LUNCHEON KEYNOTE
How Will You Clean Up the Present Mess?
(Law, guilds, evangelism, cloning, or better miracles?)

DoD Software Technology Conference
Salt Lake City, UT -- 1999 May 04

By Bob Bemer

I must begin with apologies. First to conference organizers for the length of my bio in the program. I couldn't describe 50 years of programming in 75 words. It's difficult at 1.5 words per year.

The second apology is for not making a place, in the Environment Division of COBOL, for the "Assumptions for this Program". Had I done so, programmers would have had to confess there that the world ended in 1999 for their programs.

The third apology is for my title. Originally there was a subtitle, so you would know the mess I talk about was current software, not what we see on the TV news.

In the conference title someone had the good sense to call it "software technology" and not "software engineering". The latter was our optimistic title for a meeting in 1968, at Garmisch-Partenkirchen in Bavaria. I attended, and had the chutzpah to propose the original concept for a "software factory". That's also when Doug McIlroy proposed software piece parts. More on this later. Maybe it's good that a lot of those people are dead.

But the Year 2000 problem, large as it is, is only symptomatic of a larger mistake, which was to let the computer take over the running of our lives. They have, you know.

On the word "mess" again. I think the whole software world contains a substantial portion you could call that. And even the good parts have cost needlessly by replication and not being able to exchange knowledge and software well enough. What we need are warnings on how to avoid messes. I'm going to try. And I'll do it by examining these possibilities:

- Cloning
- Standards
- Evangelism
- Better Miracles
- Books
- Guilds
- Law
- The Marketplace

BY GIVING YOU

But first I'll set the stage with three examples of the mess I'm complaining about.

EXAMPLE 1 -- CHARACTER CODES

LUNCHEON KEYNOTE
How Will You Clean Up the Present Mess?
(Law, guilds, evangelism, cloning, or better miracles?)

DoD Software Technology Conference
Salt Lake City, UT -- 1999 May 04

By Bob Bemer

I must begin with apologies. First to conference organizers for the length of my bio in the program. I couldn't describe 50 years of programming in 75 words. It's difficult at 1.5 words per year.

The second apology is for not making a place, in the Environment Division of COBOL, for the "Assumptions for this Program". Had I done so, programmers would have had to confess there that the world ended in 1999 for their programs.

The third apology is for my title. Originally there was a subtitle, so you would know the mess I talk about was current software, not what we see on the TV news.

In the conference title someone had the good sense to call it "software technology" and not "software engineering". The latter was our optimistic title for a meeting in 1968, at Garmisch-Partenkirchen in Bavaria. I attended, and had the chutzpah to propose the original concept for a "software factory". That's also when Doug McIlroy proposed software piece parts. More on this later. Maybe it's good that a lot of those people are dead.

But the Year 2000 problem, large as it is, is only symptomatic of a larger mistake, which was to let the computer take over the running of our lives. They have, you know.

On the word "mess" again. I think the whole software world contains a substantial portion you could call that. And even the good parts have cost needlessly by replication and not being able to exchange knowledge and software well enough. What we need are warnings on how to avoid messes. I'm going to try. And I'll do it by examining these possibilities:

- Cloning
- Standards
- Evangelism
- Better Miracles
- Books
- Guilds
- Law
- The Marketplace

But first I'll set the stage with three examples of the mess I'm complaining about.

EXAMPLE 1 -- CHARACTER CODES

Some of you are aware of a terrible waste of resources in that the IBM mainframe world uses EBCDIC, whereas others, and PCs, use ASCII. A 1-to-1 translation between the two exists, and it can be done imperceptibly via a chip. A few of us fought to have it that way. But a file ordered on an ASCII key just won't do when the EBCDIC ordering is needed. And that costs money and time.

I mention this because it is a classic software mistake. IBM was going to announce the 360 in 1964 April as an ASCII machine, but their printers and punches were not ready to handle ASCII, and IBM just HAD to announce. So T.V. Learson (my boss's boss) decided to do both, as IBM had a store of spendable money. They put in the P-bit. Set one way, it ran in EBCDIC. Set the other way, it ran in ASCII.

But nobody told the programmers, like a Chinese Army in numbers! They spent this huge amount of money to make software in which EBCDIC encodings were used in the logic. Reverse the P-bit, to work in ASCII, and it died. And they just could not spend that much money again to redo it.

After all, the entire 360 venture was nicknamed "You Bet Your Company", after a TV game show of that era. And IBM found the reason, or excuse, to use EBCDIC in the huge costs to their users to change their existing files to ASCII ordering. But this short- range argument fell apart when we added a lower case alphabet.

EXAMPLE 2 -- UNDETECTED FAILURES

In early 360 days, the Univac 1107 competitor had an extremely fast and good FORTRAN compiler, made by Computer Sciences Corporation -- still with us. We pitched it to a certain Government agency via a comparative compilation of their FORTRAN programs. For one program there appeared a diagnostic saying that there was a jump to the middle of a DO-loop. The person in charge said it wasn't possible; they had been running that program on IBM equipment for three years.

A programmer sent to check the source program reported back with "Sorry, sir. Three years of wrong answers".

EXAMPLE 3 -- Y2K

Dr. Edward YARDENI, the Chief Economist of Deutsche Morgan Grenfell, runs a wonderful website at:

http://yardeni.com

You've seen him on television, and he has testified often to the U.S. Congress on the Year 2000 problem, faulting the shoddy nature of existing computer programs heavily as a major deterrent to fixing the Year 2000 problem. I quote:

> "Software programming is far less disciplined and rigorous than most people realize. Two different programmers can and do write completely different programs that will perform exactly the same task. Programming is more of an art than a science. One of the biggest Y2K headaches is that few programmers take the time -- or are even asked -- to document the logic of their programs..."

His catch line is that all this software has been built

"without adult supervision"

Yes, that's how they see us programmers. But actually he was complimentary to suggest only two programmers writing the same program. How about a thousand?

Our entire problem with inadequate software was well-described in the first decade of computer usage, even better than Yardeni does it, by the Okefenokee philosopher Pogo Possum, who said:

"What we need are more expert experts
than the experts we've got"

This talk is about satisfying that need.

CLONING

We can probably dispose of this option quickly. It sounds good at first. Find some topnotch programmers and clone them.

One of my favorite examples of great programming was done by a Chris Kilgour for the original General Electric 235 timesharing system. Every line of source code had meticulous documentation which, if taken separately and adjoined, was a running commentary on the process. It read like a story; just full of realworld reasons why he did this and not that. Realtime was pretty new then.

When Dartmouth wanted to make a look-and-feel replacement with a more advanced computer, they didn't groan and say "Why wasn't this in COBOL instead of machine language, so we could move it over easily?" They just threw away Chris's GE-235 code, and wrote new code for the new machine to match his comments! Worked fine.

But the argument won't hold. Takes a while for clones to grow up; they won't get the same education; and when they are ready the software world will have changed. We know, too, that progress of a society depends upon randomness. New methods seldom come through in an regimented and artificial society.

The reason I included this as an option may be because it was once suggested that I be cloned. To which everyone objected "Oh, no! One Bemer is enough". The sociological trumps the technical. I see the names Barry Boehm and Watts Humphrey in the program. They can verify!

STANDARDS

I can't explain it, but the U.S. seems to have antipathy to standards. In the early 1900's there was a story in a New York paper about the German ocean liner Hanseatic burning to the waterline. Seems the New York Harbor fireboats were not equipped with the "so-called international standard" hose fittings!

For another instance, we know the metric system to be the legal measurement system for the U.S. for over 130 years. The foot is defined legally as so much of a metre. Yet Peter Jennings still cannot pronounce "KIL-o-metre". He says "ka-LAW-ma-ter", which, if extended to the butcher shop, would find us ordering "ka-LAW-grums" of hamburger, and tuning our radios to so many "ka-LAW-hertz"!

I hate to tell you, but the U.S. Government is to blame. They put a commission of Ph.D's together, and it decided that our people were too dumb to accept the "-re" spelling of metre as a unit of length. Never mind that the rest of the world had no problem. Why the antipathy to the "re" ending, I don't know. Our Vice President doesn't share it! And we're still spelling our land unit as "acRE", on which may grow maple trees of the genus "acER".

So Peter Jennings thinks it's an instrument, not a distance, because the Government says "Spell it METER".

We have tried to have standards for the computer business. You may be surprised to know that I created the wording for the scope and program of work for the X3 committee in the U.S. and ISO Committee TC97 of the International Standards Organization. Moreover, I did this at the behest of IBM while employed there.

And we did pretty well for a time. We made the standards for FORTRAN and COBOL. And there were Federal Information Processing Standards, a system now fallen into disarray, of which the very first was ASCII. But we still see one when we use the 2-letter token for the state.

It may have been microcomputers, and their explosive growth, that crippled the standards efforts. I was keynoting the first national microcomputer conference in Chicago in 1979. Got a lot of comments from a man in the audience -- Ted Nelson of Hypertext fame (which is why we wear out our "h" and "t" keys). He said we dinosaurs were on our way out, and newcomers were going to scrap our stupidities. Well, I couldn't argue with that, but I did ask him to be sure to salvage all we had done that was good and useful.

Why am I hopeful when someone sends me a WordPad document? My machine is supplied with WordPad. But when I see a mess of open rectangles I know it's not the same WordPad I have! And why so many incompatible versions of UNIX all these years? Thanks for much, Ted Nelson, but not for the loss of standards! You said you'd start from scratch. So now I reboot Windows 95 three times a day. And your PCs all have Y2K problems!

EVANGELISM

Here you'll have to ask Ed Yourdon and others if structured programming stemmed the tide at all. I think not. The computer business just grew faster than we could baptize converts.

Some great programmers have tried to exhort the rest of us to good practice. The best known is Edsger Dijkstra, who wrote the famous paper "GOTOs considered harmful". And so they are -- sometimes. On the other hand I once wrote a program just full of GOTOs, for efficiency, and when I showed it to Don Knuth (surely as fine a programmer as ever was) he just laughed with appreciation, because I had written the program most carefully, preparing for every possibility of things that could go wrong. Explicitly!

One must always prepare programs for failure, knowing that they will fail. In my 1979 Interface Age article that included my (and the world's) second warning about the Year 2000 problem, I made this caution:

> "If from lack of knowledge you cannot program for all conditions, at least put in a stop and display whenever you are unsure ..."

Always know where the program came from. If the story about the USS Yorktown is not apocryphal, would it not have been much better for the program, rather than crippling the ship upon a divide by zero in Windows NT, to have jumped to a program part that called the Captain and said:

> "If you don't want to spend two days in the Atlantic being towed to port, just turn the power switch OFF and then ON again to reboot!"

So evangelism may fail in what method you're selling, but not how you're selling it.

For every computer program you can show me that can adapt to change, I can show you thousands that cannot. I've known some inflexible people in my time, but none as INFLEXIBLE as an old computer program!

For every program you can show me that conforms to the rigorous methodology of the theorists, I can show you a million that do not, that are full of MISTAKES, and ignore conditions that will arise. They're documented (if at all) in a way that defeats their reuse and adaptation by others. Even now researchers still seek the truth about what makes a program satisfactory in all ways. We just don't know! And if we did, how would we get everyone to conform to that methodology?

So I must conclude that the evangelistic approach has something lacking. And also that it often gets carried to excess.

For example, take the subfield of "proof of program correctness". I wish I knew how many advanced degrees have been awarded for work in this field. I admit there are some "must-be-perfect" classes of programming where proving work might be effective. I wish we'd done it before that faulty FORTRAN punctuation killed a space probe. But if it's so useful and possible, why wasn't it applied to all those programs with the Year 2000 mistakes in them?

I admit I'm sour on this particular evangelism. I once lost out, in a Turing Award nomination, to an Englishman in this field. And at a time when video displays existed, too. Tell me how an icon designer can write proof that his output will look like a telephone.

BETTER MIRACLES

The history of software is replete with miracles. But a true miracle should last more than 6 months, and it should actually work in practice. In 50 years as a programmer I've seen more snake oil than St. Patrick saw snakes.

I guess it started when COBOL wasn't good enough, so they dreamed up PL/I. Then that wasn't good enough, so they conceived ADA. And structured programming and function points. But that wasn't good enough for PCs, so some people conceived SmallTalk, and C, and Java, etc., etc.

That's only for language miracles. Then we got networking, then client-servers, then entire run-your-business packaging. The jury is still out, but the real miracles will not occur in these hailed breakthroughs.

The real miracles might occur IF we built software like Detroit builds cars. My catch phrase in proposing the concept of the Software Factory in 1968 was "How Does Detroit Do It?" The essential ingredient was that, if programmers couldn't make a quality product that would pass programmed inspection, they'd never get paid -- a powerful incentive. And I wanted estimates before starting, scheduling, schedule review, production control, and quality assurance -- all embedded in the computer programs running the factory.

We were to have basic standard parts, interchangeability, and re-use. Just like the original Springfield rifle. Parts stocked for every purpose, with catalog descriptors to find them.

Unfortunately the U.S. did not adopt the concept, although the Japanese did construct some software factories. The most we ever got in the United States were some software garages!

BOOKS

I get catalogs of books from the IEEE Computer Society. Most of the titles are so high-flown that I just know I could not understand the content, let alone apply it usefully. And forgive me, neither could those of you that have concerns about the software production process. So these books cannot have an appreciable effect in curing the real volume of our problems. But the societies can play another role effectively, and I'll describe that at the end.

GUILDS

Guilds must be considered. They were a time-honored way of passing on good practices and knowledge to new workers.

But one master had very few apprentices in that system. And not all would eventually become masters. That was all right for the growth rate of people, but very inadequate for the growth rate of computers.

So scratch guilds. But I had to mention that possibility.

LAW

Now I'm serious. So far we've avoided any serious impact, but perhaps never again will software builders get off scot-free for their mistakes! Not after Y2K! Our deficiencies have been exposed -- to people, TV, newspapers, and to governing bodies.

Paul Strassman's ComputerWorld article of last Jan 04 basically said that in the case of Y2K mishaps the excuses and artifices won't hold up. The applicable quote is:

> "What an irony! Of all people, it will be the lawyers who finally teach the IT profession how to provide quality software. They will be paid well for teaching what could have been mastered less expensively years ago."

In my life as a programmer I've never taken any exam to prove my fitness to write good programs. I've never had to apply for a license to write programs that would affect public safety and welfare. Good thing I wasn't a CPA, or a barber, or a school

bus driver -- tasks that affect a small number of people. But no law stops me from writing a program for a bank that could, if wrong, affect millions adversely!

I must admit a bit of responsibility here. Somewhere in 1971-1973 the California Legislature considered regulatory measures. An AFIPS committee was formed. I confess to suggesting an AFIPS Good Practices series of books, and they bought it. Where is that book series today? In the hands of how many programmers? I forgot a basic principle -- laws seldom if ever disappear; there is always someone with a requirement to enforce them.

THE MARKETPLACE

Until now I have sounded quite pessimistic, I know. Software manufacture is beset with the devils of invisibility, complexity, and ever-increasing speed. To top that off, we have what I have termed the POET effect -- the POET being for the "Persistence of Established Technology". But let's do a quick trace of the general history of software, seeing how personal pride, money, individualism, and competing companies played their roles.

We really started with SHARE, which actually did interchange software piece parts. Then to operating systems and compilers. Increasing complexity led us to formal alpha-beta tests, the more so because people began charging for software as a commodity. But still under control of large companies that had reason to integrate, lest they incur liabilities outside their control.

But at the same time the mechanisms for knowing what anyone else was doing for your same needs failed, so most wrote their own special software. This required armies of less-than-experts.

Note that phrase "knowing what anyone else was doing"! What do we know that fixes that problem best? The WEB, of course. So now, just in time, comes the means to do it right, if we'll only use them. We must find the "best of breed" in interchangeable and reusable piece parts.

I know there exist monolithic programs out there for running your entire business. But they often require you to run your business *their* way. Little tailoring is possible. Maybe you'd like to maintain control of your own business, and not spend three years of agonizing changeover to the new miracle.

Look at plumbing, autos, and armaments for examples. Costs only go down, and reliability up, when manufacture is controlled and reduced in variety. Merge the computing requirements for all businesses into one storehouse of component processes, and you will find a redundancy of at least 100:1. Would you believe 1000:1 in the U.S. alone?

This looks much like Track 5 here, on collaborative processes. It might be the most important venture you have going. The IEEE, ACM, and other professional societies should mostly quit the books business and serve as guides and repository agents for software piece parts. Have them bring the whole world into the act. Have your credit card ready to buy a chunk of program off the shelf, doing what you want to do. Or, if you are integral enough, buy actual execution of such software pieces from someone's WEB server. Just like one buys power from Con Edison.

There we may find the more expert experts that Pogo says we need! The marketplace will be found, as usual, optimum.

 visits since 2000 Aug 11 Back to History Index Back to Home Page