Obstacles to Enterprise Agility

Michael James, CollabNet Certified Scrum Trainer

March 15, 2010

I often work with divisions of Fortune 500 companies that are struggling to become agile, starting with Scrum. While each organization is in a distinct business sector using different technology and management cultures, each one shares a common pathology, a kind of "giantism." This article lists common obstacles to agility in large organizations and explores the possibility that the symptoms of giantism are entirely avoidable.

At first glance, an organization's challenges will appear to be "too much to do" or "not enough resources" or "changing business climate." Upon closer inspection, the root causes will turn out to be bad habits, unexamined reflexes and misconceptions.

A division of a well-known company cited as a 1997 success story by a famous Scrum pioneer came to Danube Technologies, Inc. for help in 2009 because market forces revealed it was less agile than its competitors. The Scrum initiative that started in 1997 apparently couldn't withstand a decade of obstacles to large-scale agility. Sadly, most attempts to adopt Scrum in large organizations do not result in durable, ongoing transformation. Obstacles to Scrum adoption are usually obstacles to business success in general, and established organizations are usually reluctant to let go of them.

Obstacle #1: Naive Resource Management
The PMBOK Guide observes "often the budget needs to be increased to add additional resources to complete the same amount of work in less time." More specific to software, Fred Brooks (in *The Mythical Man-Month*) makes an apparently contradictory claim: "Adding manpower to a late software project makes it later." To resolve this paradox, let's examine the definition of "resource."

When the work is new product development, the relevant resources are intangible: task absorption, learning, interpersonal communication and innovation. Scrum teams attempt to maximize these by creating states of individual and group "flow." According to psychologist Mihaly Csikszentmihalyi, "[In flow] your whole being is involved, and you are using your skills to the utmost."

Scrum development team members collaborate intensively to build products according to goals they repeatedly negotiate with the product owner, who is responsible for making the team's business decisions. Results are demonstrated at the end of every fixed length sprint (e.g. every two weeks). During sprint execution, team members develop intrinsic interest in shared goals and learn to manage each other to achieve them. Even with ideal circumstances (including a team room) it takes a team a few sprints of stable membership to hit its stride, and a year or so to reach its potential. A popular description of this organic process is Bruce Tuckman's "forming, storming, norming, performing" model.

It is naive to think of human beings as resources. Adding people to a team will not reliably increase the intangible resources--and may detract from them. After a year of doing Scrum, one of my clients reported "Once a team is formed, we would rather lose a team member than add one!" In another case, when the Scrum team itself made the hiring decision, adding a new member went well. Even when giving the team hiring autonomy, it's inadvisable to grow it much larger than seven people.

In some circumstances, adding teams may result in more progress, if we're mindful of the intangible resources.

Obstacle #2: Teams Organized by Functional Specialization
A Scrum Team is a cross-functional group that attempts to build a properly tested product increment every sprint, gradually adding feature scope. The most popular book about Scrum uses the phrase "potentially shippable product increment" 18 times. Despite this, I meet people who claim to be "doing Scrum" while executing "analysis sprints" or "design sprints" at the beginning, deferring integration and testing to the end, and holding different teams responsible for each phase! These waterfall habits hide risk until it's too late to respond.

In Scrum, every Sprint requires a mix of development activities. We gain agility through continuous requirements analysis, continuous design, continuous integration and continuous testing, all feeding into each other.

Obstacle #3: Teams Organized by Architectural Components
Single-component teams reduce the business's ability to re-prioritize product capabilities, increase the coordination bottlenecks through managers and product owners, and introduce integration risks.

The component team approach would be efficient if new product development were as predictable as manufacturing. In practice, priorities change and estimates prove incorrect, thus it's difficult to get proper focus from the right people at the right time to build capabilities affecting multiple components.

The opposite of the component team is the feature team. A feature team spans both components and disciplines, and is thus capable of building business value features in thin, fully-tested vertical slices. The feature team approach replaces last century's efficiency thinking with a concept of team autonomy, ownership and responsibility. The product backlogs of feature teams are driven by business value, not technology dependency. If you are still living with Gantt charts, you probably don't have feature teams yet.

Creating feature teams comes at a cost: developers must learn new skills, which will slow them down initially. Fortunately, most developers enjoy learning new skills. Techniques such as appointing a technical "component guardian" for each area can help protect architectural integrity as teams are learning. As with any scaled development, continuous integration (automated tests that run much more frequently than once per day) is crucial to avoid undetected regression failures.

Once an organization has mastered feature teams, a next step could be general purpose feature teams with reduced affinity to feature areas. While such a step may be years away, this century will belong to the learning organization.

Obstacle #4: Distraction
A typical large organization wastes millions of dollars in unnecessary task switching. Teams lack focus and the months of continuous membership stability to reach the self-managing state needed for high performance and quality. Some people perpetually find themselves on multiple critical paths at the same time, severely constraining total productivity. For effective scaling, these people must become mentors rather than task executors. To increase their influence, they must relinquish some control.

Traditional management practices that reinforce specialization will exacerbate the issue. Work should be offered to a Scrum Team through the Sprint Planning Meeting. When a manager bypasses this by assigning work to an individual, there's less possibility of mentoring others in the critical skills. In a Sprint Planning Meeting I observed recently, the first question was, "Who do we have on this Sprint?" A group of people being yanked around individually isn't going to engage in the collaborative learning expected of a Scrum team. Organizations dig these ruts deeper each day.

In the higher ranks, the problems of multitasking, distraction, anxiety and inability to gain influence by letting go of control are even worse. The product owner at the same meeting rushed in nearly an hour late with a harried look on her face, then rushed out again after about 10 minutes; I was told this was routine.

That afternoon, an executive was unable to focus on a meeting with 60 of his employees because he was also trying to manage another situation via his BlackBerry. A casual observer of these scenes wouldn't guess these frantic event-driven slaves were the ones responsible for long range strategy.

Effective lateral communication between Scrum teams helps product owners calmly focus on responsibilities such as prioritizing work and being the final arbiter of requirements questions.

Obstacle #5: Reluctance to Continuously Refine, Reprioritize and Rescope
In product development, new scope discovery commonly outpaces velocity. To manage this reality, product owners should conduct a product backlog refinement meeting with the team every sprint. As large product backlog items (or "epics") emerge, find the high-value aspects of them and split those into separate smaller items (typically "user stories"). The product owner controls scope by deciding which items will fit in the release projection, given historic trends of velocity and scope discovery.

Obstacle #6: Rampant Technical Debt
An organization's management issues are ultimately visible in the source code. This "technical debt" causes production problems and high cost of change. Ideally, regression testing is automated using the same programming language our production code rather than proprietary tools that reinforce specialization. Skills such as Test Driven Development (TDD) proved their value in the 20th century. In the 21st century, agile engineering skills will be expected of any developer. Teams that don't have these skills initially must attempt thinner vertical slices of work until they've learned them.

Obstacle #7: Lack of Commitment to Transformation
Scrum allocates one person per team (the ScrumMaster) to devote full-time attention to exposing and overcoming obstacles such as these. This requires courage, imagination and support from management. Too few ScrumMasters devote proper attention to this, and management often fails to support those who do.

"Too much to do" or "not enough resources" or "changing business climate" aren't good excuses for avoiding the agile practices specifically intended for those circumstances. Change agents will be better equipped to overcome obstacles to agility by realizing the root causes are often bad habits, unexamined reflexes and misconceptions.

Michael James is a software process mentor and Certified Scrum Trainer at CollabNet with a focus on the engineering practices that enable agile project management practices. Having worked as a software developer (formerly "architect") for more than 20 years, he has deep experience with automated testing that predates the eXtreme Programming movement, including formal, phased, high-ceremony processes based on DOD-STD-2167A; chaotic non-processes of the dot-com era; and agile processes such as Scrum and XP. As an agile coach, James has found that teams learn best when he can work alongside them "in the trenches." This one-on-one interaction allows him to share agile practices and insight while tackling real organizational challenges.