Keynote Speaker

Ross Bencina
http://www.audiomulch.com/.
rossb@audiomulch.com

The AudioMulch Process: Software Development in Musical Practice

"Software development" is one of the activities I involve myself with as part of the process of making music. I view this process as one of creative interplay between technical possibilities and aesthetic tendencies. That is to say, the computer, or more specifically, the software which I develop, is not a fixed tool with which to realise creative goals, but rather a malleable medium which evolves alongside my musical projects. The search for insight into the interplay between software development and creative activity within my own practice is current and ongoing, but one thing seems clear: it is not easy to combine medium-scale software development with music making in a unified practice. This talk aims to tease out some of the nodes of opposition which I have encountered while developing the AudioMulch interactive music studio software and will, I hope, hint at some directions towards a more integrated approach to software development in musical practice.

Computer Art / The Art of Computer Programming / Software Art / Software Culture?

Artists have been developing software for some time now (for some examples see Holzman's *Digital Mantras* [1]) Some computers scientists have considered computer programming "Art" for a while too (note that I don't often draw a strong distinction between Music and other art forms when it comes to engaging with computer programming). [2] More recently, software has *become* art, in the sense that programs are written or interpreted as functioning as art objects, for example Alex McClean's Forkbomb, or Adrian Ward's AutoIllustrator. [3] But more common than any of the aforementioned categories is the familiar scenario in which software is used to make music (art) and the artists are consumers of a software "product" developed by "engineers." This appears to present a relatively clear point of demarcation between the responsibilities of, on one hand the creative practitioner and on the other the software developer (instrument builder, etc.). This division resonates with the common notion of software applications as *tools*, or perhaps "*tool-kits*," and sometimes in the musical sense as *instruments*, or even *scores and orchestras*.

I'd like to speak in favor of a different idea: that the act of developing software can be considered the act of engaging with a creative medium. By which I don't mean a medium which (in the "Software Art" sense) supports the creation of works which exist only in, and with reference to, software. Nor in the Knuthian sense of programming as *an Ar*t. Rather I'm referring to a medium which extends in a well connected fashion from software code and related development artefacts through to (in the musical case) a creatively motivated sonic articulation. When considering a creative practice which includes software development in this way a few questions arise:

How does the way in which we learn and practice software development (or if you like, computer programming) interact with our creative practices and outcomes? Can the two processes be separated? Where and what are the boundaries formed (if any) between the them?

What are the dominant technical paradigms and cultural trends influencing computer software development theory and practice? What relation do they have to the artistic and creative goals of "creative software developers"?

On the enevatability of providing a limited set of possibilities

Setting aside the question of exactly what role we give to software in relation to the creative process, it seems that it is relatively unusual to analyse the ways in which specific software systems (structures) facilitate (and/or inhibit) specific types of creative and/or stylistic outcomes. This is especially true for systems which claim to be stylistically unbiased. In her study of 1984 IRCAM culture, Georgina Born describes a universalist rhetoric prevalent at IRCAM in which it was considered possible to create software which not only avoided being aesthetically biased, but was capable of supporting any and all musical forms. Furthermore, software biased towards specific aesthetic outcomes was considered flawed. [4] Recently the creator of the Max environment expressed a related position that "the design of Max goes to great lengths to avoid imposing stylistic bias on the musician's output," [5] but earlier this year called for a reevaluation of the cultural efficacy of the search for the universalist's "elusive music representation formalism." [6]

Some will note that I have in the past expressed an interest in the elusive music representation formalism myself. [7] I have also developed a software environment (AudioMulch [8]) which has taken the

pragmatic approach of borrowing musical ideas and processes from wherever I have seen fit for my own compositional and performance explorations. I do not claim to have resolved these two positions, but my current opinion is that in the limit, a system is only stylistically neutral if it offers all of the facilities of a general purpose programming language. Even then, the structures of programming languages could bias the outcome, to say nothing of the effects of the programming culture(s) they are embedded in (see the previous section). At the other extreme it seems relatively easy to realise existing musical structures in sound (although not necessarily automatically) since any coherently organised body of knowledge has the potential to be converted to software using a variety of effective software analysis techniques.

It is between the two poles (1) using general purpose languages and (2) the application of highly specialised domain-specific software, that we encounter a familiar situation: In seeking musical freedom by employing a general purpose programming language we must choose or devise the abstractions from which "musical" software will be assembled. The entrenchment of established musical and software models is great and there are relatively few examples of music software which succeeds in taking an entirely new direction beginning with nothing but a general purpose programming language. More often, common abstractions are combined or extended in new ways to support degrees of freedom not offered by previous systems. Importantly, irrespective of which models are chosen, their selection will often exclude equally valid alternatives. It is here that the fallacy of the universalist position becomes clear. One direction forward was recently offered by Miller Puckette: to clearly document the *models and abstractions* so that they can be reused and understood in relation to each other. [9]

The Shareware situation, software in musical practice, software for other people

My AudioMulch software has been available for anyone to download from the Internet for over eight years. The software has been through over thirty public releases in that time. It's distributed as "Shareware," an honour system where people pay for the software if they think it's worth it. These arrangements have resulted in a complex web between myself and the user community (of which I am also a member). For me there are at least three very positive aspects to this relationship: Firstly, the fact that thousands of people use my software on a regular basis means that if a bug arises I hear about it rather quickly – this leads to better quality software than if I had kept AudioMulch as my "private instrument." A second aspect of the relationship is the motivation which users provide me to make high quality software and to keep making regular improvements to it. Finally, and most importantly, are the people who I have met and assisted in their own musical endeavors through their use of my software – the richness of these relationships makes the process much more than just "hacking code."

With the above in mind I'd like to acknowledge one of the aspects of the public release approach which may not reinforce the notion of software development as in integral part of a creative practice: that of the expectation for software quality: there is a large difference between what long-term users expect from a live performance instrument, and what a programmer-composer might be prepared to accept for a single studio composition or low-risk performance. Maintaining software quality means careful planning and risk management – activities that are not known for encouraging creativity.

Improvised Performance: Adapting to Software Development / Musical Composition similarities

The final area I'd like to touch on is one that I have also heard expressed by other musicians who have become involved in software development: that the development of software takes so much time that there is none left for making music. A common (but perhaps incorrect) conclusion is that a person in such a situation has become a software developer and is longer a musician. I don't want to dwell on this point too much here, but I would like to offer the opinion that this kind of thinking reinforces the separation (embodied so well by IRCAM practice) between "technician" and "composer" as if each is privileged to their sphere alone. In objection to this position I can do no better than to quote Robert A. Heinlein: "specialization is for insects."

One thing I do think is that Software Development and Musical Composition can, at least in some cases, be quite similar activities. For example, both involve structuring large, abstract systems (perhaps Douglas Hofstadter has something to say about this). I know that when I'm developing software I find less internal necessity to *compose* music, which isn't to say I find less necessity to *make* music. This has lead to a situation where I develop software and employ it to improvise performances (perhaps you could say to compose in performance although I won't go that far). This is quite a distance from the compositional approaches I was pursuing eight years ago when I used AudioMulch to produce sound materials for tape pieces and as a processing engine for instrumental performers. [10]

(no) conclusions

This is the point in an essay where I habitually sum up the main points of my argument and try to leave

the reader with a clear sense that some kind of beautiful unitary objective has been achieved. Creative practice is, however, a process. These few paragraphs and the talk which they precede serve mainly to document current points of (dis)orientation within my own process and, I believe, within some creative software development circles. With luck they will lead to some interesting discussions over dinner.

Ross Bencina, Barcelona, June 19, 2005.

References

[1] S. R. Holtzman. Digital mantras – the languages of abstract and virtual worlds. Cambridge, Mass. & London: MIT Press, 1994.

[2] D. E. Knuth. Computer programming as an art, *Communications of the ACM*, 17:12, pages 667–673, 1974.

[3] M. Weiss. What is Computer Art? 2005. http://www.medienkunstnetz.de/themes/generative-tools/computer_art/

[4] G. Born. Rationalizing culture: IRCAM, Boulez, and the institutionalisation of the musical avant-garde. Berkeley: University of California Press, 1995.

[5] M. Puckette. Max at seventeen. Computer Music Journal 26:4, pages 31–43, 2002.

[6] M. Puckette. The elusive music representation formalism, *S2S² The Future of Music Software Workshop,* 2005. http://www.s2s2.org/index2.php?option=com_content&do_pdf=1&id=71

[7] R. Bencina. The decomposing interface - reflections on the development of musical software, *Chroma - Newsletter of the Australasian Computer Music Association,* Issue 28, pages 5–6, June 2000.

[8] R. Bencina. AudioMulch interactive music studio, 2005. http://www.audiomulch.com/.

[9] M. Puckette. 2005.

[10] R. Bencina. Oasis Rose the composition - real-time DSP with AudioMulch. In *Synaesthetica '98: Proceedings of the Australasian Computer Music Conference*, pages 85–92, 1998.

Biography

Born in Melbourne, Ross Bencina is a composer, performer and software developer with a strong interest in improvised electroacoustic music. He has performed solo and with collaborators around Australia and internationally. Recent solo appearances include The Zoo Nightclub, Manchester, and Metronom gallery, Barcelona. Since 2002 Ross and fellow composers Steve Adam and Tim Kreger have performed together as the Simulus improvising electroacoustic ensemble. Ross is the creator of AudioMulch interactive music studio – software for real-time electronic music performance, which is distributed as shareware on the internet. He is the founding developer of PortAudio, an open source library for real-time audio i/o. Ross graduated from La Trobe University in 1995 with an honours degree in arts specialising in electroacoustic music composition. He is currently a visiting researcher at the Music Technology Group, Audiovisual Institute, Universitat Pompeu Fabra, Barcelona, Spain, where he is working on new musical interfaces and audio transformation techniques.