Eric Lyon
Bregman Electronic Music Studio
Department of Music
Dartmouth College
Hanover, NH 03755 USA
eric.lyon@dartmouth.edu

Dartmouth Symposium on the Future of Computer Music Software: A Panel Discussion

[Editor's note: This article is an edited transcript of a panel discussion, moderated by Eric Lyon, at Dartmouth College in October 2001. See ''About This Issue'' on page 1 for more information.]

Eric Lyon: I think it's fair to say that without the work of today's speakers, computer music as we know it would not exist. Now that we have them all together, we have an unprecedented opportunity both to focus on their individual achievements and to assess the bigger picture of where computer music software is today and where it might be heading.

At the end of a 1989 comprehensive survey of computer music languages and systems, Gareth Loy wrote the following: ''It is remarkable how few of the languages and systems documented above are generally available.'' Among the reasons postulated for software's demise were the obsolescence of the host computer system, competition from other languages, and discontinuation of support by the programs' creators. This symposium takes Gareth's observations as a point of departure. But we turn the question around and ask not why so much software has disappeared, but why a few exceptions have been so persistent and have developed a loyal and even fanatical following among computer musicians.

In the world of recorded music, which comprises most of what we listen to these days, the term ''computer music'' is redundant. The prevalence of the use of computers in today's music demands another distinction; at its outset computer music meant experimental music, carried out in laboratories and universities. This experimental work continues here and at many other institutions, but most of today's computer music is created in the field of entertainment, whether film music or the various technology-drenched genres of rap, techno, rock, and pop.

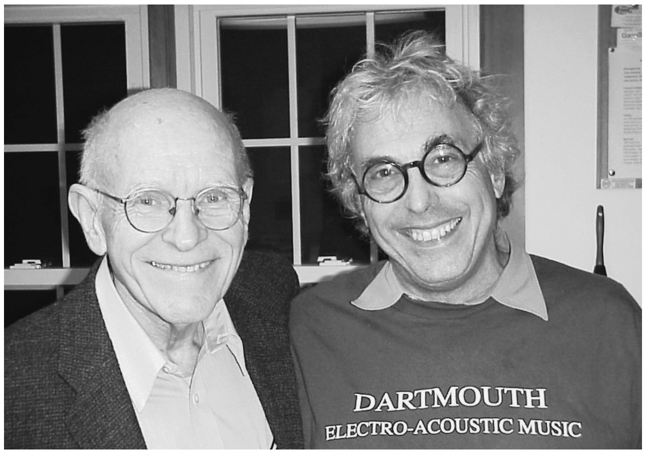
The distinction between experimental music and what might be termed ''normative music''—that is, music based on accepted stylistic norms—is mirrored in our software. On the normative side of software are utility programs such as mixers, sequencers, and reverberators. On the experimental side are the programs that we discuss today. This software is open, extensible, and invariably used in ways unanticipated by its creators. While such software does not command a market on the scale of normative utility programs, it is arguably much more influential in the long run, as it facilitates the creation of the music which today exists only in our collective imagination. And the experiments of today will lead inevitably to the norms of tomorrow. This debt is even occasionally acknowledged, such as when the Beatles put Stockhausen on the cover of their album *Sgt. Pepper's Lonely Hearts Club Band*, or more recently when Radiohead sampled (and credited) Paul Lansky on their album *Kid A* which went platinum.

My hope is that this will be a little less like a panel discussion and more like a town meeting, given the people in the audience. I am going to start by asking one or two questions, then open it up to questions from the audience.

I would like to start by discussing various aspects of different kinds of software, different specific pieces that we know and love. Perhaps this is more or less about survival paradigms, which is to say that specific pieces of software and ideas have been filtered through—have made it through—the 40-odd years of computer music. In particular, we see the ''Music N'' paradigm represented by the Music N environments and of course Csound, and we see the graphic layout and connection of signal of

Computer Music Journal, 26:4, pp. 13–30, Winter 2002
© 2002 Massachusetts Institute of Technology.

Figure 1. Max Mathews and Jon Appleton.



Max/MSP, and finally we see the pattern-based and syntactical sort of object-oriented idea of SuperCollider.

To start our discussion, I would like to ask how you feel about the idea of ''software cannibalism'' and how that may suggest a direction for the future. In other words, now that we have software models that are very clear and palpable, it also seems that we are opening them up and trying to grab ideas from other sources and absorb them into our languages.

Max Mathews: The more the better, because that's progress as well. I think the model for that is the mathematician, which I believe Miller [Puckette] is, and mathematicians do not go back to axioms to prove things if possible. They find somebody else's theorem and just continue from there. Of course that imposes rather strong requirements on the theorems. They really have to be true under all sorts of conditions to not introduce errors into other theorems that depend on them. So if cannibalism is going to be successful, then our algorithms have to be very robust with regard to the environments to which we are going to subject them.

James McCartney: Another idea that is the opposite of cannibalism and absorption is deconstruction, like the CARL system; and the way I am doing SuperCollider now is to divide things up into their simplest components. That way, each component can be worked on separately so that you don't have to keep ''glomming'' on things to one environment to make them do what you want.

David Zicarelli: I don't know if this is related to what you are asking, but I am currently working on a project to make Csound work inside MSP. It won't be ideal, because of the way each program is written, but at least you have a beginning, and it certainly should be possible to do that with other systems. I see two directions for Max: one is that, because the API is relatively simple and well-documented, it should be possible to incorporate other software into it as an object, such as Csound or even the SuperCollider language, and the other thing is that Max/MSP is eventually going to be a library that can be used for the purpose of customizing other software. The first example of this already exists, called Pluggo, which is a runtime environment for VST plug-ins that can work inside a sequencer so you develop software in Max and then put it in a particular location on your computer and open it as a plug-in to process audio in an audio sequencing environment. The same kind of thing could be true of a Web browser or video effects program, movie editor, or something like that. So, I am not sure that's cannibalism, but at least it is a way that things can work together.

Unfortunately, as any programmer knows, the interface between the two worlds is always where all the action is, and it's usually where all the bugs are, so the impulse of the programmer to do this because it seems easy is usually rewarded many times over with incredibly difficult implementation problems.

Lyon: Barry, I would be curious to hear your reaction, since in a sense, without even attempting to do so, Csound by absorbing the music and the world around it has necessarily taken in so many other things.

Barry Vercoe: Yes, well I guess the secret is to keep remaking yourself. And it is not necessarily true that you have to, in an industry analogy, gobble up all the dot-coms in order to survive; you can actually do your own thinking and grow along particular lines that are probably consistent. For

instance, we might say that SAOL is an outgrowth of NetSound, when in point of fact, there are a lot of deficiencies in the SAOL structure that I wish weren't there. We really did know better than that, but there were reasons why I needed to let a student go ahead and do something on his own. So it's ended up that the SAOL structure is much less efficient than Csound, certainly than XTCsound. I think that if I had it to do over again, I would simply take the XTCsound card away from hand-held devices and put it into MPEG. It would have been twice as fast, and what I see happening right now is that the sound community is going back and forth trying to get to the bottom of things that have already been solved many times over. When you adopt a particular line and think through all these things carefully, then you don't necessarily have to get involved in cannibalism.
Lyon: Gareth?
Gareth Loy: There is something that I don't like about the notion of cannibalism, which is that it excludes the possibility of growth—except, I suppose, it nourishes the organism that consumes the other. I would prefer to couch it in terms of genetic evolution. It seems to me what's important is what goes on in our minds, and our software is an epiphenomenon really, so in a sense, we are all like a big genetic organism thinking about these things. There are certain strains of elements of our musical beings that seem right because they help us solve the problems that we need to solve, such as, for instance, the unit generator. There are many other variations that come along, from other domains perhaps, of learning that we add to the mix to solve particular aesthetic issues that we have. So it doesn't have this quality of digesting what went before us so much as being a part of an organism that we elaborate and evolve depending on the environment in which we find ourselves.
Vercoe: I might add one more thing. It's nice when one is meandering around and it's nice at times to close the circle, complete a circle, close the loop, or something, just what we heard a few minutes ago. It is interesting because it's my belief that Max grew out of the experiments that Miller and I were running in Paris. I am not sure if that's true, but that's where the genesis seemed to be.
Miller Puckette: I have yet to sort it out. It borrows from that, but also from the synthesizer idea and also from RTSKED and who knows—it's too complicated to figure it out.
Vercoe: Yes, but the fact that David might now consider ''burying'' Csound inside of Max things is nicely self-symmetric, I suppose. So it may be self-cannibalism or something; there is a nice completeness about things like that happening within the community. I think these things are driven by the user-created process, by what composers want, what composers need. The most expansive and creative innovations in Csound or in Music11 really came from the workshops that we were running at MIT for many years. That's when the need for new things would arise, and that's when the new things would happen. That becomes a wealth of experience that is folded into the language that's certainly not just me, it's just dozens and dozens of other people's ideas of what they would want me to do, and you don't necessarily want to try to rework all of that from scratch.
Lyon: You have all developed systems over long periods of time, and they've all proved themselves. I would like to ask if you have any fantasy of what it might be like to throw away everything that you have done and just come at the question from a completely different angle. If you were to do that—if suddenly all the code that you had written accidentally was erased from every hard drive—do you have any concept of what might be a completely different way to approach the problems that we deal with?
McCartney: Well, I think most of us have already rewritten our software too many times.
Loy: Personally, I think I would start playing my guitar more . . .
Puckette: I don't think it is a problem of throwing your code away, and indeed any good programmer throws away code. Any good composer throws away paper music as well. And what you don't throw away is your memory. You can't. And, in fact if I threw out Pd and started over again at this

Figure 2. James McCartney, Max Mathews, and Gareth Loy.

point, I would like to think that I would probably end up with something rather close to Pd again, closer to Pd than Pd was even to Max. As a result, in some sense I am incapable now of going back in that same particular field and coming up with something truly revolutionary, in the sense of being revolutionarily different from Max. I would simply have to change fields, which would be fun but not something that I want to do right now.

Zicarelli: I have this idea for a program that I have never implemented, which I have called BVM, for Bad Version of Max. The idea is to exploit this tension that I talked about earlier where Max is—as Miller said earlier—the white rectangle on a piece of paper. I wanted to write a program that was always doing something and then you used its interface to modify its default behavior and something else until you got what you wanted. It would actually start out by doing something that you didn't like, rather than Max, which starts out by not doing anything.

Mathews: I certainly agree with Miller that throwing away the code wouldn't clear my memory, and, although I might bitch about having to rewrite it, I also have the feeling that at least my code over time gets dirty with various patches and things and probably bugs that I haven't found, and that it's a good idea to throw away the code and start completely over and rewrite something that is cleaner. And if I have the guts to do it myself, I will probably even do it myself before the cataclysm in the world that destroys the whole code. I suppose that's not really the basic answer to what your question is, and a program like Csound is probably a very effective program because it keeps absorbing objects, and the number of objects gets bigger and bigger. But, having extra objects around probably doesn't interfere with the subset of objects that you are using, at least not very much. So having a properly modularized program seems like one of the very important things . . . but I wish Microsoft would throw away Windows and start over.

Lyon: The last question that I am going to ask before opening it up is actually a meta-question. I would like invite each of you to ask each other questions about what you would like to see in the future.

Loy: I would like to ask how we balance the technical requirements of formal structures in the computer. Because computers operate only on formal systems, how do we balance that with the informal aspects of music composition? Suppose you start with a little motive *do re mi do*. Clearly, you have the idea of sequence going on here but there is not much harmony there yet, and if I reinforce the idea of sequence by going *re mi fa re*, now you have a sense of expectation. I've got your attention, and I believe that music is about the manipulation of peoples' attention. So, if I went *mi fa sol mi*, I would lose your attention because you are now able to predict where I am going. Instead, say I go *mi fa sol sol do*. What happens here is at the two and a half times repetition through this formal structure—it's not even an integer multiple of a formal structure—I break it off with cadence. This brings the vertical harmonic structure to the foreground, whereas over here, in the first example, I was focused on horizontal sequencing as the principle thing that I wanted you to digest. I wanted you to get that I was doing a sequence only until I could catch you off guard, just before you would start to tune me out, by shifting the focus to another representational system that would allow me to pull your attention back. And I believe that this is endemic to all music, that even if you look at the most formally crisp music of Bach or Mozart, still what's going on is a series of cataclysmic breaks in formal representation of the underlying compositional idea. And how can we get there from where we are?

For instance, the systems that I developed and which I saw demonstrated this afternoon are very good at bringing up patches that implement some idea—that implement some formal system, maybe even with random input, so that it appears that there is more information in the formal system than there actually is. But, how can we allow these things to shift levels musically in a meaningful way so that we can bring this kind of surprise to the process?

Mathews: Do you want your surprises at the level of the composing process and possibly algorithms for composing or helping composing, or do you want your surprises at the timbral level, or do you want them everywhere?

Loy: I just thought of this as an example to facilitate the question, and really I would expect the answer to occur at all levels. I could imagine also in an improvisational setting a performer might want to use a system to accompany a structure that they were creating and maybe suggest ideas to them through an accompaniment. It could arise in a number of different ways. How can we get a handle on the shifting focus that often happens in music in order to manipulate attention?

Audience Member: I don't think you should wait for the machine to give the answer to that question. It should be the composer who makes those decisions.

Loy: What if I were attempting to model the compositional process the way Lejaren Hiller did? His experiments with the *Illiac Suite* were not so much compositions as experiments, in the sense that he was trying to model the compositional process externally. And that is essentially the kind of question I am getting at here. I come up with examples like this all the time. The question is, how can I get the computer to represent them so that I could study them as a model for the process?

Audience Member: It seems to me as a composer that, as we understand these models better (à la David Cope's musical style generators and the beautifully detailed examples in James's programs)—even as we understand Bach or Mozart—we keep changing the rules. So this problem interests me from the standpoint of a teacher and from a historical standpoint. But as a composer I am very grateful that when I solve a problem like this, I am forced to create a new one for myself so that my music evolves or our music evolves and changes. Band-in-a-Box, Reason, Acid—these are great new techno tools that do a lot of stuff—even Aphex Twin's use of SuperCollider—and generate a lot of interesting new contemporary pop. They are being absorbed by a popular audience and understood as a kind of vocabulary. But, I think the great thing about our music and our musical output is that when we understand it from a formal standpoint, that we break the mold, and we continue to do that.

Audience Member: I think the artistic and aesthetic failure of David Cope's experiments in artificial intelligence simply underscore the question that Gareth Loy has posed: the issue still remains.

More useful to me would be, because we are sitting in the presence of a distinguished panel of people who have all created very valuable tools for music, for us to learn which specific musical compositions that used your software do you feel most perfectly implemented the ideas contained in that software. It is not an aesthetic question; I am not interested in whether everybody thinks they are beautiful, but, more what you felt best used the techniques that you built into the software. And I also ask that you excuse yourselves from naming your own works, Gareth and Barry.
Vercoe: Good question. I can answer that pretty clearly. For me, the two examples of good use of Music11 and Csound are *In Winter Shine* by Jim Dashow—he wrote that in Music11—and the other is called *Sphaera* by William Albright, who is now deceased. Both of these composers worked at the studio at MIT, and I think what they did was to benefit from being around the studio where there were lots of ideas floating around with the students, interacting with the students. Bill Albright had lots of interaction with Miller, for instance. The creative process of the composers builds from the environment in which they find themselves. That's part of the process I was referring to before when I say software needs to grow from something and, I think, that all the software that I have done has grown from either my musical ideas or other people's musical ideas and needs. So there's really a symbiosis that happens between composers and software developers. When that is working right, you will get both the best compositions and the best software. And I think that is why I would pick those two pieces as examples of good use of my stuff.
Loy: For the CARL system, I think my favorite compositions were the ones that the guy sitting at the end of the table did. Eric Lyon and Chris Penrose were two students who did really excellent work with the CARL system. It seemed like they really got it and were able to use it in ways that nobody else seemed quite able to match.
Lyon: Thank you very much—what a compliment!
Mathews: Jon [Appleton], would you remind me of the name of the famous composer that I met at your house—*Einstein on the Beach*—minimalist? Philip Glass? Yes. Now, I myself find Mozart delightfully surprising at almost all times. I find the Beatles somewhat surprising and rather pleasant looking back at them, 40 years later. (I didn't like them so much at the beginning.) I find Glass's music very unsurprising, and I don't think that he intends it to be surprising, or at least if it is surprising, the surprises come very slowly.
Audience Member: Well, there could be a problem with too big a surprise, right? So, it has to be just the right kind of surprise, not too big and not too little. Has anybody done any music analysis of the ''index of surprise'' in music? Analysts are doing everything else; they must have done that, too.
Loy: David Huron, perhaps? He has put a lot of thought into that, as a matter of fact. He has a theory that the element of surprise is a combination of linear sequential and leap-step types of patterns, and he has fairly convincing proof that the thing that results is the fractal behavior of music over varying expanses of time resolution.
Audience Member: For every analysis, there can be synthesis; has he turned that around and tried to synthesize it back, and is it indeed surprising to people? Does it keep your attention?
Loy: There is also the work of Voss and Clark, who in their early work on fractal music did three examples: one musical example where they chose the pitches based on a white-noise distribution, another on a Brownian distribution, and the third on a fractal distribution. People tended to prefer the fractal one because, like you were saying, it manages to balance the element of surprise against continuity.
Audience Member: I think David Rosenboom did something with surprise back in the 1970s or 1980s. He was trying to make a sequencer that maximized surprise.
Puckette: It's interesting that surprise is not the same thing as information. So the sequence of notes that would carry the most surprise in information-theoretic terms would be a memoryless random process. And that is not actually what

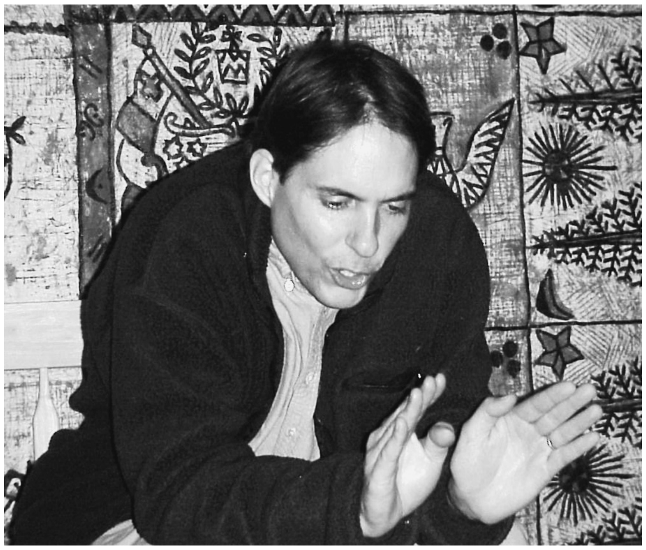


Figure 3. David Zicarelli.

would be the most surprising, because after you'd heard ten or twenty notes of it, you would know exactly what the rest of it would sound like. There is a funny paradox there that people like Xenakis had a way of getting around, but I don't know how to even describe how they did it.

Lyon: I think another point with respect to surprise is that there are really two different kinds that we are familiar with in music. One, of course, is what we consider a surprising event in a piece of music, but the second time we hear it we know it is coming, so it's not really a surprise anymore. And the other thing would be to make some kind of musical machine that would always be generating something new, which we would be able to interpret as interesting enough to keep our attention. It seems to me that the first case imposes a more stringent aesthetic requirement that the element of surprise, even after it is gone, still has some kind of impact on us, as being a deviation from a norm that we have absorbed. A good example, for those of you who know the last movement of Beethoven's Eighth Symphony where just before you have the big *tutti*, you get that unexpected sharp $\hat{5}$ that doesn't make any sense at all until much later in the movement. Every time you hear it, you know it is coming, but it hits you, because it's such an unusual event.

Loy: Surprise is relative. I remember that when I took a course in four part Bach-style counterpoint, as I got my ear more attuned to that style, I began to discover the surprising things that Bach put into these chorales which at the first sounded just banal.

Audience Member: I am a cognitive psychologist, and it seems to me that musical surprise occurs in the most satisfying way when we acquire some sort of schema based on the tonality or the sounds of our culture, whatever that culture is. And this schema leads us to expect certain rhythmic patterns and certain tonal and melodic patterns, and they have to be violated in very particular ways. If you hear something that violates it too much all at once as you've been saying, it is not satisfying. If the violations come in just the right amount and at just the right time, we find that very pleasing. And with respect to the point that you made about maintaining surprise upon repeated listenings, when we watch Charlie Chaplin and we see him walking across the room and there is a banana peel on the floor, even if you have seen that ten times before, it's still funny to watch him slip on the banana peel. There is something about this schema that you are not supposed to fall down if you're a distinguished person or you're not supposed to twist this note in this way if you are a composer. That violation, even though you know it is coming, is very satisfying, and I think that's what the great composers dance on the edge of: somehow finding a way to tweak your expectation so that it's just familiar enough and just different enough.

Puckette: It is also important that it not just be any surprise, but that it be a surprise that reveals something. When you hear a great moment in music, it is because there's a feeling of your eyes opening. That perhaps is why random sequencing, although surprising, is not musical in that it doesn't reveal as it surprises. Now what that means I can't get anywhere near to saying. What this has to do with computer music software is quite unclear, except perhaps that the software designer better stay as far away from it as possible!

Loy: Let me come back to the question then, because I'm afraid the question was not well posed. I find myself, when writing composition languages and synthesis systems, in the predicament of having to stand the system on its ear to introduce the kinds of metamorphoses that would be musically significant. I am wondering if the rest of you have encountered these kinds of dilemmas and what you have done in your software designs to take account of these kinds of things. Is that a fair question?
Zicarelli: It is a fair question. My response would be that I try to avoid those problems—to stay away from those problems as much as possible, and this isn't exactly related to what you said. One of the things that I have always liked about Max is that it has no musical concepts in it at all. That has also been one of its more controversial aspects. There was a discussion—I think it was in the ICMA journal; ''The Mins of Max''—and the criticism was that this program has nothing to do with music, and there is no way to represent musical structures in it. I think that introducing concepts that explicitly had to do with musical structures into the fundamental core of the environment would be a disaster. That's not to say that those things couldn't be added on by people who are motivated to do it, but it is personally something that I want to stay as far away from as possible.
Mathews: Gareth, I think that you need to add an adjective to this word ''surprise'' that you have brought up, and the word that I might throw in would be ''enduring.'' Now, music is to me a different animal than most anything else, like a book. I can play a Mozart sonata a thousand times until I really know exactly what is coming next, but I am still pleasantly surprised by it. I can enjoy playing it for the thousandth time far more than when I played it for the first time because, more or less, my fingers go automatically. However, I can't really read a novel a thousand times; I can read it two or three times, but that's it. And so, this surprise function must be built into a different cortical level than memory function, let's say.
Audience Member: The critique of Max that David mentions is a crucial point that really applies to all these languages. Fundamentally, they are not composition languages: they are instrument design languages. They're languages that give you a way to design different kinds of instruments and play them in various different ways. They do not have anything to do with music at the fundamental level, just in the same way that putting together some strings and bits of wood and various things has nothing intrinsically to do with music. If you are making a violin, that is not the music, the music is what you do with it afterwards. And so these kinds of discussions really go on a layer above what all of these languages are actually doing.
Vercoe: Let me just give you an example of an innovation that happened in Music11 when I was writing *Synapse*. In Music360, which came from Music4B, there was a thing called `envlp`—an envelope in which you could describe how long it was and what kind of decay there was at the end and how long the decay would last. I found, when I was writing and working closely with the violist Marcus Thompson, that he had a way of articulating a phrase that my system didn't have. I had conceived of envelopes the way a pianist would conceive an envelope: you play a note and it has a certain rate of decay and it may be pitch-dependent or whatever. A pianist would articulate by releasing the notes early so you get the articulation of a phrase by early releases. That was built into the structure of Music360 and the early part of Music11. What my violist was doing was articulating little patterns with something else; he has a way of being able—as near as I could pinpoint it, over a certain phrase that he was wanting to articulate—to drop each note down to, let's say, 20 percent of the attack amplitude of the note, no matter how long the note was. He had a consistency of coherence about this that is just very different from the way a pianist would approach the same thing. I found that what I had to do was to invent another kind of envelope to be able to match what he was doing with the computer part. And that gave rise to a thing called `envelopex`, which has an exponential decay that you could specify as part of the input argument.

Now, that was a discovery I made working with a musician playing an instrument who had control over things like that. This is the kind of evolution of ideas that comes from the act of creation, performance, rehearsals, and trying to bridge the gap between performance traditions which have so many centuries of development—just as instruments, I suppose, have developed through a process of experimentation and natural selection. And until computer music can go through that same process of natural selection and evolution, then tape and instruments are going to be quite some distance apart. So, I have had a love for combining these things, not because I just didn't like computer music by itself, but because it was a challenge to actually try and match that which has been going on in the instrumental performance traditions all this time. [Applause.]

Audience Member: I have a two-part question. The first part of the question is directed to Professor Mathews, and the second part is directed at the panel in general. Professor Mathews, you have been described in the past as the ''grandfather of computer music,'' and I am curious how you view your influence upon the landscape of music today, and I ask the panel in general what you think you might be doing if he hadn't paved the way.

Mathews: Well, the answer to the panel part of the question is that if I hadn't done something first, somebody else would have done it very shortly thereafter. I think my only claim to this recognition is simply that I happened to be in the right place at the right time, and maybe, that I also had the motivation of enjoying the violin but never being able to play it very well—so I wanted to make an instrument that was easier to play. I was first called the father of computer music, then the grandfather [laughter], but I gave some talks in Switzerland, and a publicity man from a department store filled the halls by advertising me as the ''great-grandfather of techno'' [more laughter]. Now I was happy to do that—it filled the hall with 18-year-olds, and I think they enjoyed it somewhat—but I've never done a techno piece in my life!

Lyon: That's actually not true. You made *Bicycle Built for Two*, which is a kind of techno piece. It's the first techno piece.

Mathews: Is it? Now why do you call it a techno piece?

Lyon: It is a rhythmically quantized setting of a pop tune. And it has that robotic synthesized voice singing the melody.

Mathews: Well thank you, Eric!

Lyon: That also inspired a scene from the movie *2001,* as I understand.

Mathews: They grabbed that vocal piece *Daisy Bell* of which I did the least important part. The interesting part was done by John Kelly and Carol Lochbaum, and that's really an interesting technical thing because Kelly and Lochbaum used a physical model of the vocal tract—a tube model—and what the computer did was to manipulate the shape of the tube appropriately to the sounds they wanted to make, and they put in the consonants also. If you listen to that, it's really of remarkable quality compared with things that came long after. . . . That's why the computer in *2001* sang that as a swan song.

Lyon: Any other reactions to the question?

Zicarelli: Well, I wanted to hear what everyone was going to do if Max didn't exist. But I guess no one wants to answer that.

Loy: I think I would have ended up as a failed composer of motets.

Audience Member: When computer music started, the emphasis was on a score with notes, and the notes had start times, duration, and pitches, which was very traditional. At that time, the electronic music community was doing things that were non-traditional. So, when this came out, a lot of composers—not all—said, ''This is too traditional, too note-oriented; we don't want to do the things that are note-oriented, we want to do the things that are more organic.'' I kept telling them, ''Well, you can make it organic; a note can be a whole piece.'' But they couldn't get that idea. They kept thinking that it must be a sequence of notes and this score is really imposing a structure on our ideas. It was similar to the notion that we shouldn't play a keyboard—a lot of people were saying that we shouldn't play a keyboard at that point, and they were fiddling with tape, which was more abstract

Figure 4. Gareth Loy, Miller Puckette, David Zicarelli, Max Mathews, and Barry Vercoe.

and gets us out of the habits of traditional music. The Music N programs all have that score, so they have notes, and is that a hindrance? I see these new programs like SuperCollider and so forth, and they don't have notes. Wasn't Cmix one of the first programs that got away from the note concept? Well, that's the question, then: what is a note? When I look at what James [McCartney] has up there, it doesn't look like a note. And when I look at a graphic flow diagram, I don't see notes.
McCartney: A note could just be setting some controller that is not necessarily creating a discrete event.
Audience Member: You're getting away from synchronizing the start of the note with other parameters that are changing and so forth. That makes it more free-flowing.
Puckette: There is an important psychological element to it, which is that if you do have individual cards and there is a stack of them, there is a sort of linear sequencing. And also of course, the synthesizer people—Buchla and Moog—were putting keyboards on their synthesizers at about the same time, and if you listen to the history of computer music, you hear this similar regression right around 1968 or so—right around when *Switched-on Bach* was coming out. It didn't have so much to do with computer music as it did with ''note happiness,'' which lasted at least a couple of decades. Now, in the clubs, they are starting to get away from that.
Mathews: I agree that making a score with notes is a real constraint on the kinds of music the system is able to produce. And, in my Music V anyhow, it was a very hard constraint, because I guess there was a slight facility for describing things that affected a sequence of notes as a function of time. But it was a very difficult facility to use, and in traditional instruments the sound of a given note really depends on how the preceding note at least ended and what the next note is—and that really was not possible to put into Music V. Now, my recollection, Barry, is that you improved that greatly in Csound and had a lot of ability to put in large-scale time functions that would affect a whole phrase or a section of the piece. And you probably had abilities to, let's say, play a note and maintain the final conditions of the state variables in the synthesis and use those as the initial conditions for the next note.
Vercoe: Yes, the basic principle was that unit generators could retain the state information or clear it out from event to event.

Audience Member: In the spirit of the symposium, which is to assemble some of the leading music technologists of the digital era, I wanted to ask a historical question of the panel. In all of human history, what do you think are the technological advances that have had the greatest impact or influence on music making? I am particularly interested to see how the answers fall out in terms of very old things like the ability to drill holes in pieces of wood and make flutes versus newer things, like magnetic tape or digital computers.

Vercoe: That of course depends on which culture you live in. Looking at the Western tradition, without going back too far, I would say it was around 1580. The viol had been known as an instrument primarily to double the voices. What happened around 1580 is that instrument makers made two changes as part of their experiment in natural selection. One change was to rout the frets, and the other one was to substitute a very ornate—I guess it's the beginning of Baroque thinking—resonating box. What we can see now from the work of, say, Steve McAdams is that when you move a fundamental around so that the rich harmonic structure about that fundamental is being moved in and out of very sharp resonance peaks (as occurs in a violin resonance box), then you have rapid changes of the relative strength of the component partials. For some reason that we have now begun to understand in some way (but the people at the time in 1580 only knew by experiment), that kind of sound is a very good one to separate from other instrumental sounds. That instrument can stand out from the ensemble in a way that the most of the viol family was not able.

You suddenly have in the combination of those two changes the development of a new instrument which they called the violin. And since it was an instrument that then could stand out like a sore thumb, if necessary, that began to raise new musical traditions: you get the beginning of the concerto and so forth. But, along with that came the development of instrumental music in its own right—a change of musical style—the beginnings of all kinds of new musical forms, and suddenly from 1600 onwards, you get the instrumental period of music as opposed to what you would classify as the vocal period. And I would say that that was a big innovation in the history of Western music. Composers could still write choral music, but they could write for a different kind of ensemble that was an interesting one to sustain in a concert. Now, you had instrumental music by itself or choral music by itself or the combination.

A revolution didn't occur again until the twentieth century, when in addition to those two forces you now had the addition of an electronic medium. It was many decades before instrumental music became mature, and I think that we can see the same thing with the development of the electronic medium. So we're in the midst of another revolution in music that I think is equally important. And just as we got new musical forms in the sixteenth century, we naturally have to expect big changes in the way in which electronic music is going to influence the course of musical evolution in our time. We haven't seen the last of it yet; we are still in the middle of it all. But I would expect that this would certainly qualify as being something equivalent to the impact that the development of instrumental techniques—instrument-making techniques—had in the late sixteenth century.

Zicarelli: Well, I think I understand computers and digital electronics fairly well, so I can't think of them as a profound invention. What I am impressed with is analog electronics and transducers. I took Barry's workshop in 1982, and this guy came in that worked for a company that made D/A converters. He basically said, ''You know, people say there is a 16-bit D/A, but they're lying; there isn't any such thing.'' Now we seem to have these 24-bit, 96 kHz D/A converters, and I don't hear anyone saying that they're an illusion. And I see the great emphasis in the emotions that people place on better D/A conversions, better speakers, better microphones . . . It seems as though people care a lot more about that than someone's filter algorithm.

Puckette: I head someone say about a year ago that there might have been music before there was such a thing as words and grammar. And if that is true—and I like to think that it is true—probably the invention of language itself would have had a more profound effect on musical thought and practice than anything else that's happened since. Fire certainly had very little effect on the way people make music! [Laughter.] So, that might be the answer; if not that, then I would say recording.

Mathews: Miller, did you read a little article in the *New York Times* Science page a couple of weeks ago where they have some evidence that it's a single gene that facilitates the development of language? Now, the single gene doesn't do it all, but it turns on a bunch of other genes that are involved in it. And, if you don't have this one gene, then they don't get turned on—that was my interpretation of it. So, if that gene mutated after the musical gene or whatever turns it on, then music could have come first. There are plenty of things like birds and wolves, which as far as I know don't have language but have music—maybe even dolphins and whales.

Loy: I can't go too far back, before the beginnings of language or the sounds of animals, but I can go back to the tenth century. My favorite, as I am a composer, is the work of Guido d'Arezzo, who did two important things. First of all, he invented the solmization syllables as a device for remembering plainchant melodies. People could do it on their fingers, and that would help them remember what note came next. He is also the great-grandfather of the compositional algorithm. There is a clear example of how belief in what you're doing informs how you go about doing it. His primary goal was to set the musical text of the liturgy; that is to say that the text was the jewel, and the music was the setting for it. He wanted the words to be the main focus and the music to be its setting, so he suggested the following method to aspiring composers. Find the first vowel in the first word of the Latin phrase that you are going to set. There are 5 possible vowels, and in those days there were 15 possible notes that you could sing (a compass of two octaves). And so you have three choices: you can place this vowel in one of three places within these 15 notes. And then you go onto the next vowel, and that gives you one of three possible notes.

Of course, you can do the math and figure out that there are still a very large number of compositions that can result from this procedure. I don't believe I've ever heard of a compositional algorithm that precedes this, so that would make him the originator of this concept.

McCartney: I think the one thing that's affected computer music most during its existence is not any invention but just the fact that Moore's Law has held, where a computer will continue to double capacity every 18 months. So that's really enabled computer music to go from requiring a US$ 200-an-hour computer to laptop performances. I hope that will continue.

Mathews: That certainly is my technical answer. I think that the computer programs that have been developed are pretty interesting as musical instruments, making sounds and timbres. They are pretty uninteresting—at least to me—as compositional algorithms. And, maybe they are uninteresting because we haven't found or written the right algorithms; that's a more difficult problem than just making timbres. But maybe they are uninteresting because we really personally want to write the music and we don't want to give that job to any old computer—or for that matter give it to any other person.

Audience Member: This question gets away from the actual programs themselves and goes towards the problem of distribution. I know Miller Puckette was talking about making computers accessible to people who have limited resources, and I know that the MIT Media Lab is looking at expanding into other economic environments. Do you have any ideas about ways to maximize access to these programs so that the cost can come down?

Puckette: In my own work, the answer is going to be different for each piece of software. The barrier is one of culture—of being able to really use the programs, more than just being able to get your hands on them. I don't have a good answer for that yet, and I think that is a really crucial question for me right now. How on earth are we going to make it so that people can just download the program that really does just give you a blank sheet of paper to work on and that allows you to put your own will on it? I don't know the answer to that; I certainly think that's one of the main issues right now that we face.

Mathews: Moore's Law is moving through the world in a very good direction in terms of making things more available. Certainly at the beginning, the entry price was several million dollars, and no artist could afford that, but a few sneaked in and worked at universities or companies. The PDP-11 and Music11 brought the entry-level costs down to about US$ 100,000 to make useful music. Charles Dodge had that facility at Brooklyn College for a long time at about that cost, which he was able to raise. And then the chips and the DX7 came along and brought the entry costs down to US$ 2,000, and then these popular musicians jumped in, and the number of people using these machines increased by much more than the factor of 50:1 that the price went down. There was an increase of maybe 5,000 or 50,000:1 in terms of usage. I guess the entry cost hasn't come down to much less than a couple of thousand dollars, but the value of a couple thousand dollars has certainly decreased a lot in the last couple of decades. And the power of what you are getting for these dollars has increased remarkably.

Lyon: What seems to me quite interesting is that Moore's Law does not pertain to software. There is some relationship between increased hardware power and software development, but it is nonlinear. As an example, I recall a very fine piece of software called "denoise" that was written as part of CARL by Mark Dolson. This was essentially an adaptive noise-reduction system based on spectral analysis. The price of that program, as with everything else in CARL, was zero. I recall being very surprised and somewhat amused in the early 1990s when I first discovered things called plug-ins, and these were essentially symbiotes that connect to larger programs—commercial programs such as ProTools—and they provide a particular DSP service for a price. And I was somewhat nonplussed to discover that there was a plug-in—I forget what it was called—that essentially did what denoise did, and its price was about US$ 1,000. That's an example of a non-Moore's Law progression for software. Three of the major pieces of software we've discussed here are free, and the others certainly cost less than a good flute.

Mathews: I think the major cost of a lot of software is not the dollars that you have to spend for it, but the number of hours or weeks that you have to spend learning how to use it. And that's a tough thing to reduce without imposing restrictions on the person that you don't want to impose.

Zicarelli: It's difficult to learn how to use software by yourself in many cases, too. And all of the software that we are talking about here was developed at institutions where the author was there to help. In various ways, there have been people who have made efforts to translate the technology out of that setting into places where you didn't have to know the author personally to use the program. And this is the major difficulty involved in releasing or commercializing software. Beyond releasing it, then there is the question of training and how to distribute expertise in using the software, and that's currently not free. But there are lots of kind-hearted people who seem to be sharing their expertise in various ways such as the Internet and user groups. The problem is that if you take any one random person in any one random place in the world and look at their access to the technology, it's probably not very good. But if you happen to be in the right place at the right time, in many cases someone can do something for you.

Audience Member: I feel a little compelled to compare access and costs in computer music to other musical pursuits or other disciplines. I'd also like to compare educational access to buying a flute or any other sort of traditional instrument. You can pick up a used computer or a free computer; it might not be the latest, but it's probably only a few years old and still very useful. Paint for painting and stone for sculpture aren't cheap, and neither come with user manuals and tutorials the way that Max and Pd and Csound do.

Figure 5. Miller Puckette.

Vercoe: As you look at the results of synthesis programs over the years and the benefit of a hundred years hence, I wonder which pieces will have survived. We are talking about software survival here, and let's also focus on which pieces of music will survive. I don't know how to guess at that, but it may well be that the enduring pieces will come from the systems that were the hardest to access. I'm not talking so much about the cost of them—just the hardest to get at, where you had to really sweat through what you had to say to say it in the way that you wanted to say it. I wouldn't say that making things easier and easier is going to give rise to lots and lots of new, long-surviving music. For certain, there will be lots of activity, but there's a fear I have that so much of this will be noise. And what the history of music will eventually sort out is which of these things were the giant steps that will just keep coming back. And I think it's wise for us to think about that—just what engenders the very clear statements from composers, which is after all why we've ever done this. We're not just writing software to keep machines busy.
Loy: There was a bit of philosophy that we had in the design of the CARL system to this end. Since it was designed in a university environment, it was designed to create a community of users. We always tried to keep in mind that if we had, say, ten facts about a system that we were going to require someone to learn, that the eleventh fact would somehow relate to the previous ones in some coherent fashion, so that you really could bootstrap yourself into a wider horizon. Ultimately, I think, computer software is an oral tradition. We can get somewhere without having someone to answer our informal questions about it. But there are so many questions at so many levels that I've gotten that I no longer believe it's possible for me to thread a single linguistic thread through the middle of what everybody wants to know.
Audience Member: Earlier, you were talking about what composers had done with your software, and we glanced over that question. I think two of you answered that question, and then it disappeared.
McCartney: I think that, at least for my software, the people who seem to have learned it the fastest and gotten the deepest into it are the people doing the live techno laptop stuff. They seem to be the most caffeinated groups, so they're actually the people who have written the most code.
Puckette: I think your question leads to a different issue. We've talked quite a bit about software, but what we haven't mentioned is that—almost regardless of what software we are using—I don't think that we know on a musical level what to do with computers. We don't yet know what the modality is or the modalities are that people will find to incorporate computers in their music. I think that perhaps there is some software that hasn't yet been invented that will allow that, but I don't feel that way. I feel that in fact there's a real question about where to go now and how to use this tool, the programmable instrument. People will do interesting things in spite of the limitations of the current systems, certainly.
Audience Member: You've all authored some really extraordinary software. Would you in retrospect have done anything differently to protect your own intellectual property? I want to tie that into how the MPEG-4 standard might return intellectual property to composers and authors in a meaningful way.
Vercoe: I don't know about returning to composers and authors—that's really between those compos-

ers and the music industry. I can tell you regarding the software itself that, unlike most of the rest of MPEG-4—which was contributed by industries and various forces—the core music synthesis part of it, the SAOL stuff, I put immediately into the public domain. This has always been my policy from Music 360: just to get it out there. Of course, that is the sort of thing one can do from an ivory tower. Corporate research groups cannot afford to do things like that. So the other parts of the MPEG-4 standard do in fact have licensing attached. But I think in general, wherever it is possible, we should get something out there and just spread the word.

McCartney: I think that if a lot of things in this field had been patented that are pretty basic, the field would have been stopped cold from advancing. So I don't want to patent anything that I come up with, just because I don't want to stop anybody else cold. I really don't like the idea of protecting patents in this area, but I know it is done a lot.

Zicarelli: I am similarly also opposed to software patents, and I think that they probably should never have existed. And as far as why I've fallen into being a commercial person rather than being an academic person, you know I am a bad teacher. Every time I've applied for an academic job, I never got anywhere. I am not sure I would even be good at the cocktail parties. My life just took a certain path, and I am not sure that I ever really examined it from first principles of the ideals of intellectual property issues. But what I try to do is give everything that people request. If there is some piece of source code that someone wants, I usually give it to them, but I don't give away the ability to violate the copyright on producing the commercial thing that allows me to do the work I'm doing. As long as there is one way that I can trick people into giving me money to continue to do the work that I like, I'll probably do that. I try to give away as much—other than that one little trick—as I possibly can, just because it's great to see people doing things with that information. If someone thinks the information is valuable, I'm happy to share it as much as other people in this field. And everyone sitting at this table has shared information that allows me to do the work that I do.

Audience Member: I hope James gives everyone the same opportunity to learn from his source code as he learned from Barry's.

McCartney: Well, actually if you download version 3 of SuperCollider, all the primitives are there. There is actually a lot you can read; I just don't give you the engine in the middle. [Editor's note: Supercollider was made free as of June 2002, and it was announced that the source code would soon be available.]

Loy: The general issue with what to do about protection depends of course on what it is you are trying to accomplish. You can either keep something a trade secret, attempt to patent it and capitalize on a license agreement, or simply make a wall of patents around its core technology to guarantee a revenue stream. Or if you get your revenue stream from an academic institution, as Miller was saying earlier, it is a really great opportunity to do what you think is the right thing. That was also pretty much the inspiration for the CARL system. We took our salaries from the grants that we got and from what facilities the university gave us, and what we produced was then put into the public domain, because we were compensated for it. But if you're not compensated by an academic institution, you must have some way to guarantee your revenue stream. And it's really hard, as David was saying, to support users in the field. I have worked with some companies in Silicon Valley that have the same problem where they have a technology that is difficult to penetrate. Their product fails because they cannot get enough technical support personnel, and so the business fails. This is a serious problem. You have to have enough return on your investment to be able hire the technical support required for your product to flourish. These are tough problems. It's not easy to know what the best way to go forward is.

Vercoe: The solution is to have someone else write a book on your stuff.

Audience Member: I would like to ask the panel what they think of the open-source paradigm—how that's going to influence future development of computer music software.

Puckette: I think it is certainly true that the open-source movement is a very powerful source toward higher-quality software, and I would turn the question around and ask if open-source music wouldn't also be a very good thing. If composers starting making music available in such a way that you could absolutely use it for anything, it would be amazing how fast the musical culture would then develop. The problem is that we're stuck on this outdated notion that the only way a composer or instrumentalist can survive is by forcing people to pay them for airtime. That's a revenue stream model, but I don't think it's a very good one. And the thing that made it quit being so good is that you quit distributing physical hardware with it. In other words, I think a copyright makes sense when you are talking about paper or a compact disc. I don't think it's a good thing to impose on things that are distributed by the network. And I think that the Internet is a good way of distributing music right now. I would rather see copyrighters not even get mixed up in the issue. I would rather see it tied to physical media. On the other hand, composers can't really survive anyway; the only people that are really milking us are the studios—but also the content holders like Sony. So that's the main development that I'm hoping to see in the future, and I know that right now the GATT organization is working as hard as they can to prevent us from having this and also from having open-source software. It's not clear whether the open-source movement (or the open-music movement) will really succeed against the commercial interests that are already against it, but I certainly hope so. I think it is everyone's duty to do whatever they feel they can do in keeping all manner of intellectual property restrictions out of the stream of innovation—even if it's in their income stream.
Audience Member: I would like to ask the panel how computer music has changed in the last 30 years. We know that user interfaces and programs and synthesis techniques particularly have a large influence on the type of music that composers produce.
Vercoe: There's a very big difference; there is a whole lot more audio processing of live and recorded sounds. That's just part of the evolution of this phenomenon of computer music, so certainly things keep on growing. I can't predict what is going to happen in the next ten years, but it will be something significant, I am sure. But there will be the classics that will survive the decades, and all we can do is keep writing whatever we write and hope that one of our pieces will become a classic.
Loy: At some point, quantitative changes in processor speed and disk sizes effect a qualitative change. For instance, Max went from being a MIDI control language to a digital synthesis engine, but it couldn't do that with the processors that it was originally designed for, because there wasn't enough horsepower there. So you get this qualitative shift then from a controller of synthesizers to a general-purpose synthesis engine under the hood.
Puckette: In other words, we've only got to undo the MIDI revolution!
Audience Member: Do you think the MIDI revolution was a step forward or backward?
Puckette: I think it was a huge step sideways.
Audience Member: As software developers, you've probably all experienced this: when you put out a new feature, the user community generates five more requests for other new features. It's been my experience that the user demand for new things and the ideas that flow your way is pretty strong. Is it a fact that you can't keep up with what the users would like to see, and are you frustrated by that? What do you think is inhibiting you from making more rapid progress with your software projects?
Zicarelli: If you implement a feature in x amount of time, then you get $5x$. So as x decreases, you just get more feature requests, so you could never catch up, even if you were more productive.
Audience Member: Some of the features are in high demand; let's just focus on those. I would think in your case that Max on Windows has been an issue. I think there has been a group of people who would like to have seen that happen a few years ago. Certainly, there are always these individual quirky requests that come your way that are not worth pursuing. I am thinking more of large-scale requests, where more than half the user community just wants it.

Figure 6. Eric Lyon and Gareth Loy.

McCartney: It's triage!
Zicarelli: I was going to say that that was the number one frustration, but I don't want to—there must be something else that's frustrating, too. You have a program that works on one computer, and the stupidest person in the world can say, ''Why don't you make the same program work on this other computer?'' The level of thought behind the suggestion is inversely proportional to the effort the suggestion requires to implement it. So that's what's frustrating to me—this really stupid idea to just make it work on another computer. There is absolutely no creative work in that—it's the most horrible work in the world, and yet it's the thing that people want the most. The creative, difficult, interesting problems that might even be quick to do—no one cares about as much.
Audience Member: I'd like to know what the panel thinks of today's technology-based popular music like rap and techno, and, in a broader sense, how you feel about the fact that your innovations have been absorbed into popular music and popular culture.
Mathews: Well, I myself don't really enjoy popular music, but I'm delighted that anything from our technology is being used by popular musicians. In fact, that's the biggest field of users. And I'm delighted if they have an audience that goes for it.
Loy: This is an example of a dynamic system in action . . . a chaotic, non-linear dynamic system in action. There is no way to predict how these things are going to be used.
Puckette: I am partial to Fat Boy Slim. I don't think he uses my work at all, but whatever he's got that's playing right now is pretty cool.
Lyon: James, you have a lot of people that you've contacted directly who are using your software.
McCartney: Well, I don't care about what someone's particular style is so much. Sometimes people play me these things they did, and sometimes I like them, and sometimes I don't; but it doesn't really matter, as long as they were able to use it, it's good. Sometimes people send me things in which they have used my program in some way that I think is really ridiculous. But they sometimes pose the most interesting problems.
Vercoe: I've not usually paid much attention to how my stuff ever got used in the popular field, nor have I paid much attention to the popular music styles, so I'm often caught off-guard when something does happen. But I will answer two questions at once: the one that was just posed, and also about the music of Miller Puckette. I can tell you about one of the really interesting, exciting things that I experienced when Miller was around at MIT. He wrote a little piece of music called *Cat's Ass*. That really was quite neat, and I thought it was a new use of the technology coming from someone who never claimed to be a composer, and yet it showed the kind of musical insights that he has. And that's why he's been such a contributor to the field. Thank you, Miller. You didn't expect that one, did you?
Puckette: No. I don't distribute anything that I have ever done. But for the record, it's *Cat,* space, *Sass*.
Lyon: Gordon Mumma once told me that, in his opinion, every creative act was essentially a form of subversion. And of course, as you know, Gordon is one of the most deeply technologically involved composers of the last century. When he said that, Mumma was working with a very cheap commercial FM tone generator and was digging around the operating system, discovering all kinds of bugs, things just not working the way they were supposed

to work, and then making that a part of the piece. It seems to me that one of the real questions is, ''Is it possible to subvert this software, and is it desirable?'' For example, one model that we see is the model of a blank piece of paper; how attractive or even possible is it to subvert a blank piece of paper? On the other hand, the Music N model, almost by its nature, seems to be a challenge thrown in the face of composers. Namely, here we have this new way of making music, and yet we are supposed to write a score for orchestra. That's ridiculous! Now you can take that model and subvert it by doing things like having a million notes, so that all of a sudden you have something that an orchestra wouldn't do. You have instruments behaving so a note is not what we usually think of as a note. There are all these possibilities for subversion in that first paradigm, so I think I find myself accidentally asking the last question. How do you feel about the idea of subversion of your own software?

McCartney: I think my users are secretly trying to subvert it in every way possible. I guess the whole glitch community will take all of your unit generators and find out where you didn't write them correctly and exploit that.

Puckette: Yes, that oscillator of yours is going to fail as soon as the frequency goes above the sampling rate!

Lyon: I would like to thank you all for a remarkable and enlightening day!