Do Agile Methods Marginalize Problem Solvers?

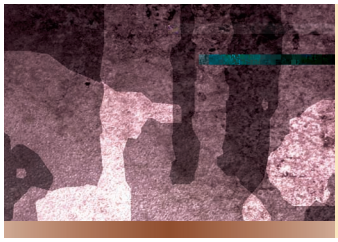
Victor Skowronski, Northrop Grumman

Isaac Newton would have been a failure as an agile programmer. Instead of sharing his work, Newton rarely communicated with other workers and hid his results from them for fear they would steal his work. Instead of collaborating, he quarreled with coworkers, particularly Robert Hooke and Gottfried Leibniz, over credit for his discoveries (James Gleick, *Isaac Newton*, Pantheon, 2003).

Newton's vendetta against Leibniz did not stop at taking credit for inventing calculus—it verged on character assassination. Embodying the antithesis of the people-oriented programming style, Newton would certainly disrupt any agile programming team. Yet his accomplishments should make any programming manager think twice about not using a person of his skills.

Thomas Aquinas also would have failed as an agile programmer. His perpetual silence earned him the nickname, "Dumb Ox." A fellow student assumed that Aquinas refrained from participating in class discussions because he did not understand the material. This well-meaning student offered to tutor Aquinas. Things went smoothly until the student came to a question he didn't know the answer to—and found to his surprise that Aquinas did.

In a programming group, Aquinas would probably be the proverbial programmer who sits silently and walks through his code all day. Like Aquinas's fellow student, the other programmers would assume that a lack of ability led to his nonparticipation in group discussions. Thus, Aquinas would be given unimportant tasks and ignored.



Agile methods could create an environment hostile to the best programmers.

A software development methodology should take advantage of programmers' strengths and avoid their weaknesses. It seems that agile methods would do exactly the opposite for the greatest mathematician of the Enlightenment and the greatest logician of the Middle Ages. This inability to adapt to the working styles of Newton and Aquinas should raise doubts about the efficacy of agile methods.

WHAT ARE AGILE METHODS?

Currently, many different agile methodologies have become popular. However, all advocate the same basic principles:

- *Individuals and interactions over processes and tools.* To implement this principle, the organization moves programmers out of their offices and cubicles into open-floorplan offices. This minimizes privacy so that programmers can see and hear what everyone else is doing.
- *Working software over comprehensive documentation.* The project leadership discourages programmers from writing documentation and encourages them to produce software instead.
- *Customer collaboration over contract negotiation.* The developers demonstrate the prototype software to their customers with the expectation that the customers will provide them with useful feedback.
- *Responding to change over following a plan.* The developers immediately use the customer feedback to guide development of the project's next phase.

Agile methods proponents claim that these principles create the ideal environment for developing software. In the case of Isaac Newton, however, a talented individual with the potential to be the star of the team would most likely fail.

This raises the question of whether agile methods provide the best environment for the best programmers. We define best in this case to mean having the most advanced problem-solving skills. This is not how a proponent of agile methods would define best, however, and that may be part of the problem—as the "Defining a 'Competent' Agile Programmer" sidebar describes. If the designers ignore the need to solve

Continued on page 118

Continued from page 120

Defining a "Competent" Agile Programmer

The paper "Empirical Findings in Agile Methods" (*Proc. Extreme Programming and Agile Methods*; http://fc-md.umd.edu/mikli/Lindvall_agile_universe_eworkshop.pdf) records a panel discussion that defined the characteristics of a good agile programmer as follows: "Participants agreed that a certain percentage of experienced people are needed for a successful Agile project. There was some consensus that 25%-33% of the project personnel must be 'competent and experienced.'" Competent in this context means that the programmer possesses real-world experience in the technology domain, has built similar systems in the past, and has good people and communication skills.

The first two of these attributes really describe experience, not skills. The only skills explicitly mentioned involve the last attribute, which addresses people skills. This is remarkable in that the description does not mention any skill that could be related to problem solving.

This quote also highlights a problem that can occur with agile methods' emphasis on oral communication. Even a cursory analysis of a statement like the one above is difficult, if not impossible, during the ordinary flow of conversation.

problems as part of software development, their methodology will not provide the proper environment for problem solvers.

AGILE PROBLEM SOLVING

Psychologists have concluded that problem solving involves the following four phases:

- *Preparation.* During this phase, the problem solver gathers information about the problem and might carry out experiments and test possible solutions.
- *Incubation.* The problem solver stops actively working on the problem during this phase and lets the brain continue working below the conscious level.
- *Illumination.* This is less a phase than the moment when the solution or the solution's central idea appears as a flash of insight.
- *Verification.* During this phase, the flash of insight expands into a complete solution, which the problem solver tests against reality.

Comparing these problem-solving phases with the principles of agile methodology reveals several inconsistencies.

Preparation

Agile methods do encourage talking to other team members and writing test programs as part of the preparation phase. Eventually, however, a problem solver will exhaust these resources and need to go outside the team for information. This is what Newton did when he invented calculus. He read the works of Copernicus, Galileo, Kepler, and Descartes. These sources gave him information he could not get by talking to his London contemporaries. Research of this sort is essential when the problem's scope exceeds the team's expertise.

Researching outside sources involves reading, which is difficult in an agile environment. The common space increases communication, but it also increases the noise level and makes concentration difficult. Libraries are quiet for a reason.

If our problem solver does manage to overcome the distractions of an open office, peer pressure can become a factor. Agile methods emphasize the production of working code. A problem solver who chooses to do research stops producing code. The problem solver could even prevent others from producing code: Extreme programming puts two programmers on the same workstation and has them work on the same piece of code. One teammate cannot code while the other does research. By doing research online, the problem solver could even tie up the workstation needed for coding.

This lack of code production will cause the rest of the team to resent the problem solver. Agile methods emphasize self-organizing teams, so the other team members are likely to make their impatience known. They will insist that the problem solver start writing code.

Worse, the problem solver cannot counter that he is producing anything that can be shown to a customer. A casual observer will rarely understand the insights the problem solver's research produces. Trying to present his results could even antagonize the customer, who is being told about a problem when he wants to hear about a solution. Management won't like this. They will want the problem solver to work on something that can show immediate results.

Incubation

The incubation phase will also be difficult for problem solvers. After spending so much time researching the problem, the problem solver appears to abandon work on it. This can be confusing to anyone unfamiliar with the problem-solving process. It might also be seen as an opportunity to get the problem solver started doing real work, particularly coding.

Although problem solvers sometimes do other work during the incubation phase, the nature of that work must be such that it lets the problem solvers unconsciously organize the facts their research has uncovered. An agile environment with lots of oral communication and personal interaction can be too distracting for the incubation phase to work properly.

Illumination and verification

The situation might improve in the illumination and verification phases. If

the problem solver can convince the team that the solution really solves the problem and that implementing it immediately is the right thing to do, it might be possible to get help from the rest of the team during these phases. However, if the team is not persuaded, our problem solver must again go through these phases alone, while the team creates peer pressure to abandon the solution and work on something more to its liking.

Some could claim that teams of problem solvers have replaced the individual working in isolation. These teams work in brainstorming sessions in which they present ideas and get immediate feedback from one another. This feedback must be carefully controlled, however, to keep ideas from being killed prematurely.

Moreover, a brainstorming session cannot replace the preparation phase. If team members do not thoroughly understand the problem, they won't be able to recognize an innovative solution when the problem solver presents one to them. At best, the brainstorming session provides a method for jump-starting the illumination phase, rather than a replacement for the entire process.

THINGS VERSUS PEOPLE

Problem solvers tend to be concerned with things, how they work, why they don't work, and how they can work better. Since software engineers must solve problems that are more concerned with things than people, generally a concern for things is an advantage. Even when the problems appear to be more about people, such as a graphical user interface design, the software engineers can best analyze and solve them if they think of these problems as involving *things*—cognitive psychology, for example—rather than people.

Too much contact with other people can cause distractions that interfere with this obsession with things. Isaac Newton invented calculus when the plague closed Cambridge University and forced him to retire to a secluded family farm. Rather than being an impediment, the absence of human companionship freed Newton to think about the problem of motion along curves. This lack of distraction arguably benefited his creative process more than any amount of conversation with his peers would have.

To the extent that problem solvers concern themselves with things, they become less concerned with people—a focus shift that typically translates into poorer people skills. A good problem solver will not spend much time listening to opinion that lacks factual backing, for example, even though this damages the opinion holder's ego. Remember, a problem solver can spend large amounts of apparently unproductive time without making any attempt to explain or justify it to coworkers.

For example, Thomas Aquinas's life as a Dominican monk let him ignore many social distractions. On one occasion he could not, however: His superiors ordered him to accept an invitation to a state dinner hosted by the king of France. Aquinas spent most of his time at the dinner deep in thought, not talking to anyone. For someone with a lesser intellect, such behavior would have been considered extremely rude and would have guaranteed that the person would never receive another invitation to such an event. Fortunately for Aquinas, his reputation encouraged others to overlook his behavior.

A good problem solver who lacks people skills appears to be much less competent in the eyes of agile methods proponents, however. The problem solver isn't as sensitive to the unspoken messages other group members send. These people focus less on problems and pick up on interpersonal messages more readily. Ironically, because they appear more in tune with the group, these individuals appear more effective in an agile environment even though they lack advanced problem-solving skills.

A programming methodology should accommodate the working style of an organization's best programmers—and agile methods may not. Agile methods could push people who excel at problem solving into the background, giving center stage to programming team members who can talk well but might lack the analytical skills necessary to do the difficult design and coding tasks.

The qualities and talents that make someone a good problem solver may not be as widespread as we would like, however. With the software industry's expansion over the past few decades, many programmers may now have better people skills than analytical skills. Agile methods might let these people work in a way they find comfortable. If the application already has a set of known solutions, applying them could be all that is necessary.

Programmers with poorer analytical skills could be better suited to this work than highly skilled problem solvers who might find such work boring. An agile method could also be the best choice for such a team because it lets them use their superior communication skills when they run into difficulties.

If an application has one or more unsolved problem areas, however, agile methods could be inappropriate. This situation requires good problem solvers, and people skills will be less relevant. The method that lets the problem solvers work in the way they find most comfortable is the most appropriate. ■

Victor Skowronski is a senior engineer with Northrop Grumman Information Technology. Contact him at victor.skowronski@ngc.com.

Editor: Neville Holmes, School of Computing, University of Tasmania; neville.holmes@utas.edu.au. Links to further material are at www.comp.utas.edu.au/users/nholmes/prfsn.