All Late Projects Are the Same

Well-known author and consultant Tom DeMarco is a regular contributor to *IEEE Software* whose viewpoints always generate some healthy discussion. His article “All Late Projects Are the Same” in the November/December 2011 issue was no different, resulting in both letters to the editor and comments on the Computing Now website. To keep the conversation going, we include a selection from both letters and Web comments in this issue. If you would like to join the discussion, feel free to email software@computer.org or comment at www.computer.org/software/lateprojects. —Forrest Shull

THE “BOTTOM LINE” of Tom DeMarco’s article “All Late Projects Are the Same” in the November/December 2011 issue of *IEEE Software* is, “What’s really wrong with us software folks is that we’re continually beating ourselves up for something that’s somebody else’s fault.” This conclusion should be a warning to all readers. As long as we blame someone else for our problems, we can’t fix them. If this is the conclusion of the article, we should view it with great suspicion. It offers us an excuse but delays any improvement.

In fact, the title of the article is wrong. As DeMarco says, it takes “a slightly more distanced view” of software to reach the conclusion that all late projects are the same. If you look closer, if you study the code and its documentation, you’ll find that there are many types of software design mistakes that lead to delays, poor-quality software, and complete failures. Until we learn to avoid these mistakes, we’ll continue to search for scapegoats and simple (nontechnical) solutions.

David Lorge Parnas

With all due respect for Tom’s 50 years in the business, I believe this isn’t the only problem for late projects. There are many, but one stands out: changing requirements (or wrong requirements that get changed over time). Or at least this was the case in the old waterfall days. In the agile world, it might no longer be a question of whether it’s late but whether it’s complete. This question will supply plenty of food for Tom’s litigation support work if he still wants to do it. Who knows, he might even get better rates for that.

Wolfgang Strigel

Maybe changing requirements is yet another instance of the “real project” starting too late? I’ve seen cases of requirements gathering totally missing the mark. I was dismissed as “too negative” when I pointed that out. The project ended up being late, over budget, and, most damning, “rejected by customer!”

I may be atypical, but I never believed that the traditional waterfall methodology should be executed in strictly separate phases. It seems that it should always allow for going back to revise antecedent phases. What it did was define a flow or derivation of requirements to specs to design to implementation to test! It allowed (in theory) a “provability” of correct project execution. The process should converge to such a sequence of documented decision steps.

Personally, I find some of the “agile” proponents similar to the “new math” that has been taught since I escaped from high school. There’s no firm ground or reference. It’s easy to keep just “muddling along” until something works well enough, or until someone or everyone just gives up. Clearly, talented people...
can make any methodology work. Methodology alone is no guarantee of success. Everyone has to really try to make it work.

I liked Larry Constantine’s observation, paraphrased: When audited, no project is seen to perfectly follow its chosen methodology, but if you don’t at least try, then you’re totally lost. I think that sums up many situations.

In summary, the main fault is that some or all of us like to lie to ourselves (about schedule, budget, market, and so on). Programmers have to be optimistic, managers should be cynical. However, just multiplying by two isn’t good enough. I like Boehm’s approaches to effort estimation and scaling.

Juhan Leemet

I agree that requirement flux can be a major contributor to project delays. Another important factor is the inability to staff the project with the right skills at the right time. Even if requirements were perfect from day one, which usually isn’t the case, staffing problems can derail the project.

Thomas Burke

This is related to Thomas Burke’s point: I’ve been disappointed that many (most?) organizations don’t seem to keep good project teams together. Before a project is even finished, other project managers are often “poaching” those that they think are the best staff from the previous project. Maybe that’s a problem with “matrix management?” That’s unfortunate, because the next team will have to spend time learning to work together as a group and not bickering. Not all teams work well together.

Juhan Leemet

Tom DeMarco’s article is an oversimplification. One of the biggest problems facing developers in the earlier days was the lack of a formalized development approach. Then along came the classic waterfall model. The major problem with this was that it was like buying a suit sold in one size only—every project had to fit the model. The reality, as we well know, is that the classic waterfall makes some major assumptions:

- The next phase doesn’t commence until the prior phase is completed,
with the project progressing in a strictly linear format, which we all know doesn’t happen.

• The requirements need to be clearly defined and complete before we move to the next phase, which we know doesn’t happen.

• The requirements don’t change throughout the lifetime of the project, but with today’s business dynamics, many do change.

One of the major flaws has always been the lack of visibility for the stakeholders and end users until the user acceptance test phase of the project. Using high- and low-fidelity prototyping addresses this issue to some extent, but not entirely, and invariably we bring change management into play or defer to a new project.

In fact, there are a number of other reasons that also cause problems, including poor estimating leading to over-optimistic scheduling, gold plating, use of silver bullets, and the list goes on. I wrote an article on this topic about two years ago, titled “17 Project Pitfalls and How to Avoid Them” for System iNews based on real-world experience.

Russ Bartlett

[DeMarco’s] conclusion is true only in the loosest sense of “starting late.” At least when dealing with the government, the so-called “need date” is determined long after the project has started, on the basis of a decision by those who have no knowledge of the project nor the resources required. Somewhat like a recent Secretary of Defense, who decided that 100,000 people were enough in Iraq, simply because he could get that many without causing political turmoil.

Durusau

All late projects are the same: they’re all research projects. Tom was close when he said, “Everything we were doing had as its unstated goal to move the hard stuff out of the hardware and into the software.” The part that gets allocated to software is the part we don’t know how to do in hardware, and we almost never get to build the same software twice. Hence, at the start of a software project, not only, do we not know what to build (requirements), but because we’ve never done this particular thing before, we don’t know how to do it.

Richard Morton

DeMarco might be revealing truths and actual problems with the business of developing software, but there’s still plenty of blame for us developers. First among these is accepting a deliverable workload that’s too large and too risky. It’s up to the developers to tell the marketers, “No, your scope is too large and we need to break this down into smaller features delivered sooner.” This could be called agile programming, but that’s really just one mechanism to get results. In the first place, the late project starts with a failure to negotiate a business and technical solution that lowers risk and proves itself to the market incrementally.

David Asher

The big release. Save the best for last. Spend as if there’s no tomorrow. Build as if there’s all the time in the world. Those are the reasons why projects are late. If something of value isn’t delivered in the first month—and value means software in hand, not an IOU like a requirements document—then I agree that the project is late from the start. Not sure that’s what DeMarco meant, though.

Chris Riesbeck

Yes, I’ve seen all of these, but the most common is this scenario:

Exec: I need this feature for the next release in six months.

Development team: It will take a year.
Exec: If you can’t deliver it in six months, I’ll find someone who can.

Development team: Okay, we’ll do it.

Outcome A: The project is late.

Outcome B: The project is delivered on time, but the dev team spends the next year fixing bugs in it and can’t deliver any new features.

Glen Seeds

In reply to Glen Seeds’ comment: unfortunately, that exec then blames the software development team—an easy target—instead of taking responsibility for his own decision.

I and my consultant/contractor buddies have had the experience of working really hard at planning and estimating to come up with solid numbers, only to get told by an exec: “I need it in x months.”

In smaller companies, someone would pipe up that their nephew could do it in y months, based on no analysis whatsoever. The story sounded good so it was accepted. We lost the job. The result was the same or worse: late, over budget, poor implementation, missing features, and so on.

The problem is that there never is a “control” case or project. It’s easy to blame incompetence, or some such, because one can’t prove there isn’t a control. Very few organizations are mature enough to track all their projects, estimates, schedules, and budgets to take responsibility for decisions made.

This should be improving with the Capability Maturity Model (many organizations have claimed Level 5 performance, but are they really?), Six Sigma, and various continuous improvement initiatives. However, it takes a long time to get a handle on process and improvements and actually turn the oil tanker on course.

Juhan Leemet

Yet Another Reason Projects End Late (YARPEL): they start on time, they have enough resources, but they do the wrong things. They don’t realize the requirements will change and need to be maintained (and the changes ripple through). They don’t plan, other than drawing an unresourced Gantt chart. They don’t maintain design coherence; and they have an infrastructure that creates an intellectual undertow making even simple tasks heroic.

Peter Farrell-Vinay

---

Showcase Your Multimedia Content on Computing Now!


If you’re interested, contact us at cga@computer.org. All content will be reviewed for relevance and quality.