# Bricolage in a hybrid digital lutherie context: a workshop study

**Jack Armitage**
Centre for Digital Music
Queen Mary University of London
London, UK
j.d.k.armitage@qmul.ac.uk

**Andrew McPherson**
Centre for Digital Music
Queen Mary University of London
London, UK
a.mcpherson@qmul.ac.uk

## ABSTRACT

Interaction design research typically differentiates processes involving hardware and software tools as being led by tinkering and play, versus engineering and conceptualisation. Increasingly however, embedded maker tools and platforms require hybridisation of these processes. In the domain of digital musical instrument (DMI) design, we were motivated to explore the tensions of such a hybrid process. We designed a workshop where groups of DMI designers were given the same partly-finished instrument consisting of four microphones exciting four vibrating string models. Their task was to refine this simple instrument to their liking for one hour using Pure Data software. All groups sought to use the microphone signals to control the instrument's behaviour in rich and complex ways, but found even apparently simple mappings difficult to realise within the time constraint. We describe the difficulties they encountered and discuss emergent issues with tinkering in and with software. We conclude with further questions and suggestions for designers and technologists regarding embedded DMI design processes and tools.

## CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; *Performing arts*; • **Human-centered computing** → *User interface programming*.


*AM'19, September 18–20, 2019, Nottingham, United Kingdom*

ACM ISBN 978-1-4503-7297-8/19/09…$15.00
https://doi.org/10.1145/3356590.3356604

## KEYWORDS

digital musical instruments, musical instrument design, dataflow programming, liveness, embedded systems

**ACM Reference Format:**
Jack Armitage and Andrew McPherson. 2019. Bricolage in a hybrid digital lutherie context: a workshop study. In *Audio Mostly (AM'19), September 18–20, 2019, Nottingham, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3356590.3356604

## 1 INTRODUCTION

In discourse on interaction design, design processes are often differentiated based on whether the media involved are physical or hardware, versus graphical or software based [14, 22]. Hardware processes are typified as being led by tinkering, or what Vallgårda and Fernaeus refer to as "bricolage practice" [22], and software processes by more conceptual approaches [14]. Digital musical instrument (DMI) designers increasingly engage with hybrid approaches [4, 21], where varied processes take place concurrently, or where the aforementioned distinctions break down. It is important to understand how tensions between tinkering and engineering are navigated in digital lutherie, in order to inform the design of tools and media suited for hybrid craft [8].

To investigate these issues, we propose comparing across DMI design activities with similar scenarios and differing processes. Following workshop studies where DMI designers physically sculpted DMIs using crafting materials [1, 13], we were motivated to run the same activity where software was instead the main focus. Software design workflows for embedded DMIs are in the early stages of development [3, 15], and while new software tools are frequently proposed, little data exists to guide them forward.

This work presents the outcomes of a workshop where groups of three DMI designers worked with a partly-finished instrument by manipulating its Pure Data (Pd) [18] software patch. Like in any implementation process, the designers confronted issues, bugs and apparent dead-ends. In the paper we delve into these issues in detail. The next section of this paper discusses Vallgårda and Fernaeus' use of Levi-Strauss' bricolage to describe interaction design practice [11, 22], and how this relates to concepts of tinkering in and with

programming systems [23, 24]. Following this the workshop design and outcome themes are described and reflected on.

## 2 BACKGROUND

### Interaction design as a bricolage practice

Vallgårda and Fernaeus propose that Levi-Strauss' notion of bricolage, elaborated from its essential usage to describe tinkering and building to encapsulate a practice that "favors making connections between the tools and materials at hand", can be a useful resource for interaction design practice [11, 22]. Bricolage complements conceptual design by facilitating in-situ structuring of events non-hierarchically, and it is suggested that this way of thinking is naturally occurring and should be formally recognised in the interaction designer's toolbelt. Levi-Strauss uses the term *treasury* to describe the bricoleur's set of "signs" and "heterogenous objects" that are used to generate solutions to encountered issues. The bricoleur continuously strives to "renew or enrich" their treasury "with the remains of previous constructions or deconstructions" [11].

Vallgårda and Fernaeus note that bricolage process enables the advantageous use of the "impreciseness of our physical world", imperfections which are characteristic of "tangible and material computing". For them, graphical user interfaces (GUI) rarely present "imperfections or unpredictable events" and behaviour is more "prescribed". A similar distinction between GUIs and non-GUIs is present in McPherson et al [14] where exploration of an unpredictable hardware device through tinkering is described in a "cautious random walk" style. They argue that "software interfaces show a distinctly different pattern" which is more top-down in nature because "there is typically no cost to moving a control across its entire range", whereas with physical or hardware exploration "time and effort scale with the complexity of the change, and the lack of undo raises the cost of a false move" [14].

Increasingly however, interaction designers confront hybrid practices where there is either an intimate co-mingling of hardware and software processes, or such boundaries become difficult to distinguish at all [4, 21]. Tracing the influence on design of the various "push and pull" effects [7] of the hardware and software tools in these scenarios becomes challenging. As researchers seeking to facilitate hybrid craft in a digital lutherie context [8], we are interested to understand more about the tensions and interplay between conceptual and bricolage ways of thinking.

### Tinkering in and with programming systems

In critiquing the design of spaces and tools for hybrid craft, Victor relates the idea of a spectrum of thinking styles spanning tinkering to conceptual understanding to discovery of scientific knowledge, all of which should ideally be supported by the craft environment [24]. In Victor's description, tinkering is the starting point where something seems to work, but the underlying principles are not understood. This is contrasted with conceptual understanding reliant on formalised models, and the discovery of new understanding enabled by scientific ways of thinking and "seeing tools". This spectrum of ways of thinking from tinkering through to engineering through to science are regarded as naturally co-occuring and mutually reinforcing, as in Vallgårda and Fernaeus' work [22]. However Victor does not tie tinkering to hardware and conceptual understanding to software, instead arguing more generally that tools should be designed to support a variety of useful thinking styles, using the most approrpriate representations.

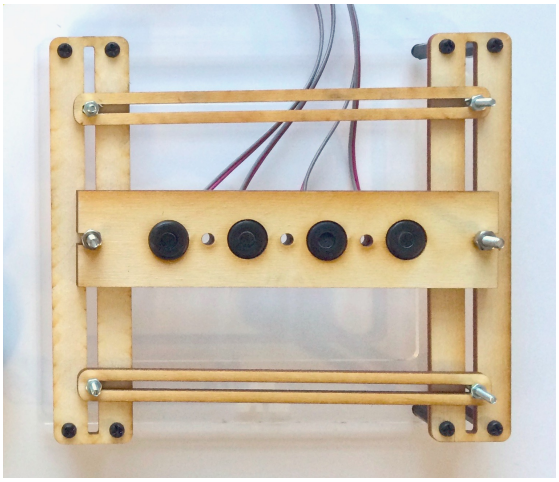
A programming system, consisting of an environment "installed on the computer" and a language "installed in the programmer's head" [23], is one such tool that ideally would support diverse thinking styles including tinkering. The extent to which existing programming systems and GUI software built with them support tinkering is a subject of debate, as highlighted in the previous section. Tinkering processes necessarily utilise under-specified, contradictory and impossible ideas, which are thought of as being difficult to represent as or in computer programs based on discrete symbolic underlying representations [6, 12]. The "materials at hand" aspect of tinkering could be related to the notion of liveness in programming systems, defined by Tanimoto simply as "the ability to modify a running program" [19], but in practice existing in a variety of forms and levels of sophistication. Nash suggests that the types of representations used in a programming system and its liveness level are aspects that co-construct programming practice [16].

It is increasingly feasible for digital musical instrument (DMI) designers to work in hybrid design contexts using integrated and embedded DMI platforms [15]. It is however unclear what balance of ways of thinking - tinkering, engineering, science or otherwise - are required by digital luthiers in these contexts, and in turn how well existing tools and infrastructure support them to create the right balance. In this workshop we wanted to explore this issue by observing digital luthiers working together in a hybrid context.

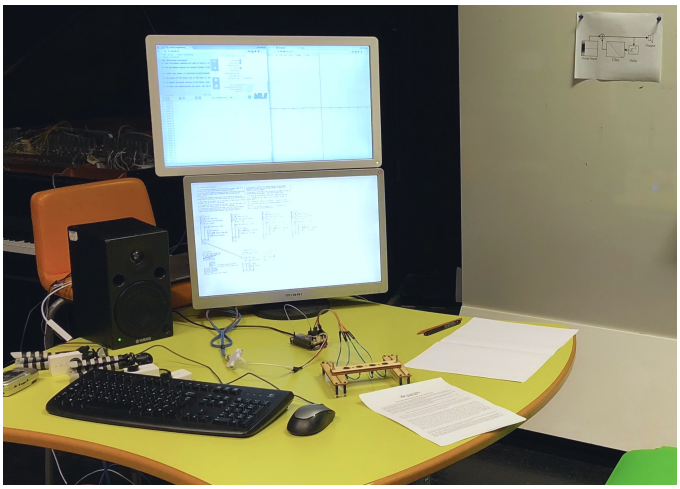
## 3 WORKSHOP DESIGN

### Workshop activity & environment

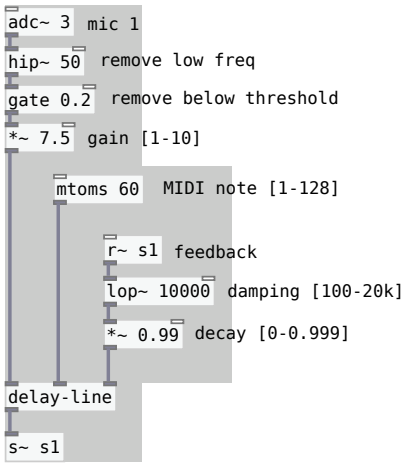
A one hour workshop activity was designed where groups of three DMI designers respond to a probe called the *Unfinished Instrument* (Figure 1a), which is deliberately simple and requires creative and technical intervention to make it more playable and interesting. The instrument and workshop environment are depicted in Figure 1. The goal of the activity



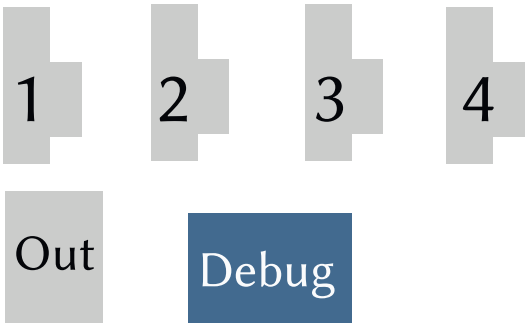
**(a) Top view of *Unfinished Instrument*.**



**(b) Instrument and software tools in situ.**




**(c) Pd patch detail of mic-to-string algorithm with visual outline for reference.**




**(d) Visual overview of Pd patch with four mic-to-string algorithms, audio output and Bela IDE in grey and debugging utilities in blue.**

**Figure 1: Above: instrument and workshop environment. Below: Pd patch mic-to-string detail and visual overview.**

is to facilitate open ended exploration of the instrument and development of its character towards the aesthetic inclinations of the designers. Our motivation in facilitating this is to gain insight into how the different elements of the material environment - the physical instrument, sensors, electronics and software described in more detail in the next section - affect design idea generation, exploration, decision making and development. Towards the end of the one hour activity, participants were asked to summarise and demonstrate the results of their work. Afterwards they filled in a brief survey, and were debriefed about the research project.

The instrument was previously deployed in a NIME 2017 workshop facilitating crafting with physical materials [1, 13], and as such it is based on a simple modular physical structure. In contrast, in this workshop a separate set of participants were given the same physical instrument but instead they work with a Pd patch for one hour. Pd offers a visual dataflow programming environment where objects and connections are represented by text boxes and lines between their inputs and outputs [18]. The specific Pd patch is described in the next section.

### Instrument design

Physically, the *Unfinished Instrument* is constructed from simple laser cut, modular parts which enable many possibilities for arranging its overall form and the positioning of the

inputs (Figure 1a). Four low cost microphone capsules are housed inside rubber grommets, which are connected to a Bela device running a Pd patch (Figure 1b). When the player taps or otherwise interacts with the mics, the signal is used to excite four Karplus-Strong [9] vibrating string models. In the Pd patch, each mic signal is pre-processed with a high pass filter, gate and gain factor (Figure 1c). The mics are clearly visible so that the designer is aware of what kind of sensor they are working with and how they should approach it [17]. The patch mixes the four string sounds together, and in addition provides debugging facilities for printing and visualising via the Bela platform's browser-based oscilloscope (Figure 1d).

Pd does not feature a formal spatial syntax, meaning the spatialisation of objects and connections are left to the author as a secondary notation [12]. Perhaps moreso than textual code comments, the layout of a Pd patch can communicate a great deal about the patch and its author(s). In this workshop, it was desirable that the patch be as transparent as possible to a broad range of experience levels, and that it be modifiable without too much concern for encapsulation, duplication and deduplication. At a high level, the patch was separated into six distinct blocks, which were connected via send and receive objects rather than graphical connections to highlight their separation. Four of these were mic-to-string algorithms, duplicated left-to-right to mimic the physical design of the instrument, and the two other blocks were for audio outputs and debugging tools. The mic-to-string algorithm can be visually broken down into mic processing, MIDI note to pitch input, and feedback input.

### Instrument editing workflow

When referring to Tanimoto's liveness levels [19], Pd can be described as having Level 4 properties (a live editable flowchart where program data is continuously displayed and is manipulable). Objects and connections can be added, edited or removed while the program is running, with the caveat is that there can be glitches when the signal graph changes. As a result one might elect not to make graph changes during an artistic performance, however during design iteration graph changes are usually frequent. Pd also support graphical user interface (GUI) elements such as sliders, buttons and plots, for direct manipulation of program data. In this case however the Pd patch was running on a remote device, Bela, so the workflow for editing the patch differed to the regular live editing workflow. While the patch is edited live in the standard desktop application, edits are not reflected on the instrument in real-time, neither are changes to program data via graphical interface elements such as sliders. Instead, the update procedure, which was measured to take around seven seconds for the given patch, is as follows:

(1) The user edits the patch using the desktop PureData application.
(2) When they want to update the instrument, they save the patch.
(3) The instrument stops running while the patch is automatically copied to the embedded device and recompiled.
(4) Once the patch has been recompiled, the user can continue to interact with the instrument.

### Participants, grouping and data collection

Participants responded to an open call published on mailing lists and social media. They were required to have at least some experience with at least one sound and music computing language. Participants were matched into groups to balance out experience, such that every group had at least one participant with some PureData or Max/MSP experience. In total 15 participants were grouped into five groups of three.

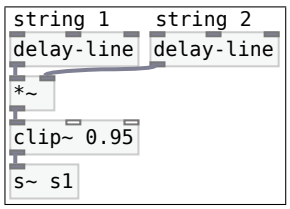
Participants self-reported their experience before the activity and were grouped in threes into two more experienced groups (G1-2) and three less experienced groups (G3-5). Workshop sessions were documented with video and audio recordings. Each patch update was automatically version controlled, and the final patches were visually annotated based on object movement, editing, deletion and addition. Each participant completed a brief post-activity survey covering their impressions and reflections on the activity, and the collected data was thematically analysed by the authors [2].

## 4 OUTCOMES

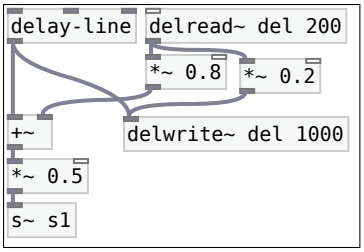
This section presents observations made about the workshop in an approximately chronological way, beginning with familiarisation and ideation, implementation patterns, evaluation and iteration, and finally demonstration of outcomes by the groups. The topic headings suggest discrete transitions from one type of activity to the next, but the actual activity was fluid across them. Some groups were already ideating while familiarising, and some skipped discussion of ideas and went straight to experimenting with implementation. Some groups stuck to implementing a few key ideas generated early on, while others continued to broaden their exploration throughout.

### Familiarisation and ideation

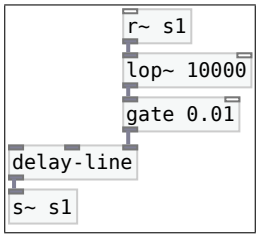
After being introduced to the workshop brief and materials, the groups showed little hesitation in accepting their task and assessing the possibilities available. The initial activity across the groups involved exploring the instrument's responses to different gestural interactions with the mics, which included tapping, hitting, rubbing and scratching. From this



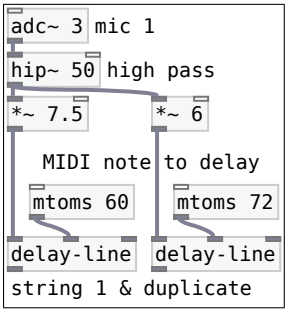

**(a) G1 A: Audio multiply String 1 & 2.**




**(b) G1 B: Delaying a string.**




**(c) G3 A: Gating string feedback.**




**(d) G3 B: Duplicating in unison (1oct).**

**Figure 2: PureData mappings across mics and strings (laid out and commented by the research author).**

the groups perceived the mic response to be lacking sensitivity, and accordingly they adjusted the mic processing parameters to alter the response. Having made these adjustments, the groups also became familiarised with the procedure for editing the Pd patch and updating the instrument. They concluded overall that the Pd patch and thus the overall instrument was "minimal" (G4), "basic" (G4), "limited" (G2), "simple, essential" (G3) and while not necessarily "inspiring" (G3), it showed "potential" (G4) as a "blank canvas of possibilities" (G1) and a "good starting point" (G4). Subsequently the groups engaged briefly in ideation through discussion, referencing gestures they were making with the instrument, making comparisons to their own experiences and seeking out common frames of reference.

## Implementation patterns

*Mappings across the Pd patch.* Rather than accepting the Pd patch structure as a given and working within it, in most cases the groups took advantage of its flexibility and explored making new connections across the structure. Here we describe four specific example mappings (Figure 2) and then reflect on their use of the patch as an ideas canvas. Two examples come from Group 1 (G1 A & G1 B), a more experienced group, and two examples come from Group 3 (G3 A & G3 B), a less experienced group.

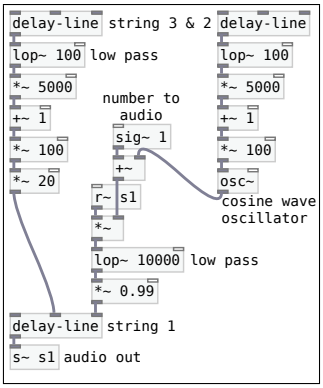
In example G1 A (Figure 2a), G1 multiplied the outputs of String 1 & 2 together. They mentioned that this approach was referencing *frequency modulation*, where one string could act as a carrier and another a modulator. In example G1 B (Figure 2b), G1 inspected the 'delay-line' abstraction, and added a similar extra delay to a string. Adding further delays to other strings, they created implicit relationships between them by giving them different delay times.

In example G3 A (Figure 2c), G3 took the 'gate' abstraction used to eliminate background noise from the mic inputs, and inserted it in place of the decay parameter of a string. This reduced the feedback signal much more rapidly than the decay parameter, creating a much more percussive envelope. In example G3 B (Figure 2c), G3 duplicated String 1 and connected Mic 1 to its input, decreasing the input gain of the mic and increasing the string pitch by one octave, creating a unison effect. With this move they were proving the concept of triggering chords or harmonies from a single input.

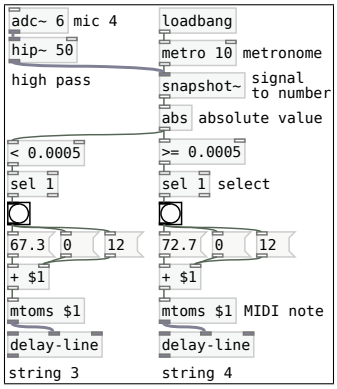
*Audio signal-based control structures.* Each group attempted at least once to use the mic input audio signal in or as a control structure (Figure 3). This could have been motivated by the lack of real-time patch editing capabilities and the lack of additional physical inputs, but equally also by extraneous factors. What was observed in each case was that the participants did not find this process intuitive, regardless of their level of experience. In this section we highlight three examples of difficulties they faced and discuss their impact on the groups' design progress.

G2 filtered and scaled the outputs of String 2 & 3 to control the decay and pitch inputs of String 1 (Figure 3a). They wanted Mic 2 & 3 to modulate String 1 such that the player would need to excite three inputs simultaneously. They chose to use the string outputs rather than the direct or processed mic inputs. Their approach was to low pass filter, scale and translate the string signals, trying parameter values ranging over a number of orders of magnitude. They were somewhat satisfied with their result but could not compensate for the low mic sensitivity.

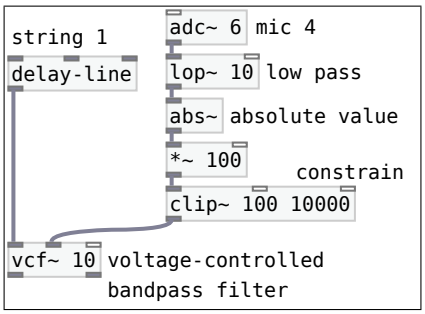
G4 used relational operators based on discrete samples of Mic 4 to control the pitch inputs of String 3 & 4 (Figure 3b). They originally wanted to control the pitches using a horizontal slider object in Pd, but realising this was not possible with the non-real time workflow, they tried to implement a control structure that would allow pitch to vary based on Mic 4 input. The figure shows their working implementation, but prior to this their patch spent some time in a state where zeroes were being sent to the string pitch input, causing a repetitive clicking sound which they assumed was occurring in time with their metronome. Their initial threshold values were several orders of magnitude too large, which they




(a) G2: String 2 & 3 to String 1 decay & pitch.




(b) G4: Mic 4 to String 3 & 4 pitch select.




(c) G5 (PM): String 1 to filter input, Mic 4 to filter centre frequency.

**Figure 3: PureData patch excerpts of signal to control structures (laid out and commented by author).**

discovered after converting the control signal back into an audio signal and visualising it on the oscilloscope.

G5 processed String 1 with a voltage-controlled bandpass filter, with a filtered and scaled Mic 4 as the centre frequency (Figure 3c). They wanted to use Mic 4 as a breath controller for String 1. Unlike other groups, they started by visualising the Mic 4 signal on the oscilloscope. They were able to create an audible effect but were not completely satisfied with the level of control.

In each of these cases we could summarise these approaches, which were recalled from memory rather than discovered for the first time, as having memorable ingredients, but forgettable recipes. The elements required to turn a signal into a control (filter, scale, etc) were not difficult to remember, but we did not observe an effective and repeatable approach to devising their ordering and parameterisation for a less than ideal sensor. Such a recipe might have included more extensive and consistent use of debugging tools, which was only observed to a degree. Overall these difficulties impacted the pace of design progression in the groups and led to some frustration, as these tasks were perceived to be simple stepping stones to the more sophisticated ideas they were attempting to realise.

### Evaluation, iteration and final demo

Opportunities to evaluate the instrument came at regular intervals due to the nature of the software update process. At these intervals, the groups would typically play, listen, consider and discuss before deciding what to do next. There were various evaluation outcomes and decision making at each stage.

Firstly regarding evaluation, the groups decided that either the implementation was working as planned or not, that in either case the result itself was interesting or not, and if any surprising outcomes occurred that were worth considering. Secondly decisions had to be made as to what to do next. If an idea was working and interesting, they could go forward and refine, reimplement or add extra features, and if not they had to decide whether it was worth persevering. In an idea was working but uninteresting, or not worth pursuing further, they could go back to older ideas or propose new ones.

In all but a few cases where discarded ideas were deleted from the patch, the groups scattered their ideas across the Pd patch. Some used different mic-to-string algorithms to trial different ideas. In this way, multiple ideas were often representated simultaneously either explicitly or implicitly in the patch (through orphaned objects for example), like in a sketchbook. The final patches represented a snapshot of their process in some cases, as much as a specific design intent.

The groups were asked briefly at the end to summarise their activity. G1 commented on the time limit as a stringent constraint: "it feels like you would need a lot of time [...] even a day would be good [...] you could even plan it a bit". G2 made a cogent reflection about their process: "realising the limitations of the inputs and then scale the design to match something that could be interesting with those inputs". G3 described that initially they "analysed the system and we outlined its parts and then we decided on which part to focus our work", although the outcomes did not follow the plan they formed. G4 felt their attention was drawn to timbre: "we had to stick to the more timbral qualities of each individual sound. Because that was the avenue of real-time-ness if that makes sense". G5 appreciated that the hybrid design context "kind of brings a reality because the sound produced in the end is a result of a physical system".

## 5 DISCUSSION

### Bricolage and tinkering in a hybrid craft context

Vallgårda and Fernaeus' bricolage explicitly encapsulates the non-digital aspects of interaction design [22], nevertheless this hybrid workshop's outcomes follow her descriptions of bricolage practice. Similarly, tinkering approaches with physical DMIs have been described as sharply contrasting with software-based explorations [14], and yet we also identify certain similarities. We believe these processes share a situated exploration of the design patterns and changes that are most immediately reachable, regardless of whether the tinkering is occuring with physical or digital tools.

It is from one perspective common sense that any complex process must be composed of simpler steps, and that the simpler steps are by definition one step away from the previous step. However at each step the materials being used - in this case mostly Pd - exert a creative pull [7], and in these workshop outcomes we saw design patterns reappearing frequently that hint at this influence. Frequently the participants' design moves were composed of individual steps, as part of an iterative process where a step was taken, re-evaluated and new steps were chosen. The programming system's representations seemed to highlight or encourage connecting disparate parts of the Pd patch experimentally, inspiring a tinkering approach which previous work has argued is more exclusive to hardware-based processes.

Another aspect of bricolage that we felt was present was the growing of a *treasury* or inventory of design ideas and implementation experiences. Examples of contents in the groups' treasuries included interaction gestures, interaction ideas, their shared Pd vocabulary, and the disconnected Pd patch parts that were kept rather than deleted. Frequently the treasury was indeed returned to when deciding how to iterate based on particular evaluations. Although the "signs" Levi-Strauss referred to were conceived as half-way points between concrete images and abstract concepts [11], it could be considered how programming environments can aid users in building, tracking and recomposing their treasuries.

### Diverging human and machine representations

For the given activity and time constraint of this workshop, and the complexity level of the ideas the participants wanted to tinker with, it appears that the level of abstraction of Pd was perhaps too low. The consistency across groups of attempting audio-signal based control by filtering and scaling, and the struggle they faced in achieving this, might suggest a latent set of primitives for audio-rate sensing oriented embedded DMI design platforms. These primitives would condition audio-rate signals into control signals, smooth them out and scale them into desirable numerical ranges, perhaps offering different interpolation methods. In some sense they would encode the recently accumulated knowledge of digital luthiers over the past two or three decades regarding effective mapping methods, and instantiate that knowledge similar to how animation tools instantiate Disney's twelve basic principles of animation [20].

Victor argues that a programming environment should enable an author to "create by abstracting – start concrete, then generalize" [23]. In our workshop we did not witness anyone creating their own abstractions explicitly, although it could be argued that copying and pasting small programs they created could be considered as a form of implicit and weak abstraction. We did not poll participants about abstractions, so we can only speculate that the available methods for composing and decomposing abstractions were too costly in time, labour and cognitive overhead to be worthwhile in this context. In turn, we suggest the need for methods of abstracting to become more responsive and contextually aware to facilitate rapid and commonplace usage.

### Workflow liveness and iteration time

Recalling Tanimoto's liveness levels described in Section 2, the update procedure in this workshop, which was measured to be approximately seven seconds in duration from saving to hearing the patch again estimates how long each group waited for compilation), shifted Pd from liveness Level 3 (or 4 if using GUI objects) to 2 - a non-live executable flowchart. The positive outcome of this trade is the increased level of performance and integration of the resulting instrument, which can run standalone, at low latency, and with high-bandwidth sensing, such that the designer or performer can evaluate the instrument holistically and in-situ [15].

In trading off liveness for integration, the digital luthiers gains certain advantages. For example, once they became familiar with the mics' physical responses, they used them to rapidly and holistically evaluate changes. They used their musical and tactile skills to quickly feel out the behaviour of the instrument, internalise it, and reconcile their expectations of the program's behaviour with the actual behaviour they observed. Based on this, we suggest that the availability of high levels of integration during the design process affords the digial luthier the ability to engage directly with instrumentality which Hardjowirogo defines as "that which defines a musical instrument as such", "the essence of the musical instrument", and "specific instrumental quality" [5].

Equally, there were a number of disadvantages to the decreased liveness level of the workflow. The seven second save-upload-recompile-run stage, which replaced live program editing, felt costly ("It was slow to develop the digital audio part of the instrument so it mainly was left unchanged" - G5) to the groups. Any change, no matter how large or trivial, required the same amount of effort to experience, which required an entirely different approach than the participants

were familiar with. Based on these outcomes, we suggest that when workflow liveness in DMI design contexts is decreased, the impact on design thinking is both quantitative and qualitative: less design moves are possible in the same window of time, entire categories of ideas are rejected, and other categories are not thought about at all.

One of the more experienced participants from G2 commented that they would rather work offline without the instrument in order to reverse the tradeoff of liveness and integration ("What I would normally do here is just work with not with that controller for a bit but just in Pd normally just listening to it in real-time"). For this participant, and perhaps others who noted the workflow speed, seeking liveness instead of integration would have been worthwhile.

### Reflections on the workshop design

Far from being passive observers, we acknowledge that we were present in the research throughout, no less than in the way our backgrounds as both researchers and designers influenced the workshop design, instrument kit and Pd patch. The time constraint of one hour, the essentially open-ended brief, and using the same kit as the physical crafting workshop [1, 13] (without iterating on its faults to facilitate comparison in later work) all impacted the outcomes. Indeed it seems that in this time constraint, Pd as a programming system was only able to facilitate tinkering, a point which deserves future scrutiny. The group context may have introduced a social pressure not to monopolise the activity by chasing complex ideas through intermediate implementations steps that do not work. The backgrounds and experience levels of the participants also impacted on which ideas were shared and pursued. We invited this richness deliberately, but correspondingly we advise caution regarding our interpretative statements based on our observations, and acknowledge that our biases will be present equally in the discussion as they were in the workshop itself [2, 10].

## ACKNOWLEDGMENTS

Research supported by EPSRC EP/G03723X/1 and EP/L01632X/1. Thanks to Augmented Instruments Lab members for workshop design contributions.

## REFERENCES

[1] Jack Armitage and Andrew McPherson. 2018. Crafting Digital Musical Instruments: An Exploratory Workshop Study. In *Proc. New Interfaces for Musical Expression*. Blacksburg, Virginia, USA.

[2] Jessica T. DeCuir-Gunby, Patricia L. Marshall, and Allison W. McCulloch. 2011. Developing and Using a Codebook for the Analysis of Interview Data: An Example from a Professional Development Research Project. *Field methods* 23, 2 (2011), 136–155.

[3] Liam Donovan, S. M. Astrid Bin, Jack Armitage, and Andrew P. McPherson. 2017. Building an IDE for an Embedded System Using Web Technologies. In *Proc. Web Audio Conference*. London, United Kingdom.

[4] Connie Golsteijn, Elise van den Hoven, David Frohlich, and Abigail Sellen. 2014. Hybrid Crafting: Towards an Integrated Practice of Crafting with Physical and Digital Components. *Personal and ubiquitous computing* 18, 3 (2014), 593–611.

[5] Sarah-Indriyati Hardjowirogo. 2017. Instrumentality. On the Construction of Instrumental Identity. In *Musical Instruments in the 21st Century*, Till Bovermann, Alberto de Campo, Hauke Egermann, Sarah-Indriyati Hardjowirogo, and Stefan Weinzierl (Eds.). Springer, 9–24.

[6] Kenneth E. Iverson. 2007. Notation as a Tool of Thought. *ACM SIGAPL APL Quote Quad* 35, 1-2 (2007), 2–31.

[7] Robert Jack and Andrew McPherson. 2017. Rich Gesture, Reduced Control: The Influence of Constrained Mappings on Performance Technique. In *Proc. International Conference on Movement and Computing*.

[8] Sergi Jorda. 2005. *Digital Lutherie: Crafting Musical Computers for New Musics' Performance and Improvisation*. Ph.D. Dissertation. Universitat Pompeu Fabra.

[9] Kevin Karplus and Alex Strong. 1983. Digital Synthesis of Plucked-String and Drum Timbres. *Computer Music Journal* 7, 2 (1983), 43–55.

[10] Giacomo Lepri and Andrew McPherson. 2019. Making up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice. In *Proc. ACM Designing Interactive Systems*. 13.

[11] Claude Levi-Strauss. 1966. *The Savage Mind*. University of Chicago Press.

[12] Alex McLean. 2011. *Artist-Programmers and Programming Languages for the Arts*. Ph.D. Dissertation. Goldsmiths University of London, London, United Kingdom.

[13] Andrew McPherson, Jack Armitage, S. Astrid Bin, Fabio Morreale, and Robert Jack. 2017. NIMEcraft Workshop: Exploring the Subtleties of Digital Lutherie. In *Proc. New Interfaces for Musical Expression*.

[14] Andrew P. McPherson, Alan Chamberlain, Adrian Hazzard, Sean McGrath, and Steve Benford. 2016. Designing for Exploratory Play with a Hackable Digital Musical Instrument. In *Proc. ACM Designing Interactive Systems*. ACM, 1233–1245.

[15] Giulio Moro, S. Astrid Bin, Robert H. Jack, Christian Heinrichs, and Andrew McPherson. 2016. Making High-Performance Embedded Instruments with Bela and Pure Data. In *Proc. Live Interfaces*. University of Sussex.

[16] Chris Nash and Alan Blackwell. 2012. Liveness and Flow in Notation Use. In *Proc. New Interfaces for Musical Expression*.

[17] Jon Pigrem and Andrew P. McPherson. 2018. Do We Speak Sensor? Cultural Constraints of Embodied Interaction. In *Proc. New Interfaces for Musical Expression*. Blacksburg, Virginia, USA.

[18] Miller Puckette et al. 1996. Pure Data: Another Integrated Computer Music Environment. *Proceedings of the Second Intercollege Computer Music Concerts* (1996), 37–41.

[19] Steven L. Tanimoto. 2013. A Perspective on the Evolution of Live Programming. In *International Workshop on Live Programming*.

[20] Frank Thomas, Ollie Johnston, and Frank Thomas. 1995. *The Illusion of Life: Disney Animation*. Hyperion New York.

[21] Vasiliki Tsaknaki, Ylva Fernaeus, Emma Rapp, and Jordi Solsona Belenguer. 2017. Articulating Challenges of Hybrid Crafting for the Case of Interactive Silversmith Practice. In *Proceedings ACM Designing Interactive Systems*. ACM, 1187–1200.

[22] Anna Vallgårda and Ylva Fernaeus. 2015. Interaction Design as a Bricolage Practice. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, 173–180.

[23] Bret Victor. 2012. Learnable Programming: Designing a Programming System for Understanding Programs. http://worrydream.com/LearnableProgramming/.

[24] Bret Victor. 2014. Seeing Spaces. http://worrydream.com/SeeingSpaces/.