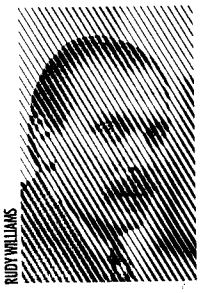

RUDY WILLIAMS

SOFTWARE ENGINEERING NEEDS VISIONARIES, MULTIPLE APPROACHES

Interview with Richard DeMillo

WHEN HE BECAME DIRECTOR OF the US National Science Foundation's Computer Research Division in fall 1989, Richard DeMillo was faced with having to help shape the basic research directions for the computer-science and -engineering communities.

The beginning of his two-year tenure also coincided with a groundswell of awareness and concern among federal policy makers about US computer science, as evidenced in the broad support for a federally sponsored national high-speed research network and congressional concerns about competitiveness and safety of software technology.

While at the NSF, DeMillo maintains his position as a full professor of computer science at Purdue University and as director of its Software Engineering Center.

In an interview with News Editor Galen Gruman, DeMillo outlined three crucial research areas — exploring techniques like automation and reuse to lower development costs, the dichotomy between process and product, and collaboration — and described the directions the NSF is encouraging.

Q: *What do you think are the most vital software research areas?*

A: It's hard to come up with a single area. Let me give you three areas that I think are really important. [First,] I don't think we've looked hard enough at novel solutions to building software.

If you look if all of the cost models for software, the thing that seems to matter in terms of what it costs to build a software system is the steps that you carry out — if you leave out steps, the cost goes down. People have different ideas about what's a cost driver and what isn't, but the one thing that's pretty common across all the cost models is that the cost is determined by the sheer number of steps that you have to carry out.

And that's one of the things that led to the current interest in reuse. Reuse is a way to leave out steps. But it's not the only way to leave out steps. This is what I mean when I say we haven't looked at alternative solutions.

Another way to leave out steps is to automate the steps. It's been proposed off and on, but it's never really been taken seriously among software engineers. I think early on in software engineering, automating the process sort of developed an AI flavor to it. And maybe like all of the early claims of AI, automating software production got tainted by the high expectations. It's just not clear to me that anyone has really gone about to revisit what's possible in that area.

> My interest right now is in getting some product-oriented research going. There are a lot of things that we should be doing in the area of product that we're simply not.

One of the mistakes that was made in those early days — in automatic programming — was to assume that "gee, this process that you're going to use to automate software production has to be everything. You have to start with the specification and then wait a length of time, and out the other end will pop a piece of software that you can run." But almost nothing else in the world gets automated that way. You automate the individual steps.

I'm saying this is an alternative to reuse — maybe not an exclusive either/or thing but certainly a way of reducing costs simply by automating the things that cost a lot.

The other thing that comes to mind as places where automation might really help is all of those parts of software development that depend on global information, for which there there don't appear to be any easy ways of doing decompositions or hierarchical approaches to control the complexity.

Q: *What's another area that would benefit from a fresh look?*

Another area has to do with the dichotomy between process and product. We've had these cycles in software engineering — even before people started to thing of it as software engineering — in which looking at programs and programming as a task that was very important.

And then — not all of a sudden, but over the course of a few years — people realized, "Well, gee, without enough context for software to develop, it's hard to figure out exactly what it is you should be measuring, so maybe we should look at process." And after a while, process becomes the most important thing to look at. I think the time has come again to look at product.

Q: *Are you suggesting perhaps product and process should be researched at the same time, rather than alternate?*

A: I think they're related, but my interest right now is in getting some product-oriented research going, speaking now from the NSF's standpoint. I think there are a lot of things that we should be doing in the area of product that we're simply not.

Q: *How would you define "product"?*

A: Let me give you an example. Suppose you wanted to know, for whatever reasons, on the average, how many references constitute a typical definition-use chain in a dataflow analysis. How do you answer it? In our current state of affairs, that's an unanswerable question.

If you've got the problem of answering a very specific, product-oriented question — What's the likelihood of this happening? Does this ever happen? Do people do this sort of thing? — the only way to answer the question is to go back and look at what people have done and say that never occurs or that occurs a lot. And I know of no other way to do that than to gather together large collections of software artifacts for people to look at. I think that this piece of infrastructure building is very important.

[Donald] Knuth, when he was writing, kept a log — a 10-year log — of the errors that he made. Presumably, at four o'clock in the morning, he had a notebook next to the terminal, and he was writing down what he did. This was actual error logged as it was produced — uncensored.

Q: *No one collects that kind of data. No one writes down what they do wrong.*

A: No. No one does. All you get is sort of post-hoc analysis based on error reports that may or may not be accurate recollections of what actually happened. And the bug reports

and the fix reports are never true recollections of what was actually fixed in the program. But this was data produced by the man who wrote the program, who was fixing the bugs. Maybe the single-person nature of it all makes it much more doable.

Q: *So this could be duplicated many times?*

A: Right. At NASA Langley, where they do a lot of testing of avionic software, there's this wonderful term for certain kinds of bugs, called "indigenous bugs." The definition of an indigenous bug is wonderful: It's a bug that an avionics software designer is likely to make. And that's its characterization. No one has been able to plop something down on the desk and say, "Here's some indigenous bugs. Show me that your test methodology covers these things." That's a terrific artifacts question.

Let's take a look at what people do and see whether or not there are errors in that code that aren't classifiable by other means — sort of like the Occam's Razor for software engineers: Let's start with the simplest plausible explanation. Let's see if we can classify the errors that people make according to simple strategies, and if something falls through the cracks there, we can go back and look at it. That requires a database of artifacts to get the study together. So someone has to take the lead and sort of pump into the world a collection of software that people can use to start the process.

Q: *What's the third vital area for research attention?*

The third area has to do with increasing the bandwidth between people. A lot of what goes on in-large scale software development — in large-scale collaborations of any kind — is simply overhead involved in getting people to behave as if they were in the same room talking to each other.

There's a lot of groups springing up around the country that are looking at the collaboration technology, looking at electronic conferencing systems. The areas of computing that depend on collaboration for their life blood are going to get a windfall from that research — it's going to be areas of software engineering, which is a collaborative venture, and of computational science, where you have groups collaborating.

Q: *What is the NSF's set of priorities for software research?*

A: The object of concern, I think, is that at some point software engineers have to step back from what they're doing and look at their problems. We've been a field for, I don't know, 10 years, that's been living and dying on solutions. People propose and market solutions to problems. That's fine if you're in the business of selling solutions.

But we at the NSF, and at a number of the other research-oriented funding agencies, shouldn't be in the business of buying solutions. We should be in the business of buying problems. And we've gotten away from that. We haven't made a big enough deal of the research orientation of this agency and the fact that there really hasn't been an effective research agenda for software engineers.

I would say — in this division, anyway — we're getting out of the mold of being research clerks who gather proposals and gather reviews and total things up.

In some of the areas that we fund, there's very little effective leadership that we can impose. Theoretical computer science is the one that comes to mind, which has an effective research agenda, which has an effective infrastructure. There's no large-scale engineering on theoretical computer science that would be effective or helpful. The people in that field propose what they think is important, and we select the best and try to put as much resources into their hands as possible.

In software engineering, I think the NSF has to be somewhat more forceful and exert some research leadership.

In software engineering, I think the NSF has to be somewhat more forceful and exert some research leadership. We need to say, "You of the community have not set your sights high enough. You've been driven by short-term concerns of other funding agencies for a long time. You've been proposing solutions instead of basic problems." If you're going to put together a successful research program at the NSF, it has to be one with a strong research scientific component. And that, quite honestly, is missing from the software-engineering program today.

Q: *Does the software-engineering community know how to do this, or has it just been preoccupied?*

A: Well, yes to both. I think it knows how to do it, but there hasn't been any particular path for doing it up until now. Most of the funding in this area has come from agencies that are, quite frankly, in the business of buying solutions. If you've got a solution, and someone's willing to pay you a million dollars for it, you'd have to be crazy to come to the NSF and say, "Well, I'm going to turn down that million dollars, and I'll take your $70,000 grant to look at problems."

Well, the world is changing in a number of dimensions. I think the solutions are becoming increasing less effective and less convincing. I think people who have good ideas haven't pumped into the [research] pipeline new problems and fresh approaches to them. This is the old software problem, but we're now in the third generation of the software crisis, and —

Q: *You can't call it a crisis for that long.*

A: Well, you can, but I think it's less compelling after two generations. I think we're locked into ways of looking at things. Let's look at more fundamental issues.

If there's one thing I can do in the years I have left at the NSF, it's to build the software-engineering program here into a real research program that over the course of some years has a chance of making some real breakthroughs.

Q: *What's the software version of the Moon landing?*

A: I don't know. No one has come up with it yet. I don't know what it is. We've had a number of workshops to try to put together sort of grand plans and grand strategies, and they've all fell more or less flat on their face. And I think there are a couple of reasons for that. One is this incremental mode that people are locked into. We really haven't had any visionaries step forward and say here's a direction that's interesting to go. We would like is a few ideas on the table to give us possible worlds to shoot for. And we don't have those right now.

From a point of view of someone who has to put together a plan and a budget for programs, that bothers me, because the other programs — particularly the successful programs — have those things. I don't see anything that grabs me as a field in which the field should charge.