What Have 20 Years Accomplished and What Is Left to Accomplish in the Next 20 Years?©

Gerald M. Weinberg
Weinberg & Weinberg

Though there have been many improvements in software development in the last 20 years, our aspirations may have grown faster than our abilities. Despite improvements in hardware, tools, and processes, we still lag in our ability to spread the best management practices to more than a small fraction of organizations.

I was asked by CrossTalk to help celebrate its 20 years of existence by writing an article on what has changed in the software business in the last 20 years. I would prefer to start with something that has not changed – people still mismanage software development at about the same rate they did 20 years ago. This figure may not be as pessimistic as it seems, because although we have made some improvements in the way software is developed, those improvements have encouraged people to undertake much more ambitious projects. We want systems that are faster (because we have faster chips), larger (because we have bigger storage devices), more connected (because we have a huge Internet), and easier to use (because we have seen quite a few easier-to-use systems throughout the past two decades).

Sadly, the number of programming languages and dialects of those languages continues to grow, though any improvements due to language changes come, at best, in small increments and people still write computer programs pretty much by hand. Whatever large quantities of juice that could be squeezed out of language improvements has dried up long ago, but computer science curricula still features compiler building as a major subject – to the exclusion of other subjects that might produce larger gains in productivity.

None of this affects me much because I no longer write code for a living. However, one of my sons writes code for a living. So, a generation has passed, but we are still building systems in much the same way we did a generation earlier (and a generation before that).

It is possible that neither of my grandsons will write code for a living because better electronic communication paths have enabled outsourcing jobs to lower wage areas around the world. My granddaughters may not either, and not just because of outsourcing. A generation ago, the percentage of women in the software development field was increasing, but now that number seems to be decreasing. (This is a personal observation based on the percentage of women participating in my workshops and conferences. For example, at a recent Amplifying Your Experience [AYE] conference, 15 out of 60 participants [25 percent] were women. That number would have been 40 percent 20 years ago. Similarly, my most recent two Problem Solving Leadership workshops each had eight women out of 24 participants, whereas 20 years ago, these workshops averaged 50 percent women.)

> *"It is a puzzle why so few software people use the information that is available to them; fewer, it seems, than 20 years ago ... one thing that has definitely changed is the instant availability of technical information on almost any topic."*

Why would the percentage of women be decreasing? Usually when wages in a field fall (as relative wages are now doing in software development), the percentage of females increases. Perhaps women have more opportunities elsewhere and are less tolerant than men of toxic work environments, as many software development environments can only be considered poisonous.

Management of software projects has improved in many places. If you want to be an optimist, you could say that half the large projects succeed to some extent. If you want to be a pessimist, you could say that half fail entirely. That is not much different from a generation ago. In absolute terms, the field is bigger, so we do have a larger number of successes, but the average manager's skill has not measurably improved in two decades.

Individual managers, however, have certainly improved. The good news is that we now have many more excellent managers who can mentor new managers. The bad news is that because the field has grown so much, there simply are not enough experienced managers to go around. We still see people abused or poorly used by their managers. All of this in spite of the fact that we have much more information about how to work with people in software development – information that, sadly, is frequently ignored. Too many managers have failed to learn that the world will not end due to a late delivery or a defect in production. Nor have they learned to relax, breathe, learn from their mistakes, and try again.

It is a puzzle why so few software people use the information that is available to them; fewer it seems, than 20 years ago. It is a puzzle because one thing that has definitely changed is the instant availability of technical information on almost any topic. Back then, mastering a new technology meant buying expensive books and accessing technical magazines on paper in remote places. Today, you can find the information in an instant (and for free) on the Internet (and sometimes it is actually accurate). So it is easier now to learn and improve, but only if you want to. It seems, though, that too few people want to take the trouble.

Perhaps software people want to improve, but lack guidance. Do we have more guidance than we did 20 years ago? Well, one thing is for certain: We do have more PowerPoint. In fact, PowerPoint started almost exactly 20 years ago under the name of Presenter for the Mac. Microsoft purchased the product shortly thereafter and renamed it PowerPoint. For me and others like me who used great numbers of transparencies and 35mm slides, PowerPoint (and its clones) was a definite change for the better. Another change that has occurred is that these days I am making movies, such as one about dog agility that I did for an Agile conference keynote address, and I have also done several more that are available on YouTube[2]. Whether movies will be an improvement over slides remains to be seen.

But an even better change, for me at least, was getting rid of PowerPoint altogether and concentrating on experiential training. Some of us understand that you cannot learn software development from watching PowerPoint presentations. You have to get your hands on the actual software, which is much easier to do today. Contrast this with 50 years ago when I had to argue for weeks with IBM managers to allow my five-week programming class to run one program, one time, on an IBM 709 computer. That was one time for the entire group of 20 students. Today, I have four machines on my desktop, and each one is more powerful than that IBM 709.

Unfortunately, too many people still believe you can learn from PowerPoint presentations and then demonstrate your competence with software development or software management by passing tests. The misuse of certifications (for both the group and the individual) has grown enormously in 20 years, as executives wish (in vain) for some sort of assurance that their software projects will be successful. The number and size of organizations promising to fulfill this hope has significantly increased, but the hope remains unrealized.

The idea of certification is only one of the management myths that has not changed in a generation. We still have an excess of heroic environments, often leading to death march projects. Many managers still maintain the fallacious perception that quality can be tested into slapdash software. Some still try to develop software without being bothered by potential users of that software. And some still believe that process models are the answer to all of our problems.

These are the things I notice on my pessimistic days. On my optimistic days, I notice more managers realizing that it is the people who make a difference, that they can hire talent, but then must also build the relationships to build a team. Quite a few organizations are now using process models successfully, but as only one of the tools providing information for informed cultural changes to their software community.

Speaking of communities, I see the formation of more virtual communities of self-selected individuals such as open source development. Curiously, open source development is a (welcome) return to the way we practiced software development 50 years ago. Maybe we are not seeing change so much as we are seeing cycles.

"... along with incremental development, timeframes have changed. Everyone seems to be in a great hurry, not stopping to think as often and as deeply as developers did a generation ago. Or maybe this is just my perception as I grow older."

Speaking of cycles, we have learned that change is not easy for anyone, so it is best to proceed slowly and show incremental improvement. But, along with incremental development, timeframes have changed. Everyone seems to be in a great hurry, not stopping to think as often and as deeply as developers did a generation ago. Or maybe this is just my perception as I grow older.

These days, we do not seem to have the luxury of time, especially when we are releasing working software every two or four weeks. Sometimes this pace is good (no more *analysis paralysis*), but often it means producing a steady stream of technical debt due to quick, shallow thinking[1]. Throughout the decades, we certainly have accumulated an enormous technical debt in our software. We have just begun to realize the way technical debt slows down progress in the long run. If we are going to look better on CrossTalk's 40th anniversary, we are going to have to address some of these things that have not changed in the past 20 years.◆

Acknowledgement

I owe many of the ideas in this article to Charles Adams, Laurent Bossavit, Marjie Carmen, Dwayne Phillips, and Dave Smith from their contributions to the AYE conference wiki <www.ayeconference.com/wiki/scribble.cgi>.

Notes

1. See "Climbing Out of Technical Debt" by Johanna Rothman <www.ayeconference.com/climboutoftechnicaldebt>.
2. See <www.youtube.com/watch?v=77xrdj9yh3m> for more information and to see other videos on a variety of other software-related subjects by the author.

About the Author

Gerald (Jerry) M. Weinberg is a principal in the consulting and training firm Weinberg & Weinberg. For more than 50 years, he has worked on transforming software organizations. Weinberg is author or co-author of more than 40 books that cover all phases of the software life cycle, including, "The Psychology of Computer Programming," "An Introduction to General Systems Thinking," "Exploring Requirements," "Rethinking Systems Analysis and Design," "The Handbook of Walkthroughs," "General Principles of System Design," and "The Roundtable on Project Management." His books on leadership include "Becoming a Technical Leader," "The Secrets of Consulting," "The Roundtable on Technical Leadership," and the "Quality Software Management" series. Weinberg is also known for his conferences for software leaders, including the AYE conference.

www.geraldmweinberg.com