Episode 67: Kevin Brock
Pedagogue podcast
Transcript

Welcome to Pedagogue, a podcast about teachers talking writing. I'm your host, Shane Wood.

In this episode, I talk with Kevin Brock about teaching and preparing graduate students to teach technical writing, digital rhetorics, and how understanding the construction of software coding impacts teaching writing.

Kevin Brock is associate professor of Composition and Rhetoric in the Department of English Language and Literature at the University of South Carolina. His research and teaching interests include digital rhetoric, rhetorical genre studies, technical and professional communication, and writing program administration. His scholarly publications include the book *Rhetorical Code Studies* (2019), several chapters in edited collections, and articles in such journals as *Kairos*, *Computational Culture*, *Computers and Composition*, *enculturation*, *JTWC*, and *Technoculture*.

Kevin, thanks so much for joining us.

SW: How do you approach teaching technical writing at the University of South Carolina?

KB: At USC, or U of SC…I know that some folks are going to hear the USC acronym or abbreviation and think Southern California, but in South Carolina we've got a class that's a 400-level class on technical writing. Although, we're just going to start offering this coming fall a 300-level introduction to professional writing course, so there's some overlap in what the goals are for those courses. I teach it not exclusively, but primarily for writing computer science and engineering related documentation, specifically because the College of Computer Science and Engineering at University of South Carolina requires this course explicitly for their students. At any given semester, about two thirds to three fourths of the students are from that college, and the remaining students come from a number of different departments, some English, some Mass Communications, Marketing, Business, and so on.

I try to make sure when teaching this course that even though we have that focus for the majority of students, there's a variety of contexts being discussed, a variety of genres that extend beyond and are not specific to computer science or software development, and attempting to move students from thinking about academic context to professional context, which I think anybody who teaches a course remotely related to technical writing or another professional writing class is going to know is super obvious. But for those who don't, it may be surprising to hear that many students think about all of their writing by the time they're about to leave the university as being university specific, because we've had so many pseudo-transactional assignments where you're really writing to the instructor and you might give some lip service to a real imagined audience.

Trying to really focus them in a way that draws attention to those actual audiences, bringing in real case studies, looking at real world examples of writing that they're doing and not just

textbook hypothetical examples, even though we also look at those, can be really helpful, and students seem to respond more positively both during the class and afterwards, when sometimes you hear back from students after they graduated, if they got a job or the like, that they want to let you know about doing this kind of writing in contrast to academic writing, or writing with an express academic purpose seems to be really useful.

Given the introduction of this 300-level course in the fall, we're going to see what we can outsource from what would otherwise be a standalone technical writing course to being now part of a growing professional writing curriculum we've got. At the University of South Carolina, in addition to the technical writing course, in addition to the introduction to professional writing course we're just offering, we also have a business writing course. Again, there's some overlapping pieces, but that class is overwhelmingly taught to students in the business school. Not surprising, right? Based on the labels here.

I think there's going to be some interesting opportunities to figure out, not only for me, but the TAs who often staff the business writing sections and occasionally the technical writing sections, what we want to do to set up foundations, to lead into the other courses while also recognizing some students won't ever take the intro class, they'll only take that one class that they're required to. It's an interesting time, and I'm looking forward to it, but I also recognize it's going to be a lot of Jenga involved, so to speak.

SW: I want to transition slightly from undergraduate teaching to teaching graduate students. You teach a class, a grad class called "The Teaching of Business and Technical Writing." Can you talk more about this class and how you prepare graduate students to teach technical communication?

KB: Sure. Just like we've got that small number of classes at the undergraduate level so far, our graduate program in rhetoric and composition doesn't really have much of an emphasis on technical and professional communication the way that Michigan State does, or another school where that's very much a strong foundation or cornerstone of the program. One of the ways that we try and approach teaching this graduate level class is to keep in mind that we've got to do some introduction to the field. We've got to do some introduction to the pedagogy related to that field. We've got to introduce these grad students frequently to many of the kinds of contexts and problem solving situations that we want the undergrads to get involved in, and many cases are more familiar than the graduate students are because this is connected to the larger curriculum they're in and they just haven't really focused on professional opportunities for writing yet.

But then also trying to think about how this class can serve, and for our department it serves as the only additional pedagogical development class that we've got at the graduate level. After the overwhelming number of our graduate teaching assistants take practica for our two first year English core sequence, only those who are really interested in expanding their teaching experience for other kinds of writing situations, like professional writing, whether that's an expressed research interest for them or not, if they're just interested in teaching writing in a

variety of situations, this is the only opportunity they get to expand in that regard. We try to emphasize how there's transfer of pedagogical knowledge from the other training they've had, and the however many semesters of teaching they've got under their belt by the time they take this teaching of this as a technical writing course.

But just like many of us are going to be familiar with dealing with grad students who may never have been in the first-year English course, but have to teach first-year English, so too do we have to talk about a number of different industrial situations, different workplace writing situations for those grad students who not only may have never taken courses related to this, but might not, if they're a student who's trying to move traditionally from undergraduate to graduate knowledge, may not necessarily have had workplace experience. Discussing that can be real tricky too, when we think about the fact that so many undergrads are juggling part or full-time work in addition to taking these courses. There's a lot of very valuable scholarship in rhet/comp that can be brought in, and in education that can be brought in, where giving students the agency to discuss these experiences and how they approach these writing contexts...

Because not only have they done it in the past, they're probably literally doing it right after they walk out the door from your class. I've had students in my technical writing course who literally walk across the street to their job after the class, where they're doing the kind of work that we just did in the classroom. It's really enjoyable to me to help introduce what's going on here and how different and similar it is to thinking about writing, where we're trying to prepare students for the university setting, for academic writing, and then say, okay, we've done that, and now we're going to move them out of that academic context. Everything that they've written about and read about in regards to knowledge transfer from FYC, suddenly, here it is. Right now, we're going to do it. We're going to move it there. Most students respond very positively when they're thinking about how they can step into that instructor role for those courses.

SW: In your book, Rhetorical Code Studies, you write about how software developers make meaning through coding, and you talk about how software codes serve as meaningful acts of communication to construct logic and arguments. What type of implications does this work have in understanding digital rhetorics and the teaching of writing?

KB: If those of us who are interested in digital rhetoric want to try and pay attention to how we are constructing meaning, communicating meaning through digital media, in addition to all the ways that we already recognize we do, from super conventional stuff like sending emails, text messages, Zoom, Skype meeting style, video sharing, website construction, and so on, it's also important for us to pay attention to how we're constructing the platforms that enable us to do all of that work. I don't want to suggest that the *Rhetorical Code Studies* book is the first to do that. I hope that the way I articulate my argument might allow for more scholars and teachers in the field who might have been skeptical of their ability to look at that, or skeptical of their ability to talk about it, to be able to do so. Very frequently, in my experience anyway, the majority of such discussion occurs when one or more scholars are prioritizing that kind of exploration. One of the best books to come out in the past couple of years, by a scholar in communication, Sophia

Nobles, *Algorithms of Oppression*, gives us an excellent opportunity to talk about how search engine technologies are reifying particular racist ideologies.

Trying to call attention to the way that we, we being software developers, or the way software developers write programs, the thought processes involved in them, the kinds of arguments they find themselves making not only in how particular logical structures are constructed to perform particular tasks, but also how that code is meant to be readable or not to other human beings who may be involved in also developing the same project, or who might come across it in the future if it's an open source or otherwise freely available set of code, that can be really valuable to us. We're talking about a kind of writing where there are billions of lines of code written every day. Sometimes we don't have access to them. Microsoft, Amazon, and so on are not necessarily always going to let us see what's going on there, but this writing's happening. For a field that calls itself writing studies, this is going to be an area, theoretically, of interest to us who are interested in digital writing.

SW: How does software coding and understanding the construction of coding impact or influence your teaching and what teaching writing does?

KB: Recognize, and we get a number of different kinds of writing classes…the first caveat I'll note, just in case anybody might be feeling super anxious right now about teaching writing and the discussion about programming, is that I really don't involve this much in my first-year writing courses at all. However, for upper-level writing courses, whether that's writing for the web or other digital environments or technical writing, it can be really useful to help students identify kinds of writing that they encounter all the time, especially those like computer science and engineering students, but also anybody who looks at web pages, if we're looking at HTML and CSS as a different kind of coding, doing that markup.

It's an interesting, for me, demonstration of the Lanham's oscillation of hypermediate and immediate at versus through...many students, overwhelmingly in my anecdotal experience, students who've done any kind of code writing in the past don't initially think about their own work as writing. They don't see it as writing, but they are absolutely and immediately happy to talk about how other people's code has given them a reaction that we would identify as rhetorically meaningful. Like, I hate how that's written, that guy sucks; or they didn't comment it at all, so I can't understand what's going on; or I disagree with the way that they've built the logic of that particular function. It's all kinds of stuff that we would expect audiences to do, but for whatever reason, there tends to be initially this blind spot of, I'm doing rhetorically meaningful writing when I do this thing. If I can see other people doing it, I'm also doing this.

That sort of shift can be really useful. Even though I'm not necessarily teaching a course on programming per se, some of the genres that I'm asking students to engage in, or sometimes they've got a range of options, and students will choose some that they might feel more comfortable with than others, like writing documentation for software, sometimes involves refactoring their code, which is rewriting it to be what we would call more elegant, as an

attribute relating to software development, where the code does something. There's aesthetic component that's hard to describe, but it involves the code being generally shorter, not repeated. That is, if you're thinking something happen, you call to it in a modular fashion. It's relatively human readable, so you're not just looking at it, but what seems to be gibberish, perhaps, for the individual who doesn't do any code, but something that approximates, to the extent that a language allows it, something that might look like English, for example.

There have been decades of computer science scholars who've made similar arguments. Some of the points I make in the book are based on computer scientists arguing for similar kinds of work, where...Donald Knuth is a luminary in the field. He says the best kind of code is one that doesn't need comments. I mentioned that a few times now: Comments are lines of code that are not read by machines. They're only there for other human beings to understand what's going on. You might write up something that adds two numbers together, so you might have a comment that says, I'm adding these together for this purpose, and the machine reads past it. Doesn't care. For Donald Knuth, the best code is code that doesn't require comments, because it's already so human readable.

Which, again, is something that I think we as rhetoricians really value, is how can we make sure that the work we're doing in our writing is accessible to both the intended audience, but also potentially secondary or tertiary audiences. When we're writing with that kind of idea in mind, how does that impact the kind of language we use, the way that we might arrange a particular argument? How are we providing a kind of style that's readable in the code? Here I'm talking about some things that might be as trivial sounding as indenting lines.

SW: What technologies do you incorporate in the writing classroom, and what would you say are some advantages to incorporating those technologies?

KB: I would say, at the broadest level, I try to adhere to very similar principles to those described in Karl Stolley's "The Lo-Fi Manifesto" that was published in *Kairos* a while ago, and then I think he published a version two of it more recently, where the argument is for plain text writing that can be formatted. Whether that's plain text or marked down versus locking in your writing to a proprietary platform or one that's not very adaptable to other environments. Very frequently we'll have conversations, my classes and I, about, do you want to use Word? Because the University of South Carolina has an Office 365 contract for all students, so they've got access to it, but I know some might prefer Google Docs, because for whatever reason, maybe they prefer writing on their tablet or phone or just in their browser versus Microsoft. Cool.

We'll talk about the file formats that interest us more than the specific program that they wrote in. If it allows for that, how does it seem to do that, what happens when we need to double check what's going on? For a long time, Microsoft Office and LibreOffice were not as compatible as they are now. While I've written in LibreOffice for a long time, I really had to always double check before I sent off a .doc file that had been saved in the LibreOffice, that it would work in

Microsoft Word. Then we'll talk PDF as a way to lock in the format, if we're really going in that direction.

For assignments or classes that expressly focused on web writing, we'll, again, talk about text editors. Those that allow syntax highlighting, which matters for code languages of various kinds. You can see even something as simple as HTML, in the markup there, are the tags opening and closed correctly? Students get some freedom in my class about what programs they choose based on what might be familiar or comfortable to them. What matters to us is what you can do, to make the files you create useful in the ways that matter to us and our audiences. Many students, again, because I'm teaching primarily computer science and engineering students, will submit files to me through Git, hosted through GitHub, but with the Git program. Not only discussing uploading documents through our LMS versus via email, which we try to avoid, but giving them this opportunity to "submit" (in air-quotes) their work through a site like GitHub, which is something that reflects the kind of work they'll do in a professional setting. It feels wrong to me to restrict that rather than to promote it as a way to reduce that pseudo-transactional feeling as much as possible. So long, again, as they're doing so as accessibly as they can, as modularly extensible as they can.

SW: Thanks, Kevin. And thank you Pedagogue listeners and followers. Until next time.