1 OPEN SOURCERY
Computer Science and the Logic of Ownership

Marvin Diogenes, Andrea Lunsford, and Mark Otuteye

This chapter participates in an increasingly important and sometimes acrimonious debate over how texts can be best circulated, shared, and, when appropriate, owned. Of course, these issues of textual and now digital ownership are not new. They have grown up, in fact, alongside print literacy, capitalism, and commodification, with copyright protection growing ever more powerful: the current protection extends to life plus seventy years for individuals or ninety-five years for corporate entities.

With the rise of the Internet and the Web, many hoped for a new era of democratization of texts that would challenge the power of traditional copyright: anyone could be an author; anyone could make work available for sharing. And to some degree, this hope has been realized, most notably in venues such as Wikipedia and in the explosion of blogging and social networking sites. Yet commercial interests are working incessantly to control the Web, and Hollywood, the music industry, and entities such as Microsoft now concentrate their efforts on getting Congress to protect digital works of all kinds. Democratic sharing of knowledge in this atmosphere is difficult, to say the least. Yet unofficially, people everywhere are sharing information and trading goods, often without any citation (or payment), from peer-to-peer music file sharing to journal article swapping to the open-source code movement in computer science.

For this project, we found computer science to be a particularly fascinating scene for questions about textual ownership.

Why? First, computer science (CS) is a new, rapidly evolving field, one in the process of defining itself in relation to traditional ideas about intellectual property, collaboration, shared knowledge, and textual production and textual value. We were drawn to this dynamic, and to the frontier mentality that seems to be an important element of the developing field's sense of itself. Moreover, at our institution, as at many others, CS is a large undergraduate major (among the largest at Stanford), and CS courses have very high enrollments from other majors, too. Thus, a significant number of our first-year composition students will eventually have at least some contact with the field. Another interesting element is that at our institution, as at many others, students in CS courses account for a disproportionately high share of the total number of plagiarism cases, and we wondered why that was the case. As these cases generally hinge on the improper appropriation of code, we found ourselves increasingly focused on the nature of code in CS, its complex relation to what qualifies as an idea, and its parallels to the kinds of texts that writing teachers and humanists work with every day.

We began our investigation by identifying eight lecturers and senior research faculty in CS who agreed to talk with us about a set of questions we sent them in advance. (See Appendix A to this book's introduction for the questions. The interview questions were adapted by Andrea, Marvin, and Claude Reichard, director of the writing-in-the-major program at Stanford. Claude was also a member of the interview team, and we thank him here for his essential contributions to that stage of this project.) These eight interviewees teach the full range of CS courses, from first-year through graduate level. Their work includes textbooks and articles as well as code, and one faculty member formerly served as co-chair of Stanford's Judicial Affairs Review Board. One of the informants works in the computer industry as a software developer. In each case, we met with our colleagues in their offices at Stanford for at least an hour, recording their remarks and later transcribing them.

Almost immediately, we could see the commitment these scholars had to the concept of open source (in general, the idea that source code is available for others to use or modify; see www.opensource.org) and to making their work available as widely as possible and as quickly as possible. These commitments lead to a tension, however, one that pits the desire to make a free space (free both in the sense of open to all who care to contribute and also free of charge) for publication of cutting-edge work against the corporate, institutional desire to control the expression of knowledge through traditional publication practices and copyright. We also began to gather information about CS ways of doing things, of their use of boilerplate, conventions, and commonplaces in code that no one owns and everyone uses. The more we talked to the respondents, the more we came to know the features and special quality of their common space—what we might call the Burkean parlor of computer science. What follows is our attempt to hear a whole range of voices and to use them to explore issues of textual ownership, particularly in CS, but also in other cultural contexts.

PARLOUS PARLORS

JZ, a CS interviewee: Here's an issue we think about: as the tools have become more and more sophisticated, we have the students do more and more things that build on the work of others. Now that work is often public domain, standard-issue, but it creates an interesting tension; we say they need to write everything themselves, but there is a lot of code that we use ready-made, and we need to make sure they know what they are allowed to use, what parts they need to build independently. Sometimes people reinvent the wheel. A lot of that code is repetitive, not interesting, you don't want students to write it anyway. So you teach them to indicate where we got this stuff from, and then build on top of that.

PY, a CS interviewee: And then of course, there's the whole issue that on the Web everybody steals everything. It's extremely easy to steal stuff. I play computer games, so I read computer game websites. And sometimes you'll see text just stolen, word-for-word, put on someone else's website. You're just like, "Okay." No attribution, no nothing. It's all hobbyist stuff, but even so, it's clear that . . . I don't know if it's a generation issue or what, but some people think nothing of just taking text from other people.

These computer science scholars are talking about two issues that came up over and over again in our conversations: the desire to keep students from having to do busywork by letting them use another's code as long as they give attribution, and the recognition that many people, including lots of students, view what's on the Web as available for use—without citation. (There's also an interesting parallel and perhaps a contrast in the attitudes toward code and word—PK notes that "on the Web everybody steals everything," but he seems to voice a special ire towards those who steal text. Apparently computer science students need to learn to acknowledge the sources of their ready-made code, but they should already know better than to appropriate text verbatim.) In these remarks, the interviewees thus point up the huge change that has taken place in terms of peer-to-peer sharing and the clash between what Lawrence Lessig calls a "permissions culture," which values absolute protection, and a "free culture," which values more open sharing of resources. The Record Industry Association of America (RIAA), for example, argues strenuously that downloading a song is tantamount to stealing a CD, while students and many others argue for a more nuanced understanding of what constitutes intellectual property (perhaps motivated by both the immediate desire to access songs easily and by long-term questions of control and ownership of music). In *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture*, Lessig (2004) outlines four distinct types of sharing and explores the ethics of each. In response to the RIAA, he says "If 2.6 times the number of CDs sold were downloaded for free, yet sales revenue dropped by just 6.7 percent, then there is a huge difference between downloading a song and stealing a CD" (71). If teachers agree with Lessig's analysis, acknowledging the claims of businesses while seeking more complete contextual information about their profit and loss, then we have an obligation to be talking with our students about these issues and helping them to articulate an informed ethic of peer-to-peer sharing.

Listening to CS scholars talk about their community and its norms led Marvin and Andrea to spend some time thinking about the assumptions we hold as scholars of rhetoric and writing studies, and about the various Burkean parlors in which we find ourselves participating. How could we begin to try to fit what the computer scientists were saying about code into what we knew about text? Sometimes correspondences appeared; at other times, we encountered distinct differences, or what seemed to be brick walls blocking understanding—so much so that we began to think not of a parlor but of a veritable carnival of parlors, which sometimes overlap but many times do not. At this point, we were fortunate to engage a former student and recent Stanford graduate to work on this essay with us. Mark Otuteye came to Stanford a computer science major but eventually graduated in African and African American Studies and English, with an emphasis on poetry. He then had an intense internship at Google, and was in the second year of a Marshall Fellowship at the University of Edinburgh, where he was working on poetry and computer science, during the writing of this chapter. As one who participates in the conversations of both humanities and computer science parlors, Mark has a special perspective to bring to this project. He describes his introduction to Google's parlor, and its attitudes toward intellectual property, in this way:

Mark: My second day on the job at Google, I had my first personal run-in with intellectual property and computer science. I have a Web page at www.markotuteye.com/google.htm that discusses ten products I thought Google should develop. I had written the page way back as a way to study for the many Google interviews I was to have. At the end of the page, I had a comment box where visitors could tell me their ideas about products Google should develop. I was proud of my comment box because it was my first attempt at building interaction into a public site. When he saw the site, Avichal, my mentor at Google, warned me that the comment box produced a conflict of interest now that I worked at Google. If someone were to submit an idea that was similar to a product Google was already developing in-house, that person could sue me after the product launch. "Oh," I said. Avichal suggested that I add some text to the site protecting me from such a lawsuit, but I thought that I would rather just take the box out because it would be safer. The price of interaction on the Internet is an acute awareness of the kind of intellectual property protected by patent laws. And, now that I work at a company which must be very open internally (for innovation) but very opaque externally (for security), I'm getting a rapid education in the *do's* and the *don'ts* of IP.

As Mark's experience demonstrates, in the corporate parlor, talk can be hazardous, ownership of ideas contested, legal remedies pursued. As the three of us immersed ourselves in the interview transcripts, we found ourselves hearing voices from other conversations about these fraught questions, voices that led us in a number of directions. Given our non-technical backgrounds and interests in popular culture, we began to make connections between the questions we asked the computer scientists and our own lived experiences, and we began to see how our interviews related to a larger conversation about ownership and control of ideas, texts, words, and codes. We also began to write together on writely.com (now Googledocs), where we could generate texts simultaneously and enter each other's texts. This technology led us to a free-wheeling meditation on concepts of ownership that we decided to weave together with the voices of our colleagues in computer sciences and other voices we hear around us every day.

In considering the key features of the computer science parlor, we sought ways to articulate and perhaps reconcile the tension between having a toolbox (the way a poet or lyricist or writing teacher might, parallel to those that new CS students are expected to develop as part of their apprenticeship) and generating and owning an idea or code. We contemplated what it means to be a member of community that owns things together and what it means to create as an individual, whether the object owned be code, a poem, a spoken-word piece, a song lyric, a joke, or a recipe.

RM, a CS interviewee: The basic issues: can you patent an idea for software, can you patent an algorithm, which is just a mathematical expression of an idea on its way to becoming a piece of software, or do you patent the software itself? The dividing line is not well-defined. I've patented ideas; for instance: I and a couple of students had an idea of doing a similarity search. When you represent objects in a computer, you represent them as a kind of number. You take a description of a table and represent it by its greatest parameters so that it becomes a sequence of numbers; you view that as a point in higher-dimensional space. Then the question: I have a huge database of these objects represented, how do I find similar objects? That becomes hard because of recursive dimensionality; instead of comparing the parameters to every object in the database, which is slow, you need to come up with something more clever. We came up with a mathematical function that takes these object descriptions and collapses them into small sets of objects, so that you compare only to the small sets. I and the two students hold that patent—well, who holds it? Stanford basically owns the idea, even though my name is listed as the inventor; we have an office of technology licensing, which handles the whole process. Whatever money

> they make off it by licensing the patent to industry, they split that up [according] to a certain formula: I'm making this up, but, say, one third to me, one third to my home department, the rest divided between the school of engineering and the university. They also pay the cost of filing the patent, which is not a small amount—ten to thirty thousand dollars. So they decide whether to file the patent; I cannot license it myself.

We're very interested here in RM's meditation on what can and can't be patented and the large grey area that currently exists in this evolving field. Other interviewees made the same point, arguing that the law is simply not yet able to distinguish effectively what is of most value in CS. In any case, as RM notes, for those working at universities, it is the institution that usually holds the patent—though the profit gained will be shared with the "inventor." What's clear is that the monetary stakes can ultimately be quite high if an invention turns out to solve a problem that needs to be solved. That context of potential vast profit suggests that scholars in CS must find ways to teach their students about these complex issues and about the grey areas of the existing law.

Mark: My grandmother used to make her own intricately spiced stews. My grandfather used to make pots and at one point he made a special mold that yielded pots perfect for cooking up stews. With a pot made from this special mold, my grandmother created a stew so piping hot and tasty that no one else in her neighborhood could figure out how she'd done it. Everyone could see the stew and taste the stew, but no one could figure out the recipe. It was Grandma's signature recipe.

On top of that, no one could figure out how to get the stew to cook in quite the same way since they didn't have grandfather's special stewing-pot. Both the vessel (the pot's mold) and the content (the stew's recipe) were "protected" or secret from the neighbors. This is analogous to the state of a Word document on the Web; both the vessel (the .doc file format) and the content (the words in the file) are protected and cannot legally be reproduced or edited without citation. This is what's currently the norm.

Well, my Grandma valued improvisation, so she gave her neighbors the recipe. Although Granddad didn't tell folks how to make their own pots by sharing his mold, he did make pots for any neighbor that wanted one. Armed with the recipe and the pot from Granddad, neighbors were free to make Grandma's stew, and innovate on top of it. The vessel (the pot's mold) is protected, but the content (the stew's recipe) is free or open. This is analogous to the state of a Word document with Creative Commons attached.[1] Given a .doc made from Microsoft Word's "mold," anyone can creatively "remix" the words that I include in the document.

Finally, my grandpa decided that it was in the best interest of the community if he taught folks how to make their own pots. So he shared the mold. Now both the vessel (the pot's mold) and the content (the stew's recipe) were "open source" in the community. This is analogous to an OpenOffice document with Creative Commons attached.

JU, a CS interviewee: Writing the code is not as important as having an idea of what code to write. The primary motivation is either how to do something, an algorithm, or "people would like it if you could do that." The famous case is the first spreadsheet: it was PC technology. . . . There were spreadsheets in the 1970s that would crunch numbers, but you needed a programmer to set them up. Then a business person said "here's what we need to do" and paid a programmer $25,000 [to create a spreadsheet program for the PC] and then made millions and millions. Some eyebrows were raised; maybe justice wasn't done; but a deal is a deal.

1. Creative Commons is an alternative to traditional copyright created by Stanford law professor Lawrence Lessig. A Creative Commons license "helps you keep your copyright while inviting certain uses of your work—a 'some rights reserved' copyright." See http://creativecommons.org.

Andrea: I was struck by the different system of values underlying JU's story about the first spreadsheet and Mark's story about his grandmother's stew, both of which show the crucial significance of cultural context to an understanding of intellectual property. But these stories don't just mark a difference between U.S. and African understandings of ownership. In fact, Mark's story immediately made me think of my maternal grandmother, Rosa May Iowa Brewer Cunningham, who made a quilt for every one of her children, grandchildren, and great-grandchildren, up to her death at the age of 96. But she did not make these quilts alone. Rather, she and her rural Tennessee quilting circle worked together—they were almost always working on a quilt or, more accurately, several at a time. Not that my grandmother didn't do a lot of the work of preparing alone: she was constantly on the lookout for scraps of fabric she could cadge or a piece of clothing or used flour sack she could cut up for the designs. Mostly, she and her friends used these pieces to make a quilt in a traditional design; the double wedding ring was one of my granny's favorites. But occasionally she or a friend—or a group of friends—would create a new design to quilt to. One I know looks a bit like a postmodern version of the log cabin quilt.

So to use Mark's language, the quilt design is the vessel, and the pieces put together are the content. Or is the design of the quilt—and all the talk that takes place around the making of each quilt—the code, and all the pieces and the slight variations stitched into each quilt are the content? In any case, no one "owns" the quilt designs because they have been developed through centuries of collaborative cultural practice. So those moments of invention fall outside the code of copyright and instead participate in the concept of open source.

Marvin: I'm the only one of the three of us who didn't have the opportunity to observe and learn from a grandmother, so I'll shift the conversation to another realm of shared cultural practices—in this case popular culture, or the sprawling family created by mass media. Here too we can see the circulation of vessels that become property held in common by all of those who add content through participation in a particular culture.

The comedy troupe called The Village Idiots appeared on *Don Kirshner's Rock Concert* in the seventies. Here's an account of a Village Idiots skit I saw late on a Saturday night at some point during that decade, though my own predilections certainly color what I remember. I'm interested in what the skit tells us about the form, or vessel, for a joke—in this case considering a joke a specific way to make meaning and comment on one's experience of the world—and how such a vessel comes to be invented, shared, and ultimately owned.

The skit begins with several cave-people in a cave, dressed in animal skins. They find a cigarette lighter, a cheap one available at the counters of convenience stores. (The unapologetic anachronisms in the skit are part of its indelible charm, at least for me.) One of the cave-people flicks the lighter, getting a flame, which terrifies all of them. The inquisitive one drops the lighter, and all scurry away, leaving it on the cave floor. At this point Ug walks in to the scene. Ug seems to have reached a later stage of evolution. He calms everyone down, picks up the lighter, and beckons them to come nearer. He flicks the lighter on, saying "Fire good. Fire cook chicken." (A rubber chicken has wonderfully been included on the set.) This indeed calms the rest of the clan, and they hold the rubber chicken over the lighter for a moment.

Ug announces a new discovery, and asks the group to listen carefully; they form an audience in front of him, squatting in the dirt. He's clearly proud of himself, preening in his animal skin as he prepares to perform. The performance begins thus:

Ug: Knock knock.

Clan: Come in.

Ug corrects the code. "No, no, no," he says. "Knock knock," he articulates, gesturing to himself. "Oo ere," he continues, gesturing to the clan.

Ug: Knock knock.

Clan: Oo ere?

Ug: Ug.

Clan: Come in.

Exasperated but persistent, Ug corrects the code again. "No, no, no," he says. "Knock Knock," he repeats, with the same gestures. "Oo ere," pointing to the clan. "Ug," he says, pointing again to himself. "Ug oo," pointing to the clan.

Ug: Knock knock.

Clan: Oo ere?

Ug: Ug.

Clan: Ug oo?

Ug: Ug-ly.

He pauses, waiting for the laugh. The clan looks at him expectantly, awaiting more direction. He tries to explain. "Joke," he says. "Joke." "Ug-ly," he repeats, pounding on his chest, thrown off by the clan's failure to appreciate the cleverness of the joke's form and the self-mocking payoff. "Ug-ly. Ug-ly. Ug-ly!"

The clan still doesn't get it, but they want to please the seemingly advanced Ug. "Joke," they say, questioningly, struggling with the concept. They pick up the previously discarded rubber chicken. "Joke good?" they ask. "Joke cook chicken?"

For The Village Idiots—and aren't we all members of the troupe some of the time—code isn't easy. First people have to learn the boilerplate, the standard structure. Then they have to weave in a flash of brilliance and hope everyone is dazzled. How do writers of code learn to reconcile the tension between having the boilerplate, the toolbox, and generating a good idea or piece of code?

PY, a CS interviewee: It was a tic-tac-toe program, and the students said, "Well, there's only one way to write a tic-tac-toe program in computer science, so of course all of ours are exactly alike," which is also totally false. Like, clearly, you guys did not learn anything in this class. And then they claimed that if they had come up with a different tic-tac-toe program, I would have just gone on the Web and found another program that worked exactly the same way theirs did. They never did get it, and they

> accused me of all sorts of stuff. They wrote this nasty letter to the Honor Code Committee. The Honor Code Committee got really pissed off at them, and made them write an apology to me. And to this day, I think at least one of them still denies that they copied it. But it literally was 100 lines of code exactly the same. And I still don't get what they were thinking. It's just bizarre. I totally see them copying, but I don't understand how they thought that once we brought it up they could just claim that they didn't copy it. It's just bizarre.

PY's bemusement and consternation are overt and heartfelt, echoing his earlier response to the stolen language on the hobbyist website discussed above. While we likely share PK's reaction to stolen words, we wonder whether the students' act of sharing the tic-tac-toe code is quite the same as stealing a CD, to return to the example from the RIAA. In CS, our interviewees told us over and over, it is the norm to use code that is out there to save time and steps: why reinvent the wheel over and over and over again? Yet the students aren't supposed to do this kind of sharing, on the theory that they need the practice of creating code from scratch. We take the point, though.

Andrea: These stories throw into stark relief the traditional humanities view of textual ownership, the by-now-familiar scene of the lone writer in the garret, struggling to compose an utterly unique text, marked with the author's genius, owned outright, and deeply protected by the web of intellectual property laws that have grown like kudzu during the last three-hundred-plus years. This is the "author" declared dead some thirty-five years ago, though the death announcement by humanists such as Roland Barthes and Michel Foucault turned out to be premature: today copyright laws are more extensive and longer-lasting than at any time in the past, and, in fact, major content producers (think Disney here) have appropriated the mantle of authorship and used it to close off larger and larger areas of creative endeavor. As the power of such authorship has grown, the public commons has shrunk; the Digital Millennium Copyright Act sanctioned the entertainment industry's appropriation of authorship and, along the way, reduced the fair use principle to a mere whisper. At the same time, scholars in the humanities, working in a relatively new field usually called "the history of the book," have resuscitated the author and theorized extensively on human agency and its relationship to textuality. Also at work in reclaiming agency have been feminist and post-colonial scholars.

If those in the humanities still cling to the possibility (or necessity) of authorial power and ownership of text, they have also moved toward a little more acceptance of collaboration. Universities as diverse as Stanford, Ohio State, and Chicago now all have "collaborative" humanities centers, which call for and fund collaborative research projects. And though the single-authored book is still the sine qua non in tenure and promotion decisions within most humanities departments, collaboratively produced articles and books are gaining some acceptability. Perhaps most important, scholars in the humanities have come to understand that very rigid and exclusionary copyright laws actually keep them from doing their work: if everything is protected, then how can one write criticism? Professor of English Carol Shloss is particularly eloquent on this issue, as a book she had worked for years to write on James Joyce's daughter Lucia was nearly blocked by the Joyce estate, which claimed ownership of so many of the sources that Shloss wanted to use that her work was put in serious jeopardy. Those sources were Shloss's "collaborators" and she needed them desperately. As her story shows, the same copyright that protects her "authorship" can be used to prevent her access to the materials she needs to establish herself as an "author." This is a potential contradiction at the heart of what we are exploring in this essay. With these contradictory tensions in mind, we were particularly interested in Mark's ideas about invention, ownership, and the poetry he writes.

Mark: Whenever I sit down to write a new poem, I first read over my previously written poems. I also read over the many African American poetry books on my bookshelf to seek inspiration from those who came before me. Because I'm both a poet and a computer scientist, I brought my two passions together and wrote a program called Heteroglossica to help automate my invention process. Heteroglossica searches over all my previous poems, blogs, essays, and email and presents me with start-material for new poems. For example, if I want to write a poem protesting the war in Iraq, I can type in "war in Iraq" and get back snippets, sentences, and lines from my previous work that have to do with the war in Iraq. Then I can craft that start material into a new poem. The best thing about Heteroglossica is that it allows me to search over multiple authors. For example, I currently have the program configured to search Shakespeare's plays and Tupac's lyrics, in addition to my own work. If I search for something like "death," Heteroglossica pulls lines from all three of these voices and populates a text box with 20 or so of the most interesting lines. Then I can edit that material into a new piece. Often, the hardest part of writing a paper is writing against the dominant thoughts and words of established authors. Heteroglossica encourages me to think of all text as open source.

The following code from Heteroglossica creates a textbox in Internet Explorer and puts lines from Tupac, Shakespeare, and me into that textbox. In writing the code, I'm aware of and sensitive to multiple audiences: the writer who will use the program, the browser (Internet Explorer or Firefox) that will show the Web page, and the server that will search across the three authors. For example, this next line is for the writer who will use the program. It lets her know that the text in the textbox can be edited.

```
echo “Edit these lines into a new poem:”;
```

The audience for this next line is the browser. It's the line that creates the textbox and puts a black border around it.

```
echo "<textarea name=\"main_text\" rows=\"40\"style=\"border
:1px solid #000000; width:100%; padding:10px\">";
```

The audience for the next chunk of code is the server, the computer that actually does the work and sends the results to your browser. These lines of "for" loops and "if" statements are supportive, boilerplate language that are written hundreds of times in programs. Someone trained in computer science would scan over these lines quickly looking for Heteroglossica's active ingredient or engine.

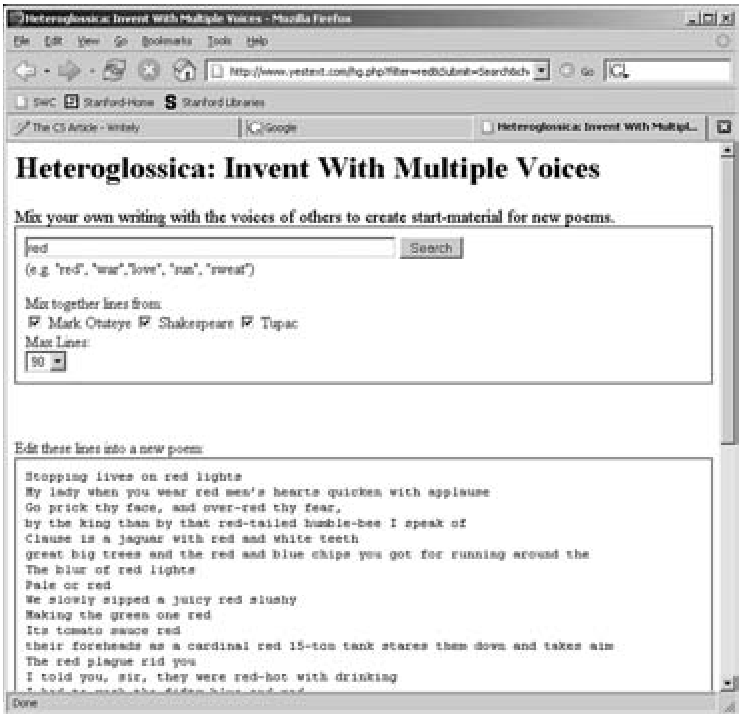
```
for ($z=0; $z<count($corpuses); $z++){
if (count($results_array) > 0 && $results_array[0] != ""){
for ($i=0; $i<$total_results/count($corpuses);){
$k = rand(0, sizeof($results_array));
if (str_word_count($results_array[$k]) > 0){
$all_results[] = $results_array[$k]."n";
$i++;
```

The line below is Heteroglossica's "engine."

```
$results = shell_exec("grep -i -h -w $query corpus/$corpuses[$z] |
sort -b -f");
```

The engine of Heteroglossica is "grep," a pre-written function well-known to computer science folk. Grep searches through lots of text and finds lines that include a given term. Because I relied on pre-written, boilerplate language for even the core functionality of my program, I announce to anyone reading my code that I am more interested in designing the experience of using the software (like interior design for a house) than in implementing a new way to search across texts (like designing the plumbing for a house). The logic of my code is expressive of my rhetorical situation and, to some extent, my individual personality.

> Finding one's voice isn't just an emptying and purifying oneself of the words of others but an adopting and embracing of filiations, communities, and discourses. Inspiration could be called inhaling the memory of an act never experienced. Invention, it must be humbly admitted, does not consist in creating out of void but out of chaos. (Jonathan Lethem "The Ecstasy of Influence," *Harper's Magazine*, February 2007—a pastiche of text from George Dillon, Ned Rorem, and Mary Shelley.)



Heteroglossica: Invent With Multiple Voices

Mix your own writing with the voices of others to create start-material for new poems.

red

(e.g. "red", "war","love", "run", "sweat")

Mix together lines from:
Mark Otuteye ✓ Shakespeare ✓ Tupac ✓
Max Lines:
90

Edit these lines into a new poem:

```
Stopping lives on red lights
My lady when you wear red men's hearts quicken with applause
Go prick thy face, and over-red thy fear,
by the king than by that red-tailed humble-bee I speak of
Clause is a jaguar with red and white teeth
great big trees and the red and blue chips you got for running around the
The blur of red lights
Pale or red
We slowly sipped a juicy red slushy
Making the green one red
Its tomato sauce red
their foreheads as a cardinal red 15-ton tank stares them down and takes aim
The red plague rid you
I told you, sir, they were red-hot with drinking
```

Figure 1.0. Heteroglossica in the Firefox browser.

Lethem's article, which doesn't announce its reliance on pastiche until its conclusion, dramatizes in another form what Mark's Heteroglossica program achieves systematically with the aid of an algorithm. The chaos Lethem describes is deeply collaborative, as is the work in CS. Students in undergraduate courses are encouraged to work together and to get help when they encounter problems writing code. In introductory classes, students are expected to write their own code, informed by discussion, and plagiarism cases generally involve students using code written by others without citation. Our informants consistently reported that plagiarism in CS is easily detected—in other words, there's no gray area when it comes to code. At Stanford, a program has been developed that finds copied code, even if the plagiarist has tried to disguise the theft by changing surface elements of the code. (One interesting element of this issue is the idea that the way an individual writes code is very distinctive.) This openness to collaboration remains constant as students advance in the field. It is standard to ask for help; it is standard for people to work collaboratively; what's not acceptable is using someone else's code without proper citation.

Except in the case of team project assignments, all the guidelines for collaboration we saw drew a very sharp line between pre-code-writing activities and the actual code-writing, with collaboration on the latter categorically forbidden—as it was in the tic-tac-toe example above. Presumably, if students copied code but *did* cite the source, they would not be charged with Honor Code violation, but it didn't seem like they would get much/any credit either. The interesting disjunction, then, is that, in most of their coursework, collaboration is expected to suddenly stop when students start writing, whereas that is definitely not the expectation/practice in industry—which most of the students probably well know and where most of them are headed.

Some interesting parallels with practices in the humanities come to mind here. Stanford undergraduates enroll in a year-long Introduction to the Humanities (IHUM) program during their first year. They learn to engage with texts (mostly canonical texts, though some IHUM courses are moving to visual texts and, in one case, the online environment of Second Life) through close reading, informed by two hours of lecture and two hours of small group discussion each week. How do beginning humanists acknowledge how they're collaborating when they compose and turn in for evaluation their single-authored work? Students are explicitly told to cite lecture and discussion when they reference them in essays and to avoid secondary sources, to avoid borrowing ideas. A premium is placed on originality, a distinctive engagement with the text that doesn't reproduce interpretations the students have already heard. Like the CS students, they are encouraged to work together on brainstorming and to visit the writing center for consultations about their drafts-in-progress: but what they write is supposed to be theirs and theirs alone.

MJ, a CS interviewee: In the early courses, it is all individual work. You are expected to implement everything that you are told to implement and use tools the way you are told to use them. In the upper-level courses, I expect them to have a toolkit, and I tell them I don't care where they get it. By 143 or 148, they are expected to know how to do those things, so the rules of 106 no longer apply. For graphics, if I ask them to implement a particular graphics algorithm, that's where I draw the line; what we are learning about, you can't copy.

Say they take 148 or 248 where they are building a graphics toolkit; then they get to an upper level course and they use that toolkit. So it keeps building. Then they get to industry, and they have their tool kit. If I am looking for copyright infringement, I am not going to look at the toolkit, which counts as shared knowledge at that point. So the toolkit will have things they have developed and also things that they have gotten from other sources. They will also modify it as they go from job to job—which can get touchy. If someone takes their toolkit plus some more meaty parts from job A to job B, that is not right. We teach them this in 201, which talks about social responsibility, ethics, etc. Sometimes, though, a person thinks "I wrote this, I can take it with me," and that gets them into trouble.

Advanced humanities students have assembled a toolkit of interpretive and analytical moves, which may be recognizable as a kind of code but cannot be patented or owned. It's also worth noting that the industry of work in the humanities does not offer the kind of financial rewards that a life in computer technology can lead to. Humanists learn to do things with words, reaching an audience of readers; programmers learn to help the much larger audience of consumers do myriad things with code. While the invention practices have parallels, the contexts and real world effects diverge dramatically.

Eventually, we began to get a sense of the kind of Burkean parlor where computer programmers/coders spend their time. It's a parlor with whiteboards, a parlor in which visual images and math have status equal to words. A parlor in which newcomers learn by listening and watching. A parlor in which one leaves the toolkit at the door, because the work of coding is done in another room, where the hardware is kept. The parlor is for ideas, for play, for testing out ways of doing things and persuading the rest of the group that one way is the best way. The ability to discover the most effective means of getting things done in a given situation. One might call this a rhetoric of programming.

One feature of such a rhetoric is the strong desire in CS to make research available as widely as possible as quickly as possible. Since knowledge is generated at such a fast pace, the traditional waiting period for publication of new work is not acceptable in CS. Thus, as mentioned earlier, one of the tensions in the field pits the desire to make a free space to encourage the quick spread of new work against the traditional methods that slow down the sharing of knowledge and subsequent synergy of minds focusing together on a problem. Many of the informants assert that this chills creativity.

The conflict has led many in CS to turn to conference proceedings as their main venue of publication. While journals generally want to hold the copyright on articles and may take a long time to get the work into print, work in conference proceedings appears more quickly and the copyright remains with the authors. What seems to have sprung up, then—amazingly quickly, as scholarly practices go—is a system for quickly and freely disseminating work, *but* with agreed upon screening/review process for making sure that work published in proceedings is in fact *cutting edge*. This has in turn led to a shift in what sorts of publications (e.g. journal articles vs. conference proceedings and even textbooks) count for promotion and tenure, and differences in practice in these areas between what the informants refer to as top tier and lower tier CS programs.

Folks in CS also publish much or all of their work on their own Web sites, contributing to the open access feel of CS, and the premium put on free access and free exchange of ideas. While stuff on the Web is easily stolen, such thefts are easy to track; one informant told us about finding his work on forty Web sites, with twenty of them not attributing the work to him. Another informant shared this motto, “Impact, not publication; conferences, not journals,” asserting that the perception in the field is that journals are more likely to be publishing what is already common knowledge, not cutting-edge. It’s important to note that the shift to alternative means of publication does not mean complete openness or lack of standards; the acceptance rate for the most prestigious conferences is 10% or less. Bypassing the traditional journal peer review process has been accompanied by the development of alternative conventions of peer review—for example, the blue-ribbon committees that select papers for each interest group at conferences.

Another interesting question in CS is the determination of what constitutes unique or new insights. One informant cited the “real misunderstanding of what’s unique in CS.” In particular, courts familiar with traditional ideas of copyrighted textual material or patent law don’t know how to evaluate work in CS. Again, there’s a gap between old IP concepts and the dynamic developing context of CS.

What can be patented in CS? What constitutes an idea in CS? How does one determine the difference between an idea and an application of an idea? One informant asserted that applications of the same idea have been patented, and that the patent granting offices simply do not know enough about CS to keep this from happening. What’s important in CS is the idea, not the execution or expression of it in code; as one informant put it, “Writing the code is less important than knowing what code to write.”

Blues and jazz musicians have long been enabled by a kind of open source culture, in which pre-existing melodic fragments and larger musical frameworks are freely reworked. Technology has

> only multiplied the possibilities; musicians have gained the power to *duplicate* sounds literally rather than simply approximating them through allusion. (Jonathan Lethem, "The Ecstasy of Influence," *Harper's Magazine*, February 2007)

Marvin: There's a British documentary from around 1988 about Paul Simon. At one point the slow-talking, near-ponderous interviewer comments that some have claimed that there can be no great art made in rock 'n' roll, because the means are too limited. He asks if Simon has felt these limits. Simon looks at him coolly, at length. He answers "No, I don't agree with that," going on to say that art can be made in any genre, including rock. He says that "rock is about rhythm," and that he can express something lasting by finding his way to the right rhythm. Rock, like code, is a well-defined structure. Much of the verse/chorus/bridge form is the same in most rock songs. The guitar/bass/drums instrumentation dominates. Rhyme is a near-constant. Somewhere in that standard form is the opportunity for cool things to happen. It's difficult to say exactly what the cool thing is. You just know the cool thing is there by the way the song makes you feel, by the way the song makes you move, by what the song allows you to do.

> The thoughts are there inside your head, Teach said to me
> Invention is easy if you take it logically
> Try these heuristics, you can call them strategies
> There must be fifty ways you can discover
>
> Don't you sit lost in thought, just waiting for the muse
> If you trust to inspiration, then the chances are you'll lose
> I'll give you options, then it's up to you to choose
> There must be fifty ways you can discover
> Fifty ways you can discover
>
> Develop the knack, Jack
> Make a new plan, Stan
> Use the topoi, Roy
> Just set your mind free

Try the pentad, Brad
Freewrite till it's not bad
Idea tree, Lee
Just set your mind free

("Fifty Ways You Can Discover," The Composition Blues Band)

The Composition Blues Band was formed in the early 1990s, motivated by the following (borrowed) (stolen) (reimagined) narrative: Imagine you enter a jam session. You come late. When you arrive, others have long preceded you, and they are engaged in a heated jam, a jam too heated for them to pause and tell you exactly what it is about. In fact, the jam had already begun long before any of them got there, so that no one present is qualified to retrace for you all the songs that had gone before. You listen for a while, until you decide that you have caught the tenor of the set; then you put in your oar. Someone answers with a verse; you answer with a verse of your own; another riffs off of your chorus; another takes a solo off the bridge, to either the delight or dismay of the room, depending upon the quality of the player's chops. However, the jam is interminable. The hour grows late, you must depart. And you do depart, with the jam still vigorously in progress.

Lyricists and musicians learn to jam just as coders do. Bits of code show up in the arcane CS conversation, recognizable to cognoscenti but not to the rest of us who just want to see what happens when we click the application. We don't know if what's underneath the screen is a bass line, a rhythm, or a bit of melody.

PY, a CS Interviewee: Yeah, so text is certainly owned, code is certainly owned, ideas are definitely under dispute. So there's this idea that you can come up with ideas and patent them. There are a lot of people in the computer field that are very unhappy with this, but I do not believe it has, in general, been challenged in court. I could be totally wrong on that, I don't really keep up with this.

But I do know that a lot of people do think that software patents are immoral. I'm not entirely sure what I think. I think there has been a tendency from the patent office to give patents for things which should not be given patents because they really are too generic. So I think at one point Groliers had a patent for people clicking on something. And it was almost like, "Clicking on something, and something happens." I don't think it was quite that loose, but it was generally considered to be extremely loose, and everybody's like, "No, this is really crummy. How could the patent office give a patent for this?" So that is generally under dispute. I think there is a substantial community that does think that software patents are immoral, as I said. I'm not quite sure how this is going to play out. So that's it for that.

KL, a CS interviewee: I think it was a grave mistake of the US patent office to allow these algorithm patents, these business process patents—they seem like a joke to me. Trying to work with these standards bodies, suddenly we are hemmed in by Cisco patenting something that is obvious, and HP has patented something very similar, ditto SUN; trying to produce open source software without infringing on these patents is tough, and these big companies just trade them back and forth in a way that freezes out the startups and the little guys.

Again, we are struck by how much is at stake in CS—and at the size of the grey area in the law. If scholars think of certain patents as "jokes" and others as so misinformed as to be immoral, then perhaps the near future will bring these issues to a head in ways that will resolve some of the uncertainty. Until then, however, those in CS might do well to follow Gerald Graff's well-known injunction to "teach the conflicts." At least then the students would be part of the conversation.

Appropriation has always played a key role in Dylan's music. The songwriter has grabbed not only from a panoply of vintage Hollywood films but from Shakespeare and F. Scott Fitzgerald and

> Junichi Saga's *Confessions of a Yakuza.* He also nabbed the title of Eric Lott's study of minstrelsy for his 2001 album *Love and Theft.* One imagines Dylan liked the general resonance of the title, in which emotional misdemeanors stalk the sweetness of love, as they do so often in Dylan's songs… Dylan's art offers a paradox: while it famously urges us not to look back, it also encodes a knowledge of past sources that might otherwise have little home in contemporary culture… Dylan's originality and appropriations are as one. (Jonathan Lethem, "The Ecstasy of Influence," *Harper's Magazine,* February 2007)

> I don't want to express myself
> Coalesce or confess myself
> Address myself, outguess myself
> Undress, assess, or duress myself
> All I really want to do is get a good grade from you
>
> I ain't lookin' to write too well
> Cite, delight, or recite too well
> Extemporize well, categorize well
> Apprise, surprise, or analyze well
> All I really want to do is get a good grade from you
>
> I don't want to describe my kin
> Explore my sin or delve within
> Be selective or reflective
> Be directive or be effective
> All I really want to do is get a good grade from you
> I don't want to explore the world
> Abhor, deplore, or implore the world
> Valorize, problematize
> Theorize, contextualize
> All I really want to do is get a good grade from you
>
> ("All I Really Want to Do," The Composition Blues Band)

So Lethem uses the example of Dylan, appropriator extraordinaire, to arrive at the possibility that "originality and appropriations" can be one. And what of less nonpareil appropriators, a category that includes most of the rest of us across the board, from rhetoric/composition to computer science, who basically seek a good grade in the eternal classroom of life? We're mixed up, so we remix, making do with what surrounds us.

KL, a CS interviewee: In CS, people build up libraries of routines for solving problems. You can get a sort function from a library without attributing it. When I take an example and build on it, all the original stuff often gets deleted, and then I might remove the copyright from it, but only if I was sure I hadn't left any code. More usually I would be happy to say at the top "portions of this code came from person X."

Marvin: In "Getting Close to the Machine," Ellen Ullman (1997) offers a version of a monkish existence for computer programmers in her account of her time in the field. She shows us an environment in which the key relationship is between the programmer and the machine, not the programmer and other programmers. There is no sense of community, no conversation, no white-boarding. She leaves the field out of a need for more consistent human interaction. She paints her colleagues as eccentric, lacking-in-social-skills, geeky Bartlebys who prefer not to deal with the mess of dealing with other people. There's just code, to them, and the uncomplicated judgment of the machine.

Andrea: In contradiction to Ullman's view, our conversations with computer scientists suggest that they do have a sense of community and that conversation and white-boarding are key elements in their creative process. What leaps out at me from a number of our interviews with them, however, is a web of contradictions in terms of ownership and collaboration. Students should work together, they say, but they must write their own code. Open source is best—but one interviewee was offended to find his work on another's website, unattributed. Students can't cut and paste code—but doing so is a common practice in the field. The "previous work" section of an article is important—but almost impossible to do (remember the architecture analogy here?). I want my name on my code—but lots of people are playing fast and loose with code on the web and I believe strongly in the open source movement.

MJ, a CS interviewee: In the context of source code, there is a set of libraries you might use if you are doing Windows applications. They are Microsoft code, and you use parts of their code in your own code, so any Windows application you might want to write would probably have that. If I was looking at a piece of software and trying to decide whether there was copyright infringement, I wouldn't consider things in libraries. If something was common knowledge, a sorting algorithm that any comp-sci student knows—professional programmers have a toolkit, and that kit has all the most common things that they use every day [examples]. So I wouldn't consider that infringement, you can get it out of any textbook. . . .
It is the nails and the screws of a building; but you still need to make something that does a particular task with unique features.

Andrea: At least some of this tension (it is "mine" versus "we should all have broad and free access") seems inevitable and, in fact, many people in all disciplines go about their work quite happily holding contradictory positions (usually unacknowledged). A case in point: when I was invited to contribute an essay on collaboration and intellectual property to PMLA, I didn't want to write a so-called single-authored essay, so I asked my longtime collaborator and friend Lisa Ede to join me, and we wrote the article together. Our collaborative practice over the years has been to alternate first authorship, with Lisa's name first on one article or book and mine on the next, and so forth. For the PMLA essay, it was my "turn" for first authorship, but just before we sent it off, Lisa found herself really wanting to be first author on this piece for several reasons, including the fact that she had never published in PMLA before. I agreed at once, though later I felt a bit awkward about this: after all, it had been my turn. Lesson learned: old habits—and proprietary feelings of textual ownership—die very hard. Lisa and I both hold collaboration and shared authorship as deeply valued practices. But apparently we also hold on to proprietary instincts as well. I think we're seeing the same kind of echo of proprietary feelings in some of our CS colleagues.

Certainly this essay reflects the tensions and contradictory impulses we have tracked in our conversations with scholars in CS. As we've woven their voices together with ours and those of others such as our grandmothers, Jonathan Lethem, Paul Simon, and The Village Idiots, we have thought about the many ways our text—a pastiche, a pot of soup, a quilt, a tapestry—resists any traditional sense of ownership. We have obviously, then, been playing with these tensions ourselves, calling on others' words or "code," experimenting with a kind of patch-writing of our own, working to create a text that is not linear in the ways of traditional academic argument, even writing in what Winston Weathers called "crots." What would it look like, we have asked ourselves, to make a kind of reference or echo map of every voice that appears in or is alluded to in this text? We envision a veritable Charlotte's Web of sources, a large and somewhat unruly chorus rather than the neat trio referred to in the listing of authors for this essay. And while the three of us take responsibility for the contents of this essay, we do not claim ownership of it except in a shared and collective way. So in the spirit of CS commitment to the open source movement, we put this text out there, ready for others to use it, to make of it what they will.

That is not to say, however, that we believe a student (or anyone) should take this text, reproduce it, and claim it to be theirs. In other words, we welcome readers to join us in swimming in what we hope is a tasty soup of voices, to slalom down the slopes of a quilted intellectual ski run, to bounce around the various parlors described and invoked in these pages, but we draw a line of ownership. As Lessig says, "This kind of piracy is just plain wrong. It doesn't transform the content it steals; it doesn't transform the market it competes in" (66). We come away from our engagement with the CS parlor, then, wanting our CS colleagues not only to recognize the tensions and contradictions that characterize their practices and their pedagogy but also to engage their students in sorting these contradictions out, in aiming to work together to make explicit what should be protected and why, what should be available for use and modification and why. And we take the same lesson for ourselves in rhetoric and writing studies: we need to tell some of the stories we've told here to our students, asking them to contribute stories and experiences of their own as a way of engaging what it means to be an author today, what it means to have—and to share—agency.