

INTERVIEW WITH JERRY WEINBERG

Magnus Ljadas

Super consultant, host of the AYE conference, a prolific blogger, and author of numerous books on consulting, software quality, writing, and most recently the sci-fi novel "The Aremac Project" - Gerald M. Weinberg is a software development legend. Although he's on a lifelong crusade against misery, he's always fun and witty in writing. This interview with Gerald was made by Magnus Ljadas.

Q: You've been active in the software development field for more than 50 years and still no signs of retirement, what's fueling you to keep on going?

A: It's like Edmund Hilary said when asked why he climbed Mt. Everest. "Because it's there." I first read about computers, when I was eleven years old--then they were called "giant brains." As a smart but miserable young kid, I understood that if we were to conquer misery, we would need more brainpower. I immediately knew that I had found my life's work.

So, the problems and opportunities of software development (and maintenance) field have been there my entire adult life. I happened to be born into the age of programming--not just computers, but as deep down as the DNA in our bodies. It's what the German's call "zeitgeist"-- we don't have a single word for it in English, but it's the intellectual and moral tendencies that characterize an age. Intellectually, it's obviously challenging enough for a lifetime, but morally it's even more interesting. The concept of programming has given us infinitely more choices for our future. We're not stuck with what we have up to now.

The Four Horsemen of the Apocalypse are described in the Book of Revelations as Pestilence, Famine, War, and Death. They were seen as inevitable curses of the human condition, but every one of the four has already been altered by programming, some for better, some for worse. And this is just the beginning. How could a person not keep going?

Q: For more than half a decade we have seen a steady decline in applications to computer science courses here in Sweden. What possibilities are young people, who want to make a difference in this world, missing out by shunning CS education?

A: Not much, not if CS education in Sweden is like so much of it in the USA: of no practical value. I'd say the students are better off preparing themselves for the future by learning to communicate with other human beings, learning to think like an engineer or physicist or business person, and learning to finish projects that always earn an A+, never settling for B or C work. Those things will open opportunities in the future, which by definition is unknown.

Yes, we'll need software developers. That seems pretty sure. But formal CS education is probably not needed to become a software developer; I never had any. And if it does turn out to be needed, computing technology is something you can study once you're out of formal schooling--when what you learn won't already be obsolete. I don't mean to say that students shouldn't take CS as a major in college, but if they do, they should pay more attention to the rest of the universe than typical CS programs do.

Q: You've written more than a handful of books about how to amplify your own effectiveness, and constant learning and relearning seems to be one of the key elements in those books. What can the typical software development organization do to support learning?

A: First, there's got to be an attitude that understands how important learning is to a software development organization. If you don't stay ahead of this ever-changing field, you fall behind. Some organizations work on a business model that hires people whose knowledge is leading edge, uses them for a couple of years without spending a penny or a minute on education, then dumps them for a fresh crop of leading-edge people. What they don't understand is how much on-the-job learning walks out the door with these "obsolete" people.

Cost accounting would help--cost and value accounting. If you really figure out how much value one correct new idea can bring to a software project, you would never cut short on any kind of education for your software people. So, assuming you value learning, what can you do as an organization? First, budget time and money for it, and don't cut corners. Have a training budget and a book budget for each person, and make each manager responsible for seeing that those budgets are actually spent. Do not reward managers for "savings" on the education budget.

Part of the budget should be time and resources for every technical person to "play" with new things--tools, languages, algorithms, just trying things out away from real projects. If you don't do this, people will play (experiment) on real projects, creating real risks. And

allow and encourage employee-generated educational events, such as brown bag lunch seminars or reports on training and conferences, reading discussion groups, membership in professional organizations, and visits to sister sites. Create an environment where social pressure favors learning, rather than simple nose-to-the-grindstone, 24/7 work on sponsored projects. You'll find that your "real" work gets done faster, cheaper, and better.

Q: Talking about education, what about miseducation? What do you think is the most widespread harmful idea about software development?

A: That by shouting at people and threatening them, you can force them to produce better software, faster and cheaper. Software development is an intellectual effort of the highest order--but in order to be able to remain consistently at the intellectual level needed to produce software, people need a safe, quiet, properly organized environment. There are no shortcuts.

Maybe I can generalize this dangerous idea into a short and simple and wrong sentence: There are shortcuts to software. Wrong. Wrong. And Wrong.

Q: You must have seen a whole bunch of ideas, about how to best do software development, grow and die over all those years. Do you see the agile movement as a pendulum swing or is it a move in a new direction?

A: How about a pendulum swing in a new direction? It's a pendulum swing because approximately every decade, there's a fresh movement to "solve the programming problem." High-level languages, structured programming, object-oriented programming, ...

But it's a new direction because it's the first movement to focus largely on social processes rather than purely intellectual ones. For that reason, I believe, it has more hope for success than the earlier movements, each of which made a little progress, then largely ran out of steam before achieving its grand promises.

Of course, agile won't achieve all its grandest promises either, given the conservative nature of human beings, but that's all right. After another dozen decades or so of incremental improvement, we'll begin to see some really fine software development. Well, I shouldn't say "we," because none of us will see them, but at least our great-great-grandchildren will be able to look back at us and laugh at our crude methods.

Q: Your most recent book is a novel called "The Aremac Project". You said to me, in an earlier conversation, that what you are trying to do with fiction is to reach more people in a perhaps more palatable way with some of the hard lessons of software development. Without spoiling the story, what hard lessons did you want to portrair in your novel?

A: Nope. The readers have to "experience" the lessons, through the experiences of the characters. That's the whole point. 90% of people will not or cannot respond to "telling," which is why all our education is experiential, and why the novels will all teach through the experiences of the characters. I wish that you could get people to use better software development practices (and drop the worse ones) simply by telling them, but that only works for a small fraction of the population. In a way, it's unfortunate that it works for even a small fraction, because some of those easy-adopters become evangelists, and can't understand why the rest of the world doesn't need to be told what's good, but needs to experience what's good. That's at least part of the reason new practices turn out to be overblown. They might be as good as their evangelists say, if only people would use them.

Playwright George Bernard Shaw has been quoted as, "The only thing wrong with Christianity is that no one has really tried it." That might be a bit extreme, but we might say about certain software practices, "The only thing wrong with X is that it's seldom really been tried." You have to try it to learn it, to evaluate it, and adopt it.

Q: Can you give some examples of "talk and no walk" software practices?

A: First, there are the people who don't walk because they don't know what walking is. During the structured programming movement, I had countless clients who claimed to be doing structured programming, but when you looked at their actual code, it was spaghetti, spaghettini, spaghettoni, linguini, vermicelli, or fusilli lunghi. Anything you could twist and tangle, they had it--and called it macaroni. In more modern times, I've seen dozens programs that claimed to be object-oriented but were merely warmed-over Fortran pudding.

As for agile, the typical words I hear are, "We're doing agile, but we've left out X and Y and Z. They're not really applicable to us." At a recent client, I found. X = user involvement; Y = test before design; and Z = pair programming. I'd say it was agile in the sense that "fragile" contains "agile."

Second, there are management promises that don't pan out. The most common case I see is "quality is number one," followed by "we can skip most of this testing we had scheduled, so we can catch up with the coding." But there are dozens of other cases, all of which management yielding to real or perceived pressures to shortcut processes--breaking configuration management rules, thawing "freezes" to accommodate some big user with last-minute patchy changes (that always fail), not providing promised people or training or hardware or software or facilities.

Enough? It pains me to write about these things, and generally people don't like to read about them, either. But that's us, the software development "professionals."

Q: You seem pretty relentless about quality, why is quality so important to you?

A: Because if you don't care about quality, everything else is trivial. (I call this 'The First Law of Software Engineering.') You need to ship on a certain date? If you don't care about quality, you just ship. You need to cut costs? If you don't care about quality, you just stop when you run out of money. You need to boost morale? If you don't care about quality, you do whatever your people want. (Oh, wait a minute. What if they want quality, even if you don't care? You want to destroy a professional software organization. Act as if you don't care about quality. The professionals will leave, either physically or psychologically.)

So, that's why quality is so important, not just to me, but to our industry. And, until we start caring, we're not going to get better. And I know we can get better, because when I've worked with clients whose entire business (or people's lives) depends on quality, they produce quality software. Curiously, it turns out that quality software is cheaper in the end, but if you're not into long-term thinking, you won't see that.

Q: I know there are people out there (in software projects that don't deliver) who want to bring change into their projects, but no one seems to listen to them or even care. Rather than resign, comply or quit; where can they look for light?

A: I keep a stature of St. Jude for these situations. I recommend that the people pray to this venerable personage, the patron saint of hopeless causes. That's if they want to stay (for some unknown reason). If they're more flexible, I recommend they leave. That's the strongest message they can give to management. If they just need the money and don't care about their work, they can comply as long as they keep drawing their pay. This is usually bad for their health, their family life, and their professional self-respect.

But your question has ruled out all three of the above recommendations, so is there anything else. Yes, there is. Whatever the situation, you can LEARN. Put yourself in learning mode at all times. Improve your technical skills by practicing and mastering new techniques and tools. Improve your listening skills by picking up subtle clues as to what disaster looms next. Practice your speaking and writing skills (outside of work, if you get punished for trying to communicate the way things are and what might improve them). Learn to say NO to meaningless or destructive assignments. All these things will serve you well in the future, and can be justified in your own mind because they're the only things that have the slightest chance of improving the situation.

You might also read Kipling's poem, 'IF', and learn to do those things he suggests, that will be a good use of your time, and on a death-march project, you'll have plenty of opportunities to practice what Kipling suggests, such as:

*If you can keep your head when all about you
Are losing theirs and blaming it on you,
If you can trust yourself when all men doubt you
But make allowance for their doubting too,
If you can wait and not be tired by waiting,
Or being lied about, don't deal in lies,
Or being hated, don't give way to hating,
And yet don't look too good, nor talk too wise*

Read the rest of the stanzas, too. And above all, learn to recognize these situations early so that next time, you can stay out of them in the first place. And, save your money so you won't have any financial excuse for not skipping out when you do find a better situation.

Q: You said "novels", was that a freudian slip?

A: I have 8 more novels circulating among publishers and agents. It's quite a bit harder to get fiction published than it's been for non-fiction, and I'm still a novice fiction writer. My most recent non-fiction book (on testing) was accepted by the publisher in less than 30 minutes. Of course, if I were J.K. Rowling, I'm sure I could beat that time with a novel.

Q: If you're the J.K Rowling of software development, who's Harry P then?

A: Well, first of all, I'm not a billionaire, so it's probably not correct to say I'm the J.K. Rowling of software development. But if I were, I suspect my Harry Potter would be a test manager, expected to do magic but discounted by software developers because "he's only a tester." As for Voldemort, I think he's any project manager who can't say "no" or hear what Harry is telling him. ■ ■ ■



Magnus Ljadas is one of the owners of Citerus, and a software developer with ten years experience.

More from Gerald M. Weinberg

Has written more than 30 books

Hosts the AYE conference - <http://www.ayeconference.com/>

Runs the popular PSL workshop - <http://www.geraldweinberg.com/Workshopstuff/PSL.html>