Scaling Scrum with Feature Teams
Agenda

Introduction
Before we start -> Some basics
Feature teams and component teams
Introduction
バスはどれでしょう？

or 八斯是谁？
Scaling Lean & Agile Development

Thinking and Organizational Tools for Large-Scale Scrum

Craig Larman
Bas Vodde
Some basics
Scrum
Continuous Integration

Continuous Integration is a developer practice with the goal to always keep a working system by making small changes, slowly growing the system and integrating them at least daily on the mainline typically supported by a CI system with lots of automated tests.
Scaling CI system
Large-scale setup
Feature teams
Conway’s law

Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure.
And...

Because the design that occurs first is almost never the best possible, the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design.

- Mel Conway
One Product Owner

Multiple Teams

Teams own a part of the system:

“Component teams”
Low value work is implemented

Everybody always busy?
“Work gets created”

Large systems... grow larger by default

Re write
Re Design
Re Architecture
One requirement does not map to one team

Dependencies never balance out

Result: Not complete requirements integrated
Assign a problem to a role

Impossible job, requirements never balance out.

Result: priority and resource fights
Large backlog items must be split in “less customer-centric backlog items”
Splitting before the iteration starts: “Architecture”

Testing after the iterations ends: “System test”
How to become good? ...
One Product Owner

3 Teams
Give complete requirements to teams: “Feature teams”

All dependencies within the team
Feature Teams

- long-lived—the team stays together so they can ‘jell’ for higher performance; they take on new features over time
- cross-functional and co-located
- work on a complete customer-centric feature, across all components and disciplines
- composed of generalizing specialists
New problem:
Dependency moved
Modern version control (e.g. svn)
Continuous integration development practice
Automated build and test
Person specialization

ABC
CDE
ABEF
Team specialization

- OK
- ABC
- ABC
- CDE
- ABEF
Team specialization
Specialization good

Don’t let specialization constrain you

Learn new specializations
Emergent design

Component guardians
Community of Practice

Architect Facilitator

Same for e.g. test, ScrumMasters
Transition can often be done by reforming teams.
What about large product development?
Always have one product owner and one product backlog per product

Or... a group of products...
Group requirements into “categories” called: “Requirement areas”

Grouping based on customer, NOT on architecture
Create “requirement area backlogs”

RA backlog is a view on the product backlog

Every PBI maps always to exactly one RA backlog
Every RA has their own “area product owner”

RA product owner specializes in “customer-centric domain”
Every RA has a set of feature teams

From 5-10 per RA

Teams specialize in that area

Areas are dynamic over time
Overall PO decides on moving teams between areas

Value vs velocity
Transition strategy
“Development areas” are groupings based on architecture.

Helps transition, has all drawbacks of component teams.
Questions?