The Cognitive Dimensions of Music Notations

Chris Nash
Department of Computer Science and Creative Technology,
University of the West of England,
Frenchay Campus, Coldharbour Lane, Bristol, BS16 1QY, United Kingdom
chris.nash@uwe.ac.uk

ABSTRACT

This paper presents and adapts the *Cognitive Dimensions of Notations* framework (Green and Petre, 1996) for use in designing and analysing notations (and user interfaces) in both digital and traditional music practice and study. Originally developed to research the psychology of programming languages, the framework has since found wider use in both general HCI and music. The paper provides an overview of the framework, its application, and a detailed account of the core cognitive dimensions, each discussed in the context of three music scenarios: the score, Max/MSP, and sequencer/DAW software. Qualitative and quantitative methodologies for applying the framework are presented in closing, highlighting directions for further development of the framework.

1. INTRODUCTION

Music and programming are two creative mediums mediated through notation. In both scenarios, notation is used to describe the behaviour of a system for subsequent execution – be that by computer or human performer. As a representation of a creative domain, notation shapes how the practitioner perceives and interacts with their art. In formal systems, such as computers and common practice music, the design of notation defines what actions and expressions are possible. However, the design of notation and techniques for manipulating can also dispose (or discourage) users to certain actions and formulations – what actions and expressions are easy.

This paper draws on research and findings in the psychology of programming and music HCI to describe a flexible approach to analysing and evaluating notations and user interfaces in a variety of digital and traditional musical practices. It begins with a discussion of the parallels between musical creativity (e.g. composition) and programming, before introducing the *Cognitive Dimensions of Notation* framework. [1] To demonstrate the application and adaptability of the framework, Section 4 explores sixteen core dimensions of notation use through three scenarios of notation-mediated music interaction: sketching and transcription using traditional score; audio/music programming using Max/MSP; and composition and production using a sequencer or digital audio workstation (DAW). Finally, Section 5 offers a survey of methodologies for applying the framework.

2. FROM PROGRAMMING TO MUSIC

There are many parallels between programming and creative musical scenarios such as composition, both in digital interaction and more traditional music practice.

Fundamentally, both practices can be mediated through notation. In Western music, formal training and practice is oriented around the musical score. Composers exploit the flexible affordances of pencil and paper to sketch and experiment with musical ideas, before transcribing their work more formally for communication to the performer, who interprets the notation to realise the written form as music (i.e. sound). The listener, as the consumer, does not see the notation. In programming, developers describe processes and interactive systems in source code, using symbol-based formal languages (such as C/C++, BASIC, or LISP). The code is compiled or interpreted by the computer to create a program that encapsulates some kind of functionality and processing of input and/or output. As in music, the end-user does not see the source code.

In both instances, the formal rules of the notation define what actions and entities can be represented with respect to the creative domain – music or program behaviour. The musical score developed over centuries to efficiently capture the formal rules of Western tonal music, during the common practice period (1600-1900). [2] While this covers a wide gamut of musical practices and styles, and continues to be relevant in modern styles, the format and conventions of the score implicitly shape the creativity of anyone working through it. [3,4,5]

Unlike music, no single standard programming notation exists; users have an element of choice over formalisms. Most coding languages are *Turing complete*, meaning they are practically capable of encapsulating any desired computer functionality. Thus, the issue with such notations is not what is possible, but what functionality is easy or quick to code, given the formal rules of the notation. [1] Different languages (and dialects) offer distinctions in syntax and semantics to facilitate different users and uses. For example: *BASIC* is designed using simple English keywords to be easily comprehended by beginners (at the expense of structure); *Assembler* more directly exposes low-level workings of hardware (at the expense of human-readability); and *object-oriented languages*, like *Java* and *C++*, are designed around creating modular systems and abstract data models that map onto user ontologies to enable notation of both low- and high-level concepts. As music notation similarly seeks to support beginners, instrument affordances, and flexible levels of abstract representation, it is instructive to analyse usability factors in notations for programming.

Beyond the format of notation, editing tools also impact the usability of a notation, and although text-based notations can be separated from code editors, other programming paradigms are more integrated with the user experience of the development environment. For example, *visual programming languages (VPLs)*, such as Max/MSP, are manipulated through a graphical user interface, the usability of which impacts how users perceive the language and its capabilities. Other coders develop using an *integrated development environment (IDE)*, offering unified platform for writing, building, running and debugging code. The integration of such tools allows code edits to be quickly tested and evaluated, accelerating the feedback cycle and thus enabling *rapid application development*, in turn facilitating experimentation and ideation. [6] Thus, any approach for analysing notation should likewise address factors in the UI.

In music, similar considerations can be made of the design of interactive modes supported by tools to manipulate notations – be that pencil and paper, ink and printer, or mouse and computer screen. Score notation supports composers in creating music, performers in interpreting it, scholars in analysing it, and learners in understanding it. In each case, practitioners use different techniques and tools to interact with the encapsulated music. Moreover, while music plays a functional role in many aspects of culture, it is also about personal, creative expression, and thus it is important to look at how the development of musical ideas is shaped by the design of notations. To consider this, the following section uses the analogue of programming to adapt an established analysis framework that might be used to reveal limitations, influences and opportunities in music notations and interfaces.

3. A USABILITY FRAMEWORK

The *Cognitive Dimensions of Notations* [1] is a usability framework originally developed by Thomas R. G. Green and Marian Petre, to explore the psychology of interaction with notation in the field of programming, breaking different factors of the software designer's user experience into cognitive dimensions that separately focus on affordances of the notation, but which collectively help to paint a broad picture of the user experience involved with editing code and crafting interactive software systems.

The definitions of each dimension (see Section 4) are borne from research in cognitive science, but shaped to operationalise the framework as a practical analysis tool for use by interaction designers, researchers, and language architects. [7] It is intended that each dimension describe a separate factor in the usability of a notation, offering properties of *granularity* (continuous scale; high/low), *orthogonality* (independent from other dimensions), *polarity* (not good or bad, only more or less desirable in a given context), and *applicability* (broader relevance to any notations).

In practice, these properties cannot always be met. [1,7] Interactions between dimensions are evident, with either concomitant or inverse relationships. For example, low *viscosity* (~ ease of changing data) contributes to *provisionality* (~ ease of experimentation); whereas, higher *visibility* (~ ease of viewing) may reduce *hidden dependencies* (~ invisible relationships). Moreover, some dimensions are value-laden; intuitively it may be difficult to see how *error proneness*, *hard mental operations*, and *hidden dependencies* are desirable. However, knowledge of these relationships can be useful in solving usability issues, where a solution to one dimension can be addressed through a design manœuvre targeted at another.

The exact set of cognitive dimensions is not fixed, and various proposals for new dimensions, designed to capture aspects of a notation or user experience beyond the original framework, have been forwarded – many arising from its expanded use in other fields in and around HCI (non-programming interaction, tangibles, computer music). New dimensions should be measured against the aforementioned requirements, but their value is most effectively gauged by how much they reveal about the interaction context in question, and arguably the greatest contribution of the framework is that it provides a vocabulary and structure for discussing and analysing notation from multiple perspectives.

As an HCI tool (and in contrast to other usability methodologies), it allows both broad and detailed analysis of human factors in a notation or user interface, adaptable to different use cases and audiences. By considering each cognitive dimension in the context of a specific system, designers and evaluators can assess how the notation fits their user or activity type, whether that's making end-user systems easier to use [5] or making musical interaction more rewarding by increasing challenge. [8,9]

For a detailed discussion of the background and definition of dimensions in the original framework, see [1]. For further publications on the subject, see the framework's resource site and associated bibliography.[1]

4. DIMENSIONS OF MUSIC NOTATION

In this section, sixteen core dimensions of the framework, adapted for a musical context, are detailed and discussed in the context of three common musical interaction scenarios. To evaluate both formal and informal music notation, each dimension is respectively reviewed in the context of the musical score and sketch (SCORE). The intersection of musical expression and programming is then similarly explored in the context of the Max audio synthesis environment (MAX/MSP). Lastly, the framework is used to review the user interfaces and experiences offered by mainstream end-user systems, through an analysis of digital audio workstation (DAW) and sequencer software (DAW). In addition to a description of the dimension, each is introduced with a simple question designed to encapsulate the definition in a form that can be used to capture feedback from end-users (e.g. a user survey [3,8,10,11]).

4.1 Visibility

"How easy is it to view and find elements or parts of the music during editing?"

This dimension assesses how much of the musical work is visualised in the notation or UI, as well as how easy it is to search and locate specific elements. While hiding

[1] http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/

data will make it difficult to find, showing too much data can also slow the search. Pages and screens limit the amount of space available for displaying data, requiring a careful balance of visual detail and coverage.

[Related dimensions: *juxtaposability*, *abstraction management*, *hidden dependencies, conciseness/diffuseness*, *closeness of mapping, role expressiveness*.]

SCORE: In sheet music, all notated elements are visible on the page; there is no feature to dynamically hide notated elements, beyond using separate sheets. However, music is hidden on other pages, where page turns also present challenges for typesetter or performer, if phrases continue over a join. This can be accounted for in layout, with forethought, but this increases the *premature commitment*. Things are easier for the composer, as a draft musical sketch need not cater for the performer, and pages can be laid side-by-side (see *juxtaposability*). Some aspects of the final musical form (e.g. expression and prosody of performance) may not be visually explicit in the musical score (see *closeness of mapping*).

MAX/MSP: As a visual programming language (VPL), *visibility* is a key dimension of Max, which explicitly represents the flow of audio and musical data. As in many programming languages, the visibility of process (code/data-flow) is prioritised over musical events (data). In Max, many elements of a system are not visualised, such as the internal state of most objects (e.g. default or current values). There is also no inherent linear / serial representation of musical time, making it difficult to sequence past or future events or behaviour. As such, Max best suits generative and reactive (live) applications.

DAW: Like most end-user software, DAWs offer a graphical user interface (GUI) that is inherently visual. However, different sub-devices (views) reveal or hide different properties of the music; no screen provides a comprehensive or primary notation. Notably, the arrange window is the only window designed to provide an overview of the whole piece, but filters low-level detail (e.g. notes), which must be edited through other interfaces (score, piano roll, data list). As a result musical data is dispersed through the UI and can be difficult to find, often involving navigating and scrolling through windows and views with the mouse. Arguably the primary and most expressive interaction medium for the sequencer is inherently non-visible: performance capture (MIDI/audio recording).

4.2 Juxtaposability

"How easy is it to compare elements within the music?"

Related to *visibility*, this dimension assesses how notated music can be compared against other data. Pages and moveable windows allow side-by-side comparison, albeit at some cost to *visibility*. How clearly elements and their purpose are represented will also affect how easy it is to compare notated passages (see *role expressiveness*). Music systems may also provide tools for non-visual comparisons – e.g. sound (see *progressive evaluation*).

[Related dimensions: *visibility*, *consciseness/diffuseness, role expressiveness*, *progressive evaluation*]

SCORE: Pages allow side-by-side comparison of elements, and the formal rules for encapsulating music make visual inspection an effective tool for assessing similarity (rhythmic patterns, melodic contour, etc.). However, some musical properties are distinguished more subtly in the visual domain (e.g. harmony, key, transposed parts – see *hidden dependencies*), requiring musicianship as well as notational literacy to enable effective comparison.

MAX/MSP: Max's windowed system allows side-by-side comparison, so long as *abstraction* (sub-patching) is applied effectively. Groups of objects can be dragged next to each other, but this becomes cumbersome as the patch grows and objects are intricately woven and linked to surrounding objects (see *viscosity* and *premature commitment*). Broad visually similarity and functional similarity may not always align (see *role expressiveness*).

DAW: As in Max, windowed systems allow side-by-side comparisons, though sizing, scrubbing, and scrolling can be cumbersome in the face of many windows, a common issue in traditional linear sequencers [4,12]. Most visualisations of musical elements are easy to compare to similar properties of other tracks, bars, etc., and generalised representations (track automation envelopes) also offer a basis for comparison across different musical properties.

4.3 Hidden Dependencies

"How explicit are the relationships between related elements in the notation?"

This definition assesses to what extent the relationships and dependencies (causal or ontological) between elements in the music are clear in the notation. Showing dependencies can improve *visibility*, but there is often a trade-off with editing *viscosity*. For example, in programming, textual source code (e.g. C/C++) can be easily edited, but the relationships between sections of code, functions, and variables are not explicitly shown. However, in *visual programming languages (VPLs)*, objects and variables are linked using arcs, making their functional connection visually explicit, but making it harder to edit, once woven into the rest of the code. [1,13]

[Related dimensions: *visibility*, *closeness of mapping*, *role expressiveness*, *viscosity*, *conciseness/diffuseness*]

SCORE: The *visibility* of the score ensures no actual data is hidden, except on separate pages, though the musical relationship between notated elements is not always explicit. Some elements are visually linked (e.g. slurs and phrasing) and there are other visual cues that events are related, as in the use of beams or stems to respectively bridge rhythmic or harmonic relationships. However, musical events are sensitive to context, as with dynamic marks, previous performance directions, and key changes – though a visual link between each individual note and the markings that affect its performance would be inefficient to notate explicitly (increasing the *diffuseness*).

MAX/MSP: A key attribute of all VPLs; the graphical connection of elements using patch cables explicitly identifies dependencies between Max objects, and help to show signal flow and the wider architecture of a patch.

However, patch execution in Max is also affected by the relative placement and spatial relationship of objects (e.g. right-to-left processing of outlets), which is not visualised explicitly and can lead to unexpected patch behaviour that confuses users. While relations between objects are shown, its specific functional purpose is not explicit and the object's current state or value is hidden. For example, default values specified as arguments can be replaced by messages, but there is no visual indication the value of the object has changed from its displayed default value.

Use of sub-patching can also hide functionality, though this is a common trade-off with the additional expressive power offered by *abstraction* mechanisms. Moreover, as a data-flow environment (and in contrast to imperative programming, as in C++), musical time and the sequence of events are not visually explicit, hiding causal and timing relationships between musical elements.

DAW: The variety of different views and UIs designed for different purposes and perspectives can lead to a large number of hidden dependencies within DAWs. [3] For example, across the different screens and settings there are dozens of variables that impact the final volume of an individual note, and often no explicit visual link between them. Similarly, the routing of audio signals through a DAW is usually not visually illustrated, but dependent on the values of (potentially hidden) drop menus. Some DAWs have attempted to address this: *Tracktion* enforces a left-to-right signal flow where a track's inputs, inserts, send effects, outputs, and other processes are aligned in sequence (in a row) within the tracks of its arrange screen; whereas *Reason* takes a skeuromorphic approach using visual metaphor to the studio, enabling users to inspect and manipulate the wired connections on the back of virtual hardware devices.

4.4 Hard Mental Operations

"When writing music, are there difficult things to work out in your head?"

This dimension assesses the cognitive load placed on users. While this is one of the few dimensions with a prescribed polarity (to be avoided in a user experience), musical immersion, motivation, and enjoyment is predicated on providing a rewarding challenge commensurate with ability, such that music may be one of the few fields where this dimension is to some degree desirable.

[Related dimensions: *consistency, hidden dependencies*]

SCORE: Formal music notation carries a high literacy threshold, making the score inaccessible to untrained or novice users. Moreover, aspects of the score also require experienced musicians to solve problems in their head, such as applying key signature, deducing fingering, etc. (see *hidden dependencies)*. By not notating these elements, scores can be more concise, as well as less prescriptive for interpretation by performers. Interaction with music notation also draws heavily on rote learning and deliberate practice to develop unconscious skills and reflexive playing techniques that would be less efficiently or fluidly performed if mediated through notation.

MAX/MSP: While arithmetic and computation tasks can be offloaded to the Max, some aspects of patch behaviour must be carefully approached. Execution order and causal relationships are not visually explicit in a Max patch; users must comprehend the flow of processing to understand the behaviour of their program. Similarly, the lack of a timeline makes time a more abstract concept, making less process-oriented styles of music harder to create and conceive, unless mentally simulated by the user.

DAW: Pro audio software is created with design principles favouring usability and ease-of-use: to be accessible to musicians and non-computer audiences. The various sub-devices in a DAW allow users to edit data through a UI style suiting their background (score, mixer, piano roll, MIDI instrument, etc.). However, because complexity is hidden from the user, there is some risk of such systems becoming less flexible and more opaque; made of black boxes supporting established artistic workflows (see *closeness of mapping*, *premature commitment*). The apparent disjunction between usability and the virtuosity musicians embrace in other aspects of their practice (performance, score literacy, composition) may suggest that such users would accept the cost of developing skill, when more flexible approaches to musical creativity is the reward, and thus design heuristics based on virtuosity rather than usability may be more apt. [9,14]

4.5 Progressive Evaluation (Audibility / Liveness)

"How easy is it to stop and check your progress during editing?"

This dimension details how easy it is to gain domain feedback on the notated work during editing. How complete must the work be before it can be executed?

In music, this is defined by what facilities are available to audition the sound or performance of the music. 'Liveness', another concept adapted from programming, defines the immediacy and richness of domain feedback available in the manipulation of notation [15,16], and is a key factor in the user's feeling of immersion in the creative process and domains such as music. [11,17]

[Related dimensions: *provisionality*, *premature commitment*, *hard mental operations*]

SCORE: Musical feedback is available through manually playing, sight-reading, and auditioning the notated music using an instrument. Material can be evaluated through performance (possibly requiring transposition) on various instruments – commonly, a piano. Crucially, the piece needn't be complete (or 'correct') to audition individual phrases or parts. Moreover, lo-fidelity musical scores (sketches) allow unfinished, informal notation of ideas that can still be interpreted by the composer. There may, however, be a disparity between notated forms and a musical performance, where performers may add their own interpretations to the notes on the page (individual prosody, articulation, *rubato*, etc.). Simulation of material on a different instrument also relies on the composer's knowledge of the target instrument and related technique – e.g. a piano may be more or less musically flexible, and offer a different timbre to the target instrument.

MAX/MSP: The environment allows patches to be run at any time, though they must be coherent and syntactically correct to evaluate the sound design or music. Good programming practice encourages a modular approach that allows sub-components and simpler configurations to be tested individually, early in development, though its function and output might be abstracted from the final sonic intent of the code.

DAW: The timeline, mixer, playback and track controls (e.g. mute, solo) enable the user flexible control of listening back to musical data and auditioning individual edits. A piece can be auditioned as it is built up, track-by-track, bar-by-bar, or note-by-note, and there is no requirement that the 'solution' or notated form be musically correct or coherent to be heard. The rigid UI prevents the entry of non-sensical data, and informal or ambiguous directions (see *secondary notation*) cannot be auditioned. For digitally-produced music, the sound output offers an exact representation of the music notated in the UI.

Sequencers designed to accelerate the edit-audition cycle enable a higher level of liveness in the user experience of notation-mediated digital music systems, as evidenced by loop- and pattern-based sequencer software such as *Ableton Live* and most soundtrackers [11], which focus editing on shorter excerpts of music, shortening the feedback cycle. This contrasts the unbroken linear timelines of traditional sequencers, where (beyond the literally live experience of recording), interaction styles for editing and arranging parts offer lower liveness.

4.6 Conciseness / Diffuseness

"How concise is the notation? What is the balance between detail and overview?"

This dimension assesses the use of space in a notation. Both pages and screens have limited space, and both the *visibility* and *viscosity* of a notation suffer when data escapes from focus. Legibility may also suffer if the notation is simply shrunk or packed tightly, such that conciseness must normally be balanced with careful use of *abstraction* mechanisms. In music, composers need to be able to access every detail of a piece, but also able to get a sense of the 'big picture'. [3,12]

[Related dimensions: *visibility*, *juxtaposability*, *hidden dependencies*, *abstraction management*, *consistency*]

SCORE: The score has evolved to provide a concise representation of music. Unlike digital notations, no abstractions or sub-views are available to hide detail; all elements are always visible, requiring economical use of space. Time is represented using a pseudo-linear scale, where notes are positioned within the bar to reflect relative position in a piece, but bar sizes are compressed such that sparse phrases consume less space. Musical time and page position are further decoupled through the symbolic representation of note duration, such that slow passages (e.g. of semi-breves) do not consume excessive space, but fast passages (e.g. of demi-semi quavers) are expanded to show the detail more clearly. This symbolic encoding of time, however, does lower the *closeness of mapping*, increasing the onus on literacy (*virtuosity*).

MAX/MSP: The layout and density of a Max patch is flexible, though readability suffers when objects are densely packed together or connecting patchcords obscure each other. When complex patches grow outside the confines of a window, *visibility* suffers and mouse-based interaction can be cumbersome. *Abstraction* mechanisms such as sub-patching are critical in managing complex systems and avoiding sprawling patches, but trade *diffuseness* over screen space for *diffuseness* over separate, possibly hidden windows.

DAW: The variety of notations and views in DAWs offer a varied level of *conciseness*. The arrange view sacrifices *visibility* of data to accommodate a broader overview of a piece in the UI. Part editors, like the score and piano roll interfaces, offer more detail (in a manner similar to the traditional score), but only partial views of the entire work. More generally, the lack of a comprehensive principle notation or interface means that information is diffused over different views within the program. Many DAWs do little to optimise window management, navigation, or searching, compounding interaction issues.

4.7 Provisionality

"Is it possible to sketch things out and play with ideas without being too precise about the exact result?"

This dimension assesses how easy it is to experiment with new ideas through the notation or UI, and how fully formed those ideas must be. Accordingly, it is a critical factor in a musical system's support for sketching, ideation, and exploratory creativity. [3,5,11,16] In digital systems, an 'undo' facility significantly contributes to *provisionality*, allowing inputs and edits ('what if' scenarios) to be trialled and reversed, reducing *premature commitment* to a particular approach [1] – reducing the risk of trying new ideas. The dimension is closely related to *viscosity* and *progressive evaluation*, where the ease and flexibility of editing and auditioning similarly facilitates exploring new ideas. *Secondary notation* also offers the opportunity to make incomplete or informal remarks, but in a non-executable form that can't be auditioned.

[Related dimensions: *premature commitment*, *viscosity, progressive evaluation, secondary notation*]

SCORE: In a musical sketch, the affordances of paper and pencil support a powerful and flexible medium for capturing part-formed ideas. [5,18] Pencil can be easily and quickly erased, facilitating experimentation and ideation. By contrast, the formality of the typeset, printed ink manuscript is less flexible and more permanent, used only to finalise a composition for archiving or communication (e.g. to performers). These two instances of score notation compliment each other in an established ecosystem that facilitates both composition (creativity) and performance (production) (cf. [19]).

MAX/MSP: The visual drag-&-drop, interactive debugging environment of Max facilitates its use as a rapid prototyping tool, useful in the exploratory design of new audio processing tools and synthesis techniques [13] – though some more involved musical constructs or expressions can be harder to develop or articulate quickly, reducing

provisionality and ideation. Conversely, as a prototyping tool, Max's focus on experimentation and early stage creativity comes at the expense of subsequent stages of the creative process ("productivity" [19]): finalisation, refinement, and continued development of designs (e.g. for consumption by end-users, non-programmers, and other musicians) is normally conducted using other development tools (e.g. C/C++).

DAW: Like other authoring tools, DAWs offer multiple ways of quickly adding, editing and deleting elements in the document (i.e. musical piece). Moreover, the presence of 'undo' functionality makes it easy to backtrack actions, reducing the risk of experimenting with new ideas, encouraging ideation [1]. The primary mode of input – digital audio or MIDI performance capture – in combination with practically unlimited storage (length, tracks, etc.) represents an improvement in provisionality over historic recording techniques (e.g. tape). Users can also address issues in live recordings using advanced overdub tools, without recourse to re-recording entire performances. Offline editing, through part editors like score or piano roll, allows experimentation with different ideas, though such interfaces are not always optimised for the rapid entry, editing and auditioning of new material to support creative exploration of musical ideas. [11]

4.8 Secondary Notation

"How easy is it to make informal notes to capture ideas outside the formal rules of the notation?"

This dimension evaluates a system's provision for recording information beyond the formal constraints of the notation. As informal notation, data is typically not executable by computer or performer, and may only be related to the encapsulated piece / performance indirectly. Decoupled from the formal rules of expression in the notation, secondary notations often allow users to make freeform notes to support their edit process, though flexibly designed facilities may be used for a variety of purposes – including evaluation (peer feedback), working out problems, highlighting relationships in the notation, sketching rough high-level structure, aesthetic decoration, to-do lists, incomplete ideas, etc. In programming, code commenting is used to annotate code with useful labels, instructions, explanations, ASCII art, etc., helping to make the code more readable, but also as a form of communication between coders. As such, *secondary notations* should be designed to be as flexible as possible, to allow users to appropriate them for their own needs.

[Related dimensions: *provisionality*, *hard mental operations*, *hidden dependencies, role expressiveness*]

SCORE: The expressive freedom of pencil and pen marks on paper allow musical scores to be annotated with any additional information, such as personal notes, decoration, as well as irregular performance instructions. Formal notation places more constraints on what is representable, though written language can be freely used in performance directions. The human interpretation of scores enables a further degree of flexibility in applying and developing new terminology, such that informal notes that break from standard semantics may still be executable. Performers can also add their own notes to manuscripts to guide their own interpretation of the piece.

MAX/MSP: Like other programming tools, code comments are an important part of developing and maintaining Max patches. Max's visual medium supports annotations using free text (comment boxes), shaded areas, and imported images (bitmaps), used to explain workings, usage, or as decoration. However, drawing facilities are very limited in comparison to pencil and paper, and even digital graphics, with no provision for freehand sketching or drawing lines, arrows, or shapes (other than rectangles). Given the proven benefits of such affordances in other music notations (e.g. the musical sketch and score [5]), their omission in such a visual medium is surprising.

DAW: Despite the proliferation of different notational styles in DAWs, each UI is rigidly structured to fulfil a defined purpose and offer specific tools for editing the underlying data. Limited provisions for annotations are provided by way of labelling and colour-coding parts and tracks, and free text is often supported for meta-data, but few mechanisms are provided for flexibly annotating the music in any of the sub-notations or views, beyond those forms formally recognised by the program.

4.9 Consistency

"Where aspects of the notation mean similar things, is the similarity clear in the way they appear?"

This dimension defines how coherent and consistent the methods of representing elements in a notation or UI are. Consistency facilitates the learning of a system (see *virtuosity*), as users used to a style of presentation can apply knowledge learnt in one area to understand others. However, consistency may also be sacrificed to improve *conciseness*, *visibility*, or *role expressiveness*.

[Related dimensions: *conciseness*, *visibility*, *virtuosity*, *role expressiveness*, *abstraction management*]

SCORE: In sheet music, notated passages that are similar musically share similar visual cues, e.g. melodic contour, repeated passages, etc. Formal rules applied consistently likewise ensure recognisable and learnable conventions. However, compromises are made for *conciseness*, and to optimise the presentation of common expressions, at the expense of readability in less canonical works. For example, the symbolic representation of note rhythm in a passage completely alters if offset within the bar (e.g. moved by a quaver). Similarly, the representation of pitch depends on key; an identical phrase requires accidentals following a change of key signature. Both scenarios present limited issues in common practice music, but the inconsistency makes the notation harder to learn and understand, and the difficulty of using it outside its intended purpose encourages conformity, discouraging experimentation and creativity. Moreover, in digital use (notably MIDI sequencers), such inflexibility markedly reduces the usability of score notation, where systems are unable to unpick the expressive prosody in a captured live performance to display a coherent visual score.

MAX/MSP: By design, programming languages offer diverse paths to produce similar code functionality. Textual languages are based on rigid, carefully designed formal grammars that ensure basic low-level consistency among programming primitives, also enabling many syntactic errors to be identified during compilation. Max's collection of objects is less formally designed and, as the accumulation of several developer's efforts (and coding styles), less consistent. Inconsistencies exist in many areas, including object-naming schemes, inlet and outlet conventions, processing behaviour, message handling, audio quality (and level), and configuration methods. These nuances produce unanticipated code behaviour that increases the learning curve for novices. Objects behave like self-contained programs or plugins; black boxes that have to be mastered individually.

DAW: The added flexibility in the visualisation of data, in the various views afforded by DAWs inevitably comes at the cost of consistency of representation throughout the program. For example, volume might variously be represented as a MIDI value (0-127), automation value (0-1), gain (dBFS, e.g. -96dB to 0dB for 16-bit audio), or using graphics (colour, bar size, rotary knob angle). The trend towards skeuromorphic visual metaphors to electronic studio equipment similarly encourages inconsistencies in representation, drawing on the conventions of previously separate, loosely connected hardware devices. Moreover, while the advent of third-party plugins brings great advantages and creative flexibility, inconsistencies in control, representation, terminology, and interaction create usability issues and a fragmented user experience that is difficult to integrate with the host application.

4.10 Viscosity

"Is it easy to go back and make changes to the music?"

This dimension defines how easy it is to edit or change a notation, once data has been entered. A common example is *knock-on viscosity*, where making a simple edit to the notation requires further edits to restore data integrity. *High viscosity* prevents or discourages alterations, forcing users to work in a prescribed, pre-planned order (see *premature commitment*); *low viscosity* simplifies and encourages making changes, reducing the investment associated with trialling new ideas (see *provisionality*). Being able to easily explore and revisit ideas (ideation) is a key factor in supporting creativity [6,19], requiring creative systems engender *low viscosity*.

[Related dimensions: *provisionality*, *premature commitment*, *progressive evaluation*]

SCORE: The provisionality of pencil marks simplifies the alteration, erasure and overwriting of notes and passages in a musical sketch. If more drastic changes are required, the reduced emphasis on neatness and third-party readability allows the composer to strike out larger sections. Inserting new material is harder, but composers can similarly sketch the inserted passage where there is space (or on a new sheet) and note the insertion. Final manuscripts are intentionally more rigid, but performers can still annotate their copy with alternative instructions.

MAX/MSP: Simple changes to values and local objects are straightforward in Max. However, as patches grow and the interconnectedness of objects increases, Max suffers from *knock-on viscosity* [1], where one change requires further edits to restore patch integrity. For example, deleting, editing, or replacing objects removes all cords to other objects. Increased viscosity is a common trade-off in tools designed to avoid hidden dependencies, often seen in data-flow and visual programming languages like Max. As a graphical notation, changes to a patch often require the layout of a patch to be reworked to make room for object insertions, and to maintain readability. In text-based coding environments, such housekeeping is simplified by the inherent serialisation of code, but in VPLs like Max, leads to increased viscosity.

DAW: As with *provisionality*, the level of viscosity in DAW interaction varies between the interfaces and interaction modes of the sequencer. By itself, a tape recorder metaphor of recording a live performance makes it easy to erase and re-record a take, but harder to edit recorded data. Audio data can be processed (e.g. EQ, mixing, FX, splicing, etc.), but musical content (e.g. individual notes or harmonies) is not easily addressed or manipulated. Recorded MIDI data is easier to edit, though visual representations (e.g. score – see *consistency*) and interaction styles can be cumbersome and unwieldy for anything but simple edits. [11,12]

4.11 Role Expressiveness

"Is it easy to see what each part is for, in the overall format of the notation?"

This dimension evaluates how well the role or purpose of individual elements is represented in the overall scheme of the notation or UI. Different elements may not be visually indistinct, or their function may be unclear in the way they are presented. For example, English language keywords in a menu or programming language can be used to express their function, whereas cryptic symbols or icons may need to be learnt. Alternatively, the visual design of GUI may impose a consistent aesthetic or layout that fails to capture the diverse functionality encapsulated, or the relationship to other elements of the UI (see *hidden dependencies* and *closeness of mapping*).

[Related dimensions: *visibility*, *hidden dependencies*, *closeness of mapping*]

SCORE: While some aspects of the score may be inferred by listening to the music (such as a general sense of pitch and rhythm), most involve learning syntax and rote practice. Similarly, while some signs offer more expressive visual cues to role (crescendo and diminuendo hairpins; tremolo marks), many do not – clefs, accidentals, key signatures, note shapes, ornaments, and foreign terms symbolise complex musical concepts that require tuition. Once learnt, however, the symbols facilitate the rapid comprehension of notated music – e.g. different note shapes and beaming conventions provide clear differentiation between different note lengths. However, recent approaches to contemporary scores tend to exploit more expressive geometric forms, rather than new symbol sets.

MAX/MSP: The role of some specialised objects, notably user controls, is clear from their representation in Max. However, beyond caption and inlet/outlet configuration, Max offers little visual distinction in the representation of most coding objects, which appear as text boxes. Patchcords help to define the context and role of connected objects, and visual distinction is made between audio and message types (though not between int, float, list, or bang subtypes) – but, despite the unidirectional flow of data, flow direction is not depicted (e.g. using arrows).

DAW: Many aspects of DAW UIs rely on a degree of familiarity with studio equipment and musical practice. However, the graphical user interfaces of most packages make prominent use of expressive icons and detailed graphics to indicate the function of controls. Visual metaphor and skeuromorphisms are commonly used to relate program controls to familiar concepts. Image schema and direct manipulation principles are similarly applied to highlight interaction affordances, in the context of both music and generic computer interaction styles.

4.12 Premature Commitment

"Do edits have to be performed in a prescribed order, requiring you to plan or think ahead?"

This dimension defines how flexible a notation is with respect to workflow, and the process of developing ideas. Notations or system features that must be prepared or configured before use entail *premature commitment*. Notations with *high viscosity*, where it is hard to backtrack, also entail forward planning and commitment. In programming, an illustrative example is the need to declare variables and allocate memory before coding actual functionality (cf. C/C++ vs. BASIC).

[Related dimensions: *provisionality*, *viscosity*]

SCORE: A degree of viscosity in altering page layout means that some forward thinking is required to commit musical ideas to the page, which generally proceeds left-to-right, bar-by-bar. However, separate pages allow sections and movements to be developed non-linearly, and the *provisionality* of the musical sketch allows some flexibility with development of musical phrases and bars. Multiple approaches to composition are possible: horizontal (part-by-part), vertical (all parts at once, start to finish), bottom up (bar-by-bar), top down (musical form). Historically, the literacy and musical experience of composers meant that musical material was often part-formed before being committed to the page – either mentally, or through experimentation with instruments.

MAX/MSP: As a prototyping tool, Max supports experimentation with partially-formed design concepts. Often, however, audio processes will be designed with a plan or general architecture in mind; in Max, forethought with respect to *abstraction* (sub-patching) or layout benefits development, though housekeeping may be needed retrospectively (see also *viscosity*). The open-ended canvas allows patches to be flexibly extended in any direction, and a modular approach to programming allows piecewise development of complex systems.

DAW: Musical parts and audio segments can be easily inserted, moved, and copied in the arrange window, though complex phrases with overlapping tracks and automation can be difficult to split and re-sequence. Furthermore, the unified linear timeline and tape recorder metaphor encourages a linear workflow. [12] In modelling studio workflows, DAWs can be seen as transcription tools, rather than environments for exploratory creativity, where artists only turn to the recording process once a work has already taken form. [3,20] By contrast, pattern- and loop-based sequencers (*Live*, *FL Studio*, tracker-style sequencers) offer a flexible non-linear approach to developing and sequencing musical forms, facilitating digitally-supported creativity and flow.

4.13 Error Proneness

"How easy is it to make annoying mistakes?"

This dimension identifies whether the design of a UI or notation makes the user more or less likely to make errors or mistakes. These can manifest as accidental interactions with a program, or incoherent, unexpected musical results arising from vagueness or ambiguity in the notation (see *role expressiveness*). In programming, for example, a notation is error prone if its function markedly alters upon the addition/omission/position of a single character. Errors are broadly undesirable, but can lead to creative, serendipitous formulations in artistic expression. [21]

[Related dimensions: *hidden dep.*, *role expressivness*]

SCORE: In scoring, the literacy threshold means mistakes are more likely during early stages of learning. Aspects of *consistency* and *hidden dependencies* contribute to a user's propensity to make errors. However, like language, fluency with the notation reduces mistakes. Sketching, as a private medium for the composer, is also tolerant of errors; they are free to misuse or invent notation, which remains meaningful to them personally. When scores are used for communication, mistakes have consequences; but the impact on early creative process is minimal.

MAX/MSP: As a formal language, it is easy to make mistakes in Max, through the creation of ill-formed code. However, aspects of the Max UI make it more prone to errors in certain situations. As a graphical UI, mouse interaction is cumbersome, and Max attempts to avoid diffuseness with compact objects, such that selecting and connecting inlets or outlets using patchcords is awkward. As with the score, *consistency* issues and *hidden dependencies* also invite mistakes relating to coding semantics.

DAW: Recording performances in real-time heightens the likelihood of input errors, though facilities exist to correct or overdub recorded data, and the occasional mistake is often acceptable for the improved flow (musical and creative) afforded by live interaction with an instrument. As in Max, DAWs invite mistakes through dependence on the mouse, where delicate pointer control is required – many edits require targeting the edge of elements (track segments, notes, etc.), and the extent of such hotspots may be small and visually indistinct. Proximate and overlapping objects can be similarly difficult to target.

4.14 Closeness of Mapping

"Does the notation match how you describe the music yourself?"

This dimension assesses how well a notation's representation of the domain aligns with the user's own mental model. In a UI, this also applies to how closely workflows and interaction styles fit a user's working methods. Music perception and aesthetics are quintessentially subjective, making it difficult to encode a universally or intuitively acceptable formalisation, so notations and systems are built around common cultural practices. This can constrain the creative expression or affordances of a notation. To mitigate this, *abstraction mechanisms* may enable users to appropriate, redefine, and extend systems.

[Related dimensions: *role expressiveness*, *abstraction management*, *virtuosity*]

SCORE: While the score is not an intuitive representation that untrained users might themselves conceive or comprehend, it remains a widespread and established technique for notation in Western music. At the same time, the canonical score systematises music in a way that makes assumptions about the musical practices and aesthetics of its users, such that modern composers identify the format as a constraint on their personal expression and creativity. However, the flexibility offered by individual sketching techniques allows composers to invent and appropriate notation techniques for their own personal use.

MAX/MSP: The data-flow model of Max maps closely to diagrammatic forms used widely in signal processing, with a shared legacy in electronics and circuit diagrams. The inherent role of electronics in the studio, and representation of audio as voltage, also make this an analogy that musicians and producers can relate to. The functional and visual resemblance to generic flow charts further helps to make the programming environment accessible to non-technical users. However, for musical applications (rather than audio processing) such as arrangement and composition, the abstract representation of time offers a poor closeness of mapping to familiar representations of music. Similarly, for traditional programmers used to imperative programming (ordered sequences of instructions), scripting program behaviour over time is difficult.

DAW: For its intended audience of musicians and sound engineers, traditional sequencers and DAWs provide a strong closeness of mapping, using visual metaphors and interaction paradigms based on studio processes and audio hardware, to allow skills transfer. Notably, MIDI and audio recording tools focus interaction on musical instruments. However, in recent years, more computer-oriented musicians, with greater technical literacy, have begun to embrace tools that rely less on analogies to the recording studio and focus on the affordances of digital and computer music technologies – as offered by *Ableton Live* and *FL Studio*. Ultimately, engagement with music, as a personal experience, should be based on articulations of the music domain crafted by the user themselves, which the rising level of computer literacy might enable, as end-users increasingly engage with programming.

4.15 Abstraction Management

"How can the notation be customised, adapted, or used beyond its intended use?"

This dimension defines what facilities a system offers for appropriating, repurposing, or extending a notation or UI. All notations present an abstract model of a domain (e.g. music, software), providing a set of fixed abstractions representing basic objects (e.g. notes, parts) and properties (e.g. pitch, time, etc.) that enable the articulation of solutions (e.g. a piece). The creative possibilities are defined by what encapsulations of objects are possible and how easy they are to extend. Notations defined for a specific purpose fix the possible abstractions and ways of working. However, the opportunity to define new abstractions (e.g. in terms of existing ones) offers the user a way to develop their own toolset and facilitates the building of more complex solutions (e.g. by abstracting low-level detail), and helps to personalise and raise the creative ceiling of a system. [6] In programming, examples include defining custom functions and abstract data types (objects). In end-user computing, systems may support automation, macros, or plugins to enable users to add new functionality. Simpler abstraction mechanisms such as grouping and naming elements are also possible.

[Related dimensions: *visibility*, *closeness of mapping*, *role expressiveness*, *conciseness/diffuseness*, *consistency*]

SCORE: In sketching the piece during the creative process, composers are able to appropriate or invent new terminology of notation technique to describe music more *concisely* (composer shorthand) or to encapsulate unconventional musical devices and practices – only when it is transcribed for communication to a performer (or computer) must it conform to established notational forms.

The canonical score format is more limited; designed around common practices and conventions in formal music, but offers some support for grouping mechanisms (e.g. brackets, phrasing) and abstraction (e.g. custom ornaments). However, a composer can use the preface to a score to introduce original notation techniques and syntax, to instruct the performer's interpretation.

MAX/MSP: As a programming language, abstraction is a key technique for building and maintaining complex solutions. Max offers several provisions for abstracting and extending code: *sub-patches* allow embedded and external patches to be nested inside other patches, represented as new objects (linked using inlets and outlets); *externals* use a plugin model to allow new objects to be coded in C/C++ with defined inputs and outputs; and *presentation mode* allows patches to be rearrange, simplified and selectively exposed in end user-oriented UIs.

DAW: Sequencers and DAWs are designed to support specific working styles in music / production scenarios. Part editors support low-level editing of notes and other musical events. In other screens, higher-level abstractions are used to structure music (tracks, parts, etc.), with some provision for grouping and organising objects (e.g. group channels, folders, track segments). Most packages also support audio plugin formats that extend FX processing

and synthesis options. However, few sequencers support more flexible abstraction mechanisms to facilitate interaction with notation, such as macros, scripting, or automation. Exceptions to this include *Live*, which can be integrated with Max, CAL Script in *Cakewalk SONAR*, and *Sibelius* plugins. In the tracker domain, *Manhattan* [23] offers end-user programming for music using an extended implementation of spreadsheet-style formulae.

4.16 Virtuosity / Learnability

"How easy is it to master the notation? Where is the respective threshold for novices and ceiling for experts?"

This dimension assesses the learnability of the notation, and whether it engenders a scalable learning curve – that is, a "low threshold" for practical use by beginners, a "high ceiling" for flexible expression by experts, affording "many paths" by which users can express themselves. In addition to supporting multiple levels of expertise and creativity, virtuosity should be understood in terms of the balance of challenge and ability experienced by the user. A slight challenge, relative to their ability, intrinsically motivates users and helps create the conditions for flow. [3,9,11,22] Too much challenge and users become anxious; too little and they become bored. The best model for systems are based around "simple primitives" (building blocks) that can be easily understood by beginners, but flexibly combined to form more complex *abstractions* and functionality. [6]

[Related dimensions: *consistency*, *prog. evaluation*, *role expressiveness*, *closeness of mapping*, *error proneness*]

SCORE: The score has a steep learning curve and beginners require formal tuition and practice to master it. Novices can be discouraged from learning music by the literacy entry threshold. [3] The complexity of the notation reflects its relatively high ceiling and capacity to flexibly encapsulate a wide variety of musical styles and pieces, though contemporary and electronic composers can find traditional, formal syntax limiting. [2,3,12]

MAX/MSP: While programming languages often present a high threshold for novices, Max is explicitly designed for musicians, and uses a visual programming model to appeal to non-coders. Tutorials present beginners with simple patches that produce useful results, enabling a working knowledge to develop quickly. Innovative interactive in-program documentation and a strong user community supports both learners and practitioners. There are aspects of the environment that also impede learning (see *consistency*, *error proneness* and *hidden dependencies*). The creative ceiling for developing audio and music systems in Max is high, further supported by *abstraction* mechanisms – though audio programmers and more music-oriented users may graduate to other tools (e.g. C/C++, *OpenMusic*, *SuperCollider*).

DAW: Music and audio production packages are designed to provide a low threshold for musicians and those familiar with studios. The use of visual metaphor and direct manipulation principles allows knowledge transfer from these other practices [4], though users without such backgrounds may struggle. Packages provide a wide array of tools and features for a variety of purposes, though few users will have need of all features. The ceiling for musical creativity is relatively high, within the confines of conventional practices, though UIs are often optimised for specific workflows and techniques, and users are largely dependent on software developers to provide new opportunities for expression. Unlike the traditional score, and programming languages (like Max), users efforts to master authoring packages can be frustrated by a lack of continuity between versions.

5. PRACTICAL METHODOLOGIES

This section briefly surveys existing applications of the *Cognitive Dimensions of Notations* in musical contexts, highlighting both qualitative and quantitative methods for analysing notations and interaction.

Blackwell (with others [7-11,16,20,24]) has used cognitive dimensions to highlight aspects of musical interaction in several settings, including music typesetting software [10,20], programming languages [16,24], and digital tools for composition (e.g. sequencers, trackers) [8-11]. In such treatments, the framework provides a language for discussing the affordances of notation, but has also lead to the development of tools to elicit feedback from end-users, such as questionnaires that probe dimensions in user-friendly, accessible language. [10] McLean's work on music and art programming languages similarly applies and develops the framework for analysis of new music coding notations and interfaces. [21]

Nash [3,9,11] extended previous qualitative analysis techniques to develop a quantitative approach to evaluating notations. Using a *Likert* scale, each dimension is formulated as a statement that users can agree or disagree with to a greater or lesser extent. The mean response from a large sample of users can then be used to plot a dimensional profile of the notation under evaluation. Figure 1 shows profiles for a survey of various music sequencer tools (*n=245*), not only highlighting relative strengths and weakness with respect to properties of each UI, but also revealing a general profile for music systems, where the trend may indicate the desired polarity of each cognitive dimension in music interaction. Moreover, the approach was combined with psychometric-style surveys of the experience of creative flow [22], using a battery of questions to also measure users' subjective experience of

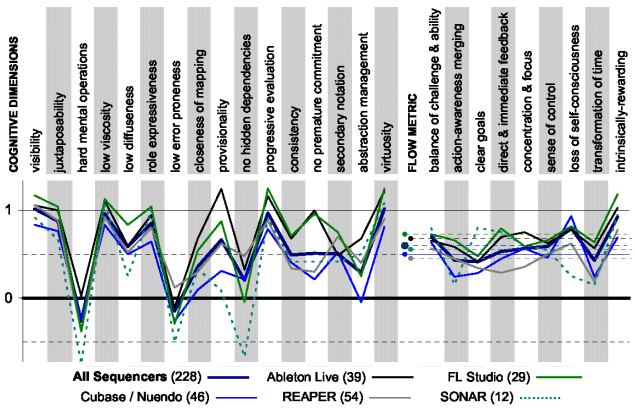

Figure 1 Cognitive dimension and flow profiles of music tools, based on quantitative user testing (see [3,11]).

nine components of flow. Using cross-correlation and multiple-regression analysis, the results for individual flow components and dimensions of the notation were used to identify the key properties of notations facilitating flow, findings of which can be used to guide the design of immersive or embodied interaction systems. The study [3,11] suggests that key dimensions in the support of flow were *visibility* (visual feedback), *progressive evaluation* (audio feedback) and *consistency* (support for learning and sense of control) – as well as *virtuosity* (balance of skill and ability), *abstraction management* (high creative ceiling), *viscosity* (ease of editing), *premature commitment* (freedom of action) and *role expressiveness* (support for learning). The findings were used to propose a set of design heuristics for music systems based around the concept of virtuosity, rather than usability (see [3,9]).

6. CONCLUSIONS

This paper has presented a musical reworking of the *Cognitive Dimensions of Notations* usability framework, and suggested methods and tools for using it to analyse music notations, interfaces, and systems. Several applications have been identified that use the framework to provide insight into the human factors of notation-mediated musical systems, including creativity, virtuosity and flow.

Future work will focus on further use and development of the framework, including its application to other music interaction scenarios and systems, the evaluation of new dimensions, and research of other dimensional profiles in other music interactions. The growing intersection of music and programming practice is also likely to reveal other parallels between these creative domains that can further inform both theory and practice.

Acknowledgments

The author wishes to thank all those who supported and contributed to this research, especially Alan Blackwell, Ian Cross, Darren Edge, Sam Aaron, the Cambridge *Centre for Music & Science (CMS)*, and many other researchers exploring and developing the CD framework in other domains. This research was funded by the Harold Hyam Wingate Foundation and UWE (Bristol).

7. REFERENCES

[1] T. R. G. Green, and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework," *Journal of Visual Languages & Computing*, 7, 1996, pp.131-74.

[2] G. Read, *Music notation: a manual of modern practice,* 1979, Taplinger Publishing Company.

[3] C. Nash, *Supporting Virtuosity and Flow in Computer Music,* PhD, University of Cambridge, 2011.

[4] M. Duignan, *Computer mediated music production: A study of abstraction and activity,* PhD, Victoria University of Wellington, 2008.

[5] M. Graf, *From Beethoven to Shostakovich,* New York: Philosophical Library, New York, 1947.

[6] M. Resnick et al., "Design principles for tools to support creative thinking," 2005, *NSF Workshop of Creative Support Tools*, pp. 25-36.

[7] A. Blackwell, "Dealing with new cognitive dimensions", *Workshop on Cognitive Dimensions*, Uni. of Hertfordshire, 2000.

[8] C. Nash, and A. Blackwell, "Tracking virtuosity and flow in computer music," 2011, *Proceedings of ICMC 2011*, pp. 572-582.

[9] C. Nash, and A. Blackwell, "Flow of creative interaction with digital music notations," in *The Oxford Handbook of Interactive Audio*, 2014, OUP.

[10] A. Blackwell, and T.R.G. Green, "A cognitive dimensions questionnaire optimized for users," 2000, *Proc. of PPIG 2000*, pp. 137-152.

[11] C. Nash, and A. Blackwell, "Liveness and flow in notation use," 2012, *Proc. of NIME 2012*, pp. 28-33.

[12] D. Collins, "A synthesis process model of creative thinking in music composition," *Psychology of Music*, 33, 2005, pp. 192-216.

[13] P. Desain et al., "Putting Max in perspective," *CMJ*, 17(2), 1993, pp. 3-11.

[14] J. A. Paradiso, and S. O'Modhrain, "Current Trends in Electronic Music Interfaces. Guest Editors' Introduction," *JNMR*, 32(4), 2003, pp. 345-349.

[15] S. L. Tanimoto, "VIVA: A visual language for image processing," *Journal of Visual Languages & Computing*, 1(2), 1990, Elsevier, pp. 127-139.

[16] L. Church, C. Nash, and A. F. Blackwell, "Liveness in notation use: From music to programming," *22*, 2010, *Proceedings of PPIG 2010*, pp. 2-11.

[17] M. Leman, *Embodied music cognition and mediation technology,* MIT Press, Camb., MA, 2008.

[18] A. J. Sellen, and R. H. R. Harper, *The myth of the paperless office,* MIT Press, 2002.

[19] T. Amabile, "The social psychology of creativity: A componential conceptualization," *Journal of personality and social psychology*, 45(2), 1983, pp. 357-76.

[20] A. F. Blackwell, T. R. G. Green, and D. J. E. Nunn, "Cognitive Dimensions and musical notation systems," 2000, *ICMC 2000: Workshop on Notation and Music Information Retrieval in the Comp. Age*.

[21] A. McLean, *Artist-programmers and programming languages for the arts,* PhD, Goldsmiths, 2011.

[22] M. Csikszentmihalyi, *Creativity: Flow and the Psychology of Discovery and Invention,* HarperCollins, New York, 1997.

[23] C. Nash, "Manhattan: End-User Programming for Music," 2014, *Proc. of NIME 2014*, pp. 28-33.

[24] A. Blackwell, and N. Collins, "The programming language as a musical instrument," 2005, *Proceedings of PPIG05*, pp. 120-130.