

Correct by Design

Jesse Poore suggests a revolution in programming -- holding software developers to the same level of rigor of training and workmanship as other professionals, developing software that's correct by design, and constraining the release of software-intensive products until they are scientifically certified as fit for use.

Jesse Poore is Professor of Computer Science at the University of Tennessee, where he holds the Ericsson-Harlan D. Mills chair in software engineering. He is Director of the University of Tennessee/Oak Ridge National Laboratory Science Alliance (UT/ORNL). His research interests include the economical production of high-quality software and federal science policy.

UBIQUITY: Let's start at the end, which is where one always starts. In the last sentence of your paper, "A Tale of Three Disciplines ... and a Revolution," [see <http://www.computer.org/computer/homepage/0104/Poore/>] you say, "Theoretically, software is the only component that can be perfect. And this should always be our starting point." What do you mean by that?

POORE: It's a comparison between analog versus digital, or hardware versus software. You can't cut a string exactly in half because there's always a further level of resolution of measurement that would show that one piece is longer than the other. However, since software is ultimately a string of bits, there is always at least one bit pattern that's exactly, perfectly correct. Hardware can fail because of wear or breaking, software cannot. If you are in the business of developing software, your point of view should be that, yes, it is possible to get it absolutely right. Of course, that is opposite the attitude of most software developers.

UBIQUITY: Why do most software developers have the opposite attitude?

POORE: Because they truly believe that it is impossible and they can get away with it. If you look back on the way that people developed software, years ago the machines were modest in size and capacity and there was only one [machine] in the computing center, and people prepared their programs and waited in line to use the computer. Typically, then, more than one person would review the code and there would be discussion and analysis about it. When a failure was

seen on execution it was disappointing and somewhat surprising, although it happened with enough frequency that I guess programmers came to think it was OK and that they would eventually get the errors out through repeated efforts to run the program. The attitude was reinforced by faster computers, quicker turnaround time, down to the present with near instantaneous turnaround time for many programs. It's a very unproductive point-of-view.

UBIQUITY: Have non-programmers essentially taken over programming?

POORE: They have in the sense that it takes relatively little in the way of training in the foundations and fundamentals to qualify people to take programming jobs. It's mostly a matter of experience or years in the trade. You don't have to say that you've done it well or that your programs generally work. You can have years of experience in writing very bad programs and it still counts as years of experience. This is OK if just money is at risk, but not when public safety is concerned.

UBIQUITY: What does it mean for a program to be "correct"?

POORE: You should have a formal specification. Failure to do so is the first point at which software development breaks down. If you have a statement of the requirements, then you can work systematically from that statement toward a precise specification. In the process of working from loose requirements to precise specification, you can discover any incompleteness or contradictions in the requirements statement. When you reach the precise specification it's complete and consistent in a mathematical sense, and traceably correct. It's easy to say whether a program is correct or not because the program has to perform according to the specification. In many instances of software development today there is no specification or statement of requirements. In that case, it's very difficult to say whether the program is correct or not.

UBIQUITY: Let's use an analogy. Take a sport, such as basketball or baseball. You know when you win the game because of the score. But there are many different ways to achieve the score and win the game. Are there different ways of writing a winning program?

POORE: Yes, typically. The precise specification of what a program should do is unique given the requirements, but there can be multiple solutions and ways to implement them.

UBIQUITY: Are they equivalent?

POORE: They are functionally equivalent. Each program meets the specification.

UBIQUITY: Would they gain equal approval of a good software engineer?

POORE: Yes, assuming comprehensive requirements and comparable workmanship. Say, for example, that you have requirements for a cruise control system for an automobile. There might be two different suppliers of the cruise control system. One from the US uses an Intel chip with software written in Java. A European supplier runs on a Siemen's chip and uses Ada. Different chips. Different programming languages. But the cruise control systems could be interchangeable in an automobile. You would have one specification (meeting the requirements) with multiple implementations.

UBIQUITY: Getting the specification right is the whole task. What is the art of doing that?

POORE: It's not an art as much as a mathematical activity. It can be done in a systematic way that would lead two different groups attempting to do the same job to reach the same statement of the specification. That's the part that's missing most often.

UBIQUITY: You've urged a revolution. Explain the nature of the revolution.

POORE: The revolution is that we can't leave software developers to freeform expression any more. We have to constrain them to follow rules and regulations in order to produce more trustworthy software.

UBIQUITY: Talk about the comparison you've made between software engineering, genetic engineering and circuit engineering.

POORE: The theme here is hardware, software and *lifeware*. In the case of hardware, everything is pretty well understood. It's a mature activity. In the case of lifeware, the field is beginning to embrace the mathematical ideas that characterize modern genetic engineering. In hardware and genetics, the goal is to be correct by design. You want the circuit that's produced to be correct, not because you've run test after test, found one problem after another and made changes. The

goal is for it to work correctly the first time it is used (or tested) because it is designed and built right.

UBIQUITY: And not so with software?

POORE: In software, the effort is seldom to be correct by design, to get it right the first time. Yet, the same degree of success is possible with software as with hardware. Software developers will often say that software is different. Software is more complex. The comparison was to show that software isn't really different. It's no more complex than these other fields. So why does software get a free pass? Why not hold software developers to the same level of rigor of training and workmanship that we do these other fields? That would mean to work in a way that let's you develop software that's correct by design, rather than "good enough" because of trial and error, and testing and changing.

UBIQUITY: Why do people think this is a controversial idea?

POORE: It's controversial because in most computer science and software engineering programs at most universities, it's not popular to teach and reinforce the mathematics of the foundations of software and to use methods and tools that would help your work to be correct by design. For most faculty and students, it's not as much fun. It's controversial in industry because it is erroneously thought to be more time consuming and more expensive.

UBIQUITY: You've taught these foundation courses. What do your students think of them?

POORE: I've had some students, good programmers, tell me afterwards that if they had to work that way then they would do something else for a living (which I encourage). I've had other students say that only when they were developing software in this way, did they feel like they were part of a profession.

UBIQUITY: The problem has been around a long time. Has it gotten worse?

POORE: Yes. The problem got worse because computers became faster and their memories bigger, which meant that the software programs could become more complex. The computers got cheaper so more companies could afford them. The growing demand for programmers far outstripped the ability of the universities or other

organizations to properly train people for the software development jobs. Many people learned on their own as well. A huge workforce evolved along with ever-increasing complexity and it got into more and more trouble. That is to say, more software fiascos, more runaway software projects, more cancelled projects, lower quality products, more rework and more waste. This has continued with the advance of miniaturization, more functionality on smaller chips. The chips and consequently software have moved into systems that impact human safety. There's software in pacemakers and automobile brake systems. About the only medical device that's not software-intensive is a dental pick.

UBIQUITY: How do you solve the problem?

POORE: We're faced with trying to constrain the people who are permitted to work on such devices or products to those who are properly trained and using proven methods, tools and processes. And to constrain the release of software-intensive products until they are scientifically certified as fit for use.

UBIQUITY: As more computers get made, and more people who are not the best programmers, or even professional programmers at all, do this work, do you feel as if you're only putting your finger in the dike?

POORE: Yes, that sounds pretty accurate, unless there's some serious licensing so that only qualified people are allowed to work on devices that have implications on public safety such as transportation and medical devices, and serious certification of the devices themselves.

UBIQUITY: Is the problem made worse as more programming is done offshore?

POORE: The movement of software development offshore will not make the problem worse and may make it better. Not only is the labor rate much lower, I've read some indications that the quality of the work is higher. It may be that the foreign workforce is better trained in mathematics than the domestic workforce. In any case, this should be a wakeup call to domestic software developers.

UBIQUITY: Are the current professionals split approximately 50-50 about whether licensing and certification are good ideas?

POORE: I doubt it's anything close to that. I would be surprised if one percent of professionals agree with me. That's why it would be a revolution if this actually came about. I don't have any idea what the split might be but if I judge it, say, by the number of organizations that are willing to use rigorous methods, then one percent would be generous.

UBIQUITY: The ACM and the IEEE have had discussions of this. Did the discussions fall apart?

POORE: They have ongoing, established committees that are working on this. They struggle along putting out some good products in the way of standards, and running some certification themes. But they are largely ignored. I don't think the certifications are good enough. They're more for generalists.

UBIQUITY: What would you do differently?

POORE: I would like to see licensing and certification be limited and specific. For example, someone could be qualified to do mathematical software, to write programs for numerical quadrature, and numerical differentiation on various architectures and machines, for example. The person would have to understand error propagation, round off error, truncation error, how it builds up, how to prevent it, and so on. That's a really complicated business. Only well trained people should go there. Likewise, those products should be certified so that applications that use them can say "certified math software inside."

UBIQUITY: What are some other examples?

POORE: People could be licensed for compiler development, which is a different type of software than mathematical software. Imbedded software is another one. You could be qualified to work in more than one business, if you have the skills and training. Medical devices probably should be a separate license: Maybe even individual medical devices. Radiation therapy systems don't have a lot in common with pacemakers.

UBIQUITY: In your most recent paper, you mentioned Dave Parnas's paper on software engineering programs and computer science programs. Talk about that paper.

POORE: Parnas is one of the people most devoted to the concept of software being correct by design, and getting specifications right. He

developed both a software engineering program and a computer science program at McMaster University. He makes a big distinction the two. The paper talks about those differences. A software engineer builds a product. Software developers should understand how the product as a whole functions, not just the software. You don't ask someone who's been writing accounting systems for a bank to program an industrial robot, because both just happened to use the same programming language. On the other hand, computer science is a science with theoretical, experimental and computational aspects. The curriculum shouldn't necessarily follow a product focus.

UBIQUITY: Talk a bit about your own intellectual evolution. How did you get to your current interests?

POORE: A long and circuitous path. I lived in computing centers right out of high school. I wrote literally hundreds maybe thousands of programs in the earlier years. I got the first Ph. D. in Computer Science issued by Georgia Tech. The faculty came from mathematics, electrical engineering, system engineering, philosophy of science, linguistics, and beyond. They looked at my background and said they couldn't teach me about computers and programming but they would teach me formal logic and how to do science and other things. My doctoral program was essentially logic and algebra, which I thoroughly enjoyed. Then I went on to run a computing center at Florida State University.

UBIQUITY: How was that experience?

POORE: Wonderful! During the 1970s, computing centers were a lot of fun. We had a very adventuresome computing center. We designed and built hardware. We designed and programmed operating systems and did a lot of interesting development projects. By the end of the '70s, the DEC VAX had come along. The computing center no longer had the only computer on campus. Everyone was becoming an expert. I saw the handwriting on the wall, that the career of computing center director was over.

UBIQUITY: What did you do next?

POORE: I returned to Georgia Tech as the computer and communications czar where we built one of the early networks in the US. But networks rapidly became a commodity and another career came to an end. So I came to the University of Tennessee to develop the computer science program and to do science. Along the way I

became more and more bothered by the fact that software products from vendors were becoming less and less reliable. I began asking, why is this the case? Why is software bad? Why is it so hard?

UBIQUITY: Is that how you became associated with Harlan Mills?

POORE: Yes. Someone told me that Mills was the only person they could think of who would insist that software should be free of errors. So I became acquainted with Mills and began working with him. I spent about 10 years and produced half a dozen Ph.D.s and 20 Master's theses focused on how to create the engineering practices that would allow software developers to implement the ideas of Harlan Mills. I understood his ideas from an algebraic and logic point of view because of my doctoral training. I thought they were absolutely correct. And if they were correct, then we should be able to reduce them to practice.

UBIQUITY: How successful were you?

POORE: We have had a large measure of success. I work with several companies that use these methods with remarkable success. We have trained several hundred software developers to use the methods.

UBIQUITY: How does your interest in cleanroom fit in to this?

POORE: Mills's software development method is called "cleanroom." I have spent the past 10 or 12 years further developing cleanroom software engineering, from Mills's original concepts to the practices, working methods and tools that are available today.

UBIQUITY: Was he using the term "cleanroom" software engineering metaphorically?

POORE: The way he told it to me, when Mills was an IBM Fellow he was touring a new fabrication facility that had "clean rooms" with only so many parts per million of contaminants, workers in bunny suits, airlocks, and so on. They were boasting about how high the quality was and how high the yield was. Then someone says, "It's too bad you software guys can't produce software of high quality." Mills said, "If we spend as much money up front getting the design right as you guys do, and if we spend as much money on the process keeping impurities and errors out, as you do, we'll have the same kind of quality and yield in software."

UBIQUITY: How does one achieve cleanroom software engineering?

POORE: You shift your focus way up stream. You develop precise specifications and then develop the code to meet those specifications. You follow processes that keep the errors out. All code is functionally verified to remove errors that were made. Finally, testing is a statistical activity designed to demonstrate that there are no errors (rather than to find errors). This supports certification of products by statistical protocol.

UBIQUITY: Your other main interest, and much of your past experience, is federal science policy. Talk about that.

POORE: I worked for the National Science Foundation for a while. A few years later, around 1977, when Carter was President, I worked with a task force that was attached to the Executive Office of the President. We looked at the need for supercomputers in the federal laboratories, and found that the need was enormous in comparison to the capability. The federal process for procuring scientific computers was somewhat awkward so we helped to clear the way. In the course of doing that, I became acquainted with the congressman who was the chair of the science committee. He was from Tallahassee (hometown of FSU) so I worked in some of his campaigns, and became even more interested in science policy. In the early '80s, I went back to Washington and served as the Chief of Staff for the Committee on Science and Technology.

UBIQUITY: What issues was the committee concerned about?

POORE: We dealt with issues like the need for supercomputers throughout the country and the need for the Internet, which was in transition at that point from just a defense network to the civilian Internet. The committee had oversight of other research too -- NASA programs, energy programs, EPA programs. Most all of the national science policy issues came through that committee. Since that time I've been on a number of NAS/NRC study panels, giving advice to agencies and to Congress about various science issues.

UBIQUITY: Would you be willing to take a federal job now if the right thing came along?

POORE: Well I suppose the answer to that is always yes. But right now I'm director of the UT-Oak Ridge Science Alliance. UT joined with Battelle to become managers for DOE of the Oak Ridge National

Laboratory. The Science Alliance program recruits joint faculty and is launching the Joint Institute for Computational Sciences. There will also be a Joint Institute for Neutron Sciences and a Joint Institute for Biological Sciences and Material Science. I interact with many activities that are based at ORNL, in all the fields of science and engineering that are strategic to Oak Ridge and to UT.

Jesse Poore may be reached at poore@cs.utk.edu

Source: Ubiquity, Volume 5, Issue 2, March 3 - 9, 2004
<http://www.acm.org/ubiquity/>