# Scaling Agile Development

## Large and Multisite Product Development with Large-Scale Scrum

**Craig Larman, Bas Vodde**

**Abstract.** Since 2005 we have worked with clients to apply the Scrum framework and to help scale agile development to product groups involving from a few hundred to a few thousand people, in multiple sites. This is organized as large-scale Scrum frameworks 1 and 2, summarized in this article and elaborated on in our two-volume book series on very large-scale agile development.

### Background

In 2003, when Craig Larman published Agile & Iterative Development [1], many "knew" that agile development was for small groups. However, we became interested in—and got increasing requests—to apply Scrum to very large, multisite, and offshore product development. So, since 2005 we have worked with clients to scale up—often for "embedded" systems. Today, the two large-scale Scrum frameworks described herein have been introduced to big groups worldwide in disparate domains, including telecom-infrastructure-equipment providers such as Ericsson [2], and investment-banking clients such as Bank of America-Merrill Lynch, plus many more.

To quantify "large", we have seen our large-scale Scrum framework-2 applied in groups of up to 1,500 people, involving seven developments sites spanning the globe. Our median experience is perhaps around 800 people on one product at 5 sites, with about 15 million lines of source code, usually C++, C, and Java.

Based on these experiences we published two volumes on scaling agile development and the large-scale Scrum frameworks summarized: (volume 1) Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum [3], that explains the leadership and organizational design changes, and (volume 2) Practices for Scaling Lean & Agile Development: Large, Multisite & Offshore Product Development with Large-Scale Scrum [4], that explains concrete suggestions for scaling, including in product management, architecture, planning, multisite, offshore, and contracting. This article summarizes concepts expanded on in those books.

### Large-Scale Scrum is Scrum: Change Implications

Scaling Scrum starts with understanding and being able to adopt standard real one-team Scrum. Large-scale Scrum requires examining the purpose of single-team Scrum elements and figuring out how to reach the same purpose while staying within the constraints of the standard "Scrum rules."

If the entire R&D group was only seven people, the implications of changing to adopt true one-team Scrum are not dramatic, since many elements will "organically" be in place—as in a startup. But when a traditional R&D group of 500 people moves to Scrum, there are major change implications, and these need full understanding and support by senior leadership and hands-on producers. These include:

**1) Standard Scrum:** A small (five to nine people) cross-functional team of multi-learning team members that do everything end-to-end to develop the product (a real feature team [5]), and no specialized sub-groups within the team, with the only title of "team member" [6].

**Change/scaling implications:**

- No separate analysis group, testing group, architecture group, user experience group, platform group, etc. And no "tester" or "architect" within the team. That implies the dissolution of existing single-function groups and the management supervising roles, and the elimination of traditional career paths and job titles.

**2) Standard Scrum:** The business-person "owner of the product" (such as, lead product manager) responsible for ROI and cost, and who can independently decide and change the content and release date becomes the Scrum product owner. The owner of the product steers development directly based on "inspect and adapt" and so is ultimately responsible for the product release, since they have the steering wheel.

**Change/scaling implications:**

- Traditionally, the owner of the product negotiated a scope-and-date milestone-based internal contract with R&D managers, who were thereafter responsible for the release. Since the owner of the product now steers directly, there is no shifting of responsibility to R&D to develop the release, and no internal contract.
- Since the owner of the product steers development directly and is responsible for the release, there is no separate R&D or IT program/project manager responsible for the release; that role is eliminated.

**3) Standard Scrum:** Each two- to four-week Sprint, from the first, the product increment must be done and potentially shippable— a potentially shippable product increment. Each Sprint the system must be implemented, integrated, fully tested, documented, and capable to deploy.

**Change/scaling implications:**

- The concept of a "big release" and the constraint, "it is not ready until the end" dissolves. This implies eliminating big release management systems, practices, roles, and policies that are predicated on a long phase of messy partially-done development before the system is ready.
- Scrum is not for the programming phase after analysis and before testing. There is no prior analysis phase or architecture phase and no following integration/testing phase. Sequential lifecycle development is eliminated, and with it, the groups that were attached to each phase (the analysis group, ...).

**4) Standard Scrum:** The team is self-organizing (self-managing) and is empowered to independently decide how to achieve their goal in the Sprint.

**Change/scaling implications:**

- There is no team lead or project manager that directs or tracks team members, which implies the elimination of related team-lead and project-manager roles.
- No organization-wide standard process that everyone must follow.

This is simply standard one-team Scrum, but its adoption especially challenges traditional R&D assumptions and organizational design at scale. Therefore, most groups do not adopt real Scrum, but instead customize it (into "fake Scrum" or "Scrum, but…") rather than change themselves.

**Observations and suggestions:**

- Real agile development with Scrum implies a deep change to become an agile organization; it is not a practice, it is an organizational design framework.
- Start a large-scale agile Scrum adoption by ensuring leadership understands the organizational implications, and they have been proven adoptable in the small scale.

## Two Agile Scaling Frameworks

After the aforementioned organizational-design changes are understood by leadership and they flip the system, then one of two large-scale Scrum frameworks can be adopted. Most of the scaling elements are focusing the attention of all the teams to the whole product instead of "my part". Global and "end-to-end" focus is perhaps the dominant problem to solve in scaling. The two frameworks – which are basically single-team Scrum scaled up – are:

- Framework-1: Up to 10 Scrum teams (of seven people).
- Framework-2: Up to a few thousand people on one product.

Framework-1 is appropriate for one (overall) Product Owner (PO) and up to 10 teams. 10 is not a magic number for choosing between framework-1 and framework-2. The tipping point is context dependent; sometimes less. At some point, (1) the PO can no longer grasp an overview of the entire product, (2) the PO can no longer effectively interact with the teams, (3) the PO cannot balance an external and internal focus, and (4) the product backlog is so large that it becomes difficult for one person to work with. When the PO is no longer able to focus on high-level product management, something should change.

A group with seasoned people who know the product and customers well and are co-located with the PO can handle more teams with one PO. A newly formed outsourcing group in India who do not know the domain, with a PO in Boston, will require less teams.

Before switching to framework-2, first consider if the overburdened PO can be helped by delegating more work to the teams. Encourage teams to directly interact with real customers to reduce handoff and reduce the burden on the PO. Most detailed analysis and project management should be done by the teams; the PO does not need to be involved in low-level details – they should be able to focus on true product management.

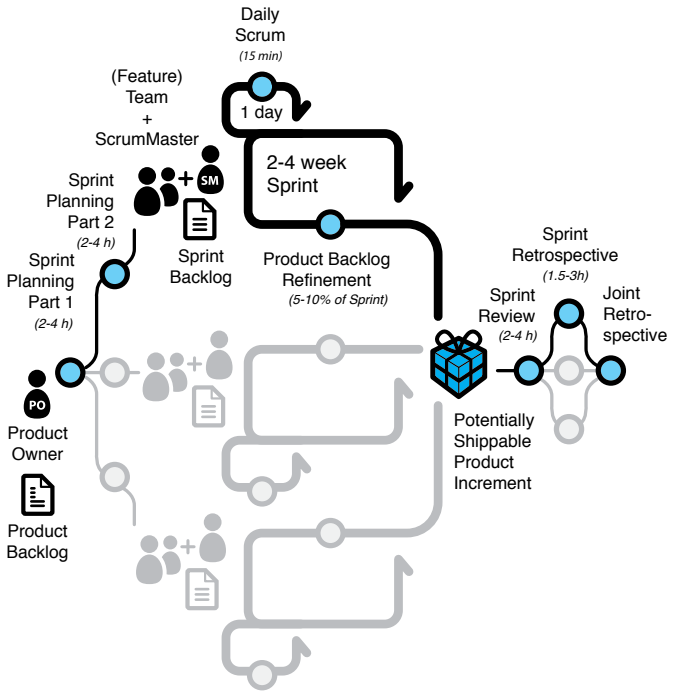


*Figure-1. Large-scale Scrum framework-1*

**What is the Same as One-Team Scrum?**

- One product backlog (it is for a product, not a team)
- One definition of done
- One potentially shippable product increment
- One (overall) product owner
- Each team is a "team" (and there are no single-specialist teams)
- One Sprint

In summary, all teams are in a common Sprint to deliver a common potentially shippable product increment.

**What is Different?**

- Role changes: none.
- Artifact changes: none; but to clarify: Sprint backlog and Sprint goal per team.
- Meeting changes: The dominant difference in large-scale Scrum framework-1 is the behavior of Scrum meetings, driven by coordination needs. This is illustrated in Figure-1 and explained next:

**1. Sprint Planning Part 1:** One meeting, and same maximum duration: one hour per week of Sprint. Rather than all team members participating, limit it to two members per team, plus the one overall PO. Let team representatives self-manage to decide their division of product backlog Items, although if "competition" exists the PO can break a tie. End with the partial-teams identifying dependencies (perhaps with a dependency matrix) between PBIs and discussing coordination.

**2. Sprint Planning Part 2:** Independently (and usually parallel) per team, though sometimes a member of team-A may observe team-B's meeting and make suggestions when there is a coordination issue between the teams.

**3. Daily Scrum:** Independently per team, though a member of team-A may observe team-B's Daily Scrum, to increase information sharing.

4. **(addition) Inter-team coordination meeting:** Several times per week, team representatives may hold an open space, town hall meeting, or Scrum of Scrums, to increase information sharing and coordination.

5. **(addition) Joint light product backlog refinement:** Maximum duration: 5% of Sprint. Only two representatives per team. Splitting, analysis, and estimation for soon-to-develop PBIs. Analysis is lightweight; for example, if using Specification by Example, only three examples per item. Note that the cross-team estimation ensures a common baseline for estimation across teams. Note that this meeting increases product-level learning and team agility – the ability of any team to take on any PBI.

6. **Product Backlog Refinement:** For this mid-Sprint meeting preparing for future Sprints, for co-located teams, hold this at the same time in one big room with all team members; each team facing a separate wall with their own learning tools (whiteboards, projectors, ...). Apply rotation writing and other large-group workshop techniques so that all members across all teams are eventually exposed to analyzing all items, which is critical for more team flexibility.

7. **(optional addition) In-Sprint PBI Inspection:** When possible, informally seek out early feedback from the PO or other stakeholders on finished PBIs as soon as possible during the Sprint, to reduce the inspection and discussion that would otherwise be required at the Sprint review; this does not eliminate the Sprint review.

8. **Sprint Review:** One meeting, and same maximum duration: one hour per week of Sprint. Limit it to two members per team, plus the PO and other stakeholders. Rather than only a common inspection of the running potentially shippable product increment, consider a "bazaar" or "science fair"-style phase during the middle of the Review: a large room has multiple areas with computers, each staffed by team representatives, where the features developed by a team are shown and discussed. In parallel, stakeholders visit areas of interest and team members record their feedback. However, begin and end the Sprint Review with everyone in a common discussion, to increase overall feedback and alignment.

9. **Team Retrospective:** Independently per team; no change.

10. **(addition) Joint Retrospective: Maximum duration:** 45 minutes per week of Sprint. Since the team retrospective ends the Sprint, this Joint Retrospective is held early in the first week of the subsequent Sprint. ScrumMasters and one representative of each team meet to identify and plan improvement experiments for the overall product or organization.

## Agility Across Teams

Notice that large-scale Scrum increases learning across teams; most can flexibly do any Product Backlog item. This is in contrast to "team A can only do A-type work", and critical for agility when scaling, so that teams are responsive to change, and all can focus on the highest-value work, rather than constrained by single specialty. Remember: agile development is for agility (flexibility) over efficiency.

## Coordination

When scaling, a dominant issue is coordination. In traditional scaling, this is (poorly) handled with major upfront "fixed" specifications and planning, private-code component teams, and extra managers. In scaling agile development, coordination is handled more by increased coordination in shared code and self-organizing teams. Besides meetings, what other coordination elements are in large-scale Scrum?

- **Continuous integration:** All code, across all teams, is integrated continuously (many times per day) and verified with automated tests, with a "stop and fix" culture of rapidly fixing a broken build.
- **Internal open source:** Rather than private-code components and "component teams", there is collective code ownership or "internal open source." Many open-source practices apply, such as standard coding style.
- **Feature teams:** Scrum feature teams develop end-to-end "vertical" customer-centric features across all shared code.
- **Communities of practice (CoPs):** To handle cross-team concerns (architecture, user experience, standards, ...) CoPs are established (and all that implies), with membership from the Scrum teams (not from external people). For example, a Design/Architecture CoP for the key concern of good design at scale; this is not composed of a separate "architecture group", but by volunteering regular Scrum team members with the skill and passion.
- **Team-controlled build system:** Rather than a separate "build group", regular Scrum teams rotate responsibility for maintaining their common build system.
- **More talking!**

Notice as a theme that coordination is handled by self-organizing teams (rather than more managers), and with fast-feedback integration cycles in code (rather than more planning and separated code).

## Multisite

If an entire product group is seven people in four sites then a co-located team is difficult. But when 50 people, it is possible to create co-located teams of five to nine people: three teams in Boston, etc. Therefore:

- **Co-located teams:** Although different co-located teams may be in different sites, avoid a single dispersed team with scattered members. The motivation for dispersion is usually specialist bottlenecks ("only Mary knows X") but a key value in Scrum is to increase learning and multi-skill to reduce bottlenecks, rather than accept them.
- **Continuous integration across all sites:** And related...
- **Free open-source (FOSS) tools:** Especially when multisite, we observe frictions in groups using commercial tools... "We cannot have more licenses", "Wait for purchasing" etc. FOSS tools (Subversion, Git, GNU tools, Eclipse, Java, etc.) eliminate friction, reduce costs, and are usually superior.
- **Free "Web 2.0" information tools:** Multisite requires more software tools; use FOSS wikis, Google Docs, and other free "pure Web" tools for information (lists, requirements, etc.), rather than commercial and pre-Web document-based tools such as Word, SharePoint, DOORS.

• Free ubiquitous video: Rapport and trust—critical! And it is degraded when people do not see each other, so replace phone calls with video. Use free, ubiquitous tools such as Google Video Hangouts and a projector.

• Multisite Sprint Planning Part 1: How? The PO is with local representatives. Other sites use video and web tools. The PO offers items via a web tool (e.g., Google Spreadsheet). Parallel discussion on the items happens on different wiki pages or chat sessions.

• Multisite Product Backlog Refinement: As in Planning Part 1, emphasizes video and web tools. If estimating with Planning Poker, use (for example) a Google Spreadsheet with different members typing estimates into different cells.

• Multisite Sprint Review: As above.

• Multisite communications CoP: Good communication requires meta-communication.

| Product Backlog | |
|---|---|
| **Backlog Item** | **Requirement Area** |
| IPv6 | protocols |
| performance 10x | performance |
| HSDPA | management |
| performance stats | protocols |
| configuration of cells | management |
| new NMS solution | continuous integration |
| speed-up of build | upgrades |
| improved upgrading support | management |
| stability to 99.999% | reliability |

*Figure-2. Requirement Areas*

## Requirement Areas

With 1,000 people on one product, divide-and-conquer is unavoidable. Traditional development divides into single-function groups (analysis, ...) and architectural-component groups (UI-layer group, ...), yielding slow inflexible development with high levels of waste (inventory, work-in-progress, handoff), long-delayed ROI, and weak feedback. And it is organized "inward" around function and architecture, rather than "outward" around customer features.

In large-scale Scrum framework-2, we do not divide by architecture; rather, we divide around major areas of customer requirements – requirement areas. For example: fault management or options trading. Then, we add a "requirement area" column to the Product Backlog and classify each item in one area (Figure-2). A filter on one Product Backlog shows distinct Area Backlog views (Figure-3).

## New Role: (Requirement) Area Product Owner

To deal with the overwhelming complexity for one PO, we introduce a new role: an area PO, who focuses on one area backlog.

The one overall PO plus all area POs form the product owner team.

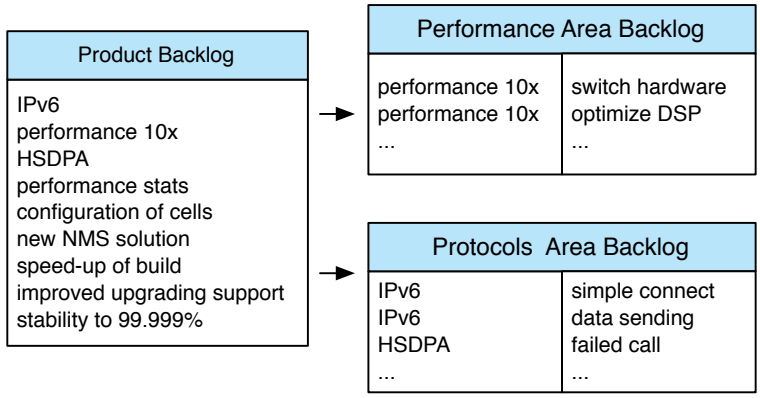


*Figure-3. (Requirement) Area Backlogs*

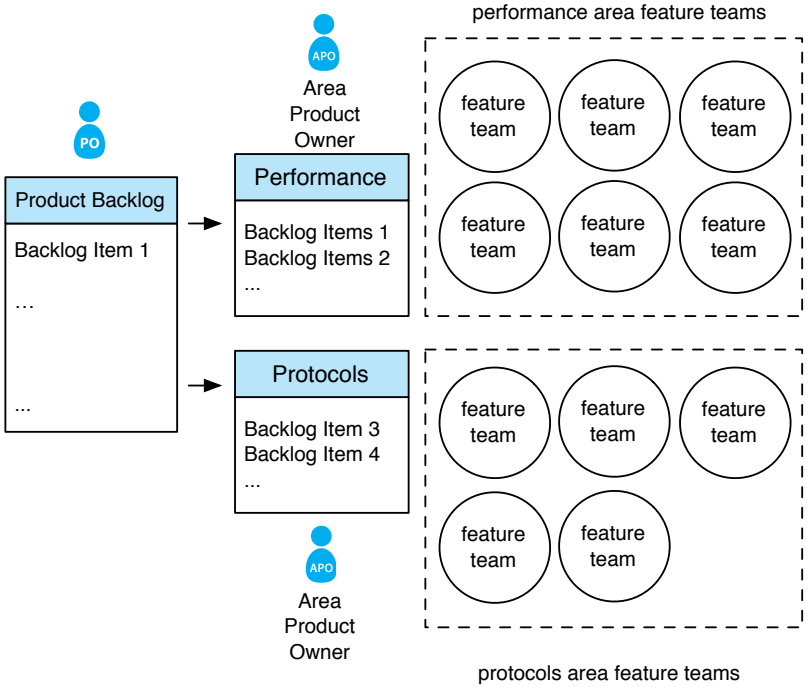


*Figure-4. Area PO and Teams*

### Area Teams

A set of three to 10 teams (area teams) are dedicated to one area PO, all who specialize in one requirement area (Figure-4). Each team is cross-functional and cross-component, doing end-to-end customer-centric feature development.

## The Big Idea?

Large-scale Scrum framework-2 is a set of several framework-1 groups (one per requirement area) working in parallel in a common Sprint (Figure-5).

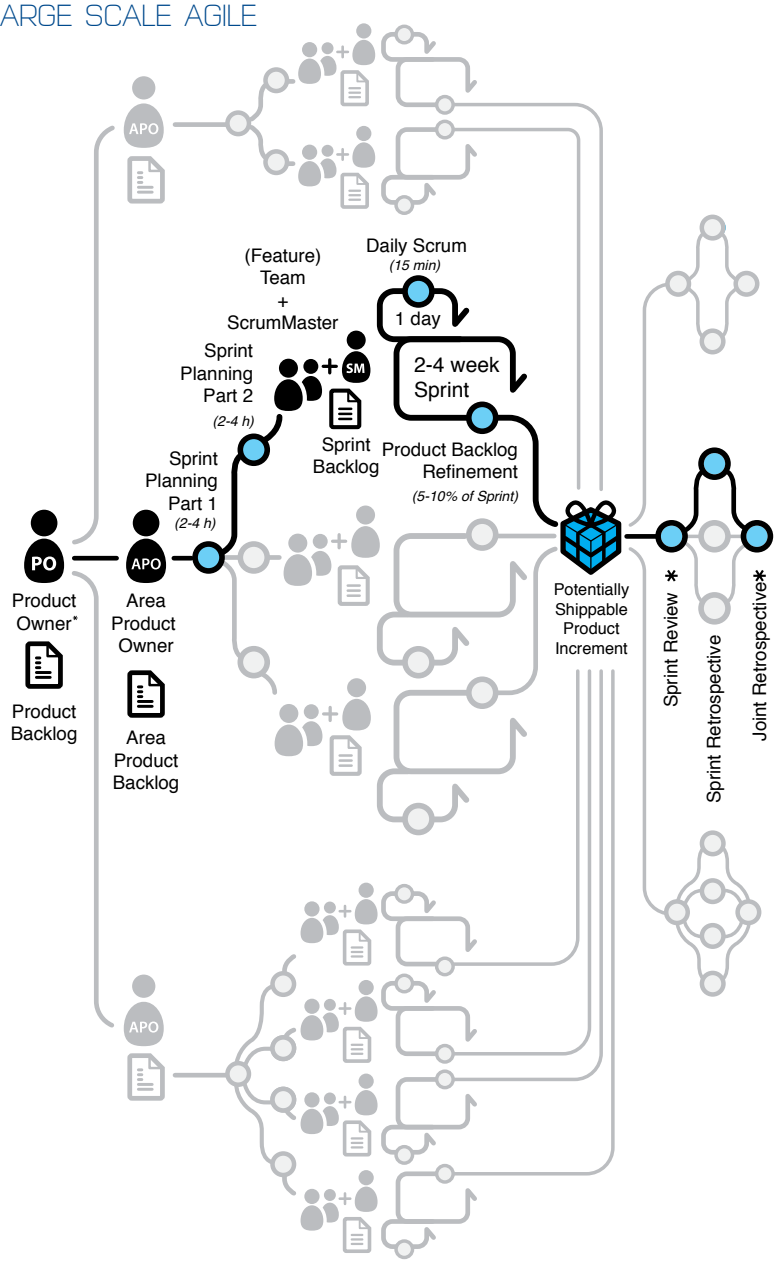


*Figure-5. Large-Scale Scrum Framework-2*

**What is the Same as Framework-1?**

One product backlog, one definition of done, one potentially shippable product increment, one (overall) product owner, one Sprint. All teams in one Sprint with one delivery.

**What is Different?**

- Role changes: area PO.
- Artifact changes: "Requirement areas" in product backlog; area backlog views.
- Meeting changes: Framework-2 is a set of parallel (per requirement area) framework-1 Sprint executions; therefore ...

**1. Pre-Sprint Product Owner Team Meeting:** Before each area PO meets in their own Sprint planning Part-1 meeting with their area Teams, they need to coordinate together and with overall PO – who focuses on product-level rather than area-level optimization. This coordination must happen before the Part-1 meetings, usually late in the prior Sprint.

**2. Area-Level Meetings:** As in normal framework-1, Sprint planning part 1, joint product backlog refinement, Sprint review, and joint retrospective need to occur for each requirement area.

**3. Overall Sprint Review:** A review is needed at the product level, not merely each area. It is not possible to review all items of all areas, so the focus is on a subset of interest to the overall PO or to many area POs.

**4. Overall Sprint Retrospective:** For system-level improvement, a retrospective is needed at the product level, not merely each area. This happens earlier in the subsequent Sprint, after area-level joint retrospectives.

## ABOUT THE AUTHORS

Craig Larman is an organizational-design consultant for enterprise adoptions and very large-scale product development with large-scale Scrum, especially in embedded systems domains, and investment banking. He serves as chief scientist at Valtech, and is the co-author (with Bas Vodde) of Scaling Lean & Agile Development: Thinking & Organizational Tools and Practices for Scaling Lean & Agile Development: Large, Multisite, & Offshore Product Development with Large-Scale Scrum, and Agile & Iterative Development.

**E-mail: craig@craiglarman.com**
**Website: www.craiglarman.com**

Bas Vodde is a coach, consultant, trainer, and author in modern agile and lean product development. When coaching, he works on three levels: organizational, team, individual/technical practices. He has trained thousands of people in software development, Scrum, and agile practices for over a decade. He is the co-author of two books on large-scale Scrum and agile development, with Craig Larman. Bas works for Odd-e, which supports organization in improving product development, mainly in Asia.

**E-mail: basv@odd-e.com**
**Website: www.odd-e.com**

## REFERENCES

1. Larman, Craig. Agile & Iterative Development: A Manager's Guide. Addison-Wesley, 2003.
2. Ericsson R&D Center Finland, "How we learn to stop worrying and live with the uncertainties". <https://www.cloudsoftwareprogram.org/results/deliverables-and-other-reports/i/27891/1941/ericsson-journey-of-change>
3. Larman, Craig & Vodde, Bas. Scaling Lean & Agile Development: Thinking & Organizational Tools for Large-Scale Scrum. Addison-Wesley, 2008.
4. Larman, Craig & Vodde, Bas. Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Addison-Wesley, 2010.
5. Larman, Craig & Vodde, Bas. Feature Team Primer. <http://www.featureteamprimer.org/>
6. Vodde, Bas. "Specialization and Generalization in Teams". <http://www.scrumalliance.org/articles/324-specialization-and-generalization-in-teams>