CrossTalk would like to thank NAVAIR for sponsoring this issue.

In the beginning, it was easy. Programming was fun. Requirements were clear and concise. Programs could be ripped out with pizza through an all-nighter. We were solo computer programmers. Then the group project appeared on the horizon. Most of us angled to go it alone. We did not need anyone else! We get stuck with two other people and do some rudimentary planning. Things do not go well and in the end, one person ends up doing 90% of the work because they do not trust the others to get anything done. Then we graduate from college and enter the real world where teams are the norm and solo work is almost non-existent. We enter a world for which we are wholly unprepared—the world of teams and the complexity it brings.

As I enter my 20th year as a professional software developer, more and more I see software projects like jigsaw puzzles in which the pieces are being shaped at the same time they are being assembled. If I am honest, software is probably more like a Rube Goldberg machine, but you get the idea. In fact, the slide in the DoD introductory acquisition training covering software project management has a drawing of the contraption from the board game Mousetrap, but I digress. Anyone who has assembled a large jigsaw puzzle knows the drill: look for the four corner pieces, find the edge pieces, start building the edges, look for color blocks, and so forth. The devilish thing about jigsaws is that they have exponentially rising difficulty relative to the number of pieces in the puzzle. Each piece has four sides, so if you have twice as many pieces, you have 16 times as many possible interconnects.

Software is similar to jigsaws in having an exponential number of interconnects but adds many more complications. The interfaces are flexible and ever changing. The functionality of the parts is ill defined. No one can lay all the pieces out on a large table for everyone to see all at once. Unlike jigsaws, and whether engineers (and their project managers) will admit it, engineering is a creative process. It exists in the minds of the creators until it is communicated in some way whether verbal or written. That is where the difficult project work begins.

Agile methods were originally developed to add two long-sought project attributes: short-term releasability and requirements churn flexibility. Agile thrives in an environment with high levels of verbal communication; the daily meetings; the on-hand stakeholders; and the pair programming. In small-scale agile, every member of the team knows what every other member of the team is doing. Cycles and tasks are short, and meetings are held often, so problems do not fester. One of the problems with agile methods are scaling up to large projects. You end up with teams of teams leading to groups of individuals not being on the same page. Programmers will know what their sub-team is working on in detail, but the other sub-teams' work will be more opaque. A common solution is documentation in the form of Interface Control Documents and formalized designs.

While the Agile Manifesto does not rule out internal project documentation, the creation of such does slow a project down and make it seem less agile. You end up with things like requirements sprints and architecture sprints before any usable product can be released to a customer. Should it be any surprise that a change in scale of a product would necessitate a change in the process used to create it? People get wrapped up in the pros and cons of one method versus another. Just because Scrum by-the-book does not fit your group does not mean that the Rational Unified Process is your only other choice.

I've been a Team Software Process (TSP) coach and I've been a ScrumMaster. At the moment, I am a TSP advocate but I'll be the first to admit that TSP is not the best method for every situation. It does not matter what you call your particular process. They all include planning, estimation, tracking, meetings, and, of course, writing software. Everything is tailorable. Dr. Deming's famous Plan-Do-Check-Act works perfectly here. Pick a process, use it, measure the results, and modify accordingly. Not one of the TSP teams I know of, and there are many, use TSP exactly by the book. They have all tailored the process according to their circumstances and metrics.

In the end, we are all just looking to get important work done on time, on budget, and with high quality. And maybe, just maybe, have some fun along the way.

Mark Stockmyer
Senior Computer Scientist
NAVAIR

[illegible]