The Trouble with “Component Teams” and alternative: “Feature Teams” or “Scaling Scrum”
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 ProductOwner
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
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 ProductOwner

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
Team specialization

0
ABCD

TEAM
ABC

CDE

ABEF

Team specialization

0
ABCD

TEAM
ABCD

CDE

ABEF
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?