

# Азбука Фиче-команд

Крэг Ларман (Craig Larman), Бас Водди (Bas Vodde)

Версия 1.3

**Фиче-команды и Области Требований** – ключевые элементы масштабирования бережливой и гибкой разработки. Они глубоко разбираются в разделе *Feature Team and Requirement Area chapters* книги *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large Scale Scrum*. Этот короткий документ обобщает несколько ключевых идей, которые также можно найти в книге *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*.

## Знакомство с Фиче-командами

**Фиче-команда** (feature team), изображённая на Иллюстрации 1, является долгоживущей<sup>1</sup>, кросс-функциональной и кросс-компонентной командой, которая выполняет от начала и до конца множество задач, описанных пользовательским языком — одну за другой.







Иллюстрация 1. Фиче-команды



Характерные черты фиче-команды перечислены ниже:



<sup>1</sup> Фиче-команды остаются вместе на долгие годы и участвуют вместе во многих задачах.

### Фиче-команда

-  долгоживущая — команда остаётся вместе, чтобы они могли 'созреть' для достижения высокой производительности; со временем они приобретают новые особенности
-  кросс-функциональная и кросс-компонентная
-  в идеале, колоцированная
-  работает над задачей целиком, описанной клиентским языком, до её завершения, над всем затрагиваемыми компонентами и во всех дисциплинах (аналитика, программирование, тестирование, ...)
-  состоит из специалистов широкого профиля
-  как и в Скраме обычно состоит из  $7 \pm 2$  человек

Применение современных инженерных практик — особенно непрерывной интеграции — является самым важным при переходе к фиче-командам. Непрерывная интеграция способствует общему владению кодом, которое необходимо, когда несколько команд работают над одними и теми же компонентами в одно и то же время.

Типичное заблуждение: каждый член фиче-команды должен знать всю систему целиком. Но это не так, потому что

-  Команда целиком — а не каждый член в отдельности — требует всех навыков для реализации от начала и до конца функциональности, ориентированной на клиента. Они включают в себя знания и функциональные навыки, такие как тестирование, проектирование взаимодействия или программирование. Но внутри команды люди по-прежнему специализируются... желательно в нескольких областях.
-  Элементы Бэклога Продукта не распределяются между командами произвольно. Текущие знания и навыки команды учитываются в принятии решения, какая команда будет работать над какими задачами.

В организации, состоящей из фиче-команд, когда специализация становится ограничением... происходит обучение.

Организация, состоящая из фиче-команд, использует преимущества в скорости от специализации, если требования покрываются навыками команд.

Требования, которые не соответствуют навыкам команд, 'форсируют' обучение, разрушая ограничения чрезмерной специализации.

Фиче-команды сочетают в себе специализацию и гибкость.

Таблица 1 и Иллюстрация 2 показывают отличия фиче-команд и более традиционных компонентных команд.

Таблица 1. Фиче-команды против компонентных команд

Фиче-команда	Компонентная команда
оптимизирована для поставки максимальной клиентской ценности <sup>А</sup>	оптимизирована для поставки максимального количества строк кода
фокус на наиболее ценных для продукта функциях и на продуктивности системы (пропускной способности поставки ценности)	фокус на увеличении индивидуальной продуктивности, реализуя 'простые' функции с более низкой ценностью
несёт ответственность за всю целиком функциональность, ориентированную на клиента	несёт ответственность только за свою часть общей функциональности, ориентированной на клиента
'современный' подход организации команд <sup>Б</sup> — избегает Закона Конвея	традиционный подход организации команд — следует Закону Конвея <sup>В</sup>
ведёт к фокусу не клиенте, прозрачности, и небольшим организациям	ведёт к 'выдуманной' работе и бесконечно растущей организации
минимизирует зависимости между командами, увеличивая гибкость	зависимости между командами ведут к дополнительному планированию <sup>Г</sup>
фокус на нескольких специализациях	фокус на одной специализации
общее владение кодом продукта	индивидуальное/командное владение кодом
общая командная ответственность	чёткие индивидуальные обязанности
поддерживает итеративную разработку	выражается в 'каскадной' разработке
использует гибкость; непрерывное и широкопрофильное обучение	использует существующую экспертизу; низкий уровень изучения новых навыков
требует развитых инженерных практик — их результаты широко заметны	использует небрежные инженерные практики — результаты имеют локальный эффект
мотивирует писать легко поддерживаемый и тестируемый код	вопреки убеждению часто ведёт к низкому качеству кода в компоненте
внедрить, по-видимому, тяжело	внедрить, по-видимому, легко

**А** - Отличия в оптимизационной цели часто дают ощущение невысокой скорости фиче-команд – с локальной точки зрения **Б** - Относительно 'современные' фиче-команды имеют длинную историю в масштабной разработке, например, в Майкрософт and Эрикссон. **В** - Мэл Конвей наблюдал нежелательные структуры в 1968, он не рекомендовал их—по факту, совершенно наоборот. **Г** –

[www.featureteamprimer.org](http://www.featureteamprimer.org)

Copyright (c) 2010 Craig Larman and Bas Vodde  
All Rights Reserved

Дополнительное планирование отражается в большем количестве “встреч по планированию релиза” или “релизных поездов” и лишней работе менеджменте.

Иллюстрация 2. Фиче-команды против компонентентных

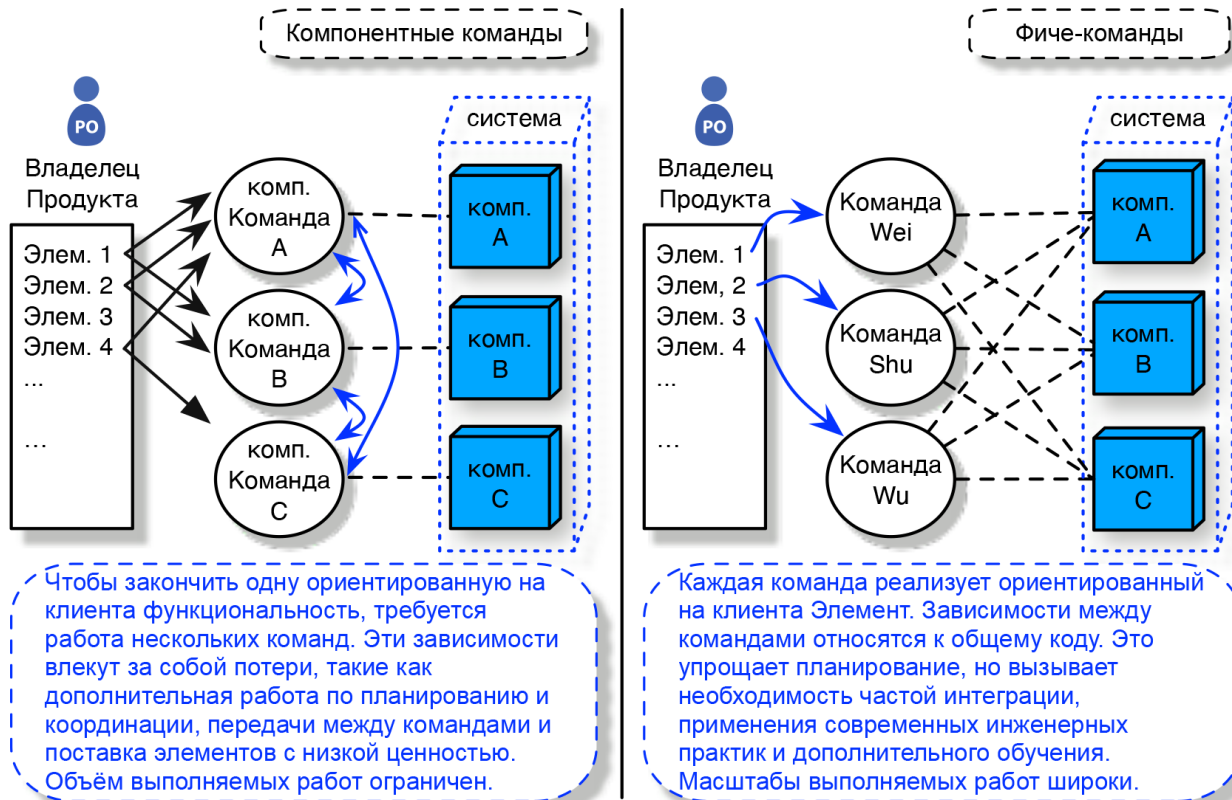
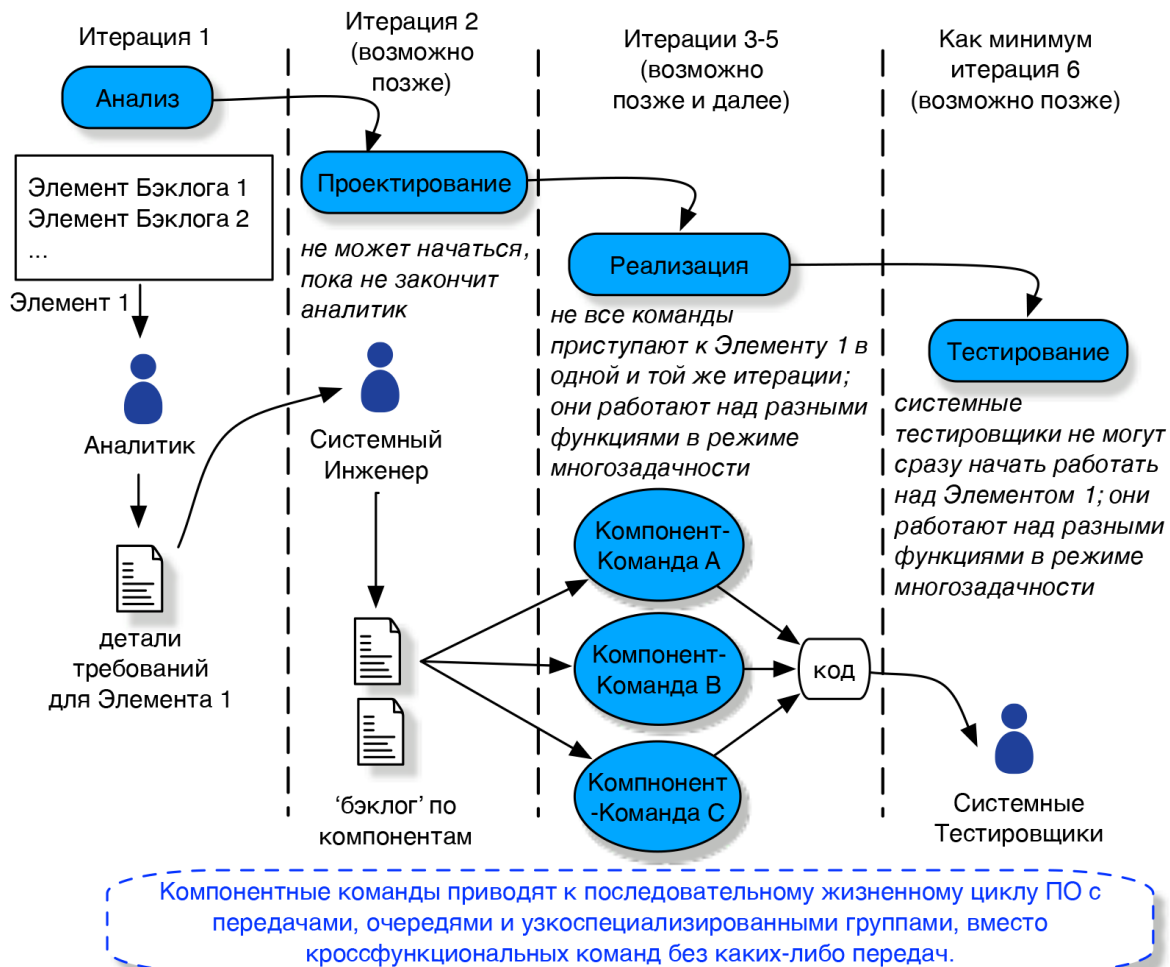


Таблица ниже подводит итоги в сравнении фиче-команд и традиционных проектных или специализированных групп.

Фиче-команда	Специализированная группа или проект
стабильная команда, остающаяся вместе на протяжении многих лет и разрабатывающая совместно большое количество различной функциональности	временный состав людей, созданный для работы над одной задачей или проектом
общая ответственность за всю работу	индивидуальная ответственность за 'их' часть работы, основанной на их специализации
самоуправляемая команда	контролируется руководителем проекта
приводит к простой плоской организации (без матриц!)	приводит к матричной организации с пулами ресурсов
члены команды выделены — на 100% — в команду	члены выделены частично на несколько проектов на основе их специализации

Большинство недостатков компонентных команд описаны в разделе “Feature Teams” книги “Scaling Lean & Agile Development”, на Иллюстрации 3 ниже можно увидеть некоторые из них.

Иллюстрация 3. Некоторые недостатки компонентных команд



Иногда этого не видно, но структура компонентных команд усиливает модель последовательной разработки ('водопад' или V-модель), с большим количеством очередей с разноразмерными пакетами работ, высоким уровнем НЗР, множественными передачами, повышением многозадачности и частичным выделением людей на проекты.

### Что выбрать: Компонентные Команды или Фиче-команды?

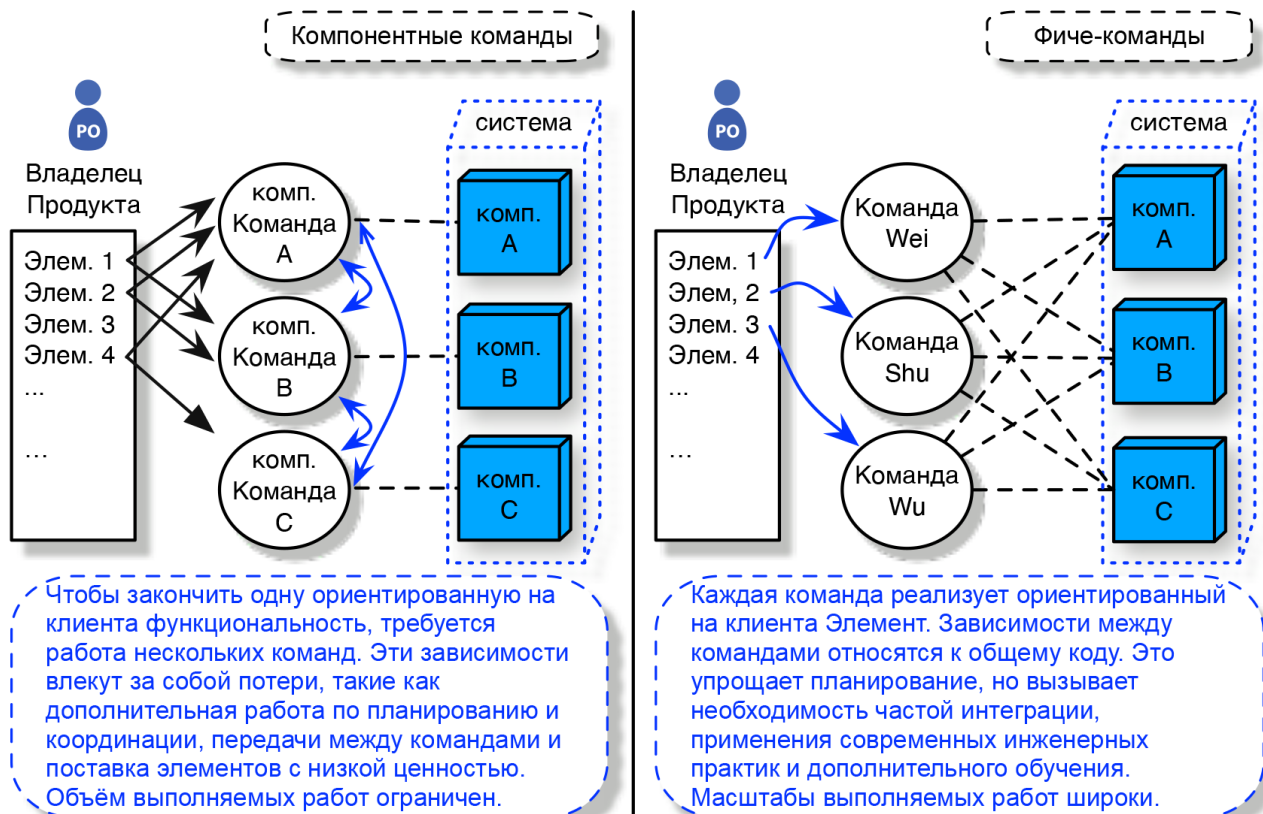
Организация, состоящая из фиче-команд, является идеалом с точки зрения поставки ценности и организационной гибкости. Ценность и гибкость, однако, не единственные критерии организационного дизайна, и следовательно, многие организации останавливаются на гибриде — особенно в процессе перехода от компонентных к фиче-командам. Внимание: гибридные модели имеют недостатки обоих миров и могут приносить... боль.

Часто высказываемая причина в пользу гибридной организации — это необходимость создания инфраструктуры, создания повторно используемых компонентов или улучшению кода — работа, традиционно выполняемая в компонентных командах. Но эти задачи могут быть сделаны в организации, состоящей только из фиче-команд — без создания постоянных компонентных команд. Как?. Путём добавления задач по инфраструктуре, повторно используемым компонентам или улучшению кода в Бэклог Продукта и передачи их существующим фиче-командам — так, как если бы они были бы задачами, ориентированным на клиента. Фиче-команда временно — на столько долго, на сколько Владелец Продукта захочет — выполняет такую работу и затем возвращается к работе над задачами, ориентированными на клиента.

### Переход к Фиче-командам

Разные организации требуют разной стратегии по переходу от компонентных к фиче-командам. Мы имеем опыт во многих стратегиях, которые работали... и проваливались в зависимости от контекста. Безопасная — но долгая — стратегия перехода состоит в запуске одной фиче-команды среди других компонентных команд. После того, как она будет чувствовать себя хорошо, можно приступить к запуску второй фиче-команды. Это продолжается постепенно со скоростью, устраивающей организацию. Это изображено на Иллюстрации 4.

Иллюстрация 4. Постепенный медленный переход к фиче-командам



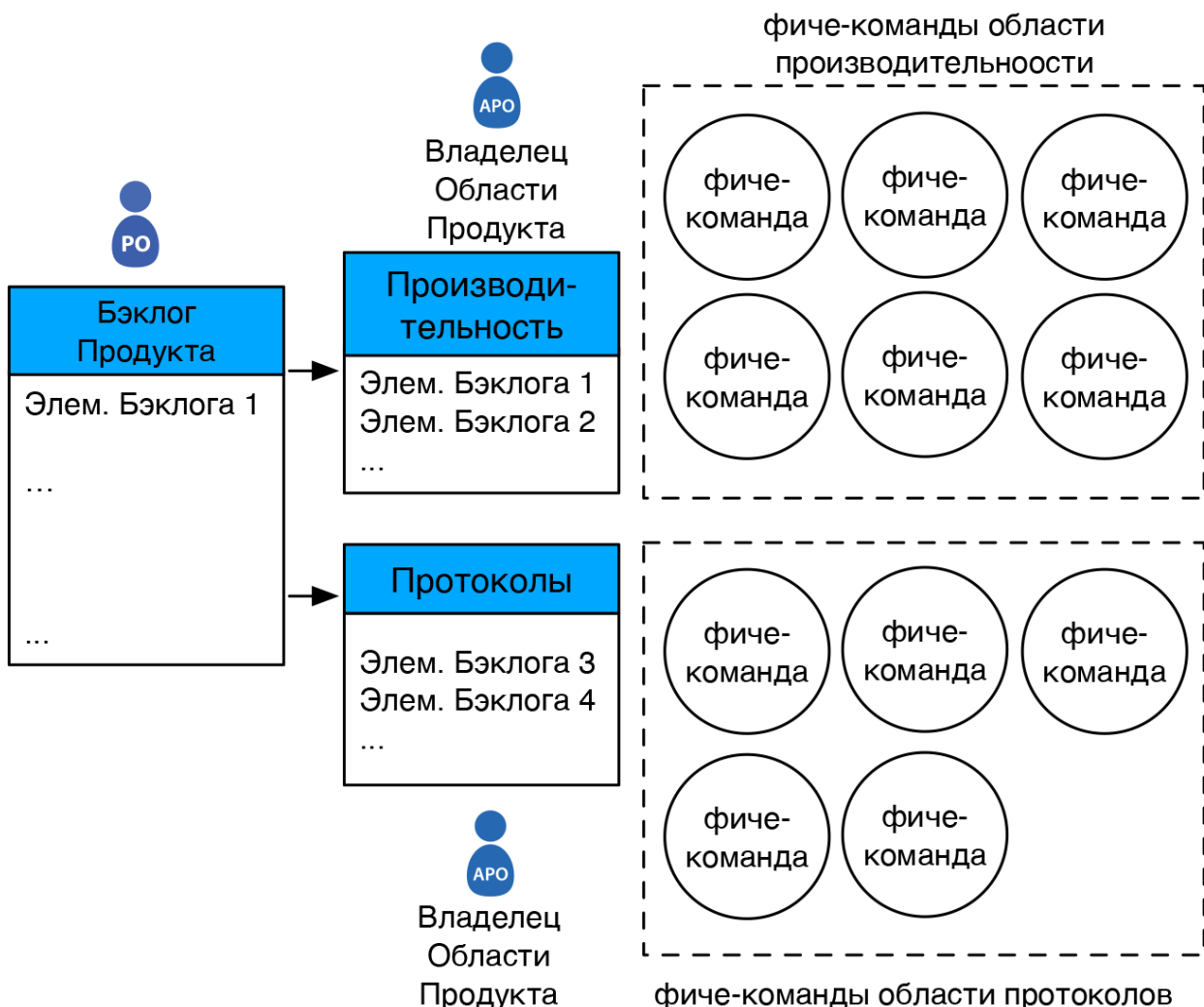


## Знакомство с Областями Требований

Фиче-команды хорошо масштабируются, но когда их количество превышает восемь команд, требуется дополнительная структура. Области Требований являются такой структурой и дополняют концепцию фиче-команд. **Область Требований** (Requirement Area) - это деление требований на категории, приводящее к другому представлению Бэклога Продукта.

Владелец Продукта группирует каждый элемент Бэклога Продукта строго в одну категорию требований - Область Требований этого элемента. Таким образом он формирует на основе всего Бэклога Продукта представления различных его частей, называемых **Бэклогами Областей** (Area Backlog). **Владелец Области Продукта** (Area Product Owner), специализирующийся на этой части продукта, определяет приоритетность элементов в Бэклоге Области с точки зрения клиентов. Каждая Область Требований имеет несколько фиче-команд, работающих над Бэклогом Области, как показано на Иллюстрации 5.

Иллюстрация 5. Области Требований



Области Требований - это масштабированные фиче-команды. Масштабирование путем структурирования команд в соответствии с архитектурой продукта называется **Областями Разработки** (Development Areas). В Таблице 3 приведены различия.

Область Требований	Область Разработки
организована вокруг, ориентированных на заказчика, требований	организована вокруг части архитектуры продукта
коллективное владение кодом всех подсистем	закрепленное владение кодом на уровне отдельных подсистем
временны по природе; должны меняться на протяжении всей жизни Продукта, но не каждую итерацию	стремятся быть фиксированными на протяжении всей жизни продукта
фокусируют внимание на заказчике, используя язык заказчика	фокусируют внимание на архитектуре, используя технический язык

Наконец, Владелец Области Продукта отличается от *поддерживающего* Владельца Продукта - человека, который работает с одной или двумя командами, чтобы помочь постоянно занятому Владельцу Продукта. Владелец Области Продукта имеет разные обязанности и разные цели, и он работает (вероятно) как минимум с четырьмя командами, а не только с одной. Это позволяет избежать локальной оптимизации в отношении деятельности одной команды.

### Заключение












Фиче-команды - это стабильные команды, которые делают всю работу в задачах, ориентированных на клиента. Эти команды устраняют локальную оптимизацию и дополнительные затраты на координацию, вызванные организацией компонентных команд. Тем не менее, фиче-команды не лишены проблем.

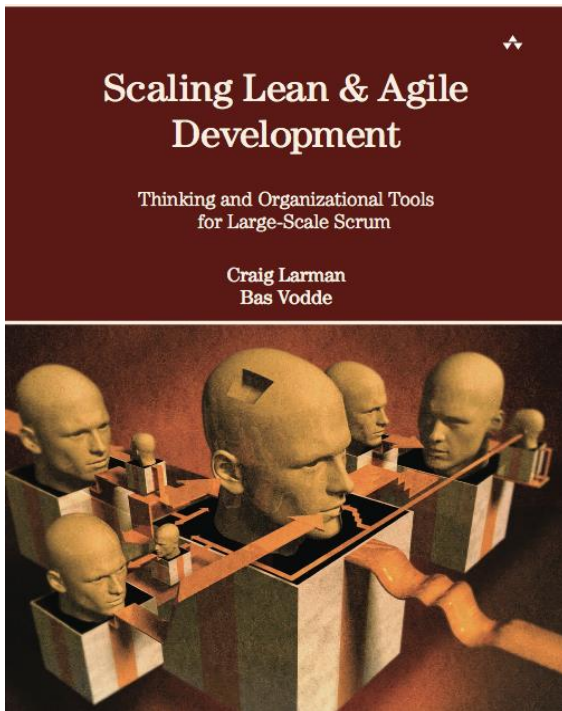
Области требований масштабируют концепцию фиче-команд, создавая ориентированные на клиента представления общего Бэклога Продукта и, таким образом, создавая структуру, которая позволяет масштабировать фиче-команды до любого размера.
















## Ссылки

### Главы:

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



### Главы:

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

