BEST PRACTICES WHITE PAPER

Succeeding with Geographically Distributed Scrum

Jenny Stuart, Vice President of Consulting, Construx Software

Version 2, March 2018

Contributors

Melvin Perez, Senior Fellow

Alex Sloley, Senior Fellow

When organizations adopt Agile throughout the enterprise, they typically apply it to both large and small projects. The gap is that most Agile methodologies, such as Scrum and XP, are team-level workflow approaches. These approaches can be highly effective at the team level, but they do not address large project architecture, project management, requirements, and product planning needs. Our clients find that succeeding with Scrum on a large, geographically distributed team requires adopting additional practices to ensure the necessary coordination, communication, integration, and architectural work. This white paper discusses common considerations for success with geographically distributed Scrum.

This white paper was downloaded from www.construx.com/whitepapers

Construx®
SOFTWARE DEVELOPMENT BEST PRACTICES

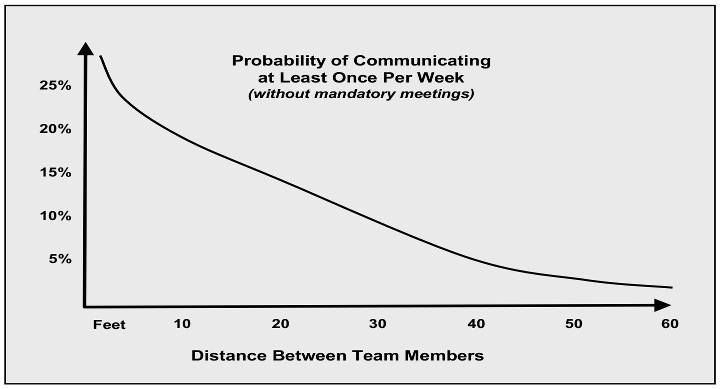
Contents

Establish Co-Located, Cross-Disciplinary Teams

Most of the companies that Construx works with are geographically distributed, with at least two, and often six or more major locations. For most large organizations, having sites around the world is simply the way that business is done today.

While some companies initially established multiple sites to conduct "follow-the-sun" development, this model has not proven successful in practice. Most organizations have found the best way to successfully use multiple locations is to divide the work into logical chunks for each location. Whenever possible, they establish teams in each location that have end-to-end responsibility.

Software development is a human endeavor and requires communication and coordination between team members. The figure below shows how communication decreases as distance between team members increase.



Adapted from: *Building a Project-Driven Enterprise*, by Ronald Mascitelli

[1]

Figure 1 *Communication decay over distance*

When teams are distributed around the world, communication and coordination is significantly limited, becoming slow and time consuming. Organizations that have the most success with geographic distribution create teams in each location that focus on specific products, features, components, or other pieces of work that are as independent as possible. Each team includes the Product Owner, Scrum Master, and all members of the Development Team.

Whenever this is not possible, teams need to put in place additional practices and infrastructure to support the necessary level of communication and coordination.

[1] Mascitelli, Ronald. *Building a Project-Driven Enterprise: How to Slash Waste and Boost Profits Through Lean Project Management*. Technology Perspectives, 2002.

Prefer Feature Teams

Whereas component teams focus on a layer in the architecture (the database, the user interface, and so on) or on a component (the kernel), feature teams include staff who can cross all components of a system to deliver fully tested, potentially shippable increments of user value.

Use of feature teams simplifies dependency management and coordination between teams. Some of the major benefits of feature teams include:

- **Better cycle time through reduced handoffs**. Since the feature team can complete an entire end-user element of user value, every sprint can result in something meaningful to the user. Component teams, on the other hand, require handoffs throughout the layers in the application before meaningful user value is present.
- **Improved code/design quality.** Perhaps unexpectedly, the general industry experience is that using multiple feature teams who work on shared components creates pressure to keep the code modular, maintainable, and well tested. This is an advantage over component teams, which tend to perpetuate obfuscated code, since they have the tribal knowledge on how it works.
- **Simplified planning and tracking.** Planning and tracking is much more complex when the output of multiple component teams is needed to produce meaningful user value. When features can be given to a single team, progress tracking is simpler.

Organizations and product teams should seek to form feature teams wherever possible. The most common approach to this is to create a set of feature teams that have representatives from the various component teams.

Establish a Project-Wide Definition of Done

A frequent issue with large Scrum projects is the lack of a common Definition of Done[2] (DoD) across all teams. Without this, the quality practices vary widely from team to team. For example, one team is doing test driven development and including its system testing in the same sprint. Another team does not automate unit testing, and its system testing occurs in the subsequent sprint. This lack of consistency makes it hard to understand the state of the product and to track progress toward delivery.

To address this inconsistency, establish a minimum set of criteria (Definition of Done) that teams must meet to consider a Product Backlog Item (PBI) complete within a sprint. Individual Scrum teams can build upon this standard DoD with specific items they need beyond the shared fundamentals the project team establishes.

[2] See Construx's *Definition of Done* Practice Paper for further information on what is commonly included in an individual Definition of Done.

Converge Frequently

Integrating completed work and validating it with the Product Owner is a fairly straight forward activity for a single Scrum team. The importance of this frequent convergence does not go away for a large, distributed team, but it does become more complex.

If you have 25 Scrum teams, how often do they integrate into the mainline? The answer depends on your situation. A team with continuous integration and a significant amount of unit and system test automation might integrate PBIs from individual Scrum teams into the mainline throughout the sprints. Teams that rely solely on manual system testing will struggle to keep a high quality mainline with that approach.

The goal is to converge as frequently as possible for your project team given its situation. Most teams do not have the automation in place to practice continuous convergence. In that case, teams might integrate each sprint or establish convergent points after a set number of sprints, when all of the teams integrate their completed work together. This fully converged product can then be reviewed by the Product Owner Council or its equivalent.

Using long convergence points increases the risk that integration issues will be discovered late. Teams using this approach should invest in the automation, continuous integration infrastructure, and technical practices necessary to introduce continuous convergence. Continuous convergence ensures that a potentially releasable product is available at all times.

Implement Release Planning

The most common recommendation that Construx makes for large Scrum project teams is the adoption of release planning. Release planning is a working session that includes at least the Product Owners, Scrum Masters, technical and quality representatives, and relevant Subject Matter Experts (SMEs).

Release planning[3] results in a shared understanding of the vision for the release, overall prioritization guidance, cross-team working agreements, allocation of the Product Backlog(s) to teams, and agreed-upon completion criteria for work that crosses teams.

The complexity of your project will determine the details of how often release planning occurs, who participates, where it occurs, and what types of planning happen above and below it. The requirements process used for the project also has an influence.

[3] For more details, see Construx's *Release Planning* white paper.

Define a Requirements Process

For small Scrum teams, the requirements process is clear—a Product Backlog is maintained by a Product Owner and feeds work to the team. But how does it scale? What happens with a 250-person team working on a product or product suite? Isolated Product Backlogs are likely to result in a disjointed product. For most teams, a single, shared Product Backlog with views for individual teams works well.

Project teams need to determine the process they will use to identify and prioritize epics, decompose epics to stories, allocate epics and/or stories to team(s), and validate their acceptance. An example project workflow is shown in Figure 2.

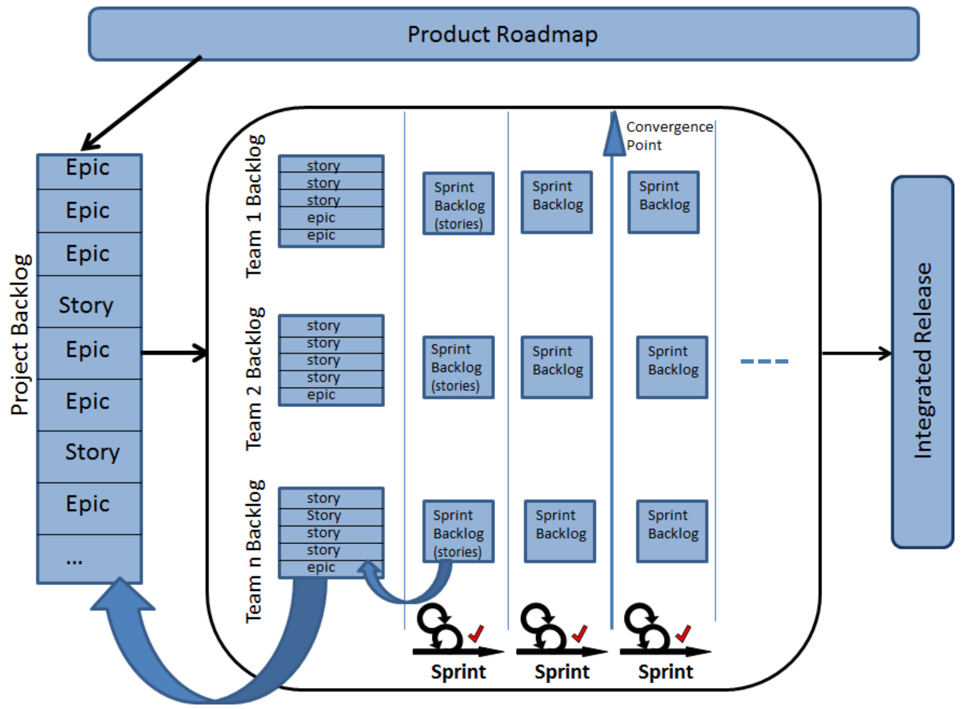


Figure 2 *Example project workflow*

One common approach is to use a Chief Product Owner and a Product Owner Council. The Chief Product Owner (CPO) is the ultimate decision maker at the product level. The Product Owner Council (POC) includes all Product Owners and relevant SMEs. In most organizations, the POC meets once or twice a week for the following:

- reviewing priorities, especially at the product level
- discussing and refining epic level requirements and acceptance criteria
- decomposing epics for upcoming sprints
- verifying the epic level acceptance criteria are complete
- identifying and discussing dependencies
- allocating the backlog to teams

- tracking overall progress
- reviewing and revising the release plan

A distributed Scrum team needs to have the right people participating in requirements and effectively decomposing larger pieces of work across all of the teams.

For teams that have a remote PO, consider investing in a Behavior Driven Development (BDD) approach to documenting acceptance criteria for user stories. This structured requirements approach reduces ambiguity and ensures alignment between the Development Team and Product Owner.

Address System Architecture Implications

Construx's 10 Keys to Successful Scrum Adoption white paper discusses the need to steward the architecture on Scrum projects. We believe there is a balance between too much architecture (over-engineering a system and wasting time creating things that are not need) and too little architecture (not building fundamental underpinnings of the system and finding later on that it is too weak to support the objectives for the project).

Large, geographically distributed teams need to ensure there is enough architectural guidance so that individual teams have what they need. If teams are building duplicate functionality, releasing updated API's that cause unexpected failures in other components, or creating unique approaches for sending data through the system, then your project has a gap in its system architecture guidance and oversight.

Large projects need someone (a single architect or virtual team[4]) who is actively involved with the teams to provide guidance, support the creation of a high-level architecture description, and trouble shoot issues.

Implement a Multi-Level Testing Approach

One common pitfall with Scrum at scale is that individual project teams validate their functionality is working, but insufficient focus and attention is spent on validating the end-to-end system.

For example, Construx once worked with a team which was doing an excellent job of testing at the team level. They had 75% unit coverage of the code, automated integration test suites, and system testing of the PBIs completed by the team. The gap was the limited resources validating the product from the user interface through the business layer and into the databases. Without this, they had insufficient visibility into the actual completeness of the major epics they were building. As they invested in staff in this area, they discovered their project was months behind schedule.

[4] See Construx's *Virtual Architecture Team* Practice Paper for more information.

Large, distributed Scrum teams typically find that they need these two parallel testing activities:

- **Team-level testing.** Individual teams validate that the PBIs they have completed are integrated and work within the overall product. This includes unit, integration, and system testing of those PBIs. The expectations here are typically captured as the Definition of Done.
- **Product-level testing.** A product level test team executes the overall product to validate that end-to-end customer requirements are working, ensure the overall product is stable, identify regressions caused by new functionality, and perform non-functional testing (performance, scalability, load, etc.). This testing does not replace the testing done by each Scrum team; rather, it supplements it and focuses on how changes impact the end-to-end product.

Having a two-prong testing approach helps the individual teams focus on the work they are doing without neglecting the stability of the overall product.

Document Appropriately

The adoption of Agile methods, such as Scrum, reduces the amount of documentation that project teams need to produce. Small, co-located teams can rely on stories as a reminder to have the necessary conversations before and/or during individual sprints.

This approach is difficult to scale to large, geographically distributed projects. These project teams often have dependencies on one another but are separated by distance and time.

Documentation is one solution to the distance and time problem in communication. Documents address the need to communicate to someone else (or to ourselves) at some future time and/or in some other place. Code level documentation helps a future maintainer or the original author remember why a particular design approach was selected. Requirements documents capture information needed by developers and testers to build and validate the software.

Agile methods seek to reduce the need for formal documentation by putting people together in the same place working on a small bit at the same time. This increased face-to-face communication by all parties reduces the need for documentation.

When Scrum projects scale to hundreds of people and/or across multiple geographic locations, we are adding distance and time back in. These project teams need to determine what documentation is necessary to enable communication. For example, we might create an architecture document to ensure everyone (on this project and future projects) understands how the components of the system work together; an API document might capture the interfaces (and changes to those interfaces) so that teams can work as independently as possible; or user acceptance testing might be added to user stories so that a remote team can create test cases that are reviewed before the sprint begins.

Wikis or other infrastructure should be adopted to make capture, communication, and storage of the information simple.

Invest in Travel

Software development is a human endeavor and relies heavily on the relationships between people. This was true in more traditional development approaches, such as waterfall, and it is still true with Scrum. Whenever there are dependencies between teams or coordination needed, it works better when the people know and trust each other. One VP that Construx worked with stated it this way, “the half-life of trust is 6 weeks.” And that trust is only built when people have face-to-face interaction. Another VP commented, “when you start seeing mistakes, it is time to put people on a plane.” One common issue we see in troubled projects is that the team is fractured, with mistrust between teams, geographies, functions, or some other divide. Investing in travel and face-to-face time can be an invaluable way to avoid or minimize issues before they arise.

Contributors

Jenny Stuart, VP Consulting

jenny.stuart@construx.com
+1(425) 636-0108

Melvin Perez, Senior Fellow

melvin.perez@construx.com
+1(425) 636-0120

Alex Sloley, Senior Fellow

alex.sloley@construx.com
+1(425) 636-0118

About Construx

Construx Software is the market leader in software development best practices training and consulting. Construx was founded in 1996 by Steve McConnell, respected author and thought leader on software development best practices. Steve's books *Code Complete*, *Rapid Development*, and other titles are some of the most accessible books on software development with more than a million copies in print in 20 languages. Steve's passion for advancing the art and science of software engineering is shared by Construx's team of seasoned consultants. Their depth of knowledge and expertise has helped hundreds of companies solve their software challenges by identifying and adopting practices that have been proven to produce high quality software—faster, and with greater predictability. For more information about Construx's support for software development best practices, contact us at consulting@construx.com, or call us at +1(866) 296-6300.

Construx®

SOFTWARE DEVELOPMENT BEST PRACTICES

© 2015-2018, Construx Software Builders, Inc. All rights reserved.
Construx Software Builders, Inc.
10900 NE 8th Street, Suite 1350
Bellevue, WA 98004
U.S.A.

This white paper may be reproduced and redistributed as long as it is reproduced and redistributed in its entirety, including this copyright notice.

Construx, Construx Software, and CxOne are trademarks of Construx Software Builders, Inc. in the United States, other countries, or both.

This paper is for informational purposes only. This document is provided "As Is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Construx Software disclaims all liability relating to use of information in this paper.