# Feature Team Primer

by Craig Larman and Bas Vodde

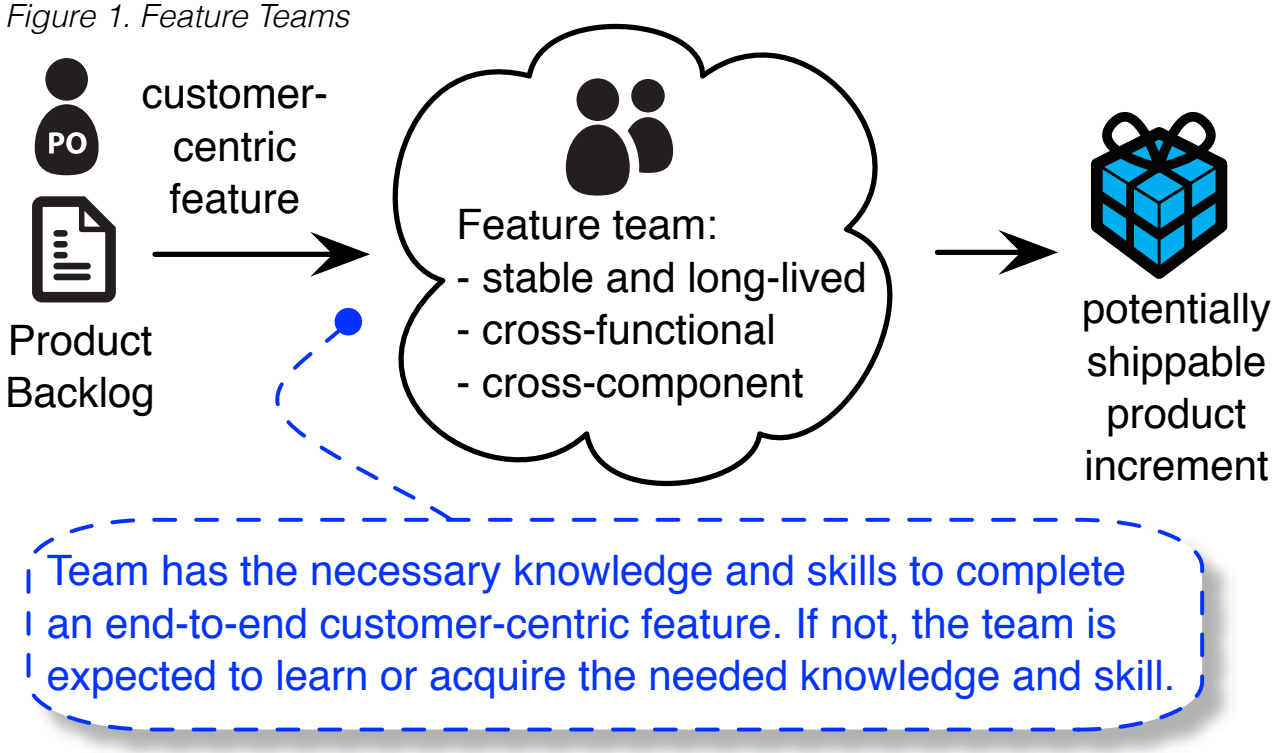
Version 1.3

***Feature teams*** *and* ***Requirement Areas*** *are key elements of scaling lean and agile development. They are analyzed in depth in the Feature Team and Requirement Area chapters of* Scaling Lean & Agile Development: Thinking and Organizational Tools for Large Scale Scrum*. This short paper summarizes a few key ideas and can also be found in* Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*.*

## Introduction to Feature Teams

A **feature team**, shown in Figure 1, is a long-lived[1], cross-functional, cross-component team that completes many end-to-end customer features—one by one.

*Figure 1. Feature Teams*




[1] Feature teams stay together for years, implementing many features.

The characteristics of a feature team are listed below:

| Feature Team |
| --- |
| • long-lived—the team stays together so that they can 'jell' for higher performance; they take on new features over time<br>• cross-functional and cross-component<br>• ideally, co-located<br>• work on a complete customer-centric feature, across all components and disciplines (analysis, programming, testing, ...)<br>• composed of generalizing specialists<br>• in Scrum, typically 7 ± 2 people |

Applying modern engineering practices—especially continuous integration—is essential when adopting feature teams. Continuous integration facilitates shared code ownership, which is a necessity when multiple teams work at the same time on the same components.

A common misunderstanding: every member of a feature team needs to know the whole system. Not so, because

- The team as a whole—not each individual member—requires the skills to implement the entire customer-centric feature. These include component knowledge and functional skills such as test, interaction design, or programming. But within the team, people still specialize… preferably in multiple areas.
- Features are not randomly distributed over the feature teams. The current knowledge and skills of a team are factored into the decision of which team works on which features.

Within a feature team organization, when specialization becomes a constraint…learning happens.

A feature team organization exploits speed benefits from specialization, as long as requirements map to the skills of the teams.

But when requirements do not map to the skills of the teams, learning is 'forced,' breaking the overspecialization constraint.

Feature teams balance specialization and flexibility.

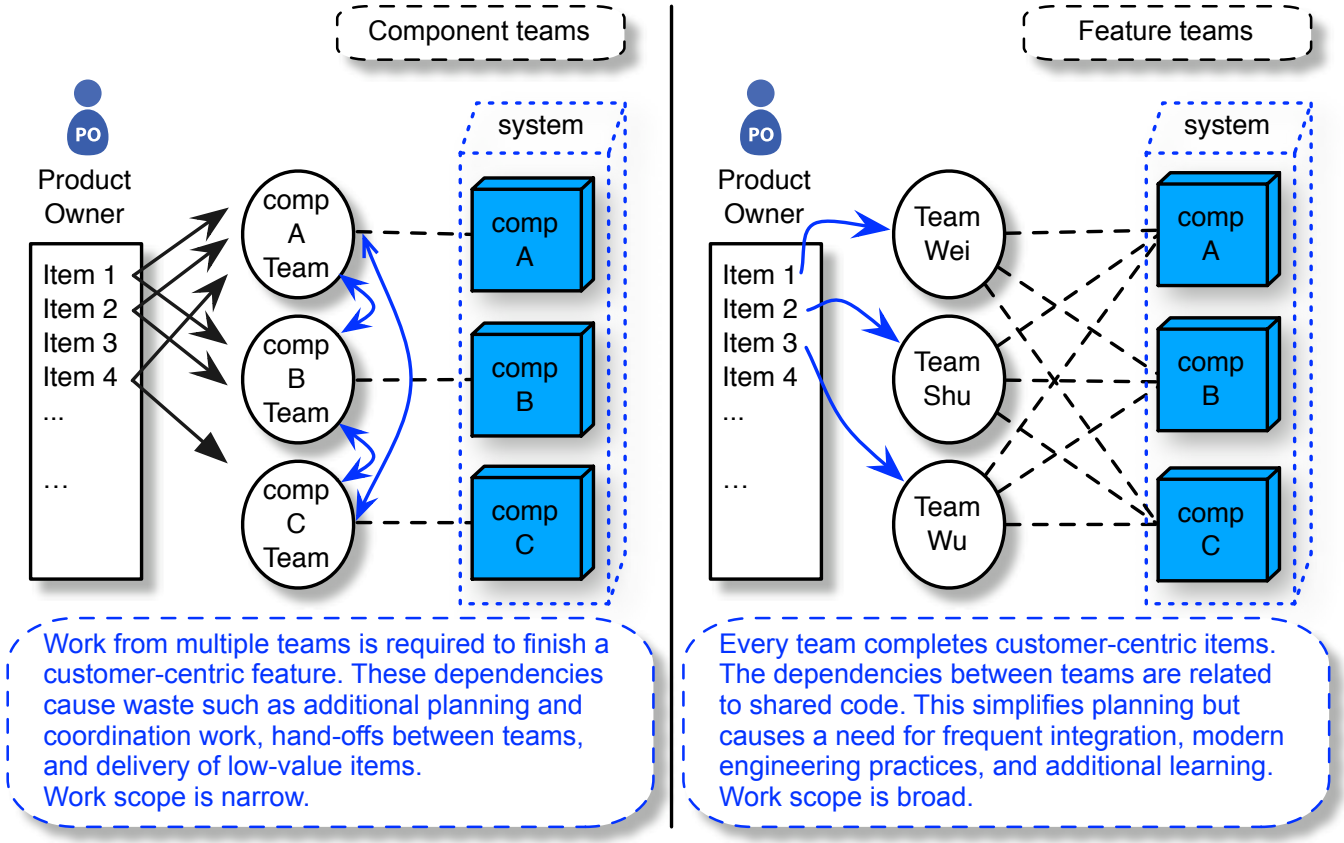
Table 1 and Figure 2 show the differences between feature teams and more traditional component teams.

*Table 1. feature teams vs. component teams*

| Feature Team | Component Team |
|---|---|
| optimized for delivering the maximum customer value [a] | optimized for delivering the maximum number of lines of code |
| focus on high-value features and system productivity (value throughput) | focus on increased individual productivity by implementing 'easy' lower-value features |
| responsible for complete customer-centric feature | responsible for only part of a customer-centric feature |
| modern' way of organizing teams [b] — avoids Conway's law | traditional way of organizing teams — follows Conway's law [c] |
| leads to customer focus, visibility, and smaller organizations | leads to 'invented' work and a forever-growing organization |
| minimizes dependencies between teams to increase flexibility | dependencies between teams leads to additional planning [d] |
| focus on multiple specializations | focus on single specialization |
| shared product code ownership | individual/team code ownership |
| shared team responsibilities | clear individual responsibilities |
| supports iterative development | results in 'waterfall' development |
| exploits flexibility; continuous and broad learning | exploits existing expertise; lower level of learning new skills |
| requires skilled engineering practices—effects are broadly visible | works with sloppy engineering practices—effects are localized |
| provides a motivation to make code easy to maintain and test | contrary to belief, often leads to low-quality code in component |
| seemingly difficult to implement | seemingly easy to implement |

a. *The different optimization often makes the feature team feel slower—from the local view.*
b. *Relatively 'modern' as feature teams have a long history in large-scale development, for example, Microsoft and Ericsson.*
c. *Mel Conway observed this undesirable structure in1968,he did not recommended it—in fact, quite the opposite.*
d. *This additional planning is visible in more "release planning meetings" or "release trains" and more management overhead.*

*Figure 2. feature vs. component teams*

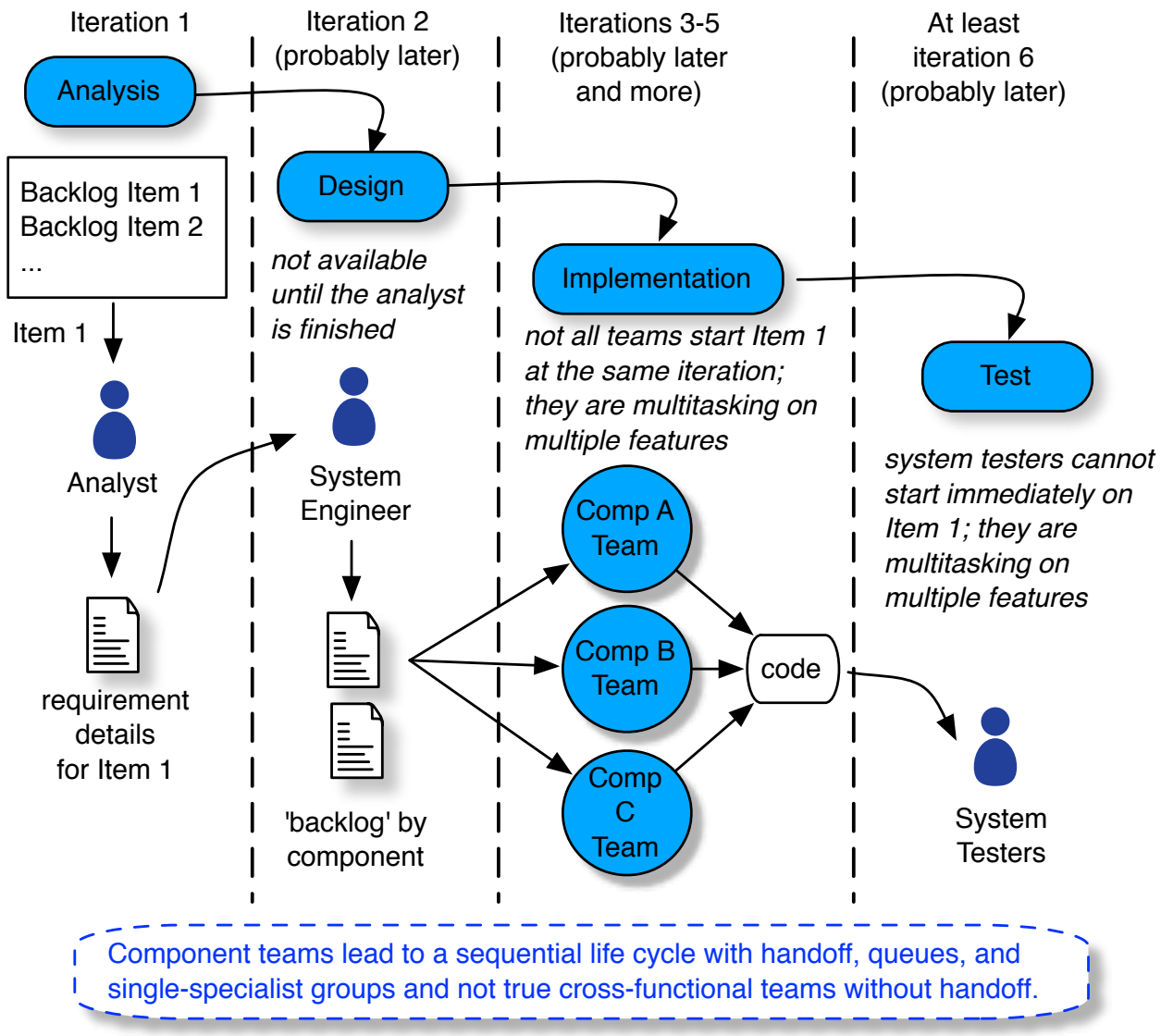



The table below summarizes the differences between feature teams and conventional project or feature groups.

| Feature Team | Feature group of feature project |
|---|---|
| stable team that stays together for years and works on many features | temporary group of people created for one feature or project |
| shared team responsibility for all the work | individual responsibility for 'their' part based on specialization |
| self-managing team | controlled by a project manager |
| results in a simple single-line organization (no matrix!) | results in a matrix organization with resource pools |
| team members are dedicated—100% allocated—to the team | members are part-time on many projects because of specialization |

Most drawbacks of component teams are explored in the "Feature Teams" chapter of Scaling Lean & Agile Development, Figure 3 summarizes some of these.

*Figure 3. some drawbacks of component teams*




What is sometimes not seen is that a component team structure reinforces sequential development (a 'waterfall' or V-model), with many queues with varying-sized work packages, high levels of WIP, many handoffs, and increased multitasking and partial allocation.

## Choose Component Teams or Feature Teams?

A pure feature team organization is ideal from the value-delivery and organizational-flexibility perspective. Value and flexibility, however, are not the only criterion for organizational design, and many organizations therefore end up with a hybrid—especially during a transition from component to feature teams. Caution: hybrid models have the drawbacks from both worlds and can be…painful.

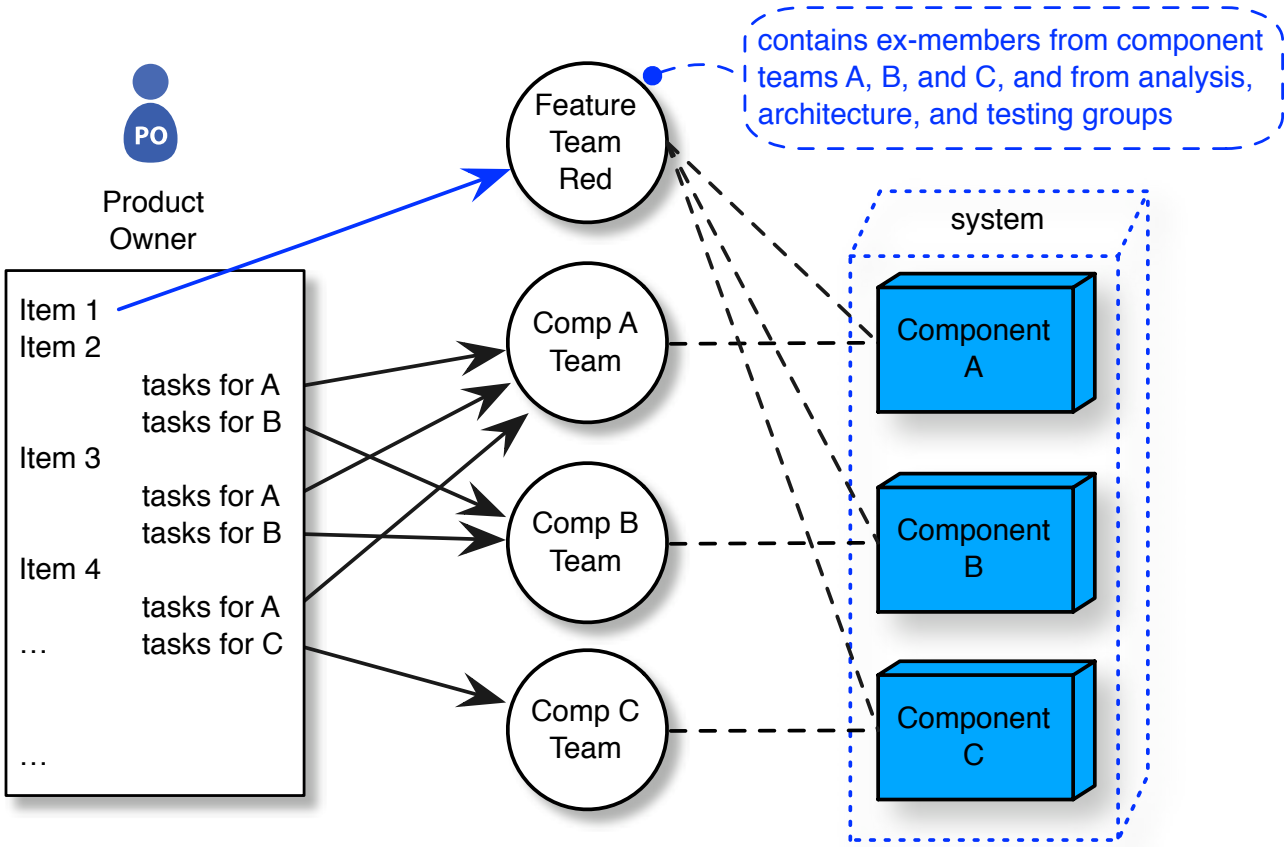
A frequently expressed reason in favor of a hybrid organization is the need to build infrastructure, construct reusable components, or clean up code—work traditionally done within component teams. But these activities can also be done in a pure feature team

organization—without establishing permanent component teams. How? By adding infrastructure, reusable components, or cleanup work to the Product Backlog and giving it to an existing feature team—as if it were a customer-centric feature. The feature team temporarily—for as long as the Product Owner wishes—does such work and then returns to building customer-centric features.

## Transitioning to Feature Teams

Different organizations require different transition strategies when changing from component to feature teams. We have experience with many strategies that worked…and failed in a different context. A safe—but slow—transitioning strategy is to establish one feature team within the existing component team organization. After this team performs well, a second feature team is formed. This continues gradually at the speed the organization is comfortable with. This is shown in Figure 4.

*Figure 4. gradual transitioning from feature to component teams*

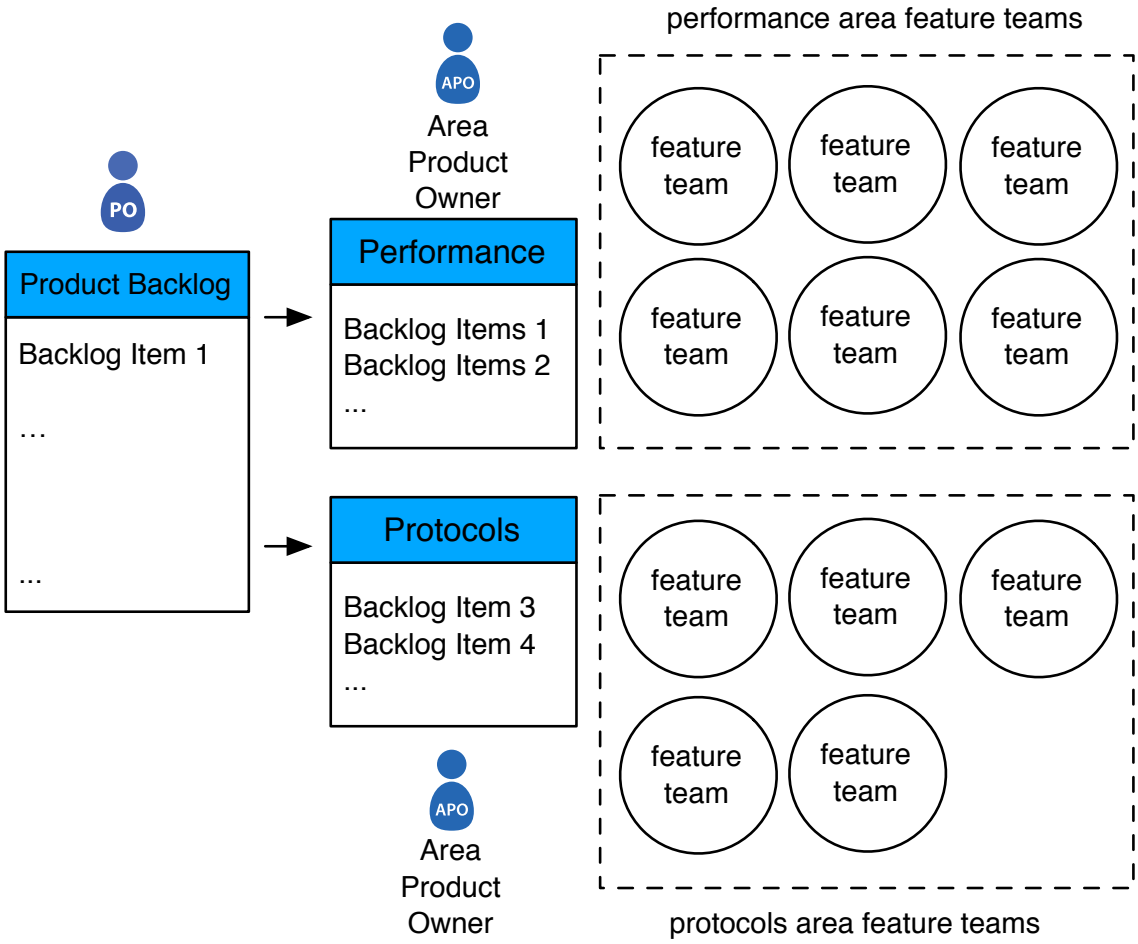


## Introduction to Requirement Areas

Feature teams scale nicely, but when their number goes above ten teams—about a hundred people—additional structure is needed. Requirement areas provide this structure and complement the concepts behind feature teams. A **requirement area** is a categorization of the requirements leading to different views of the Product Backlog.

The Product Owner (PO) groups every Product Backlog item under exactly one requirement category—its requirements area. After this, he generates different views on the overall Product Backlog—called an **Area Backlog**. The Area Backlogs are prioritized by an **Area Product Owner** who specializes in part of the product—from a customer perspective. Each Requirement Area has several feature teams working from the Area Backlog, as shown in Figure 5.

*Figure 5. requirement areas*




Requirement areas are scaled-up feature teams. Scaling up by structuring teams according to the product's architecture is called **development areas**. Table 3 summarizes the differences.

*Table 3. requirement areas vs. development areas*

| Requirement Area | Development Area |
|---|---|
| organized around customer-centric requirements | organized around product's architecture |
| no subsystem code ownership | code ownership per subsystem |
| temporary in nature; should change over the lifetime of the product, but not at every iteration | tends to be more fixed over the lifetime of the product |
| focus on the customer, using customer language | focus on the architecture, using technology language |

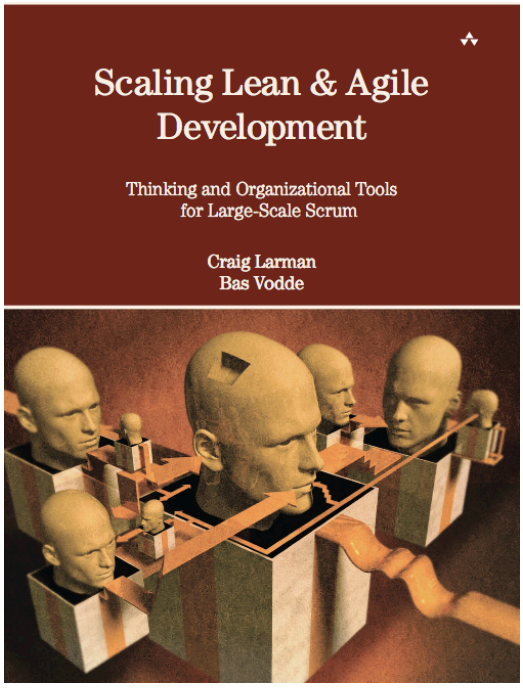
Finally, an Area Product Owner is different than a *supporting* Product Owner—someone that works with one or two teams to help a busy *overall* Product Owner. An Area Product Owner has different responsibilities and focus, and works with (probably) at least *four* teams, not just with one. This avoids local optimization toward the activities of one team.

## Conclusion

Feature teams are stable teams that are given complete customer-centric features. These teams resolve local optimizations and extra coordination overhead caused by component team organizations. However, feature teams are not without challenges themselves.

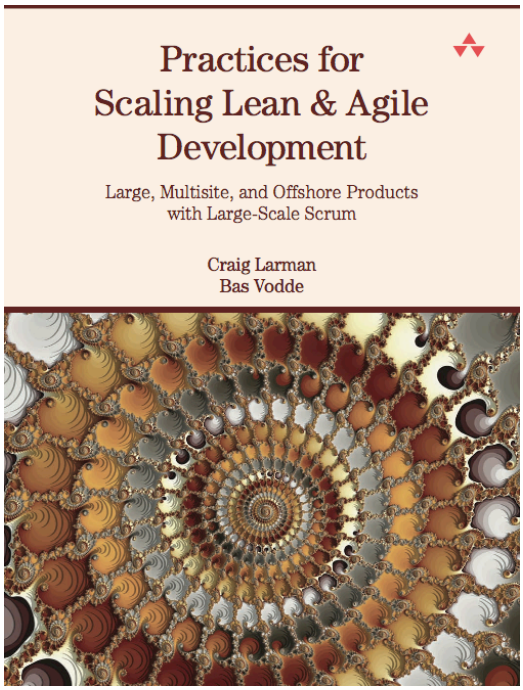
Requirement areas scale the feature team concept by creating customer-centric views on the overall Product Backlog and thus creating a structure that allows feature teams to be scaled up to any size.

## References




**Chapters:**

- Introduction
- Systems Thinking
- Lean
- Queueing Theory
- False Dichotomies
- Be Agile
- Feature Teams
- Teams
- Requirement Areas
- Organization
- Large-Scale Scrum




**Chapters:**

- Large-Scale Scrum
- Test
- Product Management
- Planning
- Coordination
- Requirements
- Design
- Legacy Code
- Continuous Integration
- Inspect & Adapt
- Multisite
- Offshore
- Contracts