

**Title:** The human side of software engineering

**Summary:** Janice Singer refutes the notion of the antisocial software engineer in favor of the scientific collaborator.

**Featured researcher:** Janice Singer

**Duration:** 6 minutes, 40 seconds

### **Introduction**

Nearly every discipline in the arts and sciences has its “loner” stereotype, and computer science is no exception. In this episode of [Podium](#), Janice Singer - a cognitive psychologist and researcher in the area of human-computer interaction - talks about her efforts to develop a Web community of people interested in the social habits of highly effective software engineers and computer scientists.

### **Presentation**

My name is [Janice Singer](#). I'm a senior research officer with [The National Research Council of Canada](#) and I'm going to talk about the human side of software engineering.

When I started my work, there was a lot of really technical people around, and really what these people were working on were things like clone detection in program source code and formal methods and operating systems for embedded software - so, really very, very technical things.

I remember going to one of my colleagues one day and saying, “What do software engineers need? What kind of

tools do they need? What do they do in their everyday work?" He kind of shrugged his shoulders because he didn't know.

So I thought, "Okay, well, you know, he's just one person and he doesn't know everything, right?" So I decided that what I would do is look at the literature. So I started to do a massive literature review on software engineering and try to understand what their work practices were.

At that point in the CSCW [[Computer Supporter Cooperative Work](#)] and HCI [[Human-Computer Interaction](#)] literature, there was a big push towards understanding work practices and then trying to build tools around the actual work practices of people. And in software engineering, there were maybe two papers at most.

There was a seminal paper written by [Bill Curtis](#) and colleagues that looked at a large software development effort. And I think there was some research by [Robert Kraut](#) as well. But that was it.

What I decided to focus on in my research area was trying to understand what is it that software engineers do on a day-to-day basis.

A lot of the challenges that software engineers face are challenges that anybody would face in any job. So, problems with coordinating your work, problems with communicating with your colleagues, time pressures. I think probably what's different in software engineering than in some of these other domains is that in software engineering, there is a real culture of using tools to solve problems.

So, for instance, when versioning of software became a problem, you create version-control systems, right? When trying to find defects in code became a problem, then we had debuggers. So, there's a real culture of using software tools to solve software problems.

What I'm hoping some of my research can shed light on is, what kinds of tools can we build to help solve maybe some of the problems in coordinating work and being aware of what other people are doing in finding someone that can help you do your work more effectively in generating better code, understanding how to test code better. So, if we can understand some of these practices, then we can build better tools.



It is a myth that software engineering is largely about interacting with computers, because it's not. If you're a software engineer, it's largely about interacting with

other people to create software, and then some small portion of that time is spent actually writing code. There's a lot of processes that you have to follow. You really have to coordinate your work amongst a great number of people.

I mean, this actually takes up a large amount of your work, and it doesn't matter whether you're following a [Waterfall](#) practice on the one side to an [Agile](#) practice on the other. Interaction with other people is a key part of creating and building software.

What kinds of processes do you use to create software and under what conditions do those different processes matter? Software engineers almost always work in teams. You know, anybody who is working on a commercial product is working on a team. You're just not going to be working on your own.

So, there's all sorts of successes and difficulties in working in a team. How do you coordinate your work? How do you understand who's working on what? When you have dependencies in the team, how do you make sure that that person finishes what they're supposed to do in time?

It's really important to be a good communicator. Not simply to communicate your ideas and present your ideas clearly,

but also to be able to listen to other people's ideas and to be able to understand those people's ideas as really super important. And it's a skill that's frequently lacking. It can be taught, it can be learned, but it's not.

Almost by definition when you're working on a team, when you're working on a product, you're going to understand just a small portion of the source code for that product and being able to go in, understand just enough that you need to know and being able to make changes to the source code is a very, very difficult task.

Not only are there all these social and teamwork issues involved, but in the end, individual cognition really matters for how you're able to create effective software.

It's not like you can never be depressed or can never have a bad day, but overall, your personality has to be such that you're extremely professional in how you deal with people.

What's conveyed to students in their educational process is that you have to be the best technically, and it's not conveyed to them that these other things matter, and I think that that's a mistake.

There was a prima donna on a team. And this guy, just, he was always right no matter what. And apparently he was technically quite brilliant but he wasn't willing to listen to anyone else.

And even though his solution may have been technically superior, eventually they had to let him go from the team because he just could not work with others. He just couldn't play in the same sandbox as other people.

So, it didn't matter how technologically brilliant he was. In the end, he was not an asset to the team. He was, in fact, drawing the team down and preventing them from moving forward on solving the problems they needed to solve.

What we're trying to do is build a community of researchers, of people who are interested in the human side of software engineering.

**You've been listening to Podium, a project of IBM Research - now available on iTunes. For more information about the human side of software engineering, visit**

**<http://janicesinger.com/>.**

**Series producer:** Barbara Finkelstein

**Music:** ["One Sly Move" by Kevin MacLeod](#)