Live Coding Practice

Click Nilson
University of Sussex
Department of Informatics
Falmer, Brighton, BN1 9QH
+44 (0)1273 877837
N.Collins@sussex.ac.uk

ABSTRACT

Live coding is almost the antithesis of immediate physical musicianship, and yet, has attracted the attentions of a number of computer-literate musicians, as well as the music-savvy programmers that might be more expected. It is within the context of live coding that I seek to explore the question of practising a contemporary digital musical instrument, which is often raised as an aside but more rarely carried out in research (though see [12]). At what stage of expertise are the members of the live coding movement, and what practice regimes might help them to find their true potential?

Keywords

Practice, practising, live coding

1. INTRODUCTION

Rather than the composer-pianists of the 19th century, the 21st century is seeing a rise of composer-programmers, who can explore electronic music through its natural tool, the computer, even in live performance situations. With the computer such a ubiquitous feature of modern life, art which confronts its mechanisms and processes has great contemporary relevance. In this respect, live coding, the art of programming a computer under concert conditions, can enable a novel engagement with the notions of algorithm, and the mapping from code to musical resultant [4, 20]. Since the computer is the instrument practised most readily by computer musicians, why not let this programming practice become the basis of new performance practice side stepping the heritage of gestural acoustic instruments? Live coding provides an alternative opportunity to keep the human being involved in live electronic music.

Live coding has attracted a certain amount of press (see [1,3], and the hornet's nest of slashdot commentaries on [9]), some of it bad, and certainly raises controversies, from those of physicality and immediacy, to obscurantism and intellectualism. Certain aspects of the physical question related to practice and automaticity will be tackled later in this paper, but my primary focus will avoid such debates (and I would argue that a New Interface for Musical Expression does not have to rely on an overtly physical central substruct to be expressive in human music making).

Nime'07, New York.

I wish to discuss the practice of live coding, not only as an area of investigation and activity, but also in the sense of actually practising to improve ability. I shall describe the results of a month long experiment where live coding was undertaken every day in an attempt to cast live coding as akin to more familiar forms of musical practice.

Exercises will be discussed that may be found efficacious to improve live coding abilities. In relation to these, literature from developmental and educational psychology (both of music and computing), and theories of expertise and skill acquisition, were consulted. The possibility of expert performance [7] in live coding is discussed.

I should state at the outset that I will be focusing primarily on one particular form of live coding in this paper, that of programming a computer on a concert platform as a performance act, often as an unaccompanied soloist. I will not treat cases of those interactive explorations, outside of realtime audience environments, enabled by interpretative programming languages and collaborative coding, interesting as these are [13] (Julian Rohrhuber: 'To me (personally, not as an observer) live coding is about conversation and experimentation only. The only way to "practise" conversation is to converse' [personal communication]). Instead, the emphasis is on that situation which demands potentially the greatest cognitive load, and hence is of great subsidiary interest as a compositional constraint, psychological test, and end-user programming study [2].

1. THE BATTLEGROUND: BURNT FROM EXPERIENCE

By way of setting the scene and highlighting the challenges implicit in live coding, I shall try to be brutally honest concerning three concerts I found myself participating in. The first two were dubbed as sporting duels, couched in such hard hitting description as '13 rounds of hard typing action till Code Out; For the World Programming Federation Fingerweight Belt (Turingsberry rules apply)'. Such public contests show some (lighthearted) affinity with Ancient Greek dialectics, the slow public mathematical competitions of Renaissance Italy (solving the cubic equation (Fior vs Tartaglia (1535), Tartaglia vs Ferrari (1548)); the meetings of piano virtuosi (Mozart vs Clementi, Beethoven vs everyone); chess (Kasparov versus Deep Blue?) and in literature, Hermann Hesse's *The Glass Bead Game.*

In the first concert, the 'Battle of the Belge' (Dorkbot Ghent, May 21st 2005), I struggled against Alex McLean, who at the time was live coding with his own Perl system. This was possibly the better of my two professional fights, and in truth we contestants were conspiring in some rounds. Whilst exploring certain algorithmic musical devices we otherwise could not have navigated, we were both disappointed in the

level of actual original coding achieved during the bout, for we fell back on certain presets and tricks too frequently.

The second concert saw me take on that fearsome inventor of ChucK, Ge Wang [19], in the 'Battle of Barcelona' (off-ICMC, 7th September 2005). Due to a build-up before the bout, and amusingly partisan SuperCollider and ChucK crowds, the audience's expectations were somewhat at odds to our own preparations. We found it hard to 'fight' in the way that had become expected, and missed many tricks that would have suited the occasion (such as fast coding of snippets for trading chops, or the 'code out' ending of deliberate coded crash). In an ICMC panel on interaction the next day, David Wessel accurately observed that there was a lot of extra musical content, and was skeptical (like many of us) about the depth of programming possible under pesky realtime constraints.

In general, my disappointment after these two events might be traced to some idealism concerning the level of musical coding achievable in front of an audience. Despite a week of daily practice ahead of the Barcelona bout, I was left feeling unresolved whether improved practice procedures might assist with the general facility of live coding.

In contrast, the third concert was an experimental theatre performance interlude in an audiovisual tour (Brisbane, 7th July 2006), which slid by with some small live coding adventure, but was not overly dramatic nor incisive. Sharing the bill, however, was *aa-cell*, an Australian duo (Andrew Brown and Andrew Sorensen) who live code using the Impromptu system based on the Scheme language [3, 17]. I found myself highly impressed with the Andrews' framework for improvisation in concert. A week or so later I was able to see them repeat the concert, with similar success, and sufficient variation, to convince me of its improvisational lucidity. Whilst the algorithmic engagement was still somewhat weak compared to my idealistic stance, the careful balance of precoded structures (particularly used at the beginning to cover initial set-up times) and on-the-fly construction within a musical form, were the best I'd seen. It turned out that this pair had been practicing together frequently.

1. PERSPECTIVES FROM PSYCHOLOGY

It is worthwhile to pause at this point and consider a general overview of pertinent psychological literature on expert performance and educational strategy.

It is recognised that expert computer programming can take as much development as musical skills. Norvig [11] satirises the 'Learn Programming Language X in 14 Nanoseconds' books and advocates a practical ten year course of obtaining programming proficiency. In a review of expert performance, Ericsson et al. [7] place programming alongside practical musicianship, and outline 'The 10-Year Rule of Necessary Preparation'. Importantly, the necessity of 'deliberate practice' is proposed as essential to obtaining an internationally recognised level of expert performance: 'the levels of performance individuals attain after years of experience alone are much lower than those of experts who have adhered to regimens of careful training and practice' [7, p.297]

Such concentrated formal training is often at odds with entertaining, informal practice sessions. This distinction is raised by Lucy Green in a study of popular musicians and their (predominantly informal) practice habits [6]. Popular musicians often characterise their learning as requiring fun, though they may also become obsessively involved. This may reflect the situation for many live coders. In contrast, John Sloboda has noted the compromise required in teenage skill development for classical musicians: 'rapid progress is often achieved only by repetitive practice far in excess of what is pleasant or intrinsically rewarding' [16, p.225].

How much deliberate practice have live coders been engaging in anyway? The standard is set by those hard working classical musicians: professional violinists will typically have done 10000 hours of practice by the age of 20 [5, p.183]; since the authors also note that music education before the age of ten should consist mainly of fun and sociable informal lessons, with intense practice commencing in teenage years, that might work out at around three hours a day consistently over 10 years. Sloboda [15] gives a figure of 7000 hours of dedicated practice by the age of 18 by the better violinists in an academy compared to half that for those rated by the teachers more likely to remain amateur.

From an email survey of prominent live coders, the number of hours of dedicated practice that seems to have been undertaken works out at around 100. Considering the number of hours spent practicing so far as an indication of grading with respect to [15, p.400, fig. 1], on this basis live coders in general might be at ABRSM Grade 2 or 3 (out of a possible 8, and not including further diploma exams).[1]

Of course, there are other sorts of relevant activity – many thousands of hours of general programming experience for some individuals, and many live coders also have extensive instrumental experience. Thus, depending on the transferability of general programming skills to live coding, some may be post Grade 8. Yet the specialization of expert skills can lead to difficulties in transference [14, 18]; in some cases, the cognitive task of live coding may be sufficiently novel to undermine conventional programming experience, forming its own alternative software engineering skill. Live coding can demand producing functioning code to a strict time limit, to find ways to introduce or modify code with low latency. High level planning routines [7, p.285] and standard working methods can be undermined by these requirements just as instrumental improvisation delivers a very different cognitive load to non-realtime composition.

1. REFLECTIONS ON PRACTICE

In an attempt to study the implications of practice for live coding, Fredrik Olofsson and myself devised and self-administered a study last August, in which we were partly joined by other live coders including Julian Rohrhuber. For the duration of the month, we each committed to an hour of live coding practice (using SuperCollider) each day, on top of whatever other programming and musical activity we were engaged in. We documented all the code we produced online (http://swiki.hfbk-hamburg.de:8888/MusicTechnology /819). The aim was to assess any sense of improvement and refinement in our live coding. Alberto de Campo's History class was used to record the time and content of any

[1] As a tongue in cheek spur to action, TOPLAP already proposed a set of Live Coding Grades for tuition (predominantly written by Alex McLean and Adrian Ward).

interpreted code, providing a mechanism to store the code improvisation sessions exactly as they unraveled, for later listening and study from the perspective of an audience. MP3 excerpts are also online for the curious reader.

Both Fredrik and I worked with blank slate coding, starting from an empty document each day. Whilst we carried out some preparatory admin outside the hour, the hour of practice always consisted of a simulated concert performance, so as to emulate the target situation. Fredrik characterised the practice as 'lots of fun' whilst I often found it more like a chore. For me it was a great mental effort after a day of research, and not the release of piano practice, for instance, where I am so much more highly automatised and able to relax in easy 'flow'. We both found quality varying from day to day, though there were positive experiences amongst more awkward days; I noted in my open letter to the TOPLAP mailing list 'I could often reach a critical point of complexity and breadth of material where unexpected and interesting results would flow, and there were enough voices to play with to maintain a musical exploration simultaneous to the algorithmic exploration.'

We were both still challenged by the formal construction of the performance, especially by the unavoidable and problematic delay at the beginning of each set. Attentional demands were high, and there were compromises between mathematical thinking and musical mapping especially for more esoteric treatments of material like the classic '3x+1 problem'. Fredrik particularly noted the formal strictures: 'often just a single 'theme' or process. I feel I'd have to rehearse a lot more to be able to do abrupt form changes or to have multiple elements to build up bigger structures over time with. I sort of got stuck in the A of the ABA form.'

Yet we also both felt we got better at it, by introducing various shortcuts, by having certain synthesis and algorithmic composition tricks in the fingers ready for episodes, and just by sheer repetition on a daily basis.

To present a contrasting example, Julian's approach was much more in the vein of interactive programming [13]. On the first day he joined he commented 'slow coding is my thing. I'd like to do live coding for airports'. A correspondence chess player compared to a timed tournament competitor.

2. PHYSICALITY AND AUTOMATICITY

2.1 The controversy

Programming a computer during a concert performance is a controversial practice in current live electronic art. This act is particularly enraging for advocates of the essential role of the human body in musical expression and interfaces to electronic music. Live coding also seems to show the (not necessarily beneficial) asymptote of silent respectful audiences for Western concert hall fare, translated to an idolization of the abstruse murmurings of programmers.

It is helpful to first disassociate control and physicality. Fredrik Olofsson differentiates physical immediacy from a feeling of musical control in reflecting on his own live coding practice sessions, arguing that instant feedback is not lacking: 'In many of the later sessions I felt like I was in total control over the sounds and where the music was going - at least for parts of the practising time. Sometimes it was enough changing just one discrete parameter, other times I kept re-coding say an LFO to take different shapes - great fun and the feedback was absolutely quick and direct (although maybe not as predicable as a real instrument all the time. Randomness, syntax errors and brain farts added to the sounds. But humans are brilliant in auto adapting and making them crazy sounds their own (or quickly find excuses like: yes I *really* wanted to distort everything here right now ;-))'

In terms of direct physical control, typing which leads to note events is trivial (spawning one event for each key tap), but uninteresting for a live coder (it is very interesting for a pianist, I'm not against individually manipulated notes!). This is the only exemplar of note-level control with live coding: the rest is score-level, of an order unachievable with a conventional instrument. We can argue about the appropriate 'score to code' and 'conductor to live instrument builder' analogies for live coding, but I would concede the failure to specify a musical stream note by note with direct feedback control.

Yet there are levels of abstraction that don't have an immediate physical analog, and this is a fundamental brickwall we shouldn't beat ourselves up against. It is an inherent 'price' of live coding that directness is exchanged for greater abstract power. The notion of *as all-consuming as possible* sensorimotor action is surely not to be privileged as the only valid state, regardless of the situated physical basis of cognition. If we define music as requiring a certain sensorimotor engagement, live coding can be excluded, but perhaps we wouldn't want to define music so as to privilege particular performance practices alone? I would argue that live coding engages with musical materials within a different framework of attentional resources, yet is of potentially equivalent expertise and sense of flow. There are countless reports of programmers (and by extension live coders) utterly losing themselves in their activity which should not be discarded.

I find it particularly intriguing in this context that recent research [14] posits the deep equivalence of sensorimotor and cognitive skill acquisition. In a psychological study of developing physical musicianship Nielsen [10, p.289] notes that 'the theory of learning strategies developed in reading, mathematics and similar learning areas where the cognitive aspects predominate, can be used in a learning area where motor performance is critical.' This raises the possibility that perhaps motor musicianship could learn from live coding?

2.2 Quick physical fixes

I shall deal both seriously and jocularly with this reoccurring issue of the physical. My current feeling is that live coding and conventional instrumental control are simply different, and should be celebrated for that. But as with all such sweeping categorisations, there is murky artistic fun to be had in the middle… so as an interlude I present a selection of (sometimes amoral) ideas that can bring the physical back into live coding for those who rue its absence.

2.2.1 Physical results after coding (errors lead to physical punishment)

Beethoven's father would strike his hands with a ruler if he made mistakes while practising. I suggest electric shocks applied to the programmer, linked to syntax errors or bugs of certain graded seriousness, with associated degrees of pain. A full system crash would be matched with death for the programmer from a loaded pistol, or a drop from a great height

as a trapdoor opens, thus incorporating a real concert tightrope.

2.2.2 Physical coding

The performer dictates a program in sign language. The performer plays with some tangible computing interface. The performer jumps around a symbol mat, etc. Both Amy Alexander in her *Thingee* system (with a dancing mat) and Dave Griffiths with *BetaBlocker* have explored these issues; in the latter work, Griffiths uses a joystick controller to struggle (http://www.pawfal.org/index.php?page=BetaBlocker) with an assembly language like game interface!

2.2.3 Physical data as an input

The data to be sonified is the posture of the live coder at their desk, as they unconsciously slump, fidget, fail to move an eyelid etc.

2.3 Automaticity

Joking or artistic indulgence aside, we can now return to a subject of great pertinence to practice regimes; the notion of automaticity inherent in skilled performance. Expert performance can be characterised by improved representations for action, often taking advantage of the special co-option of long term memory, and manifested in neural changes reflecting our inherent brain plasticity.

Musicians learn to automate many physical actions because they otherwise could not control everything at once (this is why they show expanded cerebellums and primary motor cortex representations in neuro-imaging studies!). But more abstract cognitive skills can be automated too, though perhaps not quite to the same degree as perceptual-motor skills; VanLehn [18] cites a study of the automatisation of subtasks of the skill of trouble shooting digital circuits, though the whole skill could not be automatised to the degree of driving a car.

The attentional demands of live coding would be substantially benefited by the automatisation of subtasks (I shall not consider but don't wish to preclude the possibility of AI adjoints that might assist automations in live coding, nor systems designed to hide certain avoidable repetitive tasks). As noted already, the human learning system is set up to assist us in automatising both cognitive and sensorimotor tasks if we only put in enough practice! The aim of this automaticity would be to free up as many resources as possible to deeper algorithmic engagement and long term formal musical planning: an expert 'performer can choose how and when to monitor his performance, knowing which aspects can be safely left to learned programming procedures' [16, p.102] (Sloboda means cognitive programming here, but the sentiment hopefully carries across to live coding).

Nevertheless, it is tempting to go too far in hoping for automaticity: 'contrary to the belief that expert performance is highly automatised, most types of expert performance are mediated by reportable thoughts involving planning, reasoning, and anticipation.' [7, p.291]. Because much of this remains speculative, I suspect we need to put some live coders in brain scanners.[2]

[2] This is potentially problematic due to the large number of confounds for tasks (i.e. motor noise from typing, language activation). Programming imagery might be one avenue. Behavioural studies of attention would also be applicable.

3. LIVE CODING EXERCISES

Following Czerny, Hanon and Kreutzer, practice exercises are provided below for live coding of various kinds. These are intended in the spirit of building up a repertoire. We are in a fragile state where what works is not entirely known, because it can seem as if none of us have done sufficient practice to claim true expertise! Further, some live coders have argued that the priority is systems development, and that 'internalising musical style through regular practice seems the wrong approach, for me at least. What we should be doing is reflecting upon our livecoding style, then externalising by adding functions and operators to our livecoding language of choice' [Alex McLean, TOPLAP mailing list communication]. Nevertheless, discussion of practice and development tactics can be assisted by a base of suggestions.

3.1 Isolation Exercises

Oore [12] considers the importance of reduced practice exercises for working on specific subtasks. Suggestions for isolation exercises might include:

Typing practice – (the world record for typing speed currently stands at an average of 12 characters per second)

Memory – Trying to keep all the processes and details in mind (especially without any graphical reminder in your live coding system) may be aided by memory practice exercises

Algorithmic building blocks – From language mechanisms for encapsulation/recursion/iteration and conditional looping to specific algorithms for sorting or organising data.

Mathematical constructs – Particular discrete mathematical problems as riffs to show algorithmic engagement: the 3x+1 problem, the Babylonian Square Root, more number theory (i.e. sieves, Goldbach summands, prime number algorithms, finding primitive generators modulo a prime), group theory (symmetries, permutation chains).

Computer music constructs – Sound synthesis and algorithmic composition techniques, representations from non-standard tunings (i.e. Just Intonation to x-limit) to timbral parametrisations, musical mappings including knowledge of psychoacoustics and music cognition.

Attention and awareness – Whilst automaticity may assist better potential resource management, the concentration to take advantage of this might be profitably practiced.[3] Meditation and reflection may also play a part in managing stress and concert anxiety, though any adrenalin overload tends to be diminished for experienced performers after clocking up hours.

All of these exercises are carried out with respect to a specific language under timing constraints. Whilst conceptual

[3] Whilst avoiding the circumstance of concert virtuoso performance, Julian Rohrhuber has also commented on the critical issue of dividing attention: 'The main difficulty of live coding I find is to keep aware of the relation between text and sound - this is a balancing act between the ability to change and the ability to understand a certain situation' [personal communication].

'language-free' pseudo code study might be of some mental benefit, and might be associated with certain instrument (read, language/system) independent scoring practices for code, in practice, the live coder will have chosen the environment for which they are preparing just as one might select a violin, a piano or an augmented tombola.

3.2 Connectivity Exercises

A critical and difficult aspect, much overlooked even in general algorithmic composition, is that of longer term form; this was a critical observation of the August practice sessions. The live coder may find it productive to explore and rehearse techniques for planning and integrating material over larger timescales and groupings [16, p.101].

Layering and mixing – Relative placement of multiple voices in space, spectrum, volume and time. Appropriate construction of multiple supportive or oppositional layers.

Enveloping – Use of time-varying envelopes and tendency masks for parameters, or stochastic distributions and the like.

Tension/release – Calculation of future structures in terms of emotion, surprise and audience anticipation.

The fast swap – Can you quickly adapt to a new scene? As raised by Alex McLean on the TOPLAP list, one goal, particularly in performance with acoustic musicians, would be to gain within perceptual present turnaround (<3 second). This may have to be achieved by certain innate preprogrammed facilities of a live coding system (for example, one fast and effective Wesselian fix already available in an operating system is the ability to turn off all sound quickly).

React to code – Given existing code, how quickly can you comprehend and modify? You might also practice starting from certain recorded positions, so as to practice your endgame or opening play.

The live coder may also wish to work on particular themes, that provide unifying structures. These might draw upon certain mathematical algorithms or computer music ideas, developing wider applications from those mentioned under isolation exercises, or indeed work within any accessible artistic style. An example motif might be that of sonification, perhaps of audience demographic data or TOPLAP mailing list traffic, programming language family trees and history of computing, or recursively, programming language sonification constructed live to operate on itself.

3.3 Working on Concert Pieces

The form and content of live coding concert works is itself an open area of investigation; we lack (perhaps fortunately) a rich institutionalised repertoire such as the piano commands. Practitioners have nevertheless investigated a number of improvisation frameworks. One challenging situation is the 'Blank Slate', where a live coder attempts to start from an empty document. Depending on the level and standard libraries of the programming language itself, and the nature of any user-prefabricated libraries and facilities in their live coding system, the actual challenge can very much vary [4, 9].

Whether a major music publishing house would ever be brave enough to publish them or not (and internet distribution would remain the viral best tactic), it would be a Cagean gesture to compose an intently serious series of etudes providing frameworks for improvisation founded on certain technical abilities. Just as many classical etudes escape the constraints of repetitive pedagogical exercises (for instance, by varying the harmonic basis in inventive ways underneath the bravura ostinati figurations) so might live coding etudes reinforce certain isolation exercises but in a manner palatable to presentable concert works. Without wishing to strain the western acoustic analogy too far, the sight reading of pseudo code notation might form one future practice!

Lacking such etudes and works, to cover up current solo deficiencies, due to the immense cognitive demands, various settings have been proposed for multi-player collaboration and competitive scenarios [4, 13, 20], often formulated as particular games.

Scrabble – points scoring for use of certain code constructs as if upon a scrabble board, for alternate contributions or simultaneous construction

Tetris Challenge – each player sets the other the next code element they must utilise and incorporate into their patch (A one player version is also plausible, perhaps via preselected challenges from an independent jury, or algorithmically generated challenges, as for instance exemplified by Alberto de Campo's Oracle class for SuperCollider, which randomly assigns SuperCollider language tokens and can even rate the coverage of any patch which utilises these)

Chinese whispers/cadaver exquis – Players take it in turn to modify iterations of a co-authored piece of code, or juxtapose programs or outputs (within an agreed protocol, perhaps) blind to the other's construction.

Root war – A hacker technique in which competitors try to undermine each other's systems. In other competitions for resources, performers might compete for low vs high spectral occupation (like four hands piano with the duettists fighting to grab the whole keyboard)

Shared code – A more benevolent situation, even leading to extreme programming techniques where two coders sit at one system. Alternatively, a constant remix might be in place where live coders can take over or at least copy each other's patches at any moment, allowing constant convergences and divergences.

Laptop Battle – An overlapped (perhaps with synchronization challenges before taking over) or abruptly juxtaposed session; can turn into a real fight, with competitors taking turns.

Laptop Orchestra/Jam – Hopefully avoiding an insensitive fight to be heard, but more reasonably a mutually supportative session of live coders familiar with each other's playing styles. With overlapping, the pressure is reduced on individuals to always code under full realtime responsibility.

Tag team - 2 teams of 2 coders say, where only one on each team can be live at a time.

Of course, certain improvisation games from the ensemble works of indeterminacy composers, 60's text pieces or Zorn's game pieces may all provide interesting templates.

4. CONCLUSIONS

Does it undermine live coding to judge it against traditional instrumental performance, failing to celebrate it for its own unique take on computer music? What can come out of a comparison is perhaps a healthy emphasis on the importance of expertise, as one of the great human abilities, and as more

surely exhibited in current practice by the legions of pianists than the few live coders. Whether the live coders can show the discipline to establish their credentials, and thus truly establish their field, must be a product of time. Fundamentally, the discipline required of live coders can exceed that of classical instrumental virtuosi, because the former lack the automatic sense of 'history' and weight of practice procedures that the latter musicians can draw upon. Thus, the feedback available to live coders (as for all truly experimental ventures) is hard to interpret with respect to best learning practice, and there is an additional confound of systems development absorbing effort.

To be fair, whilst there are scattered historical precedents, the modern live coding movement has only been around for seven years or so (dating it from the first slub and jitlib performances, though the former in particular have substantially revised their approaches as the movement has become more self-aware). And even ten years of informal practice will not guarantee expertise to rival a concert pianist. This paper has been an attempt to discuss certain policies of deliberate practice and the potential of future live coding concert performances. The stage is waiting for the first true virtuoso of algorithm.

Right now, I think live coders are equivalent to potentially talented 11 years olds. Give us seven years of intensive practice, and come to one of our gigs in 2014. In terms of expertise, we are perhaps beginning our Middle Years, past the Early Years of fun (even if nobody played code to us in the womb nor provided a nurturing home programming environment), and certainly not yet at the Late Years of networking and professional careers; the chief psychological skill demanded of us at this juncture might be commitment to practice [8]. It would be helpful if there were some live coding instruction videos available in the local music store with manifest examples of expert performance to aspire to! But we currently have no master teachers to guide us, and must risk the possibility of ineffective practice [8, p.287] whilst enjoying the sense of open discovery. Since I'll never get the time to practice properly if I keep writing papers like this, I look forward to enrolling in retirement as a mature student at a future academy of live coding.

5. ACKNOWLEDGMENTS

Many thanks to TOPLAP and the livecode mailing list for always stimulating discussions on these issues. Additional individual thanks for communications and live code battles to Andrew Brown, Andrew Sorensen, Fredrik Olofsson, Ge Wang, Alex McLean, Dave Griffiths and Julian Rohrhuber. And thankyou to Thor Magnusson for proof reading and debate.

6. REFERENCES

[1] Andrews, R. Real DJs code live. *Wired: Technology News*, 6 July 2006. http://www.wired.com/news/technology/0,71248-0.html

[2] Blackwell, A. and Collins, N. The programming language as a musical instrument. In *Proceedings of PPIG05 (Psychology of Programming Interest Group)*, 2005.

[3] Brown, A. R. Code jamming. *M/C Journal 9*, 6, 2006. http://journal.media-culture.org.au/0612/03-brown.php

[4] Collins, N., McLean, A., Rohrhuber, J., and Ward, A. Live coding techniques for laptop performance. *Organised Sound, 8*, 3 (2003), 321-330.

[5] Deliège, I., and Sloboda, J. (eds.) *Musical Beginnings: Origins and Development of Musical Competence*. Oxford University Press, New York, 1996.

[6] Green, L. *How Popular Musicians Learn*. Ashgate, Burlington, VT, 2002.

[7] Ericsson, K. A., and Lehmann, A. C. Expert and exceptional performance: evidence of maximal adaptation to task. *Annual Review of Psychology, 47*, 1996, 273-305.

[8] MacNamara, Á., Holmes, P., and Collins, D. The pathway to excellence: the role of psychological characteristics in negotiating the challenges of musical development. *British Journal of Music Education, 23*, 2006, 285-302.

[9] McLean, A. Hacking Perl in nightclubs. 2004. http://www.perl.com/pub/a/2004/08/31/livecode.html

[10] Nielsen, S. G. Learning strategies in instrumental music practice. *British Journal of Music Education, 16*, 3 (2006), 275-91.

[11] Norvig, P. *Teach Yourself Programming in Ten Years*. 2001. http://norvig.com/21-days.html

[12] Oore, S. Learning advanced skills on new instruments (or practising scales and arpeggios on your NIME). In *Proceedings of NIME* , 2005, 60-65

[13] Rohrhuber, J., de Campo, A., Wieser, R. Algorithms today - notes on language design for just in time programming. In *Proceedings of the International Computer Music Conference*, Barcelona, 2005.

[14] Rosenbaum, D.A., Carlson, R. A., Gilmore, R. O. Acquisition of intellectual and perceptual-motor skills. *Annual Review of Psychology 52*, 2001, 453-70.

[15] Sloboda J.A. Individual differences in music performance. *Trends in Cognitive Sciences, 4*, 10 (Oct. 2000), 397-403

[16] Sloboda, J. *The Musical Mind: The Cognitive Psychology of Music.* Oxford University Press, New York, 1985 (2004 reprint)

[17] Sorensen, A. Impromptu: an interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference*, 2005, 149-153.

[18] VanLehn, K. Cognitive skill acquisition. *Annual Review of Psychology 47*, 1996, 513-39.

[19] Wang, G. and Cook, P. On-the-fly programming: using code as an expressive musical instrument. In *Proceedings of the NIME*, 2004.

[20] Ward, A., Rohrhuber, J., Olofsson, F., McLean, A., Griffiths, D., Collins, N., and Alexander, A. Live algorithm programming and a temporary organisation for its promotion. In *Proceedings of the README Software Art Conference*, Aarhus, Denmark, 2004.