Special Report

SEI-89-SR-13
ESD-89-SR-42

Continuing Education in Software Engineering: Teaching Tricks of the Trade

Maribeth B. Carpenter, editor

September 1989

Special Report

SEI-89-SR-13
ESD-89-SR-42
September 1989

Continuing Education in Software Engineering: Teaching Tricks of the Trade

Maribeth B. Carpenter, editor

Video Dissemination Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This work is sponsored by the U.S. Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government.

Continuing Education in Software Engineering: Teaching Tricks of the Trade

Abstract: This document is an edited transcript of the opening session of the Continuing Education Workshop held at the Software Engineering Institute (SEI) November 1988. It contains welcoming remarks by Maribeth Carpenter, a senior computer scientist at the SEI and organizer of the workshop, and by Larry Druffel, director of the SEI. Dr. Druffel establishes some context for the workshop and introduces the keynote speaker, Dr. Jon Bentley of AT&T Bell Labs. Then follows Dr. Bentley's talk entitled "Teaching the Tricks of the Trade".

1. Opening Remarks

MARIBETH CARPENTER: Good morning. I'd like to welcome you all to the Software Engineering Institute. My name is Maribeth Carpenter. I'm in charge of the workshop and also the SEI's Continuing Education Series, which we will be talking about for the next two days, hopefully to get you involved in it.

I would like to introduce you to Larry Druffel, who is the Director of the Software Engineering Institute, who wishes to welcome you.

LARRY DRUFFEL: Well, let me just say welcome. And I really do mean welcome to the SEI. When I look around, I see some faces here who have been here more often and longer, over a period of time, than I've been myself. And then again, I see some new faces. And so, particularly for those of you who are new, welcome to the SEI. I know that many of you know more about us than I can describe in a short period of time. But I am going to speak a little bit this morning, for those of you who aren't familiar, a little bit more about some of the other things the SEI is doing, in addition to the things that we will be talking about here at this workshop.

The point I would like to make this morning is that our mission specifically calls for us to transition technology to practice. One aspect of that and one important part of our mission has been education. To date, most of our efforts in education have been on the academic side. (I think most of you know Norm Gibbs and his folks.) We have focused on developing a curriculum for a master's degree in software engineering. We've done that jointly with 39 other universities, whose representatives have participated actively in building the material for the master's degree. This year we began to teach courses at the SEI that are offered for credit at Carnegie Mellon.

The academic program is now on pretty solid footing and we want to turn our attention to continuing education. And that is what this workshop is all about. We want to help the practicing engineer, who might not want the academic credit but needs the information.

As I look around, at least some of you are old enough, that you didn't grow up as software engineers, with formal education in software engineering—because there wasn't any. And so, we got into this business through experience. What we don't have is the benefit of the work that has gone on in the academic side of education, for the last five years, in cataloging the information. We have had to read the papers, and if you are like me, it was kind of a random walk. I read some of the right papers and some of the wrong papers, trying to figure out where the field was going.

So now we want to focus on how we reach the practicing engineer. And we want to do that in very much the same style as we have on the academic side, where most of the development effort has been cooperative. In fact, we have been more a facilitator than anything else. We worked with the academic community to define the requirements two years ago, when we had the first workshop to begin building a consensus for what a master's degree in software engineering might look like. Now we are interested in your perspective, based on your experience in the industrial world and in the government world, to help us determine what the need is and how we can fulfill that need. And its not enough to just come and give us advice, because that's not the end of the story. We want our continuing education effort to be much like the academic effort, and we want your participation.

Many of you know the model that we use in the Academic Series to catalogue the information, in software engineering. This information is written and developed by folks in other universities, who come to the SEI for periods of time to work with us. We hold Faculty Development Workshops every six months and share that knowledge among other folks, who then develop and teach the courses and report back to us on their efforts. We want to do the same sort of thing, build the same sort of model, where you become involved.

Another point I want to make is that we want to do this in a cooperative way. We believe there is a need, and we want to fill that need. We don't want to compete with you, just as we don't compete with universities. We create videotapes of our courses. We send those out to participating universities, who offer those courses for credit. They give the credit. We don't. We are simply working with them in a cooperative mode. We want to foster the same sort of active involvement from you folks as we have on the academic side.

So, the charge is to figure out what your role is and how you can work with us to create a really effective, world-class software engineering program for the practicing engineer. That's what we are about today. I urge you to focus your attention on how we can achieve that.

Now that you feel welcome and you understand what we are here for I'd like to introduce our keynote speaker, Jon Bentley. Jon has been a regular participant here at the SEI, gave one of our Distinguished Lectures, and has been a long-time advisor to the academic program. Jon has a bachelor's degree in Mathematical Sciences from Stanford and a master's and PhD from the University of North Carolina. He was on the faculty here at CMU and was given a distinguished award for his undergraduate teaching. He is now with AT&T Bell Labs. He is the author of three books; and as I think many of you know, he is the author of

the series "Programming Pearls" in the *Communications of the ACM.* Please welcome Jon Bentley.

2. Teaching the Tricks of the Trade

JON BENTLEY: Before you do anything else, quick, please, either close your book [of workshop materials] or promise you won't look ahead. I don't look into your minds, to see the nasty things you are going to say to me, so you shouldn't look at my pieces of paper, to see the things I am going to say to you. That's only fair, after all.

Right now is the time when I relate the talk to the whole purpose of the workshop, and if I could, I really would. You would be the first I'd tell about how this relates to what the workshop is doing, if I could.

I guess what I have is an example of continuing education. It is the kind of education I have found fairly handy in my job at Bell Laboratories. It's the kind of thing that I like to teach to professionals. But apart from that, apart from the aspirations, I think what I can do here is promise you a few fun things to teach. Instead of talking about education, I'm just going to *do* something.

What I want to talk about today are some tricks of the computer trade—tricks that most programmers know, most expert software engineers known, tricks that are a large part of what makes a software engineer a really good software engineer. What I want to do is start off with four tricks of the trade, review what they are, think about them for a while, and talk about teaching them, especially how one teaches them in the context of professional programs.

A definition: In any kind of field, one can say that there are at least three different kinds of knowledge that one needs to have. There is science, there is project management, and then there are tricks of the trade. In medicine, for instance, if one is going to cut into a human body, it really pays to know something about biology. How embarrassing it can be, indeed, if you cut in and say, "Gee, what's that sort of pinkish thing in the middle there?" Likewise, if you are going to prescribe drugs for another human being, it's really important to know about chemistry and how drugs work, at a basic biochemical level. Science is a big one. You can't argue about that.

Likewise, project management is a huge one. Operating teams are a fine example of getting ten or eighteen hands working together under one mind. Medical records allow one to trace a unit of blood or establish a history, to call up an x-ray from a dozen years ago—providing an excellent example of documentation.

All those things are worthwhile and important. Science. Project management. Fundamental. But in medicine, there are also tricks of the trade.

I believe that every talk should have at least one useful thing in it. Here's the useful thing for this talk. For those of us who wear these kinds of clothes [indicating casual dress], as opposed to the kind of clothes that some of you wear [indicating military uniforms] with the short haircuts, this is a trick that we can use when we donate blood. When the Blood Mobile

comes around to work, before they put the horse needle in your arm to get out a quart or two, they take a little sample from the finger. And what they usually do is go into the finger that you use the most, right here [indicating index finger] and go into the most innervated part of the finger, the most sensitive, the most useful part of the finger. A friend of mine here in Pittsburgh taught me the trick: insist that since it's your blood that they want, it's your finger that they are going to poke, they are going to do it your way. Those of us who don't wear uniforms can make this clear.

What you can say is, "I'd rather you please go to a finger that I don't use nearly as often and go into a part of the finger that has a fine blood supply, over here on the side [indicating]. Look at it. It's a nice, healthy pink, not white or grey or anything. And yet, it has almost no nerves. If you just take a sample from right here (indicating), instead of right here (indicating), it makes my day a tad more comfortable." That's a trick of the trade that is not based on any heavy precepts in science or project management, but it's incredibly useful. Try it the next time you donate blood.

Well, in software, I'm going to say that one can't deny that some of computer science is, in fact, useful in doing software engineering. Program verification and database theory give you fundamental insights in how to build software. In project management, schedules are what we in my center, Bell Laboratories, proudly refer to as crowd control when you've got large herds of industrial programmers who are trying to walk in roughly the same direction. All that stuff is very important. You can't deny that.

However, there are a few things in software that are at a similar intellectual level to this trick of getting blood out of this finger [indicating] and I hope that they are at a similar level of usefulness also. That is the topic of this talk.

I've given a predecessor of this talk as an after-dinner talk on the ACM lecture circuit. So, this is really one of the first times that I've ever given it to a sober audience. It's certainly good for the after-dinner context. I hope it's useful here, to give you a few amusing stories and a prod to think about some tricks. My goal here, in addition to getting a chance to see Pittsburgh (and you thought it wouldn't be fun, until you came over the bridge last night, did you?), is to get a few more stories and tricks. That's one of the fun things about doing this. Mandatory here are real time thought and some discussion. There will be embarrassing places where you won't have anything to do if you don't speak up, because I'm not going to talk for a while. It's illegal to take notes. I didn't think you'd do that, anyway, but some of the nerds up front always do this.

So, I want to begin this part of the talk by referring to an excellent engineering book. Jim Adams of Stanford wrote this book, *Conceptual Blockbusting: A Guide to Better Ideas*. It's used in many universities in the freshman "so you want to be a nerd" course, where they talk about that subject and they talk about engineering. Engineering spans many disciplines, and in this book, Adams talks about a number of conceptual blocks that one has and how one can break through them. He also gives a number of stories from his own fascinating experience, of an aero and astro type. He talks in particular, at one point, of being on

the Mariner project and about sending the vehicle out to Mars. As soon as it goes exo-atmospheric, it wants to throw open some solar cells to get power for the trip. I thought, gee, why not use an extension cord? But then, maybe they were old fashioned. So as soon as it goes outside the atmosphere, they have to throw open the solar cells. They have a very short flight. But at the end of this flight, they have to retard the throw (???) somehow. So, they had to design several retarders. They started off in the zero version with one they had used on an earlier lunar craft. But it was an oil-based device that on this long Mars mission could have coated the spacecraft with a lethal coat of slime. It wouldn't work. They tried the next version, but that was too heavy. They went to a third version, and again that was fairly heavy. It wouldn't work.

As you know, you have to wait until things are in the right position when doing a planetary shot. The launch window was closing and they still hadn't solved the problem. At that point, they had a number of engineers working really hard—they were really going at it. Then, at the last minute, they decided to do a study and ask, "What happens, in the worst of all possible worlds, if—for this particular new set of solar cells—if we blow it? Just, for the very worst case analysis, what happens if our retarder just totally fails, if it just goes out there and stops? What happens?"

Well, they did the test. Anybody want to guess what happened? Nothing. So, with that bit of insight, the solution to the problem that they had been working on furiously for many months is shown here in it's entirety [indicating a blank screen] as number four. They had been struggling furiously to solve a problem that wasn't. Fortunately, that never happens in computing.

I was so interested, reading about that experience, that I tried to think about similar ones in my work. At one point, I was working for a public opinion polling firm on part of a three-stage process for drawing a sample. First you choose the right places in the state; then you choose the right precincts in the area; then you choose the right people in the precincts. For the second step, the polling firm wanted to draw a random sample from a hard copy list of precincts. Well, they were doing it in a typical way, with this long, boring table of random numbers in the back of a math handbook. They'd go through the list, and they'd stop at a place and count forward this many—a really dreadfully boring thing, and they can do this job in a couple of hours. The users said, "Gee, this is dumb. We have a new computer. Hey, Mr. Programmer, why don't you write a program for me like this: the input is that I will type in the precinct names, ten character strings, and an integer M less than N. I want, say, about 20 strings out of 200. And your output is then a random selection of M names."

Easy enough program to write. And that really is a trivial program. It is, however, morally wrong. What is morally abhorrent about this notion? What is evil? What are you asking a person to do?

FROM THE FLOOR: To provide information that's already there.

JON BENTLEY: Well, more to the point, you're asking them to type information that the

computer is going to promptly ignore—type in 200 names and spend an hour of your time doing it. If I want somebody to ignore 90 percent of what I type, I can teach undergraduates. I mean, you don't need a computer to do that for you.

What's a better way of solving this problem? Rather than typing that information in, to have it ignored, what can you do instead?

FROM THE FLOOR: Determine what the random numbers are and go down the list.

JON BENTLEY: Okay. I was so terribly proud that I said, "Gee, you don't really want to do that. Here's what you can do instead. You just type in two integers, M and N. For instance, if you want a sample of three out of eight precincts, you type in, 'I want three out of eight.' The output might be 2, 3, and 5. You just count through the list one, two, bingo three, bingo, four, five, bingo. You just count through this list." This is infinitely easier to program, certainly.

Now let's get down to the important point, not how easy it is to use but how easy it is to program. I think when I had ended the software engineering efforts—after I had done careful requirements and specifications and had gone through the high-level code design—the result was a 12-line BASIC program, 60 minutes later. I mean, it's not a major development effort, to do this.

Furthermore, it's eminently useful to the user. If you want to do this, you just go through typing the two integers. Type it out and it's a matter of a few minutes, within an hour.(???) It was a genuinely useful program. I was ever so proud of this solution. Here this person who thought of it originally had a conceptual block. The block was, "If I want output from a computer, I must provide it as input to the computer in the first place." I overcame that conceptual block. And unfortunately, I was tripped viciously by another conceptual block that I have. This is from a deficit. I have a handicap that many of you in this room have, also: education.

I talked about this at West Point. A cadet there had a much more eloquent solution than I. What's a really elegant way of doing this? What was the conceptual block that I have?

FROM THE FLOOR: Put the names in a hat?

JON BENTLEY: Put the names in a hat? Excellent. It turns out my conceptual block is I have this degree in computer science. I think all problems have to be solved with a computer. What the cadet said was as follows.

You have a hard copy list. What you can do is walk over to a photocopier, make a copy of the list, take a paper slicer, slice it up appropriately, then get a shopping bag. Put them in and shake vigorously. As long as the slips of paper are roughly the same size and as long as you shake really vigorously, you get a good sample.

As a matter of fact, these two incidents are roughly comparable. It's just that I got so down

on the creators of the first solution for having a conceptual block that I didn't realize that I, myself, had a conceptual block. I think that if most of us introspect more than a little, we can identify a number of programs that we've written, that...you know, should they have really been programs?

Well, after I discovered this conceptual block, I had another interesting experience with random objects. A friend of mine was writing a thesis in psychology. She called me and said that she wanted me to write a program for her, for N equals 72 psychological subjects. She wanted me to randomly permute an order of experimenters, 1, 2, and 3, and stress conditions high, medium, and low. So, when subject number one comes in, he sees experimenter number 3 in low stress first, number 2 in medium stress next, and finally number 1 in high stress. When number two comes in, he sees first experimenter number 3 in high stress, 1 in medium stress, and 2 at the low stress. Everyone see how that works? Each one of these [indicating chart] is an independent permutation of 1, 2, and 3, and high, medium, and low. That was the program that she wanted to write.

I thought about this for a while. I sketched it out and said, "Gee, I think I know how to write this program. Do you really want things without replacement?" That's what she wanted. I sat down and sketched the program and finally I said, "Wait! Wait! What's the next step? What can you see here? [indicating chart again] How many permutations are there on three objects? Six. Who here has ever generated one out of six random objects? I mean, how do you do this?

Now, I was so terribly proud, I said, "Wait! You don't want a program. What you want is two child's blocks. Write on each one the permutations 1-2-3, 1-3-2, 2-1-3, 3-2-1, 3-1-2, 2-3-1. And when the subject enters the room, just take these two dice, roll them across the floor, do whatever it says and you're guaranteed. There's no question that it's random. This is exactly the solution that you want." It turns out that this solution was abhorrent to her. This was horrible. Why was I ever so wrong? I, in fact, wrote her program for her.

What problem did she really want to solve? Did she really care about getting random orders? You've been there. What was the problem that you wanted to solve when you did your thesis? You wanted to get your committee to sign it. How did she think that she could get her committee to sign it? The real thing was that she wanted was the authority of the computer. What the user really meant to say was "I want an appendix that has this program and this computer produced output." Make sure that you know what problem the user really wants to solve—before trying to solve it.

Well, that's the first mini-sermon this morning: trying to solve the right problem. Think of what the problem is and what some solutions are. We will return later for a number of suggestions.

For the second topic, another trick of the trade. I'd like to discuss "back of the envelope" calculations. Shortly after arriving at Bell Laboratories, I had a talk with Bob Martin. At this point, he was an executive director and had about a thousand folks working for him. He was

widely known as a semi-human guru who did everything right. He did these amazing things. Here I was a rookie. I'd been there a few months. We were having this great talk about what are the real insights of software engineering.

In the middle of all this really technical stuff, he looked me in the eye. He actually made eye contact. He said, "How much water flows out of the Mississippi River every day?"

Gasp! "Pardon me?"

"How much water flows out of the Mississippi River, each day?"

Well here, the poor fellow had obviously cracked under the pressures of running this huge software shop. I had no choice but to try to humor him until help got there. How do you respond in this context? How much water flows out each day?

FROM THE FLOOR: A lot.

JON BENTLEY: A lot? I can't argue with thoughts like that. It's certainly true, a lot. But suppose you wanted to keep your job. Suppose you weren't independently wealthy—how much flows out?

FROM THE FLOOR: Well, you have to take the sum of how much flows in from all the tributaries and account for evaporation.

JON BENTLEY: Okay. So, one answer is "as much as goes in." That's a smart answer. But how can we do that? Give me a number.

FROM THE FLOOR: Well, make an assumption.

JON BENTLEY: Okay. Like what? What do you want to assume? How wide is the river?

FROM THE FLOOR: Width?

JON BENTLEY: Yeah. At the moment, it's pretty low. But on the average, in a good year, how wide is the Mississippi River?

FROM THE FLOOR: A mile wide.

JON BENTLEY: A mile wide. How deep is it? About a couple of hundred feet deep?

FROM THE FLOOR: Thirty-one.

JON BENTLEY: You know, those guys with the poles must have been really tough hombres. And how fast does it flow?

FROM THE FLOOR: Slow.

JON BENTLEY: Slow? Twenty miles an hour?

FROM THE FLOOR: Three miles an hour.

JON BENTLEY: Okay. My guess was as follows [writing on a viewgraph]. The numbers here are roughly the same. I guess, the river is, what, a mile wide. I figure it's 20 feet deep on the average, and it flows maybe 5 miles an hour. If it's a mile wide, times 20 feet, (5,000 feet in a mile), 24 hours a day, I figure it's half a cubic mile a day. Ballpark.

But why? There we were having this wonderful conversation, and Bob Martin asked, "What's the matter with you?"

At that point, he walked over to his desk and picked up a proposal. It was about one inch thick. It was a proposal being done in his organization for the mail system that AT&T was building—and putting the AT&T corporate logo on—for the '84 Summer Olympics. This was in February of '83.

The proposal went through a very similar sequence of back of the envelope calculations: How many users are there going to be? How many times a day do they send mail? All that sort of stuff.

Then he asked, "And how long does it take for one piece of mail on the system that we're using here?" He walked to his terminal, typed a message, mailed it to himself, timed it. It came back. Beep. "Ah, that long."

In two minutes, he showed me that the system that had come up through five levels of management to a guy who had a thousand people working for him could work if and only if there were at least 150 seconds in each minute. Here was a system that people had built, that had gotten to a proposal that thick [indicating]. There was this really detailed design, and no one had sat down to do a very simple, back of the envelope calculation to ask, "Is this at all possible?"

It turns out that it's the kind of mistake, too, that one wouldn't see and wouldn't find on a system test. You would only see it after it got loaded. At that point, whom were you doing in? On what poor, innocent victims were you inflicting this horrible evil, if this didn't work? Forget the athletes. The main user of the system is the world press. If there is one group that you don't want to have really mad at the AT&T corporate logo, it's certainly that group.

He convinced me. Engineers, before they build something, have to sit down and apply this little trick of doing ballpark estimates. They have to ask, "Is this in the right ballpark? Is this answer plausible? Well, can we check it somehow?" How would you check this [indicating figures on the Mississippi River]?

MARY SHAW: A suggestion over here. How much rain falls and how much evaporates?

JON BENTLEY: Okay. How much water flows in the river and how much flows out? As much as flows in. I figured the basin is, what, a thousand by a thousand miles wide? Maybe a couple of feet of rainfall, a foot. You take that thousand miles, times a thousand

miles, times one five-thousandth of a mile per year. That's a fifth of a thousand, 200 cubic miles a year. You can then use the well-known fact that there are 400 days per year, to give half a cubic mile per day.

Well, at this point, I talked to my colleague, Peter Weinberg, and asked him the question. He did this [indicating] calculation. I was amazed that his answer and my answer came out so close to each other. I had to know what the real answer was and I knew that my buddy, Mary Shaw, has written guides on canoeing in Western Pennsylvania. So, foolishly, I called her up and said, "Mary, Mary, I have to know. You have access to this Corps of Engineers data. Please, please tell me how much water flows out of the Mississippi River."

It was so terribly humiliating. I've never been so embarrassed—well, not often have I been so embarrassed— as when she observed that I had recently written a column about solving the right problem. I was trying here to solve the wrong problem, that of consulting an expert. The humiliating part was that she made me reach for my own almanac. I found the almanac that I had in my desk, opened it up, and got the calculation, which I'm sure that some second lieutenant did himself using this same stuff.

But I looked at the almanac. The discharge there was 640,000 cubic feet per second. If you multiply this through, it turns into four-tenths of a cubic mile per day. When both answers came out to half a cubic mile a day, I was pretty sure that the real answer was maybe within a factor of two or three. Now, I'd be surprised if it were more than a factor of three away from half a cubic mile a day. I'd be shocked if it were an order of magnitude off these two answers. The proximity of these two answers to one another, one the "correct answer," is a fine example of a really fundamental engineering technique: sheer dumb luck. I heartily recommend its use, whenever you can get it.

Here's a trick that a lot of folks learn in physics or chemistry. How much water flows in per year? You can make this table or else say that it's a thousand miles by a thousand miles by a five-thousandth mile per year. I'll go through here now and cancel out these three miles (indicating), give a cubic mile. One-fifth of that is 200. So, that's two or three miles per year. Now you multiply by a year is 400 days. You cancel out again. This gives half a cubic mile per year.

Who has seen this technique before? Who didn't raise their had just then? Okay. I hadn't seen it until I gave a talk at the ACM Chapter in Chicago, as a matter of fact, similar to this. Evidently it's taught in a number of physics classes and chemistry classes under the title "Dimensional Analysis" or whatever. This is a really handy way of doing this computation. It's a little trick, about the same intellectual level of poking here (indicating side of finger), rather than here (indicating end of finger) but incredibly useful for going through and making sure that you are multiplying by the number of users, times the average number of requests per day, times the number of disks, times the number of machines and that sort of stuff. It's a nice way of writing this stuff out.

A few reminders about the calculations. One is how to do them. Two answers are always

better than one and especially when they are quick answers. You can use this table (indicating), this dimensional analysis. You can do dimension checks. You can add V plus ... you can't add V plus meters or V plus seconds. There are a lot of quick checks that one can do, slide rule checks, to make sure that the exponent is right, to make sure that the leading digit is right, to make sure, if you doing the complete answer, that the trailing digit is right; all these things that people used to know when they were taught how to solve interesting mathematics, with real-life slide rules. People nowadays don't know as much as they should.

Common sense. I said that I had a problem of education, of having a degree in computer science. At one point, I asked a horribly handicapped person (he had a PhD in mathematics) how to solve his problem. He thought for a long time about how much water flows out, thought for a long time and said, "What 100,000 gallons?" I said, "Wait a second. You have been in pools that have more than 100,000 gallons." You know, they don't have to stop the Mississippi River for a day to fill that pool.

When to do estimates. This is an example of a really trivial object, a really trivial concept, a trivial engineering trick. I don't know about you, but at least in my company, it goes unused way too often. There are people who can become executive directors, fifth line managers and have a thousand folks working for them, that get to be that way simply by applying these tricks at the right time. In the case I mentioned, in Martin's case, he did it after a system was designed, before starting the coding. Rather than building this thing and then seeing that it doesn't work, early on in the project, really early on, he asked, "Is this design at all plausible?" You can get a lot of insight that way, certainly, before you do any efficiency improvements. You can also use this for cost benefits analysis. This is a little trick that, at least where I work, there are few people who do this and they are regarded by all as a superior race. They must just have better genes than the rest of us. In fact, I think they are just using a few tricks like this.

Questions or comments at this point about this stuff? Mary?

MARY SHAW: Lest anybody think that this is all irrelevant, I saw a report not long ago about the rating of the advanced placement calculus exams. This is all the kids in high school who were real hotshots in mathematics and trying to get out of taking college calculus. The graders were grading the exams and after some grading for a while, they decided to grade the results based upon a problem. The problem had to do with the volume of a water tank. They tabulated the range of the results. The largest of the results was somewhat larger than the body of a universe. The smallest of the results, if you ignore the negative results, was somewhat smaller than an atom. Now, these are kids who are in the business of showing off how much they know.

JON BENTLEY: I'll return to that shortly, with a test exam that I once gave on this material, after lecturing on it. That was the really embarrassing part.

To do these calculations, though, you have to have some data. I bet that most people in this

room have a lot of data they don't realize. For instance, I would guess that most folks here, in fact can, if they really want to, answer the question of when is a boy on a bike with a mag tape faster than a 56 kilobyte line? Answers of the form "When the line is down", are not acceptable. Over what distance is a boy on a bike with a mag tape faster? Well, that depends. How much information is on a mag tape? What's the density of a mag tape?

MARY SHAW: Two thousand bytes per inch.

JON BENTLEY: Boy, you're dating yourself. This is like saying, "Gee, I really enjoy watching Howdy Doody." How many?

FROM THE FLOOR: Sixty-two fifty.

JON BENTLEY: Sixty-two fifty what per inch?

FROM THE FLOOR: Bytes per inch.

JON BENTLEY: How many feet of mag tape?

FROM THE FLOOR: Twenty-four hundred.

JON BENTLEY: Twenty-four hundred feet. Is all that for data? Well, you lose some for a blocking factor, but basically you have this many bytes per inch, times 2400 feet, times, say , half for a blocking factor, which gives you about 90 megabytes. A 56 Kilobyte line is 7000 bytes per second, is 25 megabytes per hour. Basically the boy on a bike with a mag tape has about three and one-half hours to get there.

Back in the days of the Bell System, there were certain tapes that had to be distributed every night. They looked at a number of ways of doing this. Finally they settled on shipping them Greyhound on the late night run. They had to go a couple hundred miles. It turns out that a Greyhound bus full of mag tapes has a really awesome bandwidth to it. Optical fiber pales in significance, compared to this.

But for the right context, there are other examples. When Lockheed, out in Sunnyvale, had to get some test data over some mountains in Santa Cruz every morning, they used an automobile courier service. It was costing one hundred bucks a day to get it back and forth. It arrived late. They had to transport a few engineering drawings each morning. What did Lockheed really do? Hint: this solution is for the birds. They photographed it and used carrier pigeons. It replaced a hundred bucks a day, literally. It's cheap but it's early.

These simple calculations show you that certain schemes are very effective sometimes. Given there were some conceptual blocks, too, but if we want a data line, we have to have a data line. Who says you can't use a Greyhound bus or a pigeon?

Brooks, in The Mythical Man-month offers a rule of thumb, that in building a system, you should allow one-third of the time for design, one-sixth for coding, a quarter for component tests, a quarter for system tests (all components in hand), rules about balancing things, all

sorts of ninety-ten or eighty-twenty rules. All these are common things that people have. Some of this information, though, is not contained in quantitative rules, so much as qualitative rules. In designing an interface, there is the "principle of least astonishment." If you work for a company, every programmer in your company who designs an interface for a human being should know the words "principle of least astonishment." That way, if he does a poor job, you can dump on him with four words. You can sit down at the terminal, gasp in horror and say, "principle of least astonishment" and he will be mortified.

Gordon Bell has observed correctly that the cheapest, fastest and most reliable components in the computer system are those that aren't there. It is oh so true. All these things are things that people should think of, when they sit down to build things. In Strunk and White, White talks about being in Strunk's class and having Strunk say, "Vigorous writing is concise. Vigorous writing is concise." There are these things that come back to him 15 years later to haunt him, as he is doing his own writing. When you're building systems, you should think about things like this, "plan to throw it away." You will, anyhow. A lot of these things have a lot of truth to them.

I, at least as a programmer, spend a lot of time (I don't know if it is popular to admit this but it certainly is true and obvious) debugging. Most program analysts spend a lot of time debugging. People almost never teach debugging anywhere. It's the kind of thing people learn about in locker rooms and in the back seats of '57 Chevy's. But there are all these approaches to debugging that are useful. Program verifiers just simply don't make any mistakes. Well, you know, if you can sell it, it's fine and it in fact works. There are some big theories of test data selection, advanced debuggers. All these things are really handy, in their place. The best approach to debugging is the right attitude. I've seen exactly one book a really excellent book on debugging. There's now a second book, volume two, which is by Burton Vouchez, who writes "The Annals of Medicine" in the New Yorker. He collects really fascinating stories about debugging large, complex systems.

In one case, he talks about debugging Steubenville, Ohio. Steubenville, around Christmas ... it's just up the river from here ... around Christmas of 1980, was broke. There was a salmonella outbreak and they couldn't figure out what happened. It went from the individual physicians, to County Health, to State Health. They formally invited CDC to send an intelligence officer down to work on the problem. This guy would have been (it's a story of a tragic life wasted) a fine programmer, who wasted his life as a physician. But this guy would have been a really great programmer, who had the right attitude. When he came to Steubenville, he said, "I was prepared for hard work, for plenty of old-fashioned shoe leather epidemiology."

In such outbreaks, there's always a common denominator. Maybe they stop at the same place on their way home from work; maybe they've all been to the same church social, to the same picnic or something. There's always a common factor in any outbreak like that. He went through there, in this case, and couldn't find any common factor. It just wasn't there. They tried everything and they couldn't find it. The only common factor that they could find, in this case, was the age. They noted that in each household that had sal-

monella, there was one youngish person, one person in the range of 20 to 30, to quantify it. And, in fact, they did a study. They found that there was at least one person in each household in that age bracket. Is that typical? How do you answer that question? They went to the phone book, chose 50 numbers at random, called up, "Do you have a person in the age of 20 to 30?" They found out that the answer was yes about half the time. So, the fact that in the 20 infected households, the answer was 20 for 20 and in 50 random, it was 25 for 50, was highly significant.

That really is a lead. It's not just a statistical fluke. That's something very important.

What does that age group have in common, that might lend to this, in Christmas of 1980, in Steel Valley? Any guesses? In walking into the supermarket, this physician, at one point looked around and saw a person stumbling around drunk and said, "Ah ha!" What could it be? They were worried about substance abuse. It was a really depressed area around Christmas time. People do a lot of substance abuse. They went back and they did a study. They asked all the parties, "Have you been taking any drugs, of any form, any alcoholics, whatever?" And the answer always was, of course, "Oh, no. We never do drugs. Now and then, sometimes we smoke marijuana." In every single instance, they found that every party had smoked marijuana. Well, those of you who have been to college in the past couple of decades, probably remember what it smells like. It smells a lot like burning horse manure. What they found out was when they actually did the clinical study on some donated substance the people gave them, was that the substance they were being sold on the street was, in fact, half marijuana and half ... The distributor did not have a high quality control program. This was finding its way into their final product. In fact, at that point, everyone became deliriously happy. The County Health was fretting, "Oh, what have we done wrong? What sort of horrible, dreaded disease have we let out on our public?" In fact, this was an illegal substance. It just made everyone absolutely happy. Furthermore, it explained several other salmonella outbreaks that weren't explained before. All this by solving a really hard, complex system, finding the one thing that was different, the one thing that was strange and pursuing that vigorously.

There are a lot of similar stories about computer systems. I heard one story one at an ACM Chapter meeting, where a guy had written a banking system that worked just fine. The banking system worked really well. They used it in a number on contexts. But the first time that they used it on international data, the banking system quit in the middle of a run. It just quit. Programmers were assigned to go fix this problem. "It quit for no reason, at all. I want you to look through the source code and find this missing quit command." There was no way that it could quit on this data but it was quitting. How could it do this?

Well, they spent literally a week, having people pour through the code. They couldn't find any commands. Finally, one excellent debugger walked into the room and said, "Wait a second. Exactly where is it quitting?" "On international data." "Yeah. But from where?" "From South America." "From where in South America?" "From Ecuador." "Well, on what city is it quitting?" "Well, when I type in the name of the capital city, for no reason, ..." You know, "quit-o" or "Quito"? Okay-o, I'll stop-o right now-o. It's times like these when you

want to wander up to a programmer, look him squarely in the eye and sadly mutter, "Principle of least astonishment." This system grossly violated the principle of least astonishment, when by typing five letters, instead of four, you get the first four back.

At IBM T. J. Watson Laboratory in Yorktown Heights, there was a programmer who got a new terminal. It worked just fine when he was sitting down, but the programmer couldn't log in standing up. What's going on? There's something that knows whether they're sitting down or standing up? What is it that knows? Well, there are all these theories that people throw out. "Gee, is it a wire here? Is it this, is it that?" But what is the main difference between the way that you interact with a terminal sitting down and the way that you interact standing up?

MARY SHAW: Which half of your bifocal?

JON BENTLEY: Which half of your bifocals? Well, that's part of it. Hold your hands up. Show me how you type when you're sitting down. Okay. How do you type when you're standing up? What's the difference between this (indicating sitting position) and that? When you're doing it this way (indicating standing position), who tells you where the keys are? Your brain stem or something south of that?

When you're doing it this way (indicating standing position), who tells you where the keys are? Your eyes. What had happened here? there is a difference between what your brain know and your eyes knew. What could have been the difference?

FROM THE FLOOR: The keyboard was ...

JON BENTLEY: Exactly. The tops of two keys were swapped. A, S, D, F, ... K, L; two keys were swapped. When he was typing this way [indicating sitting position], his brain didn't care about the tops of the keys. He said he couldn't see them anyway. But as soon as you type this way [indicating standing position], your eyes guide you. People were perplexed for a week over this. What could be the difference?

Finally, one person asked the right question about the main difference between typing this way [indicating standing position] and that way [indicating sitting position]. How many of you, if you looked at a keyboard, would notice if two keys were swapped? You know they are roughly in the right order. If only four percent of the keys were wrong, and you wouldn't ever notice it. But once you know to look for that ....

There are a lot of examples of great stories of debugging that I think need to be told more often. I think this is what the super-programmers do, what a lot of the really excellent systems people do. And too often, we don't exploit this fact. I think that a lot of this behavior can be taught. Again, there are these two excellent books that teach it.

I've now surveyed four tricks. Time is almost up. There are a lot of other tricks from monitoring tools. Electrical engineers never wander far away from their oscilloscope. When they build something, they have the tools for monitoring, the tools for understanding it. We

need more of that in computing—looking at the data or going out and doing the statistical study, whether it is a Knuth study of FORTRAN programs to see how shall I build my compiler or other studies. You have to look at the problem to solve, graphic split, data testing. All of these things are other kinds of tricks of the trade that most engineers learn in their undergraduate education, not in a class, but just by being immersed in a culture for four years.

I think this is one thing that software engineers have not had in their education. I learned a lot of it around the lunch table at Bell Labs. It is a cultural issue. I think it's the kind of thing that we can teach by example. I have written about these things in various books and columns. There are some references there, if you really want to know.

Teaching the tricks. I was at a really humbling meeting, within Bell Labs, of a bunch of really excellent software designers, people who were known for really good software design. A few of us were on a committee that was appointed by the president of Bell Labs to figure out what they knew that the rest of us didn't. At one point, one guy said, "Wait a second. What is your undergraduate degree in?" The answers were really quite varied. They varied all the way from electrical engineering, to mechanical engineering, to aeronautical engineering. I was the only person in the room—and how embarrassed I was—who didn't have an undergraduate engineering degree. One thing that we deduced from this was that maybe, in an engineering program, you learn how to think like an engineer. Maybe you learn some of these basic things that aren't taught in math classes, for instance.

When you teach, these stories are fine in small doses. You can't have a whole lecture, I mean, this much is almost unbearable. You can imagine what two hours in a row would be like. Little doses are fine. You typically learn by osmosis. This is the kind of thing that in corporate education, you can put in in little places, in the rest of the course. First-person stories are best, especially when you can show how you are the butt of the joke. The best stories span disciplines. If I had more time, I would tell you some of the stories that were very useful at West Point, in talking to the undergraduates there, about "some engineering techniques that are illustrated in an Army context but apply equally in any kind of system."

I want to review a few tricks for teaching these things. In problem definition, for people who are teaching at universities, what you're essentially trying to do is to undo 16 years of education. If there's one thing people learn by the time they're a senior in college, they know for sure that the problems at the end of chapter 7 are solved using the techniques in chapter 7. I mean, there are certain things that a college education teaches you, and that is foremost amongst them. When I taught mathematics at Carnegie, I used to really upset students by having a problem at the end of chapter 7 that you could, in principle, solve using the techniques in chapter 7. You'd have to stand on your left ear and squeeze it one way and you could use the techniques. If, however, you used the techniques in chapter 6, the problem was trivial. The students would yell and scream and say, "You aren't fair! Your aren't fair!" It really is a case that in defining the right problem, too many students take that with them, into their place of employment. The people sort of assume that the problem I really want to solve is the problem I say I want to solve. They don't sit back and say, "Now, what problem

do you really want to solve? Why are we doing this? What is the benefit from doing it that way?" This is one of the fundamental things. You can't teach this off in some education department somewhere. It has to be part of the atmosphere where you work. "But wait, is it really what's best for the company ... to do it this way?"

The back of envelope. To underscore Mary's example, after I had taught this material in a college course, I asked the cost of a one-semester, 15-student class (at a particular college, if you assume such and such about the budget, et cetera)? Most answers were near my answer, about $50,000. The answers varied, though, from a high of $100,000,000 down to a low of $38.05. This nickel offended me so deeply that the student lost a point. I think that for $38.07, he would have lost two points. But the nickel really bothered me. People should learn how to do this, even giving a lecture on using it in other classes and teaching it. This is important stuff. You have to do it. You have to show the appropriate use about the course. (??)

Where I work at Bell Labs, we oftentimes go out for a walk around the grounds after lunch, trying to solve the great questions of our time, like what weighs more, mankind or mosquito-kind? It's part of the culture. It's simple, back of the envelope things. How many G's were the shuttle astronauts put in, for how long? All these are a part of the culture; they're things that you really can do fairly easily.

Rules of thumb. Polya, in his great book *How To Solve It*, says that students learn how to solve problems by imitation and practice. You have to use these rules of thumb, and all the lectures have to season the people. They pick it up. There are some things like the principle of least astonishment. I honestly believe that if you see that six more times in the next three minutes, you might actually take it away from here. You might get two useful things out of the lecture, about this and the principle of least astonishment. I'm going to try every way I can to say that at least five more times. It's the kind of thing that people have to hear. You can actually get it down into the brain stem somewhere.

Again, White talking about Strunk, says that 50 years after hearing the lecture, he could hear his voice calling out, "Omit needless words." Boy, there's a self-verifying sentence: "Omit needless words." He said that he got his prose down to be so sparse that he would have no more than a two-minute lecture, and so he would say the same thing several times. "Omit needless words. Omit needless words." It's true in programming, as well as in writing.

Debugging. Like at any college, the students at West Point hate buying books. It used to really irritate them because, when I taught there, the U. S. Government had to pay me. For some silly reason, if you tried to give a tank to the government, if you'd just drive it to Fort Knox and say, "Here's a tank. I made it for you. Don't you like it?" They throw you out. I tried to teach at West Point in the same way, and it was, "Are you crazy, kid? Go away." So finally, they had to pay me. I couldn't take the money, so I bought copies of Vouchez's book for all the students in the class instead. I assigned them each to read one chapter from this. Then, for those awkward, five-minute pauses at the end of the lecture, instead of

just saying, "You guys can go away early now," it made one guy stand up. They would fight to do this. They would stand up and tell their story from the medical detectives and an interpretation for how it relates to computer systems. Incredibly popular. We actually had students (I think it was the first time almost in the history of the Academy) voluntarily read the book given to them by the professor. It was just amazing.

Anyway, what I've tried to do here is certainly a bunch of tricks. I've covered four tricks in detail. I've talked about how you can teach them in various contexts. Why should you teach them? Because they are useful. These are the kinds of things that, more often I think that we care to admit, make projects or, by their absence, break projects. They are useful in the real world. Now you can learn these things in any college class. They're fun, both in school and in a career. What more can you want?

MARIBETH CARPENTER: Thank you very much, Jon. I think we'll find Jon's talk very appropriate to our workshop. He has taught us the importance of questioning requirements, some interesting debugging techniques, and probably the most important of all, the importance of lateral thinking.

Table of Contents