Scaling Agile: Best Practices for Large, Multi-Site Projects

Common-sense practices don’t apply to multi-site Agile projects. The Agile methodology is based on a need for efficiency in small-team settings, so is it possible to apply it to broader efforts?
It’s All in the Name

In some ways Agile software development is a victim of its own success.

With a focus on defining requirements as you go, testing code as it’s written and delivering small chunks of working software, Agile has proved so advantageous that many software organizations seek to expand its use beyond single team projects.

But how do you go large with a software methodology that derives its strength from keeping things small? Agile was created to benefit small teams, with members sitting together to solve development challenges quickly and creatively. How can companies take this small group focus and translate it to larger projects? And should they? This three-part guide offers practical answers on the weighty subject.

In the first article, Agile experts weigh in on the challenges of managing projects across multiple teams and time zones. They offer advice on how to divide workloads, deal with cross-team dependencies and retain momentum to stay Agile.

Next, we look at why—even though it may seem promising at first glance—moving members from a team that is thriving to one that is not is a bad idea.

And finally, we tackle one of the toughest issues an Agile team can face: how to make sure the software delivered does what the business needs it to do.

If you don’t get that right, it doesn’t matter how Agile you are.

Jennifer Lent
Site Editor, SearchSoftwareQuality.com
Managing Large-Scale Agile Projects? Think Small

**With short development** cycles, continuous software delivery and shared roles and responsibilities among team members, Agile software development is all about efficiency. In fact, Agile originated as a way to eliminate the widely chronicled inefficiencies of the Waterfall methodology: Too often, software was delivered late and did not meet the needs of those it was designed for.

Unlike Waterfall, where testing did not begin until coding was complete, Agile does a good job of keeping the software development process moving. Practices like testing software at each iteration and daily meetings help projects to make steady progress—provided the project involves only one team.

But what happens when Agile development efforts expand to include multiple teams, working at different, often geographically distant, locations? How can team members stay true to Agile principles when they aren’t able to even sit down in the same room?

“Agile projects run into trouble when they scale [beyond one team],” said Scott Ambler, founder of Toronto-based Agile consultancy Scott Ambler and Associates. In taking Agile to a larger format, there is the risk of creating management overhead issues that Agile was designed to eliminate in the first place. Therein lies the challenge: How do software organizations manage multi-team Agile projects without losing their agility?

According to Agile experts, it’s impossible to completely eliminate that risk. After all, multi-team Agile projects by definition require more management than a single-team undertaking. While some duplication of effort is unavoidable, there are techniques that help multiple teams working together on a single project maintain the efficiencies of Agile software development.

How teams are structured, how work is
divided, how dependencies among teams are managed can make the difference between success and failure, said Johanna Rothman, founder of Rothman Consulting Group Inc., in Arlington, Mass. The key, Rothman explained, is “thinking small to go large.”

**THE SIZE OF SUCCESS**
Moving to multi-team Agile is no small feat. It’s important not to take that step until you have first mastered the Agile development process with a single team, said James Shore, an Agile consultant. This sounds obvious, but Shore has seen many organizations attempt large-scale Agile projects without single-team Agile experience. It’s a blueprint for failure, he explained. “Make incremental jumps. If you don’t, you will quickly run into problems with communication, problems with code quality.”

Single-team Agile works well for teams with fewer than 10 people. Once you reach 10 or 12 members, a second team is the only effective way to grow, Shore said. He noted that his colleague Diana Larsen, a partner at Agile consultancy FutureWorks Inc., in Portland, Ore., believes the optimum team size is “seven plus or minus two.” Once a team grows too big, it starts breaking down into sub-teams, Shore said. “Even if, on paper, members are part of the same team, people form cliques and stop working as a single unit. Whether or not you admit it, you end up with multiple, interdependent teams.”

**THE RIGHT SKILL SET**
Rothman believes keeping teams small is key to maintaining agility when working on multi-team projects. She recommends five to seven members. Some clients she works with say that’s not enough people to cover the range of skills Agile development projects require. “They say ‘we don’t have enough testers, DBAs, user experience experts or whatever.’” Rather than increase team size, Rothman recommends that the team decide how best to address the need for the skills that team members lack. Ambler agreed. The idea that the team itself has all the skills needed to get the job done sounds great from an enterprise point of view, he said. “But it’s dangerous. Teams need to
take advantage of enterprise resources, working with architecture people, reuse people, database people.”

**DIVIDE AND CONQUER**

With teams in place, the next step is to delegate responsibilities. One tempting approach is establishing a team to build the architecture—for example, a banking platform—then assigning additional teams to customize that architecture for the various financial services. This sounds like a sensible way to divvy up the work, but it’s not, Shore said. “It always results in bottlenecks. The services team can’t get started until the architecture is fully established.” Another drawback? “All the smart developers end up on the platform team.”

Rothman explained that the right way to organize the work is around software features—behaviors of the system that carry out a specific user need, such as placing an order. She recommends embedding architects in the individual teams and allowing the architecture to evolve as the project progresses. Of course, it’s important to begin with a general idea of the architecture shared among the teams. “But don’t be wedded to it,” she cautioned. “As you finish features, you update the picture of the architecture.” As the architecture becomes solid, the teams understand what works and what doesn’t.

**KEEP IT SHORT**

Another mistake practitioners of large-scale Agile projects make is setting up iterations that are too long, Rothman said. Iterations—a single development cycle within a larger project—should not exceed two weeks. “Shorter iterations force teams to integrate [new code with existing code] more often, learn more quickly and obtain feedback more often,” she said. Short iterations also allow

“As you finish features, you update the picture of the architecture.”

—JOHANNA ROTHMAN, founder, Rothman Consulting Group Inc.
teams to throw away small increments of work that fail. “That is the beauty of Agile,” she explained. You deliver software in small increments, solicit feedback and if it doesn’t work, you can go back to the drawing board. Throwing away two weeks of work doesn’t spell disaster for the project. But any more than that might, she said.

DEPENDENCIES
It’s impossible to eliminate dependencies in large-scale Agile projects. Minimizing them reduces wasted time and prevents projects from running off the rails. “The more dependencies between teams, the more hand-offs. The more hand-offs, the greater the opportunity for delay and error,” Shore said, noting that delays and errors are why large-scale Agile systems are so much more wasteful than individual Agile teams.

Dependencies that can’t be eliminated must be managed. How do you do that? Shore’s preferred approach is to rely on a concept known as bounded contexts, a term coined by Eric Evans, in his book Domain-Driven Design.

Here’s how Shore explains the idea in his blog The Art of Agile: “A bounded context is a collection of code, database schema, and other artifacts that are managed as a single unit. Within the bounded context, changes to one part of the system are allowed to affect any other part of the system. Any part of the bounded context may be refactored, enhanced, or changed at any time, and everybody working within a bounded context is expected to be aware of what everyone else in that context is doing.”

By limiting the scope of dependencies, bounded contexts allow team members to deal with them without running into problems that could cause chaos for the entire project.

It’s impossible to eliminate dependencies in large-scale Agile projects. Minimizing them reduces wasted time and prevents projects from running off the rails.
TAKE A STAND
Best efforts aside, Rothman cautions that large-scale Agile projects can still run into communication problems with managing dependencies, and they can slow all teams involved to a crawl. “When you wait for others to fix the problem, a state of inertia can set in.” Most team members don’t have the resources to figure out what ails the entire multi-team project, but a single team member on one of many teams can take steps to get things moving. “Ask yourself ‘What can I deliver today? What can I do to help my team deliver today?’”

Taking action, no matter how small, and letting others follow in your footsteps builds momentum, and momentum only is the way to overcome inertia. “It’s a way of staying Agile, even though you are one contributor among a dozen teams and hundreds of developers,” Rothman said. —Jennifer Lent
In Agile, Shaking Up Teams Doesn’t Work

Across the organization, the Agile process is running smoothly. With multiple teams in place, Agile is well on its way to being fully accepted, understood and utilized. The majority of the teams are productive. They work well together, and development is progressing at a solid, sustainable pace.

The teams are busy, but some are less so than others. Why not move team members around? This resource-switching approach could support the Agile process by allowing project managers to plug team members in and pull them out as needed.

With that reasoning, shuffling members among different Agile teams sounds like a good idea, doesn’t it? But don’t do it.

Forming, Storming, Norming, Performing
It might not be evident at first, but on any established team, members have already learned how to work together to support the Agile process. Disrupting a stable team doesn’t improve software quality or boost team productivity. In fact, it can have the opposite effect. Resource switching doesn’t work because the team has already gone through the “forming” process.

The process of forming a productive team isn’t easy, and it requires a significant commitment of personal energy, time and concentration from each member. When teams first come together, they go through the stages of forming, storming, norming and performing. These are Agile terms for getting to know one another and mapping out strategies that turn strangers into members who grow into stable, productive and high-performing teams over time.

When project managers switch out team members, that process has to start all over again. When a stable team is disrupted, you force them to regroup and re-form. When a
team stabilizes, it’s a sign that members have learned how to optimize their skills within working relationships. In the most successful Agile teams, members develop a bond that allows them to be productive and interact on a personal, genuine level.

**BREAKING THE ICE**
Another advantage of stable teams is they have learned how to optimize their processes. By processes, I mean Agile methods and technical processes, such as unit testing, giving demos to testers, code reviews and test reviews, to mention a few. Continuous improvement processes tend to bind members together so they work efficiently. Each time you move members in or out of a team, you force the team to redefine or rework processes. If you think about it as a manager, the members of each team spend a great deal of time and effort getting to know on another other—understanding how each person responds to varying viewpoints, egos and differences of opinion until they hash out an effective professional working relationship. Once that balance is reached, it’s best to maintain the balance and not disrupt the formative, personal work that’s already been accomplished. The team’s energy needs to be spent in developing and testing, not in re-establishing team connections.

**NO POWER IN NUMBERS**
Stable teams have higher productivity and established velocity. They already know how each member works and what they can expect from one another. They’ve established a working order that results in a set velocity that results in more accurate planning. That enables project managers to reliably predict the outcome of a team’s work. But if the team is disrupted—and forced to incorporate new members—velocity and productivity will be in flux until new members find their places in the established team flow.

How do stable teams affect software quality? It’s easy to fall into the trap of thinking that adding more testers will boost quality. It’s easy to believe that the more testers you have, the more testing that occurs, the greater the quality and lower the chances that defects will
make their way into production.

In Agile, the whole team bears responsibility for testing, not just the testers. And in my experience, stable and productive teams devote more time to quality. They keep the customer experience in mind and focus on reducing customer issues and defects. Software professionals don’t want to hear that customers have run into a problem. And on an established team, it’s even more personal: The team, not the testers, missed an issue.

**Software professionals don’t want to hear that customers have run into a problem. And on an established team, it’s even more personal: The team, not the testers, missed an issue.**

**STABILITY BREEDS SATISFACTION**

Stable teams allow members to experience a higher degree of personal satisfaction. “What?” you might be asking. “Personal satisfaction? What has that got to do with anything?” Here’s what: In small groups, most people are able to shine and showcase the depth of their abilities. Stable Agile teams produce star members who might not have been noticed otherwise. As a project manager, it’s vital to provide a consistent, stable work environment to improve employee satisfaction and retention.

Yes, it’s difficult to resist the temptation to switch out team members to fill gaps and holes in other projects. Yes, even stable teams are likely to change at some point. However, to provide the maximum positive impact on quality, productivity and employee satisfaction, it’s critical that stable teams be kept intact. “Plug and play” simply doesn’t work with real, working people. —Amy Reichert
Collaboration Is Crucial for Business-Ready Software

**Many teams are** good at delivering working software throughout the Agile process, but not so good at building software that meets business needs. Often, tight deadlines spur teams to start coding before fully identifying business and user requirements, according to application requirements experts Ellen Gottesdiener and Mary Gorman. They laid out the value of techniques and tools for discovering those needs in the Agile process.

Gottesdiener and Gorman are principals of EBG Consulting, based in Sudbury, Mass., and authors of the how-to book *Discover to Deliver*.

“In the attempt to go lightweight, many Agile teams end up going no-weight, skipping analysis,” Gottesdiener said. “They pay a big price for that. Once they get into the work, into delivery—whether it’s an iteration or release—they realize how many business needs and questions they can’t answer.”

**The road to nowhere**
The lightweight requirements analysis approach usually leads to “traveling stories,” in which developers discover new user needs during development. Because they can’t get those requirements done in an iteration or release, they have to extend the Agile process.

Traveling stories can also happen and weigh down development if teams spend an inordinate amount of time planning and still not understanding the product needs they have to deliver. “They end up often having huge backlogs or bulging backlogs instead of lean backlogs,” Gottesdiener said.

**From bad to worse**
Insufficient analysis also leads to poor estimates. *Estimating doesn’t go away in the Agile development process*, but it’s common for Agile teams’ estimates to be way off,
Gottesdiener said. They get into an iteration, into the building and they find that they forgot an interface, or they come across detailed business rules they haven’t explored. The team hasn’t collaborated enough on refining the backlog to figure out how wide and deep they need to build individual stories. “So they can’t get any traction and understanding of how much work they can get done or meet service-level agreements,” she explained.

Within the requirements themselves, the project team has to build certain aspects of the product to enable other aspects to work. That’s particularly true around the data.

According to Gorman, not anticipating dependencies is another project problem. Within the requirements themselves, the project team has to build certain aspects of the product to enable other aspects to work. That’s particularly true around the data; developers must create data and make it available before they can update it or perform some other action on it.

“Without having a shared understanding of the actual business needs, of the product needs, they really can’t come up with reasonable estimates,” Gorman said.

DONENESS AND GOODNESS
A process the authors call “structured conversations” can bring efficiency to the discovery, shared understanding, estimating and building of applications that meet business and user needs.

In the structured conversation, the development team explores, evaluates and confirms business and technical needs.

“The structured conversation is a meta-framework that you would use at any planning horizon,” Gorman said. It’s an ongoing and iterative process of exploring, evaluating and confirming product needs. It includes scheduled and spontaneous, just-in-time conversations among all team members, including business, user and technology partners.

In the exploration phase, the team considers options for product dimensions, including user
needs, interface, data, control, environment, quality and more. In exploration, the team researches and evaluates such criteria as competitors’ markets and products, existing and potential markets and technology trends. That information is the foundation of assessing the organization’s expectations for the product and the technical teams’ ability to meet those expectations.

The level of granularity in the exploratory process will vary as the process goes on. At first, features will be “big chunky stories,” Gottesdiener said. In previews, Gorman explained, “you’re starting to connect the dots and draw clear boundaries.” Eventually, those stories will become thinly sliced iterations that can be delivered in a few days.

In evaluation, the second part of structured conversations, the Agile project team weighs the software options, dependency risks and feature delivery order, Gorman said. Estimation is a key part of evaluation. This is when systems get measured for size, complexity and other attributes that determine how long it will take to build the product. Considerations here include product lifecycle, user acceptance and, of course, business needs. Note that evaluation of the value of product options is critical to establishing feature delivery order.

The structured conversation ends only when all team members agree that chosen software is working well and meeting business needs.

Checking the software in each phase of delivery is the third step in the structured conversation process. The team must continually evaluate and verify that business needs are being met by the application. The structured conversation ends only when all team members agree that chosen software is working well and meeting business needs.

“In planning, the team has to define its ‘doneness’ and ‘goodness’ criteria,” Gottesdiener said. She worked on a project in which the lead developer was the only person who really knew the code base. A key part of the project’s acceptance criteria—doneness and goodness—for each iteration and release was doing
code walk-throughs with junior developers. “They realized they were not done unless they had paid the learning price, so to speak,” she said. Without that, the future of the feature or product was compromised.

**CONVERSATION STARTER**

According to Gottesdiener and Gorman, one or more skilled facilitators are needed to lead and educate cross-functional development teams doing Agile collaboration in the structured-conversation style. Leaders need product, technology and business knowledge, of course, but Agile methodology, business analysis and facilitation skills are must-haves, too. Leaders should help people set up a framework for collaborating and using analysis tools quickly, so requirements can be explored, planned and delivered in a lightweight but rich manner. Facilitators should “lead conversations in a facilitative way without driving to the answer [and slice] the options for product needs based on values that have been articulated by the whole community,” Gottesdiener said.

The concept of value is the guiding star of requirements discovery and delivery, Gorman explained. From exploration to evaluation to confirmation, always ask, “What is the value to those being served?” —Jan Stafford
JENNIFER LENT is the site editor of SearchSoftwareQuality.com. A veteran technology journalist, she was most recently a contributing writer and senior editor at Software Development Times, where she covered application lifecycle management, Agile, requirements management, application security and software testing.

AMY REICHART is a software tester with 16 years of experience. She has a wide variety of QA experience in companies producing complex software from ERP systems, architectural design, e-commerce and health care system software. Her experience also encompasses Agile, extreme and traditional software development practices. Email her at rabowreich@gmail.com.

JAN STAFFORD is executive editor in TechTarget’s Business Applications and Architecture Media Group. Stafford has covered the computer industry for 20-plus years, writing about everything from personal computers and operating systems to server virtualization and application development. She has worked at several national trade publications, as a contributor to such publications as PC Week Inside and Investor’s Business Daily and as a speaker at industry conferences. Email her at jstafford@techtarget.com.