TWO APPROACHES TO INTERDISCIPLINARY COMPUTING+MUSIC COURSES

Jesse M. Heines, Gena R. Greher, S. Alex Ruthmann, and Brendan L. Reilly
University of Massachusetts Lowell

This is the submitted version of a paper published in the December 2011 issue of IEEE *Computer* magazine featuring articles on Computers and the Arts. The published version of this paper can be found at:

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6086524

The bibliographic reference for the published paper is:

Heines, J.M., Greher, G.R., Ruthmann, S.A., & Reilly, B. (2011). Two Approaches to Interdisciplinary Computing+Music Courses. *IEEE Computer* 44(12):25-32. Special Issue on Computers and the Arts, December, 2011.

ABSTRACT

The intersection of computing and music can enrich pedagogy in numerous ways, from low-level courses that use music to illustrate practical applications of computing concepts to high-level ones that use sophisticated computer algorithms to process audio signals. This paper explores the ground between these extremes by describing our experiences with two types of interdisciplinary courses. In the first, arts and computing students worked together to tackle a joint project even though they were taking independent courses. In the second, all students enrolled in the same course, but every class was taught by two professors: one from music and the other from computer science. This course was designed to teach computing and music *together*, rather than one in service to the other. This paper presents the philosophy and motivation behind these courses, describes some of the assignments students do in them, and shows examples of student work.

It's the first day of a new semester. Two students walk into your class. You've never seen them before. You know nothing about them. You identify them and check their majors (primary fields of study). One is in Computer Science, the other in Music. Which do you assume is the more *creative*?

Surely, most of us would answer "the Music major." The general perception is that people in the arts are more creative than those in the sciences, particularly those in computing. But is this truly the case?

Consider the types of learning experiences that characterize each field. In Music, students mainly focus on the *re-creation* of art. They learn to master their instruments by studying someone else's original creations. The *composition* of original works is advanced study, typically pursued by only a handful of Music majors, and typically at the graduate level.

In Computer Science, students may initially re-create programs that implement known algo-

rithms, but they quickly progress to writing original programs to solve problems. Those problems may be carefully bounded, but solutions devised by good students come will typically exhibit a wide range of approaches.

It is interesting to note that Music students have to learn concepts and syntax, too. Think of staves, notes, key signatures, accidentals, fingerings, etc. The difference is what they *do* with these. In general, they apply what they've learned to try to play a piece exactly as their teachers say it should be played. Computer Science (CS) students try to apply what they've learned to *solve a problem* outlined by their teachers.

So, on reconsideration, which do you now judge to be the more creative?

INTERDISCIPLINARY LEARNING

It is not our purpose, of course, to instigate an argument over who is more creative than whom. But it certainly *is* our purpose to break stereotypes and to stress that when one looks at science and engineering majors vs. their peers in the arts, business, and other supposedly non-technical majors, it is clear that they have much to learn *from each other*. It is not much of a stretch to assert that the technologies most of our CS graduates will be working on 5-10 ten years after they graduate probably haven't been invented yet. This can make it a bit hard to decide what or how we should teach them. We have therefore based our work on the following postulates.

(1) ***Once our CS students graduate, it's very likely that they will never again write a program of any significant size by themselves.*** Instead, they will work in teams, and those teams will undoubtedly be interdisciplinary. Even if certain members of the team do not write a single line of code, they will have a say in not only what a program does, but also in how it is implemented.

(2) ***Basic skills will remain basic.*** An array will always be an array, and a linked list will always be a linked list. With all the buzz about students wanting CS programs with concentrations in game development, programmers who succeed in that subfield will be those who understand that interesting games are built on the fundamentals of algorithms and data structures, just as musicians understand that interesting music is built on the fundamentals of melody, rhythm and harmony. Zyda states: "The game industry ... wants graduates with a strong background in computer science. It does not want graduates with watered-down computer science degrees, but rather an enhanced set of skills" [1].

(3) ***The need for everyone to have basic computer skills will only increase.*** Jeanette Wing writes that the basic skill in problem solving is "computational thinking," which "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" [2]. She feels that this "is a fundamental skill for everyone, not just for computer scientists." We strongly agree, and we feel that by exposing arts students to computational thinking *within their own field* has huge potential for enhancing their education.

(4) ***Everyone has something to learn from everyone else.*** Virtually all jobs today involve interdisciplinary teams, and working in such teams almost always requires that assumptions about one's coworkers' fields be abandoned. Reflecting on one of the assignments in our interdisciplinary course, a CS major wrote: "It was great to work with someone as musically (and graphically) inclined as Maria [a Music major]. I lack a lot of knowledge about both of those, and her ideas made very notable improvements in the programming as well as the music and graphics." Note that the CS major specifically mentions improvements to the *programming* based on ideas from the Music major.

COMPUTING+MUSIC COURSES

To address these issues, we developed two interdisciplinary course models that our colleague Fred Martin dubbed *synchronized* and *hybrid* [3].

The *synchronized* model pairs two independent, upper-level courses in different disciplines and requires interdisciplinary teams of students to complete a joint project collaboratively. The *hybrid* model is a single course taught by two professors from different disciplines, with both in the classroom throughout the semester.

These are, of course, but two of myriad models employed in interdisciplinary computing+music courses. To put our work in perspective, we took an *informal* look at 52 courses at 40 colleges and universities that cover computing through music or music through computing. Some of these were identified by attendees at a March 2011 workshop on this topic under the auspices of the ACM SIGCSE Music Committee [4] and sponsored by the NSF-funded LIKES project [5] (www.likes.org.vt.edu). Additional courses were found by the student researcher on our team, who searched the web for syllabi that combined computing and music in interdisciplinary courses.

Our search criteria specifically *excluded* audio recording and production courses that have the shaping of sound through electronics and signal processing as their primary objectives. These courses fall at the intersection of computing and music, to be sure, but they focus on *using* technology to achieve desired sounds, rather than *teaching* computational and musical concepts together. With those caveats in mind, Table 1 presents general information about the courses we discovered and gives an overall picture of the landscape.

Table 2 presents the content of the 52 courses, as best we could glean from their posted syllabi. This is an inexact measure, to be sure, but it still gives a somewhat reasonable view of the field. (The Ns in each section do not add up to 52 and the percentages do not total 100% because some entries fall into more than one category.)

Where Our Work Fits

One can see that there are indeed large ranges of courses offered, subjects covered, perspectives taken, teaching styles employed, and software systems used. Reviewing these data and reflecting on our familiarity with some of the people who teach these courses, the following overall picture emerges.

- At the upper end of the curriculum, virtually all courses that cover computing+music are advanced offerings by Music departments. We know of no upper-level CS courses dedicated to addressing issues faced by musicians (although of course there may be some unknown to us).
- Courses and research at the upper end require deep understanding of *both* computation and music. (See, for example, the algorithmic composition work by Edwards [6] and Brown and Sorenson [7].)
- At the lower end of the curriculum, music is typically used to demonstrate or to introduce concepts. This is music *in service to* computing, not music *integrated with* computing. (See, for example, the media computation work by Guzdial and Ericson [8].)

Table 2. Computing+Music course offerings.

Listing Department	**N**	**%**
Music	40	77%
Computer Science	9	17%
Co-Listed	3	6%
Type of Instruction	**N**	**%**
Single Instructor	28	54%
Team Taught	8	15%
(*unable to identify*)	16	31%
Student Level Targeted	**N**	**%**
1st- & 2nd-year Undergraduates	21	40%
3rd- & 4th-year Undergraduates	19	37%
Graduate Students	5	10%
Multiple Levels	7	13%

Table 2. Computing+Music course content.

Disciplines Covered	**N**	**%**
Sound/Audio	37	71%
Computer Science	36	69%
Music (composition)	22	42%
Music (theoretical)	12	25%
Media	5	5%
Primary Focus	**N**	**%**
Composition	31	60%
Sound Synthesis	27	52%
CS (introductory)	18	35%
Sound Processing	17	33%
CS (specialized)	12	23%
Music Theory	7	13%
Interactive Media	1	2%
Software Used	**N**	**%**
Max/MSP	11	21%
Audacity	4	8%
Processing	4	8%
SuperCollider	3	6%
ChucK, Disklavier, Pro Tools, Reason	2 each	4% each
Audition, Garage Band, MATLAB, Peak, PureData, Reaktor, Scratch, Sibelius	1 each	2% each

Our work attempts to fill some of the gaps between these types of courses by providing integration of computing+music at a high conceptual level. Our synchronized course targets mid- to upper-level Music and CS majors with the intent of furthering students' knowledge of *both*. Our hybrid course is a General Education ("GenEd") offering open to all students in the university. It attempts to provide an understanding of where computing and music interact, at a level that is accessible to students without deep knowledge of one or the other.

Thus, our work is at both ends of the *instructional* spectrum. The remainder of this paper describes our courses, the topics they cover, and the types of assignments our students complete.

GUI PROGRAMMING + MUSIC METHODS

One of the ways to get started in interdisciplinary teaching and learning is to connect the students in two existing courses through a joint project. Administratively, this is a "low-hanging fruit" approach, because it doesn't involve getting a new course approved or making any changes to the course catalog. All that's needed are professors who agree to collaborate with each other to build an interdisciplinary project into their courses.

In our case, the CS professor teaches a project-based course in graphical user interface (GUI) programming, which fit nicely with a project-based course on teaching methods taught by the Music professors. After reviewing the projects that we assign in our respective courses, we decided to make our initial foray into interdisciplinary teaching using a "Found Instruments" project that has been used in Music for years.

The Music Assignment

For the musicians, the purpose of our assignment is similar to one described by Hugill [9]: "to strip away previous ideas of 'musicianship,' [by] reevaluating the sounding properties of objects, how they may be made into instruments, how playing techniques might be developed, and how music may be created as a result." Here's what Music students are asked to do:

(1) Using only household object(s), create a musical "instrument" that can produce several different pitches and/or timbres. Your instrument must be able to produce several different types of sounds, or sounds with several different characteristics.

(2) Create a composition for your instrument that employs a specific musical form of your choice. It need not be long. A 2-3 minute piece is sufficient, but it must include distinct sections

that give it form. That is, your composition must include distinctive opening, middle, and closing sections.

(3) Devise a system of creative notation that others will be able to understand well enough to perform your composition. Your notational system should not resemble traditional musical notation in any way.

(4) Bring your instrument and notated composition to class. Come prepared to explain your work and to perform your piece.

To achieve camaraderie and pique interest, the CS majors are also given this assignment. Our experience is that the instruments "found" by the CS students exhibit just as much novelty as those of their Music counterparts. When we get the students from the two courses together, we do a number of things to build community, including having them jam on their instruments in mini-ensembles. Again, the CS students "get into" this project just as much as the Music students, and the resultant "music" is, well, "interesting" to say the least!

Another class activity has students try to play *each other's* found instruments from the notations created for those instruments. We have them do this *without* first hearing the original composer play the piece and *without* any verbal explanation of the notational system. This is a good test of the communicability of the notation by itself, and it opens up a number of avenues for discussion of human factors. As an example of this activity, please see www.youtube.com/watch?v=IJuGoYnCxSs.

The Computing Assignment

So how did the Found Instruments project connect to computing? Through the creative notation. Here's how it worked.

(1) We introduced CS students to standard music notation software using Finale Notepad (www.finalemusic.com/NotePad) and Noteflight (www.noteflight.com).

(2) We assigned CS and Music teams and charged the CS students with creating a music notation program for the notation devised by their Music partners.

(3) We scheduled several joint classes in which the Music students could work with the CS students on the programs' designs, review the CS students' works in progress and offer comments and suggestions for improving the programs, and finally act as usability test subjects on the finished products.

Some of the programs produced as a result of these collaborations and the lessons learned from them were truly astounding. We describe below one of the best.

Mike, a music student, used his jacket as a found instrument, creating sounds by slapping it, rubbing it, working the zipper, etc. (see Figure 1). He then created a piece satirically named *Eine Kleine Jacket Music*. An excerpt from Mike's creative notation is shown in Figure 2, and performances of Mike's piece first by a Chase, a CS student, and then by Mike himself are posted at www.youtube.com/watch?v=iD4dEZOTiIg.

Figure 1. Mike playing his jacket as a "found instrument."

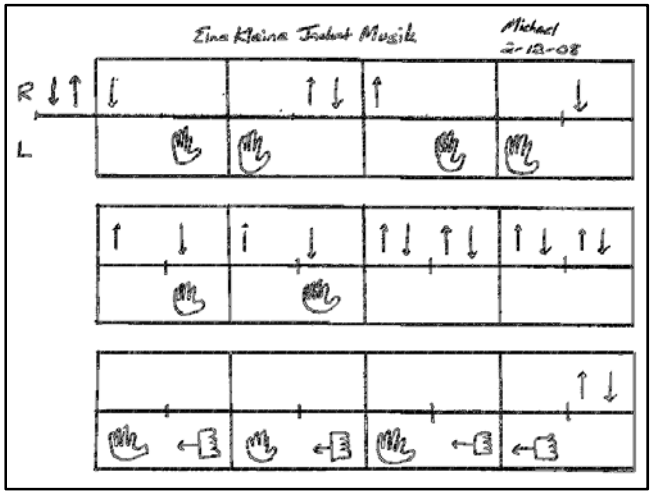


Figure 2. Mike's notation for his composition.

Figures 3-6 walk through part of Mike's partner Chris's composition program to demonstrate the CS concepts and skills involved in developing such a program and one important lesson that Chris specifically learned from this project.

Figure 3 shows Chris's composition program after a few icons from the tool palette on the left have been placed onto the right- (R) and left-hand (L) staves in the composing area by either dragging-and-dropping them or double-clicking them in the tool palette.

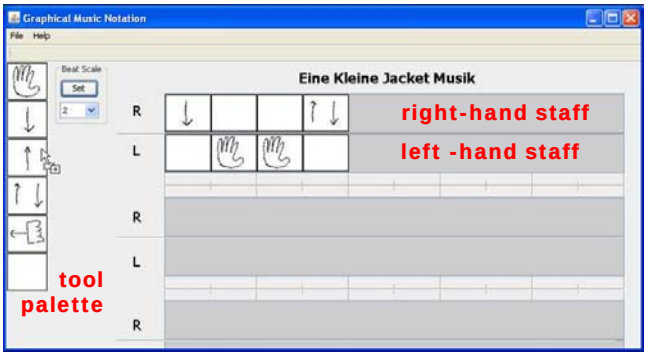


Figure 3. State 1 of Chris's music composition program for Mike's jacket notation.

Figure 4 shows the program with the insertion cursor positioned between the 6th and 7th icons on the left-hand staff, as indicated by the thick vertical bar. If an icon in the tool palette is double-clicked at this point, that icon would be inserted to the right of the insertion cursor, which is to the left of the last hand icon on staff L.

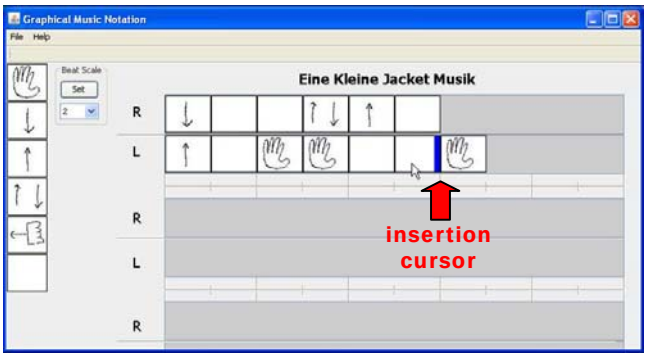


Figure 4. State 2 of Chris's program with the insertion cursor positioned between the 6th and 7th icons on the left-hand staff.

In Figure 5, the `Backspace` key has just been pressed, and the blank (or "rest") icon pointed to by the arrow cursor in Figure 4 has disappeared. The issue is that the thick vertical bar insertion cursor has also disappeared, leaving users to wonder where the insertion point is. In most editors, the insertion point would not change. That is, if an icon in the tool palette is double-clicked at this point, that icon would still be inserted to the left of the last hand icon on staff L.

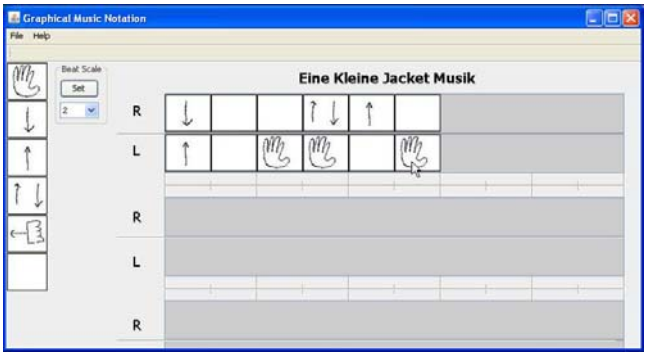


Figure 5. State 3 of Chris's program after the icon pointed to by the arrow cursor in Figure 4 has been deleted.

But unfortunately, this is not what happens. Instead, when the "scratch" icon is double-clicked it is inserted at the beginning of the staff, as shown in Figure 6. This may be fully logical to a programmer who has implemented the compos-

ition area as a pair of linked lists, but it is not at all logical to someone used to working with any sort of text editor.

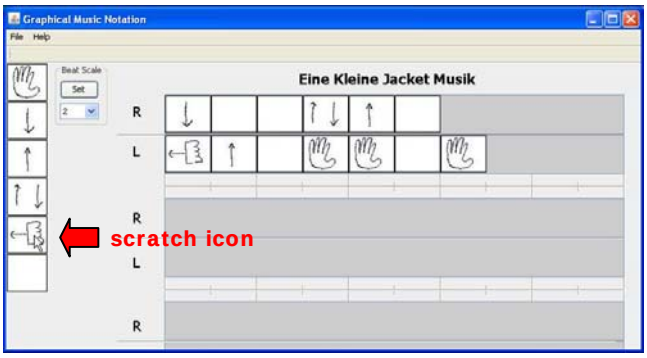


Figure 6. State 4 of Chris's program after the "scratch" icon in the tool palette (indicated by the arrow cursor) has been double-clicked to insert it into the composition.

When the anomaly was pointed out to Chris, he immediately recognized the problem and said, "I can't believe I didn't notice that." But that's exactly why usability tests are needed. Programmers are often "too close" to their work to see even the most obvious of user interface issues. Teaching this point in a lecture setting requires students to mentally connect theory and practice. When it is learned from a peer while testing one's own software, the connection is far more concrete and the lesson is learned at a deeper level that is more personal and, therefore, more effective.

Thus, not only the different, but also the *fresh* views of students in other disciplines can teach valuable lessons to our computing students. Likewise, for Music majors, helping non-musicians translate their musical concepts into computer programs can shed light on the clarity of their thinking—or lack thereof. Such *reciprocal learning* [10], in which students learn from each other instead of just from the professors, exemplifies one of the very best characteristics of interdisciplinary courses (see Figure 7).

Figure 7. Music and CS students working on the design of a composing program.

SOUND THINKING

Our *synchronized* courses worked well at the upper end of our curricula, but we also wanted to work at the lower end so that we could introduce more students to the benefits of interdisciplinary courses. Following the pioneering work of Yanco et al. in combining Art and Robotics at our own university [11], we developed a new *hybrid* course that could be offered to all students in the university. This is *Sound Thinking* (please see soundthinking.uml.edu).

There are two critical characteristics about the way in which *Sound Thinking* was put into the course catalog that contributed significantly to its success. First, it was co-listed in both the Music and CS departments. Second, we applied for and were granted General Education ("GenEd") status for the course. Arts students who take it register using the CS Dept. number and receive Science & Technology GenEd credit. Science students register using the Music Dept. number and receive Arts & Humanities credit. These characteristics were essential to achieve the critical number of registrations needed for the course to run, especially with two professors present at all class meetings.

Revisiting Found Instruments

We used the Found Instruments project at the beginning of *Sound Thinking*, too, but we took it

in another direction. After students created their instruments and notations, we had them record the sounds their instruments could make and then used those as an introduction to sound editing with Audacity as explained below.

Eric, a CS student, created what he called a "lever drumitar," shown in Figure 8. He strung a guitar string across the opening of a cup, secured it with strong tape, and rigged up a carabiner to use as a lever for changing the cable's tension. This allowed him to produce different sounds when he strummed the cable with a pop top.

Figure 9 shows the original notation that Eric created for his instrument. Each row represents an action. If the square in the second column is filled in, the string is to be strum. A V in the third column indicates that the time duration is to be shortened. The length of the line in the fourth column indicates the position of the carabiner.

For the next assignment, students recorded the various sounds their found instruments could generate and loaded them into Audacity. They then created original compositions by looping and combining those sounds. To hear Eric's original lever drumitar sounds and his remixed composition, please see www.youtube.com/watch?v=_zA_hn_4T8k.

Figure 8. Top view of the lever drumitar.

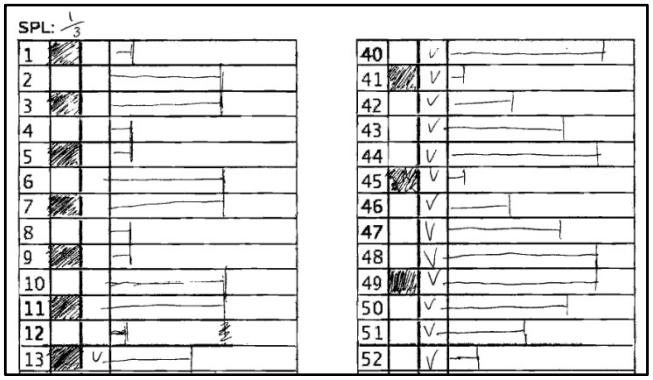
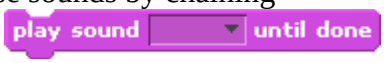


Figure 9. Notation for playing the lever drumitar.

Extending Found Instruments

We weren't done with the sounds the students created. For the next assignment, students loaded their sounds into Scratch [12] and sequenced those sounds by chaining



blocks together. Initially, they just created linear chains like that shown in Figure 10. When they wanted to repeat a sound or just use it again, they simply dragged in another block and selected the sound they wanted it to play.

Figure 8. A Scratch program to play a straight sequence of sounds.

With a bit of experimentation, *all students* were able to create Scratch programs that used looping as shown in Figure 11. With a bit more instruction and encouragement, most were able to incorporate variables, nested loops, and conditional structures as shown in Figure 12, as well.

Figure 11. A Scratch program to play a sequence of sounds using loops.

Figure 12. A Scratch program to play a looped sequence with conditionals.

Finally some students, *with help from each other rather than from the professors*—which indicates true student involvement in the course and is the best way for them to learn: by teaching others—were able to figure out how to do more advanced things, such as playing two or more sounds simultaneously using the

and

and its complementary

blocks, leading to interesting and sometimes quite complex discussions about synchronization.

There are a lot of CS concepts at play here, and we use that word "play" intentionally. The Scratch development group at the MIT Media Lab is called the Lifelong Kindergarten Group for good reason. The ability to learn through *thoughtful* play that involves the use of creativity is at the heart of what we are trying to achieve. The music and arts students learn about computing, to be sure, *but so do the computer science and engineering students.*

Using a visual programming environment like Scratch forces CS majors—who have been "brought up" on languages like C/C++ and Java and text-based coding environments—out of their comfort zone. It is amazing how many of them stumble when they discover that a Scratch loop doesn't provide access to its index (counter) variable. It's pretty easy to implement a counter themselves, but solving of this problem requires a bit of creative thinking. In addition, explaining what they're doing to their non-technical peers not only increases their partners' understanding, but solidifies their own, as well. As the saying goes, "If you really want to learn something, teach it to someone else."

Sound Thinking builds on the Found Instruments project and its related assignments by introducing MIDI concepts and generating music using Scratch's various "sound" blocks (see Figure 13). We have created a number of different types of assignments using these blocks, including having students create a composition based on only major 2nds and perfect 5ths (to break Music majors out of their Western music comfort zone), writing algorithms to transpose lists as either MIDI values or interval deltas into different keys, and coding multiple parts that need to be carefully synchronized.

These and other assignments are described in detail at soundthinking.uml.edu. Through these assignments, not only do Music majors learn about computing, but CS students simultaneously learn about music.

INTERDISCIPLINARY TEACHING

One measure of the success of our work is the lasting effect it has on students. This is difficult to assess, but the number of students who come back to us semesters later to tell us how they applied the concepts they learned in a different context gives us confidence that at least some of the activities we developed resulted in lasting effects. We are currently working to devise more rigorous evaluations to substantiate this belief.

Figure 13. Blocks available from the Scratch Sound panel.

In addition, the effects of our interdisciplinary experiences were not limited to the students. The professors also learned from each other, not only about discipline-specific content, but also about teaching and pedagogy. As a result of the collaboration, the lead author felt totally revitalized by the experience and made significant changes to the way he teaches even his regular CS courses. The NSF evaluator of our *Performamatics* project wrote in her final report:

> One CS faculty member ... changed his approach to teaching significantly in some situations, assigning more open-ended projects, a change well received by students. ... Change in faculty is an essential but often overlooked element of institutional and curricular change.

The professors' experiences in teaching with each other were so positive that they continued to do so even after the original NSF funding expired. Then In 2011 we were awarded a grant from the NSF TUES program to disseminate our work in a series of workshops for interdisciplinary pairs of professors. The first of these free workshops will be offered on June 21-22, 2012. Faculty interested in attending are invited to visit www.performamatics.org for further information and to apply.

Our explorations of ways to bridge the gaps in computing+music education are really just beginning. We believe that there are many more ways to introduce arts majors to computing and science and engineering majors to the arts, and that our approaches offer effective ways to work toward that goal in an undergraduate institution. We are constantly working to improve our current approaches and to extend our work into more advanced offerings that move into live coding [13-15] and text-based music coding environments such as SuperCollider, Impromptu, Processing, and Max/MSP.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation Awards No. 0722161 and 1118435. In addition to the authors, Dr. Fred Martin and Dr. Sarah Kuhn of the Univ. of Massachusetts Lowell and Dr. Scott Lipscomb of the Univ. of Minnesota are members of these project teams. Dr. Lipscomb thoroughly review of an early draft of this paper and provided significant suggestions for improvement. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Please see www.performamatics.org for further information, including how to apply to attend our NSF-sponsored workshop.

AUTHOR BIOS

Jesse Heines is a Professor of Computer Science at UMass Lowell with a strong interest in music and its power to interest students in computing. He was PI on the original NSF CPATH Performamatics award and is PI on the current NSF TUES award. Gena and Jesse are writing a book on interdisciplinary teaching that is currently under contract with Oxford University Press.

Gena Greher is an Associate Professor of Music Education at UMass Lowell. Her research focuses on creativity and listening skill development in children, and examining the influence of integrating multimedia technology in urban music classrooms. Before entering the education profession, Gena was a music director in advertising, working for several multinational agencies producing jingles and underscores for hundreds of commercials.

S. Alex Ruthmann is an Assistant Professor of Music Education at UMass Lowell. Beginning as a middle school music teacher and computational musician, he now teaches courses at the intersection of music education and arts computing. Alex's research explores social/digital media musicianship and creativity, as well as the development of technologies for music learning.

Brendan Reilly is an undergraduate Computer Science major at UMass Lowell. He has played bass since grade school, participating in every musical group available to him.

REFERENCES

1. M. Zyda, "Computer Science in the Conceptual Age," *Comm. of the ACM*, vol. 52, no. 12, 2009, pp. 66-72.
2. J. M. Wing, "Computational Thinking," *Comm. of the ACM*, vol. 49, no. 3, 2006, pp. 33-35.
3. F. Martin, G. R. Greher, J. M. Heines, J. Jeffers, H.-J. Kim, S. Kuhn, K. Roehr, N. Selleck, L. Silka, and H. Yanco, "Joining Computing and the Arts at a Mid-Size University," *Jrnl. of Computing Sciences in Colleges*, vol. 24, no. 6, 2009, pp. 87-94.
4. R. Beck and J. Burg, "Report on the LIKES Workshop on Computing and Music," ACM SIGCSE Music Committee, 2011.
5. W. Chung, E. Fox, S. Sheetz, and S. Yang, "LIKES: Educating the Next Generation of Knowledge Society Builders," in *Americas Conf. on Information Sys-tems*, San Francisco, CA, 2009.
6. M. Edwards, "Algorithmic Composition: Computational Thinking in Music," *Comm. of the ACM*, vol. 54, no. 7, 2011, pp. 58-67.
7. A. R. Brown and A. Sorensen, "Interacting with Generative Music through Live Coding," *Contemporary Music Review*, vol. 28, no. 1, 2009, pp. 17-29.
8. M. Guzdial and B. Ericson, *Introduction to Computing and Programming in Java: A Multimedia Approach*. Upper Saddle River, NJ: Prentice Hall, 2005.
9. A. Hugill, *The Digital Musician*. New York, NY: Routledge, 2008, p. 255.
10. H. F. Silver, R. W. Strong, and M. J. Perini, *Strategic Teacher: Selecting the Right Research-Based Strategy for Every Lesson*: Assoc. for Supervision & Curriculum Development. Chapter 13, 2008.
11. H. A. Yanco, H. J. Kim, F. G. Martin, and L. Silka, "Artbotics: Combining Art and Robotics to Broaden Participation in Computing," in *AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, Stanford, CA, 2007.

12. M. Resnick, J. Maloney, A. Monroyhernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch Programming for All," *Comm. of the ACM*, vol. 52, no. 11, 2009, pp. 60-67.
13. A. Brown. (2011, 11/7/2011). *Generative Structures Performance*. Available: vimeo.com/26193440
14. A. Ruthmann. (2011, 11/7/2011). *Live Coding & IchiBoard-Enhanced Performance*. Available: www.youtube.com/watch?v=qehSEroHn4E.
15. A. Sorenson and A. Brown. (2007, 11/7/2011). *aa-cell Live Coding at The Loft 1*. Available: www.youtube.com/watch?v=OBt4PLUv2q0.