CUTTER CONSORTIUM
Agile Software Development & Project Management Advisory Service
E-Mail Advisor

19 January 2006

# AGILE METHODS, SYSTEMS THEORY, AND "MIND-BLENDING"

by Michael Mah, Senior Consultant, Cutter Consortium

In a recent Cutter Business Trends *E-Mail Advisor* entitled "Complexity's Rising Tide" (1 December 2005), I talked about modern systems and software development being primarily about the "blending of minds" among IT professionals. I consider this idea at the heart of successful innovation in a tech-driven world, where the paramount challenge is to successfully build more complex systems under increasingly tight deadlines. Many are looking to agile methods as a way to accelerate their cycle time given the pressures that they're facing.

To me, mind-blending is the essence of knowledge work, which many technologists consider software development to be. However, if you take this idea further and want to understand why agile methods can reduce defects and thereby shorten schedules, then it helps to appreciate concepts like cybernetics and systems theory. Systems theory (a modern synonym for cybernetics) can be seen at many intersections within the agile development philosophy.

What is cybernetics? The concept was first introduced by an American scientist named Norbert Wiener. In 1906 at the age of 11, Wiener entered Tufts University to study mathematics. He then pursued philosophy and mathematics at Harvard and Cornell before heading to Cambridge, England, to study under Bertrand Russell, the influential British mathematician. Wiener ultimately took a teaching post at the Massachusetts Institute of Technology, where he was known for his poor lecture style, absent-mindedness, hypersensitivity to criticism, and fits of depression.

In places like the Principia Cybernetica Web (http://pespmc1.vub.ac.be/DEFAULT.html), you will find cybernetics described as:

> The study of the communication and control of regulatory feedback, both in living beings and machines, and in combinations of the two. It emphasizes the interactions and connectedness of the different components and in practice, focuses on complex, adaptive, self-regulating systems.

Principia Cybernetica goes on to say that,

> The systems approach distinguishes itself from the more traditional analytic approach by emphasizing the interactions and connectedness of the different components of a system. From Cybernetics also comes Shannon's information theory, which was designed to optimize the transmission of information through communication channels, and the feedback concept used in engineering control systems.

Agile methods capture the essence of the dynamic interaction and interconnectedness among knowledge workers when you consider practices like Test-Driven Development (TDD), pair programming, and colocated teams. It recognizes the mind-blending that occurs through these iterative processes. That's why they work! The transmission of information -- think user stories, story cards, software design and code iterations, validation, and refactoring -- are all about optimizing the information transmission between people, solving problems together. The discipline inside agile practices are about "feedback control" between members of a team.

My appreciation for Shannon's information theory and its application to software trace back to reading Larry Putnam's early research on software lifecycle behavior in the early 1980s. At the time, I was working on software for the Trident II nuclear submarine and ironically, implementing an early form of Test-Driven Development (TDD) known as Integrated Test Program Planning (ITPP). I only realized that we were doing TDD and calling it ITPP after reading Jim Highsmith's descriptions years later [1].

TDD and ITPP were all about reducing defects and thereby shortening cycle time. To do this, we had to understand systems theory, causality, and software lifecycle "laws of nature," since we were scaling-up with large teams under a tight deadline, on a system that was one of the most complex engineering undertakings I had ever seen. It's a mammoth engineering feat to steer a ballistic missile submarine the length of two football fields underwater for long periods of time without hitting an undersea mountain, while carrying a stockpile of nuclear warheads undetected. Predicting defects that would be discovered during integration testing required us to run forecasts using computer models like SLIMtm, and that's where Putnam's research came into play. Serious defects left undiscovered in a nuclear submarine navigation system is really bad. (We predicted the number of defects to greater than 90% accuracy, conducted tests successfully, and made the deadline.)

As an engineer who studied mathematics, electromagnetic physics, and feedback control theory at Tufts (while trying not to get into trouble as part of a fraternity), I understood the real-world behavior of software expressed by Putnam's math. His early work, found in relatively obscure journals of the IEEE, blew apart the reductionist view of software development as factory work. He uncovered a systems behavior that showed the interrelationships between time pressure, effort/cost expenditure, and software reliability -- all as a function of team productivity, project size, and complexity. Putnam published a unified systems theory of software development as knowledge work in those early papers, and in four subsequent books.

## So What Does This All Mean?

One thing that we discover is an understanding of why agile methods often result in higher quality. A reductionist view uses terms like "software factory," where code is reduced to piece parts that come off a manufacturing assembly line a la Frederick Taylor (think of lines of code per day). Agile proponents reject this view and appreciate software development as research and invention.

Another thing we appreciate is why colocated teams and paired programming works. The factory mindset asks, "Why use twice the people on a task and thereby double the cost?" The cybernetic or systems view realizes that the instant communication and direct feedback between paired programmers and colocated teams eliminates design mistakes at the source and actually lowers the cost. Mind-blending is all about getting the information for a feature or story card right, getting the way to validate that in a test right, and getting the code right -- all in a very fast iteration cycle. The client as an active participant gets to see firsthand when things work, much earlier instead of later (which might be too close to a looming deadline).

Mind-melding -- as they say on the planet Vulcan -- is where it's at. Mister Spock would be proud.

## Note

[1] In a strange twist, in 1981-1986 I was working on a Naval surface ship called the USNS Vanguard, which was converted to test submarine navigation systems, after having once served as a tracking vessel for Apollo space launches. I later discovered that Jim Highsmith worked on the Vanguard during the Apollo program. (Now you know how long we've been around.)