of Programming Languages". In: *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*. OOPSLA '09. New York, NY, USA: Association for Computing Machinery, pages 1055–1056.

Floyd, Robert W. (1979). "The Paradigms of Programming". In: *Communications of the ACM* 22.8, pages 455–460.

Frankjær, Raune and Peter Dalsgaard (2018). "Understanding Craft-Based Inquiry in HCI". In: *Proceedings of the 2018 Designing Interactive Systems Conference*. Hong Kong China: ACM, pages 473–484.

Frich, Jonas, Lindsay MacDonald Vermeulen, Christian Remy, Michael Mose Biskjaer, and Peter Dalsgaard (2019). "Mapping the Landscape of Creativity Support Tools in HCI". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, pages 1–18.

Fukuda, Takuto, Eduardo Meneses, Travis West, and Marcelo M. Wanderley (2021). "The T-Stick Music Creation Project: An Approach to Building a Creative Community around a DMI". In: *NIME 2021*.

Fyans, A. Cavan, Adnan Marquez-Borbon, Paul Stapleton, and Michael Gurevich (2012). "Ecological Considerations for Participatory Design of DMIs". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Ann Arbor, Michigan: University of Michigan.

Gagniuc, Paul (2023). "Paradigms and Concepts". In: *An Introduction to Programming Languages: Simultaneous Learning in Multiple Coding Environments*. Cham: Springer International Publishing, pages 41–59.

Gamma, Erich, editor (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Reading, Mass: Addison-Wesley.

Gaster, Benedict and Ryan Challinor (2021). "Bespoke Anywhere". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Shanghai, China.

Gaster, Benedict and Max Cole (2020). "Audio Anywhere with Faust". In: *Proceedings of the 2nd International Faust Conference*.

Gaster, Benedict, Renney Nathan, and Parraman Carinna (2019). "Fun with Interfaces (SVG Interfaces for Musical Expression)". In: *7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*.

Gaster, Benedict R., Nathan Renney, and Tom Mitchell (2018). "Outside the Block Syndicate: Translating Faust's Algebra of Blocks to the Arrows Framework". In: *Proceedings of the 1st International Faust Conference (IFC-18)*.

Gauthier, Robert P. and James R. Wallace (2022). "The Computational Thematic Analysis Toolkit". In: *Proceedings of the ACM on Human-Computer Interaction* 6.GROUP, 25:1–25:15.

Gaver, William W. (1991). "Technology Affordances". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Reaching through Technology - CHI '91*. New Orleans, Louisiana, United States: ACM Press, pages 79–84.

Gibson, James J. (2014). *The Ecological Approach to Visual Perception: Classic Edition*. 1st edition. Psychology Press.

Gill, Andy (2014). "Domain-Specific Languages and Code Synthesis Using Haskell". In: *Communications of the ACM* 57.6, pages 42–49.

Glass, R. L., I. Vessey, and V. Ramesh (2002). "Research in Software Engineering: An Analysis of the Literature". In: *Information and Software Technology* 44.8, pages 491–506.

Glass, R.L. (1994). "The Software-Research Crisis". In: *IEEE Software* 11.6, pages 42–47.

Glass, Robert L., V. Ramesh, and Iris Vessey (2004). "An Analysis of Research in Computing Disciplines". In: *Communications of the ACM* 47.6, pages 89–94.

Goel, Vinod and Peter Pirolli (1992). "The Structure of Design Problem Spaces". In: *Cognitive science* 16.3, pages 395–429.

Goodman, Elizabeth, Erik Stolterman, and Ron Wakkary (2011). "Understanding Interaction Design Practices". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Vancouver BC Canada: ACM, pages 1061–1070.

Gough, Brendan and Anna Madill (2012). "Subjectivity in Psychological Science: From Problem to Prospect." In: *Psychological Methods* 17.3, pages 374–384.

Gouyon, Fabien (2007). "Microtiming in "Samba de Roda" Preliminary Experiments with Polyphonic Audio". In: *Brazilian Symposium on Computer Music* January 2007.

Graafsma, Irene L., Serje Robidoux, Lyndsey Nickels, Matthew Roberts, Vince Polito, Judy D. Zhu, and Eva Marinus (2023). "The Cognition of Programming: Logical Reasoning, Algebra and Vocabulary Skills Predict Programming Performance Following an Introductory Computing Course". In: *Journal of Cognitive Psychology* 35.3, pages 364–381.

Green, T R G (1989). "Cognitive Dimensions of Notations". In: *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. New York, NY, USA: Cambridge University Press, pages 443–460.

Grgic, H, Branko Mihaljević, and Aleksander Radovan (2018). "Comparison of Garbage Collectors in Java Programming Language". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, pages 1539–1544.

Gurevich, Michael (2016). "Diversity in NIME Research Practices". In: *Leonardo* 49.1, pages 80–81.

Gurevich, Michael and Jeffrey Treviño (2017). "2007: Expression and Its Discontents: Toward an Ecology of Musical Creation". In: *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*. Edited by Alexander Refsum Jensenius and Michael J. Lyons. Cham: Springer International Publishing, pages 299–315.

Halabi, Ammar and Basile Zimmermann (2019). "Waves and Forms: Constructing the Cultural in Design". In: *AI & SOCIETY* 34.3, pages 403–417.

Hanenberg, Stefan, Sebastian Kleinschmager, Romain Robbes, Éric Tanter, and Andreas Stefik (2014). "An Empirical Study on the Impact of Static Typing on Software Maintainability". In: *Empirical Software Engineering* 19.5, pages 1335–1382.

Harlin, Ismail Rizky, Hironori Washizaki, and Yoshiaki Fukazawa (2017). "Impact of Using a Static-Type System in Computer Programming". In: *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 116–119.

Hatchuel, Armand and Benoit Weil (2003). "A New Approach of Innovative Design: An Introduction to CK Theory." In: *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*.

Hayward, Robin (2015). "The Hayward Tuning Vine: An Interface for Just Intonation". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Edgar Berdahl and Jesse Allison. Baton Rouge, Louisiana, USA: Louisiana State University, pages 209–214.

Helbling, Caleb and Samuel Z. Guyer (2016). "Juniper: A Functional Reactive Programming Language for the Arduino". In: *Proceedings of the 4th International Workshop on Functional Art, Music, Modelling, and Design - FARM 2016*. Nara, Japan: ACM Press, pages 8–16.

Hempel, Brian, Justin Lubin, and Ravi Chugh (2019). "Sketch-n-Sketch: Output-Directed Programming for SVG". In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. UIST '19. New York, NY, USA: Association for Computing Machinery, pages 281–292.

Hesmondhalgh, David (2005). "Subcultures, Scenes or Tribes? None of the Above". In: *Journal of Youth Studies* 8.1, pages 21–40.

Hinrichsen, Haye (2016). "Revising the Musical Equal Temperament". In: *Revista Brasileira de Ensino de FÃsica* 38.

Hirschfeld, Lawrence A. and Susan A. Gelman, editors (1994). *Mapping the Mind: Domain Specificity in Cognition and Culture*. Cambridge ; New York: Cambridge University Press.

Hoc, Jean-Michel and Anh Nguyen-Xuan (1990). "Language Semantics, Mental Models and Analogy". In: *Psychology of Programming*. Elsevier, pages 139–156.

Hoc, J.M. (2014). *Psychology of Programming*. Computers and People Series Visual Studies. Elsevier Science.

Hoda, Rashina, James Noble, and Stuart Marshall (2011). "Grounded Theory for Geeks". In: *Proceedings of the 18th Conference on Pattern Languages of Programs*. PLoP '11. New York, NY, USA: Association for Computing Machinery, pages 1–17.

Hudak, Paul (1986). “A Semantic Model of Reference Counting and Its Abstraction (Detailed Summary)”. In: *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*, pages 351–363.
— (1996). “Building Domain-Specific Embedded Languages”. In: *ACM Computing Surveys* 28.4es, 196–es.
— (1997). “Domain-Specific Languages”. In: *Handbook of programming languages* 3.39-60, page 21.
— (1998). “Modular Domain Specific Languages and Tools”. In: *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pages 134–142.
Hudak, Paul, Tom Makucevich, Syam Gadde, and Bo Whong (1996). “Haskore Music Notation – An Algebra of Music –”. In: *Journal of Functional Programming* 6.3, pages 465–484.
Hughes, John (1995). “The Design of a Pretty-Printing Library”. In: *Advanced Functional Programming: First International Spring School on Advanced Functional Programming Techniques Båstad, Sweden, May 24–30, 1995 Tutorial Text 1*. Springer, pages 53–96.
Hunt, Andy, Marcelo M. Wanderley, and Matthew Paradis (2002). “The Importance of Parameter Mapping in Electronic Instrument Design”. In: *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*. NIME ’02. SGP: National University of Singapore, pages 1–6.
Imai, Saki (2022). “Is GitHub Copilot a Substitute for Human Pair-Programming? An Empirical Study”. In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. ICSE ’22. New York, NY, USA: Association for Computing Machinery, pages 319–321.
Inie, Nanna and Peter Dalsgaard (2017). “How Interaction Designers Use Tools to Capture, Manage, and Collaborate on Ideas”. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’17. New York, NY, USA: Association for Computing Machinery, pages 2668–2675.
Jack, Robert, Jacob Harrison, and Andrew McPherson (2020). “Digital Musical Instruments as Research Products”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Romain Michon and Franziska Schroeder. Birmingham, UK: Birmingham City University, pages 446–451.
Jack, Robert, Adib Mehrabi, Tony Stockman, and Andrew McPherson (2018). “Action-Sound Latency and the Perceived Quality of Digital Musical Instruments: Comparing Professional Percussionists and Amateur Musicians”. In: *Music Perception: An Interdisciplinary Journal* 36.
Jacobs, Jennifer, David Mellis, Amit Zoran, Cesar Torres, Joel Brandt, and Theresa Jean Tanenbaum (2016). “Digital Craftsmanship: HCI Takes on Technology as an Expressive Medium”. In: *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems*. Brisbane QLD Australia: ACM, pages 57–60.
Japikse, Philip, Kevin Grossnicklaus, and Ben Dewey (2017). “Introduction to TypeScript”. In: *Building Web Applications with Visual Studio 2017: Using .NET Core and Modern JavaScript Frameworks*. Edited by Philip Japikse, Kevin Grossnicklaus, and Ben Dewey. Berkeley, CA: Apress, pages 241–280.
Jin, Wuxia, Dinghong Zhong, Zifan Ding, Ming Fan, and Ting Liu (2021). “Where to Start: Studying Type Annotation Practices in Python”. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 529–541.
John Varney (2014). *A Different Way to Visualize Rhythm*.
Jordà, Sergi (2004a). “Digital Instruments and Players Part II: Diversity, Freedom and Control”. In: *International Computer Music Conference*.
— (2004b). “Instruments and Players: Some Thoughts on Digital Lutherie”. In: *Journal of New Music Research* 33.3, pages 321–341.
— (2005). “Digital Lutherie Crafting Musical Computers for New Musics’ Performance and Improvisation”. PhD thesis. Universitat Pompeu Fabra.
Jules Storer (2016). *Only an Idiot Would Write a New C-like Language in 2016, Jules Storer*. Audio Developers Conference 2016.
Jung, Young-Wook, Youn-kyung Lim, and Myung-suk Kim (2017). “Possibilities and Limitations of Online Document Tools for Design Collaboration: The Case of Google Docs”. In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. Portland Oregon USA: ACM, pages 1096–1108.
Kaijanaho, Antti-Juhani (2015). “Evidence-Based Programming Language Design : A Philosophical and Methodological Exploration”. In: *Jyväskylä studies in computing* 222.

Kaltenbrunner, Martin, Sergi Jorda, Gunter Geiger, and Marcos Alonso (2006). "The reacTable*: A Collaborative Musical Instrument". In: *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06)*, pages 406–411.

Kantaros, Antreas and Olaf Diegel (2018). "3D Printing Technology in Musical Instrument Research: Reviewing the Potential". In: *Rapid prototyping journal* 24.9, pages 1511–1523.

Ko, Amy J. et al. (2011). "The State of the Art in End-User Software Engineering". In: *ACM Computing Surveys* 43.3, 21:1–21:44.

Koch, Janin, Jennifer Pearson, Andrés Lucero, Miriam Sturdee, Wendy E. Mackay, Makayla Lewis, and Simon Robinson (2020). "Where Art Meets Technology: Integrating Tangible and Intelligent Tools in Creative Processes". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. Honolulu HI USA: ACM, pages 1–7.

Koeppe, Ian (2018). "An Investigation into the Imposed Cognitive Load of Static & Dynamic Type Systems on Programmers". PhD thesis. University of Nebraska at Omaha.

Kosar, Tomaž, Sudev Bohra, and Marjan Mernik (2016). "Domain-Specific Languages: A Systematic Mapping Study". In: *Information and Software Technology* 71, pages 77–91.

Król, Karol and Dariusz Zdonek (2019). "Peculiarity of the Bit Rot and Link Rot Phenomena". In: *Global Knowledge, Memory and Communication*.

Kuznetsov, Stacey and Eric Paulos (2010). "Rise of the Expert Amateur: DIY Projects, Communities, and Cultures". In: *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*. NordiCHI '10. New York, NY, USA: Association for Computing Machinery, pages 295–304.

L. Haven, Tamarinde and Dr. Leonie Van Grootel (2019). "Preregistering Qualitative Research". In: *Accountability in Research* 26.3, pages 229–244.

Laguna, Christopher Patrick and Rebecca Fiebrink (2014). "Improving Data-Driven Design and Exploration of Digital Musical Instruments". In: *CHI '14 Extended Abstracts on Human Factors in Computing Systems*. Toronto Ontario Canada: ACM, pages 2575–2580.

Landin, P. J. (1966). "The next 700 Programming Languages". In: *Communications of the ACM* 9.3, pages 157–166.

Lauer, Michael S, Harlan M Krumholz, and Eric J Topol (2015). "Time for a Prepublication Culture in Clinical Research?" In: *The Lancet* 386.10012, pages 2447–2449.

Lepri, Giacomo and Andrew McPherson (2019). "Making Up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice". In: *Proceedings of the 2019 on Designing Interactive Systems Conference*. San Diego CA USA: ACM, pages 113–126.

Lertwittayatrai, Nuttapon, Raula Gaikovina Kula, Saya Onoue, Hideaki Hata, Arnon Rungsawang, Pattara Leelaprute, and Kenichi Matsumoto (2017). "Extracting Insights from the Topology of the JavaScript Package Ecosystem". In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 298–307.

Lessig, L. (2009). *Remix: Making Art and Commerce Thrive in the Hybrid Economy*. Bloomsbury Publishing.

Levy, Steven (1984). *Hackers: Heroes of the Computer Revolution*. 1st ed. Garden City, N.Y: Anchor Press/Doubleday.

Lindell, Rikard (2014). "Crafting Interaction: The Epistemology of Modern Programming". In: *Personal and Ubiquitous Computing* 18.3, pages 613–624.

Lohman, Kirsty (2017). "Theories of Punk and Subculture". In: *The Connected Lives of Dutch Punks: Contesting Subcultural Boundaries*. Edited by Kirsty Lohman. Palgrave Studies in the History of Subcultures and Popular Music. Cham: Springer International Publishing, pages 23–59.

Loy, Gareth (1985). "Musicians Make a Standard: The MIDI Phenomenon". In: *Computer Music Journal* 9.4, pages 8–26. JSTOR: 3679619.

Lubin, Justin and Sarah E. Chasins (2021). "How Statically-Typed Functional Programmers Write Code". In: *Proceedings of the ACM on Programming Languages* 5.OOPSLA, 155:1–155:30.

Madgwick, Sebastian and Thomas J. Mitchell (2013). "X-OSC: A Versatile Wireless I/O Device for Creative/Music Applications". In: *SMC Sound and Music Computing Conference 2013*.

Madison, Guy (2001). "Variability in Isochronous Tapping: Higher Order Dependencies as a Function of Intertap Interval". In: *Journal of Experimental Psychology: Human Perception and Performance* 27.2, pages 411–422. PMID: 11318056.

Magnusson, Thor (2006). "Affordances and Constraints in Screen-Based Musical Instruments". In: *Proceedings of the 4th Nordic Conference on Human-computer Interaction Changing Roles - NordiCHI '06*. Oslo, Norway: ACM Press, pages 441–444.

— (2009). "Of Epistemic Tools: Musical Instruments as Cognitive Extensions". In: *Organised Sound* 14.2, pages 168–176.

Magnusson, Thor (2010a). “Designing Constraints: Composing and Performing with Digital Musical Systems”. In: *Computer Music Journal* 34.4, pages 62–73.
— (2010b). *Ixi Lang: A Constraint System for Live Coding*. Edited by Judith Funke, Stefan Riekeles, and Andreas Broeckmann. Berlin.
— (2019). *Sonic Writing: Technologies of Material, Symbolic and Signal Inscriptions*. New York, NY: Bloomsbury Academic.
Magnusson, Thor and Alex McLean (2018). “Chapter 14: Performing with Patterns of Time”. In: *Oxford University Press*, pages 1–17.
Magnusson, Thor and Enrike H. Mendieta (2007). “The Acoustic, the Digital and the Body : A Survey on Musical Instruments”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. New York City, NY, United States, pages 94–99.
Maguire, Sandy (2018). *Thinking with Types*. Leanpub.
Mariano, Benjamin, Yanju Chen, Yu Feng, Greg Durrett, and Işil Dillig (2022). “Automated Transpilation of Imperative to Functional Code Using Neural-Guided Program Synthesis”. In: *Proceedings of the ACM on Programming Languages* 6.OOPSLA1, 71:1–71:27.
Marlow, Simon (2010). *Haskell 2010 Language Report*.
Marquez-Borbon, Adnan and Juan Pablo Martinez-Avila (2018). “The Problem of DMI Adoption and Longevity: Envisioning a NIME Performance Pedagogy”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Thomas Martin Luke Dahl Douglas Bowman. Blacksburg, Virginia, USA: Virginia Tech, pages 190–195.
Marquez-Borbon, Adnan and Paul Stapleton (2015). “Fourteen Years of NIME: The Value and Meaning of ‘Community’ in Interactive Music Research”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Edgar Berdahl and Jesse Allison. Baton Rouge, Louisiana, USA: Louisiana State University, pages 307–312.
Marshall, Mark T. and Marcelo M. Wanderley (2006). “Vibrotactile Feedback in Digital Musical Instruments”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Paris, France, pages 226–229.
Matavire, Rangarirai and Irwin Brown (2008). “Investigating the Use of ”Grounded Theory” in Information Systems Research”. In: *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*. SAICSIT ’08. New York, NY, USA: Association for Computing Machinery, pages 139–147.
Mathew, Marlene, Jennifer Grossman, and Areti Andreopoulou (2016). “Women in Audio: Contributions and Challenges in Music Technology and Production”. In: *Audio Engineering Society Convention 141*. Audio Engineering Society.
Mathews, Max V. and Curtis Abbott (1980). “The Sequential Drum”. In: *Computer Music Journal* 4.4, pages 45–59. JSTOR: 3679465.
Mayer, Clemens, Stefan Hanenberg, R. Robbes, É Tanter, and A. Stefik (2012). “Static Type Systems (Sometimes) Have a Positive Impact on the Usability of Undocumented Software : An Empirical Evaluation”. In.
McArthur, J.A. (2009). “Digital Subculture: A Geek Meaning of Style”. In: *Journal of Communication Inquiry* 33.1, pages 58–70.
McCartney, James (2002). “Rethinking the Computer Music Language: SuperCollider”. In: *Computer Music Journal* 26.4, pages 61–68.
McCauley, Renée, Scott Grissom, Sue Fitzgerald, and Laurie Murphy (2015). “Teaching and Learning Recursive Programming: A Review of the Research Literature”. In: *Computer Science Education* 25.1, pages 37–66.
McLean, Alex (2011). “Artist-Programmers and Programming Languages for the Arts”. PhD thesis. Goldsmiths, University of London.
— (2014). “Making Programming Languages to Dance to: Live Coding with Tidal”. In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design - FARM ’14*. Gothenburg, Sweden: ACM Press, pages 63–70.
McPherson, Andrew (2017). “Bela: An Embedded Platform for Low-Latency Feedback Control of Sound”. In: *The Journal of the Acoustical Society of America* 141.5, pages 3618–3618.
McPherson, Andrew and Youngmoo E. Kim (2012). “The Problem of the Second Performer: Building a Community Around an Augmented Piano”. In: *Computer Music Journal* 36.4, pages 10–27.
McPherson, Andrew, Robert Jack, and Giulio Moro (2016). “Action-Sound Latency: Are Our Tools Fast Enough?” In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Volume 16. Brisbane, Australia: Queensland Conservatorium Griffith University, pages 20–25.

McPherson, Andrew and Giacomo Lepri (2020). "Beholden to Our Tools: Negotiating with Technology While Sketching Digital Instruments". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Romain Michon and Franziska Schroeder. Birmingham, UK: Birmingham City University, pages 434–439.

McPherson, Andrew, Fabio Morreale, and Jacob Harrison (2019). "Musical Instruments for Novices: Comparing NIME, HCI and Crowdfunding Approaches". In: *New Directions in Music and Human-Computer Interaction*. Edited by Simon Holland, Tom Mudd, Katie Wilkie-McKenna, Andrew McPherson, and Marcelo M. Wanderley. Cham: Springer International Publishing, pages 179–212.

McPherson, Andrew and Koray Tahıroğlu (2020). "Idiomatic Patterns and Aesthetic Influence in Computer Music Languages". In: *Organised Sound* 25.1, pages 53–63.

McPherson, Andrew P., Alan Chamberlain, Adrian Hazzard, Sean McGrath, and Steve Benford (2016). "Designing for Exploratory Play with a Hackable Digital Musical Instrument". In: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. Brisbane QLD Australia: ACM, pages 1233–1245.

McPherson, Andrew P. and Youngmoo E. Kim (2013). "Piano Technique as a Case Study in Expressive Gestural Interaction". In: *Music and Human-Computer Interaction*. Edited by Simon Holland, Katie Wilkie, Paul Mulholland, and Allan Seago. London: Springer London, pages 123–138.

Meyer, Miriah and Jason Dykes (2020). "Criteria for Rigor in Visualization Design Study". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pages 87–97.

Michon, Romain, Catinca Dumitrascu, Sandrine Chudet, Yann Orlarey, Stéphane Letz, and Dominique Fober (2021). "Amstramgrame: Making Scientific Concepts More Tangible Through Music Technology at School". In: *NIME 2021*.

Michon, Romain, Yann Orlarey, Stéphane Letz, and Dominique Fober (2019). "Real Time Audio Digital Signal Processing with Faust and the Teensy". In: *Sound and Music Computing Conference (SMC-19)*.

Michon, Romain, Yann Orlarey, Stéphane Letz, Dominique Fober, and Dirk Roosenburg (2020a). "Embedded Real-Time Audio Signal Processing with Faust". In: *International Faust Conference (IFC-20)*.

Michon, Romain, Daniel Overholt, Stephane Letz, Yann Orlarey, Dominique Fober, and Catinca Dumitrascu (2020b). "A Faust Architecture for the ESP32 Microcontroller". In: *Sound and Music Computing Conference (SMC-20)*.

Mikkonen, Tommi and Antero Taivalsaari (2007). *Using JavaScript as a Real Programming Language*. Technical Report. USA: Sun Microsystems, Inc.

Milanesi, Carlo (2022). "Object-Oriented Programming". In: *Beginning Rust*. Berkeley, CA: Apress, pages 309–335.

Milne, Andrew, William Sethares, and James Plamondon (2007). "Isomorphic Controllers and Dynamic Tuning: Invariant Fingering over a Tuning Continuum". In: *Computer Music Journal* 31.4, pages 15–32.

Miranda, André and João Pimentel (2018). "On the Use of Package Managers by the C++ Open-Source Community". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. SAC '18. New York, NY, USA: Association for Computing Machinery, pages 1483–1491.

Miranda, Eduardo Reck and Marcelo M. Wanderley (2006). *New Digital Musical Instruments: Control and Interaction beyond the Keyboard*. The Computer Music and Digital Audio Series v. 21. Middleton, Wis: A-R Editions.

Moro, Giulio, Astrid Bin, Robert H Jack, Christian Heinrichs, Andrew P McPherson, et al. (2016). "Making High-Performance Embedded Instruments with Bela and Pure Data". In: *International Conference on Live Interfaces*. University of Sussex.

Moro, Giulio and Andrew P. McPherson (2021). "Performer Experience on a Continuous Keyboard Instrument". In: *Computer Music Journal* 44.2-3, pages 69–91.

Morreale, Fabio, S. M. Astrid Bin, Andrew McPherson, Paul Stapleton, and Marcelo Wanderley (2020). "A NIME of the Times: Developing an Outward-Looking Political Agenda for This Community". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Romain Michon and Franziska Schroeder. Birmingham, UK: Birmingham City University, pages 160–165.

Morreale, Fabio and Andrew McPherson (2017). "Design for Longevity: Ongoing Use of Instruments from NIME 2010-14". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Aalborg University Copenhagen, pages 192–197.

Morreale, Fabio, Andrew P. McPherson, and Marcelo Wanderley (2018). "NIME Identity from the Performer's Perspective". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Thomas Martin Luke Dahl Douglas Bowman. Blacksburg, Virginia, USA: Virginia Tech, pages 168–173.

Morreale, Fabio, Giulio Moro, Alan Chamberlain, Steve Benford, and Andrew P. McPherson (2017). "Building a Maker Community Around an Open Hardware Platform". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. New York, NY, USA: Association for Computing Machinery, pages 6948–6959.

Mulder, Axel (1994). "Virtual Musical Instruments: Accessing the Sound Synthesis Universe as a Performer". In: *Proceedings of the First Brazilian Symposium on Computer Music*, pages 243–250.

Myers, Brad and Andrew Ko (2009). "The Past, Present and Future of Programming in HCI". In: *Human-Computer Interaction Consortium (HCIC'09)*.

Nash, Chris and Alan F Blackwell (2011). "Tracking Virtuosity and Flow in Computer Music". In: *Proceedings of International Computer Music Conference*.

Newell, Allen (1993). "Reasoning, Problem Solving, and Decision Processes: The Problem Space as a Fundamental Category". In: *The Soar Papers (Vol. 1) Research on Integrated Intelligence*, pages 55–80.

Norman, Donald A. (1999). "Affordance, Conventions, and Design". In: *Interactions* 6.3, pages 38–43.

Nosek, B. A. et al. (2015). "Promoting an Open Research Culture". In: *Science* 348.6242, pages 1422–1425.

Okon, Sebastian and Stefan Hanenberg (2016). "Can We Enforce a Benefit for Dynamically Typed Languages in Comparison to Statically Typed Ones? A Controlled Experiment". In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10.

O'Modhrain, Sile (2011). "A Framework for the Evaluation of Digital Musical Instruments". In: *Computer Music Journal* 35.1, pages 28–42.

Orlarey, Yann, Dominique Fober, and Stéphane Letz (2009). "FAUST : An Efficient Functional Approach to DSP Programming". In: *New Computational Paradigms For Computer Music*, pages 65–96.

Page, Rex (2001). "Functional Programming, and Where You Can Put It". In: *ACM SIGPLAN Notices* 36.9, pages 19–24.

Parr, Terence and Kathleen Fisher (2011). "LL(*): The Foundation of the ANTLR Parser Generator". In: *ACM SIGPLAN Notices* 46.6, pages 425–436.

Parr, Terence J. and Russell W. Quong (1995). "ANTLR: A Predicated-LL (k) Parser Generator". In: *Software: Practice and Experience* 25.7, pages 789–810.

Patterson, Daniel and Amal Ahmed (2017). "Linking Types for Multi-Language Software: Have Your Cake and Eat It Too". In: *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Edited by Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi. Volume 71. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 12:1–12:15.

Payne, Stephen J., Helen R. Squibb, and Andrew Howes (1990). "The Nature of Device Models: The Yoked State Space Hypothesis and Some Experiments with Text Editors". In: *Human–Computer Interaction* 5.4, pages 415–444. eprint: https://www.tandfonline.com/doi/pdf/10.1207/s15327051hci0504_3.

Pearce, Marcus and Geraint A Wiggins (2002). "Aspects of a Cognitive Theory of Creativity in Musical Composition". In: *Proceedings of the ECAI02 Workshop on Creative Systems*.

Pham, Michel Tuan and Travis Tae Oh (2021). "Preregistration Is Neither Sufficient nor Necessary for Good Science". In: *Journal of Consumer Psychology* 31.1, pages 163–176.

Polak, Rainer (2010). "Rhythmic Feel as Meter: Non-isochronous Beat Subdivision in Jembe Music from Mali". In: *Society for Music Theory* 16.4, pages 1–26.

Posch, Irene and Geraldine Fitzpatrick (2021). "The Matter of Tools: Designing, Using and Reflecting on New Tools for Emerging eTextile Craft Practices". In: *ACM Transactions on Computer-Human Interaction* 28.1, pages 1–38.

Puckette, Miller (1996). "Pure Data: Another Integrated Computer Music Environment". In: *Proceedings of the second intercollege computer music concerts*, pages 37–41.

— (1997). "Pure Data". In: *Proceedings of the 1996 International Computer Music Conference*. San Francisco: International Computer Music Association, pages 269–272.

— (2002). "Max at Seventeen". In: *Computer Music Journal* 26.4, pages 31–43. JSTOR: 3681767.

Quille, Keith and Susan Bergin (2018). "Programming: Predicting Student Success Early in CS1. a Re-Validation and Replication Study". In: *Proceedings of the 23rd Annual ACM Conference*

on Innovation and Technology in Computer Science Education. ITiCSE 2018. New York, NY, USA: Association for Computing Machinery, pages 15–20.

Ramirez V, Gabriel M, Yenny A Méndez, Antoni Granollers, Andrés F Millán, Claudio C Gonzalez, and Fernando Moreira (2021). "State of the Art of Human-Computer Interaction (HCI) Master's Programs 2020". In: *World Conference on Information Systems and Technologies*. Springer, pages 405–414.

Rastogi, Aseem, Nikhil Swamy, Cédric Fournet, Gavin Bierman, and Panagiotis Vekris (2015). "Safe & Efficient Gradual Typing for TypeScript". In: *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 167–180.

Renney, Nathan, Benedict Gaster, and Tom Mitchell (2018). "Return to Temperament (In Digital Systems)". In: *Audio Mostly*. Edited by Hunt Samuel.

Renney, Nathan, Benedict Gaster, Tom Mitchell, and Harri Renney (2022). "Studying How Digital Luthiers Choose Their Tools". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI '22. New York, NY, USA: Association for Computing Machinery, pages 1–18.

Renney, Nathan and Benedict R. Gaster (2019). "Digital Expression and Representation of Rhythm". In: *Audio Mostly*. Association for Computing Machinery (ACM).

Renney, Nathan, Benedict R Gaster, Thomas J Mitchell, and Harri Renney (2021). *Data and Methodology : Exploring How Digital Luthiers Choose Their Tools*. Version 1.0.0.

Repp, Bruno H (2003). "Rate Limits in Sensorimotor Synchronization with Auditory and Visual Sequences". In: *Journal of Motor Behavior* 35.4, page 16.

Riley, Terry (1983). *Songs for the Ten Voices of the Two Prophets*. Germany.

Rist, Robert S. (1991). "Knowledge Creation and Retrieval in Program Design: A Comparison of Novice and Intermediate Student Programmers". In: *Human-Computer Interaction* 6.1, pages 1–46.

Roads, C. and Max Mathews (1980). "Interview with Max Mathews". In: *Computer Music Journal* 4.4, page 15. JSTOR: 3679463.

Robins, Anthony, Janet Rountree, and Nathan Rountree (2003). "Learning and Teaching Programming: A Review and Discussion". In: *Computer Science Education* 13.2, pages 137–172.

Robson, Samuel G. et al. (2021). "Promoting Open Science: A Holistic Approach to Changing Behaviour". In: *Collabra: Psychology* 7.1, page 30137.

Rossmy, Beat and Alexander Wiethoff (2019). "The Modular Backward Evolution — Why to Use Outdated Technologies". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Marcelo Queiroz and Anna Xambó Sedó. Porto Alegre, Brazil: UFRGS, pages 343–348.

Sakulniwat, Tattiya, Raula Gaikovina Kula, Chaiyong Ragkhitwetsagul, Morakot Choetkiertikul, Thanwadee Sunetnanta, Dong Wang, Takashi Ishio, and Kenichi Matsumoto (2019). "Visualizing the Usage of Pythonic Idioms Over Time: A Case Study of the with Open Idiom". In: *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 43–435.

Sayago, Sergio (2023). *Cultures in Human-Computer Interaction*. Synthesis Lectures on Human-Centered Informatics. Cham: Springer International Publishing.

Schaeffer, Pierre (2017). *Treatise on Musical Objects : An Essay across Disciplines*. Volume 20. Univ of California Press, page 569.

Selic, Bran (2008). "Personal Reflections on Automation, Programming Culture, and Model-Based Software Engineering". In: *Automated Software Engineering* 15.3, pages 379–391.

Silva, Eduardo S, Jader Anderson O de Abreu, Janiel Henrique Pinheiro de Almeida, Veronica Teichrieb, and Geber Lisboa Ramalho (2013). "A Preliminary Evaluation of the Leap Motion Sensor as Controller of New Digital Musical Instruments". In: *Recife, Brasil*, pages 59–70.

Silver, Mike (2006). "Towards a Programming Culture in the Design Arts". In: *Architectural Design* 76.4, pages 5–11.

Sivaraman, Aishwarya, Rui Abreu, Andrew Scott, Tobi Akomolede, and Satish Chandra (2022). "Mining Idioms in the Wild". In: *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '22. New York, NY, USA: Association for Computing Machinery, pages 187–196.

Small, Christopher (1998). *Musicking: The Meanings of Performing and Listening*. Music/Culture. Hanover: University Press of New England.

Snape, Joe and Georgina Born (2022). "Max, Music Software and the Mutual Mediation of Aesthetics and Digital Technologies". In: *Music and Digital Media: A Planetary Anthropology*. UCL Press, pages 220–266. JSTOR: j.ctv2pzbkcg.11.

Spohrer, James C and Elliot Soloway (1989). "Simulating Student Programmers." In: *IJCAI*. Volume 89, pages 543–549.

Sprankle, Maureen (2003). *Problem Solving and Programming Concepts*. 6th ed. Upper Saddle River, N.J: Prentice Hall.

Stacy, Webb and Jean MacMillan (1995). "Cognitive Bias in Software Engineering". In: *Communications of the ACM* 38.6, pages 57–63.

Star, Susan Leigh (1989). "The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving". In: *Distributed Artificial Intelligence*. Elsevier, pages 37–54.

Stefanus DuToit (2014). *Hourglass Interfaces for C++ APIs*. CppCon 2014.

Steffen, Bernhard (2019). "Methods, Languages and Tools for Future System Development". In: *Computing and Software Science: State of the Art and Perspectives*. Edited by Bernhard Steffen and Gerhard Woeginger. Lecture Notes in Computer Science. Cham: Springer International Publishing, pages 239–249.

Stolterman, Erik and James Pierce (2012). "Design Tools in Practice: Studying the Designer-Tool Relationship in Interaction Design". In: *Proceedings of the Designing Interactive Systems Conference on - DIS '12*. Newcastle Upon Tyne, United Kingdom: ACM Press, page 25.

Suh, Nam P. (1998). "Axiomatic Design Theory for Systems". In: *Research in Engineering Design* 10.4, pages 189–209.

— (2001). *Axiomatic Design: Advances and Applications*. The MIT-Pappalardo Series in Mechanical Engineering. New York: Oxford University Press.

Szabo, Claudia and Judy Sheard (2022). "Learning Theories Use and Relationships in Computing Education Research". In: *ACM Transactions on Computing Education* 23.1, 5:1–5:34.

Tagg, P (1997). *Understanding Musical Time Sense: Concepts, Sketches and Consequences*.

Tahiroğlu, Koray (2021). "Ever-Shifting Roles in Building, Composing and Performing with Digital Musical Instruments". In: *Journal of New Music Research* 50.2, pages 155–164.

Tahiroglu, Koray, Michael Gurevich, and R. Benjamin Knapp (2018). "Contextualising Idiomatic Gestures in Musical Interactions with NIMEs". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Thomas Martin Luke Dahl Douglas Bowman. Blacksburg, Virginia, USA: Virginia Tech, pages 126–131.

Tanaka, Atau (2010). "Mapping out Instruments, Affordances, and Mobiles". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Sydney, Australia, pages 88–93.

Tanenbaum, Theresa Jean, Amanda M. Williams, Audrey Desjardins, and Karen Tanenbaum (2013). "Democratizing Technology: Pleasure, Utility and Expressiveness in DIY and Maker Practice". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Paris France: ACM, pages 2603–2612.

Théberge, Paul (1997). *Any Sound You Can Imagine: Making Music/Consuming Technology*. Music/Culture. Hanover, NH: Wesleyan University Press : University Press of New England.

Thomas, John C. and Michael L. Schneider, editors (1984). *Human Factors in Computer Systems*. Human/Computer Interaction. Norwood, N.J: Ablex Pub. Corp.

Tilbian, Joseph and Andres Cabrera (2017). "Stride for Interactive Musical Instrument Design." In: *NIME*, pages 446–449.

Tilbian, Joseph, Andrés Cabrera, Steffen Martin, and Lukasz Olczyk (2017). "Stride on Saturn M7 for Interactive Musical Instrument Design." In: *NIME*, pages 503–504.

Tilkov, Stefan and Steve Vinoski (2010). "Node. Js: Using JavaScript to Build High-Performance Network Programs". In: *IEEE Internet Computing* 14.6, pages 80–83.

Tod, Machover and Chung, J (1989). "Hyperinstruments: Musically Intelligent and Interactive Performance and Creativity Systems". In: *International Computer Music Conference Proceedings* 1989.

Torvalds, Linus (1999). "The Linux Edge". In: *Communications of the ACM* 42.4, pages 38–39.

Trott, Peter (1997). "Programming Languages: Past, Present, and Future: Sixteen Prominent Computer Scientiest Assess Our Field". In: *ACM SIGPLAN Notices* 32.1, pages 14–57.

Turchet, Luca and Carlo Fischione (2021). "Elk Audio OS: An Open Source Operating System for the Internet of Musical Things". In: *ACM Transactions on Internet of Things* 2.2, pages 1–18.

Turner, D (1986). "An Overview of Miranda". In: *SIGPLAN Not.* 21.12, pages 158–166.

University of Technology Sydney and Eindhoven University of Technology and Kees Dorst (2016). "Design Practice and Design Research: Finally Together?" In: *Design Research Society Conference 2016*.

Vallgårda, Anna and Ylva Fernaeus (2015). "Interaction Design as a Bricolage Practice". In: *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*. Stanford California USA: ACM, pages 173–180.

van den Berg, Harry (2008). "Reanalyzing Qualitative Interviews from Different Angles: The Risk of Decontextualization and Other Problems of Sharing Qualitative Data". In: *Historical Social Research / Historische Sozialforschung* 33.3 (125), pages 179–192. JSTOR: 20762306.

Van Nort, Doug, Marcelo M. Wanderley, and Philippe Depalle (2014). "Mapping Control Structures for Sound Synthesis: Functional and Topological Perspectives". In: *Computer Music Journal* 38.3, pages 6–22.

Van Wyngaard, C. J., J. H. C. Pretorius, and L. Pretorius (2012). "Theory of the Triple Constraint — A Conceptual Review". In: *2012 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1991–1997.

Wadler, Philip (1992). "The Essence of Functional Programming". In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '92. New York, NY, USA: Association for Computing Machinery, pages 1–14.

Wan, Zhanyong, Walid Taha, and Paul Hudak (2002). "Event-Driven FRP". In: *Proceedings of the 4th International Symposium on Practical Aspects of Declarative Languages*. PADL '02. Berlin, Heidelberg: Springer-Verlag, pages 155–172.

Wanderley, Marcelo M (2001). "Gestural Control of Music". In: *International Workshop Human Supervision and Control in Engineering and Music*, pages 632–644.

Wang, Ge, Perry R. Cook, and Spencer Salazar (2015). "ChucK: A Strongly Timed Computer Music Language". In: *Computer Music Journal* 39.4, pages 10–29.

Wessel, David and Matthew Wright (2002). "Problems and Prospects for Intimate Musical Control of Computers". In: *Computer music journal* 26.3, pages 11–22.

Williams, Joan C. (2014). "Double Jeopardy? An Empirical Study with Implications for the Debates over Implicit Bias and Intersectionality," in: Harvard Journal of Law & Gender.

Winslow, Leon E. (1996). "Programming Pedagogy—a Psychological Overview". In: *ACM SIGCSE Bulletin* 28.3, pages 17–22.

Wright, D. (2009). "Mathematics and Music". In: Mathematical World. American Mathematical Society. Chapter 4, pages 45–46.

Xambó, Anna (2018). "Who Are the Women Authors in NIME?–Improving Gender Balance in NIME Research". In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Edited by Thomas Martin Luke Dahl Douglas Bowman. Blacksburg, Virginia, USA: Virginia Tech, pages 174–177.

Yamada, Yuki (2018). "How to Crack Pre-Registration: Toward Transparent and Open Science". In: *Frontiers in psychology* 9, page 1831.

Zappi, Victor and Andrew McPherson (2018). "Hackable Instruments: Supporting Appropriation and Modification in Digital Musical Interaction". In: *Frontiers in ICT* 5, page 26.

Zhang, Cheng and David Budgen (2012). "What Do We Know about the Effectiveness of Software Design Patterns?" In: *IEEE Transactions on Software Engineering* 38.5, pages 1213–1231.

Zhang, Weixin and Bruno Oliveira (2019). "Shallow EDSLs and Object-Oriented Programming: Beyond Simple Compositionality". In: *The Art, Science, and Engineering of Programming* 3.3, page 10. arXiv: 1902.00548 [cs].

Zhu, Shuofei, Ziyi Zhang, Boqin Qin, Aiping Xiong, and Linhai Song (2022). "Learning and Programming Challenges of Rust: A Mixed-Methods Study". In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. New York, NY, USA: Association for Computing Machinery, pages 1269–1281.

Zoran, Amit (2011). "The 3D Printed Flute: Digital Fabrication and Design of Musical Instruments". In: *Journal of New Music Research* 40.4, pages 379–387.