
Less



Amsterdam - 11 Oct and 6 Dec

Munich - 12 Dec

LARGE-SCALE SCRUM CONFERENCE

Embrace Agility

27–28 September 2023 | Berlin



Gojko Adzic | Bjarte Bogsnes | Jutta Eckstein | Craig Larman | Bas Vodde

#LeSS2023
http://less2023.works

LeSS Courses
LeSS Supporting Courses
LeSS Events
LeSS-Friendly Scrum Course
LeSS Conferences
Becoming a LeSS Trainer
Becoming a LeSS-Friendly Scrum Trainer

Summary Upload Contacts Edit Admin Actions

Maximizing Dependencies with Interdependent Teams, with Bas Vodde

(Online) New York Link Type: Meet-up Date: August 11, 2023 (1) Time: 12:00 ~ 13:00

By: Gene Gendel Language: English

Share Tweet

Description:

Synopsis:

To get Agile to work, we need to reduce dependencies to create independent teams. That is common sense! And it is wrong.

When multiple teams work on one product, the teams need to work closely together to ensure it truly becomes one product. Teams need to learn from each other, help each other, and together build one product. They need to have dependencies with each other, a lot of them! However, many people and teams have bad experiences with dependencies. In their experience, the dependencies blocked them from making progress. It doesn't have to be that way!

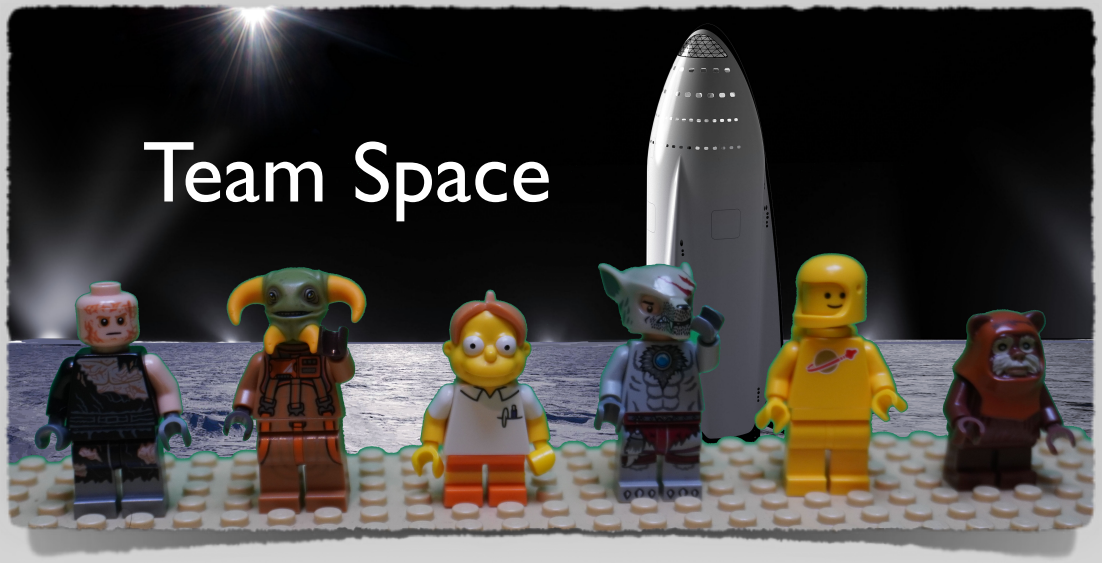
So, how can teams work together and what enables this?

Speaker's Bio:

Bas Vodde is a coach, programmer, trainer, and author related to modern agile and lean product development. He is the creator of the LeSS (Large-Scale Scrum) framework for scaling agile development. He coaches organizations on three levels: organizational, team, individual/technical practices. He has trained thousands of people in software development, Scrum, and modern agile

Maximizing Dependencies with Interdependent Teams




Team Space


Team Lake


Team Halloween


Team Criminals


Team Power
POLICE
千葉県茂原警察署 茂原駅前交番

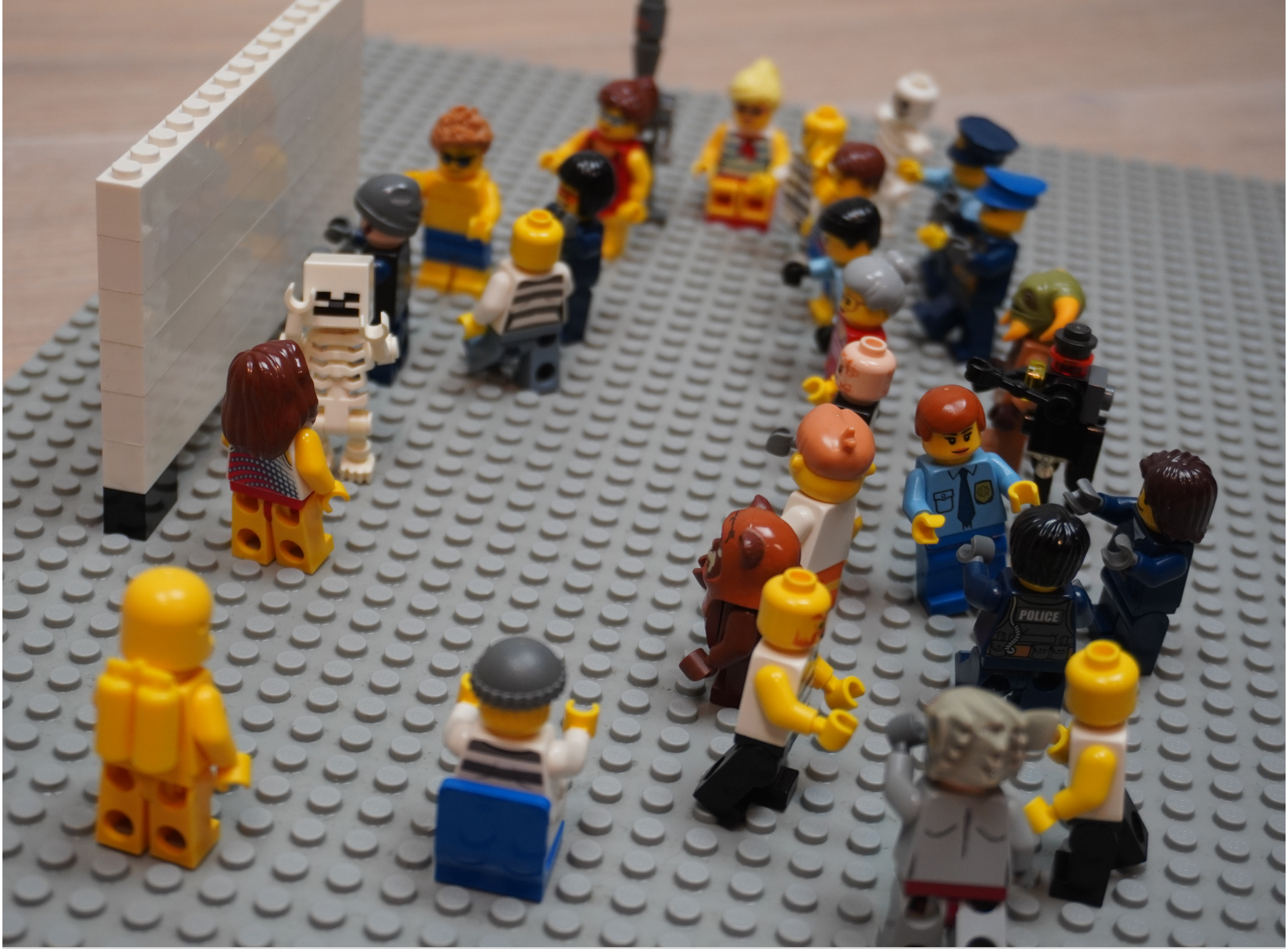
Features

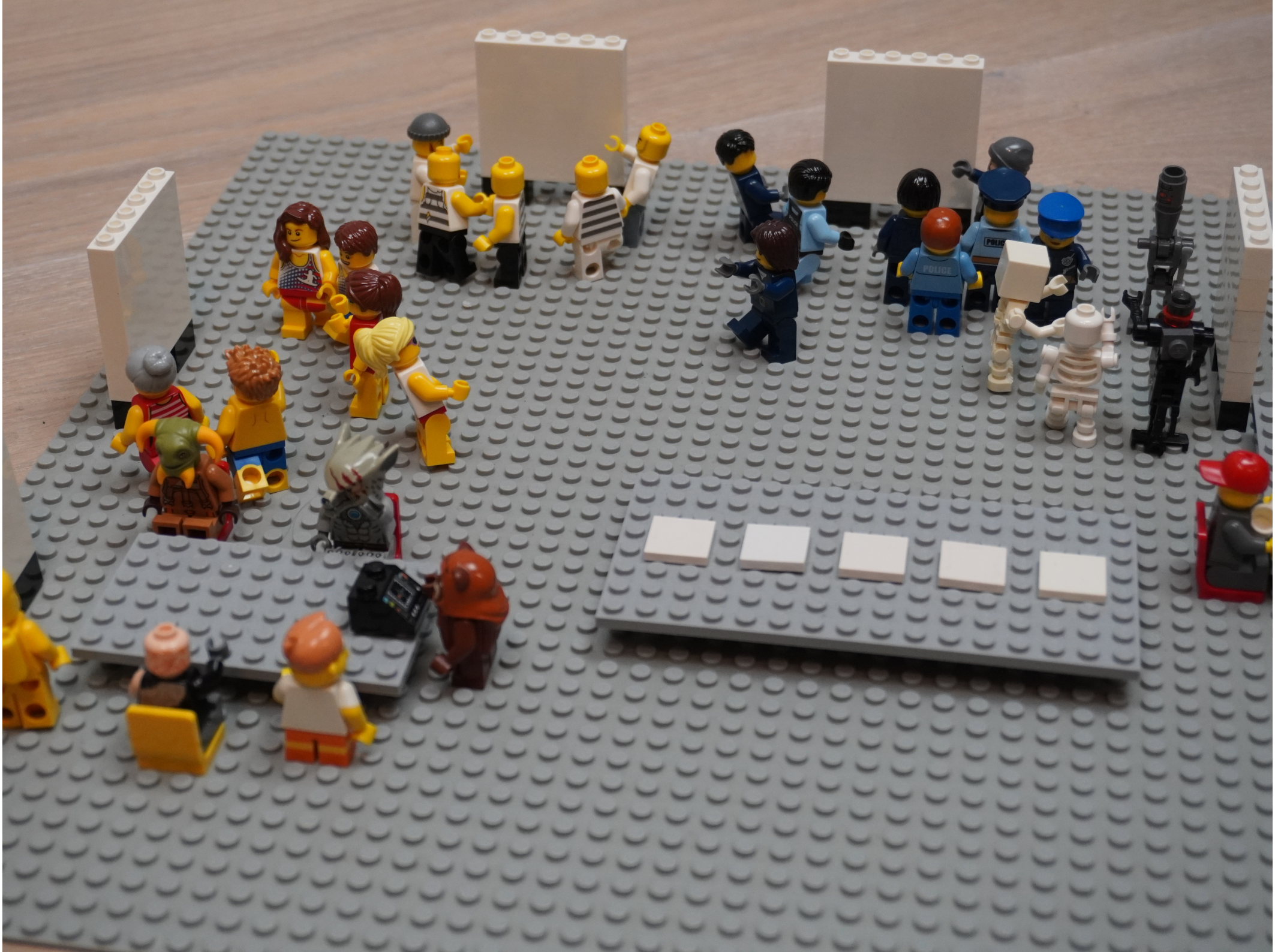


Top of Product Backlog

Viewing courses with direct link
Course registration
Basic integration with External Scheduler
Migrate main page to New Tech
Course registration with invoice
External Scheduler support for timezones
Migrate menus to New Tech
Course registration through payment provider
Cancelling registration without payment


POLICE
POLICE


POLICE

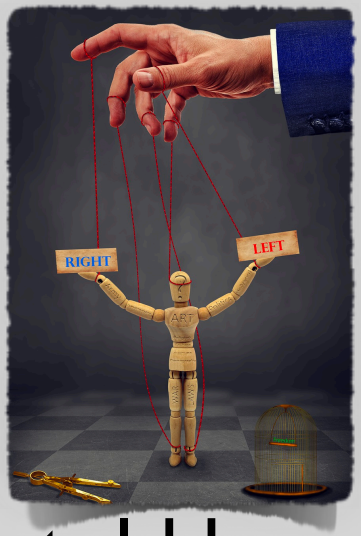

POLICE
POLICE


POLICE

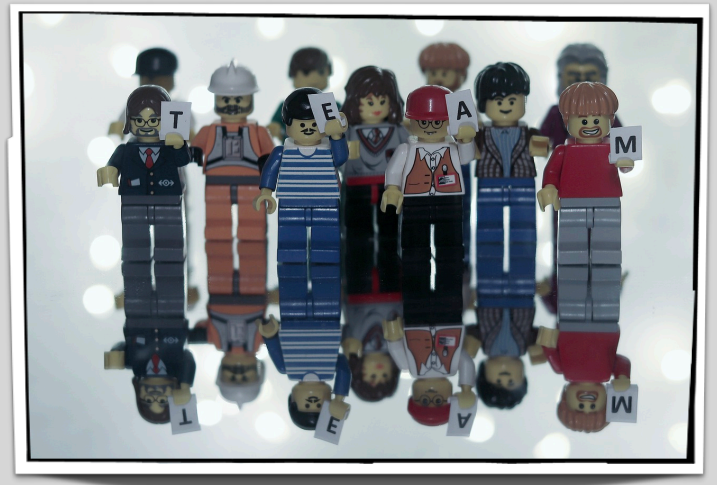

POLICE
POLICE

Enablers

Two Journeys



Being told how to work



Self-managing Team



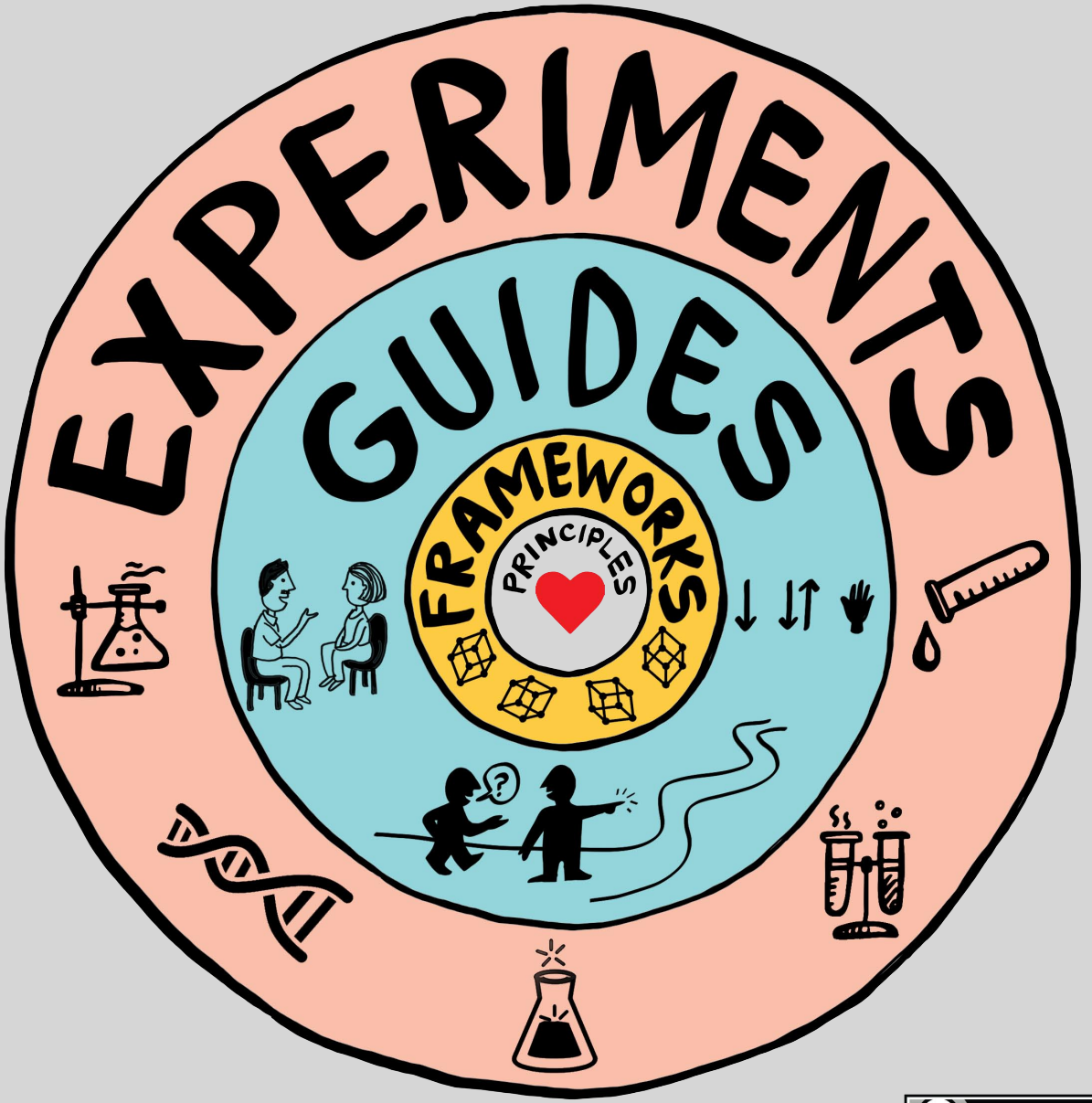
Programmer



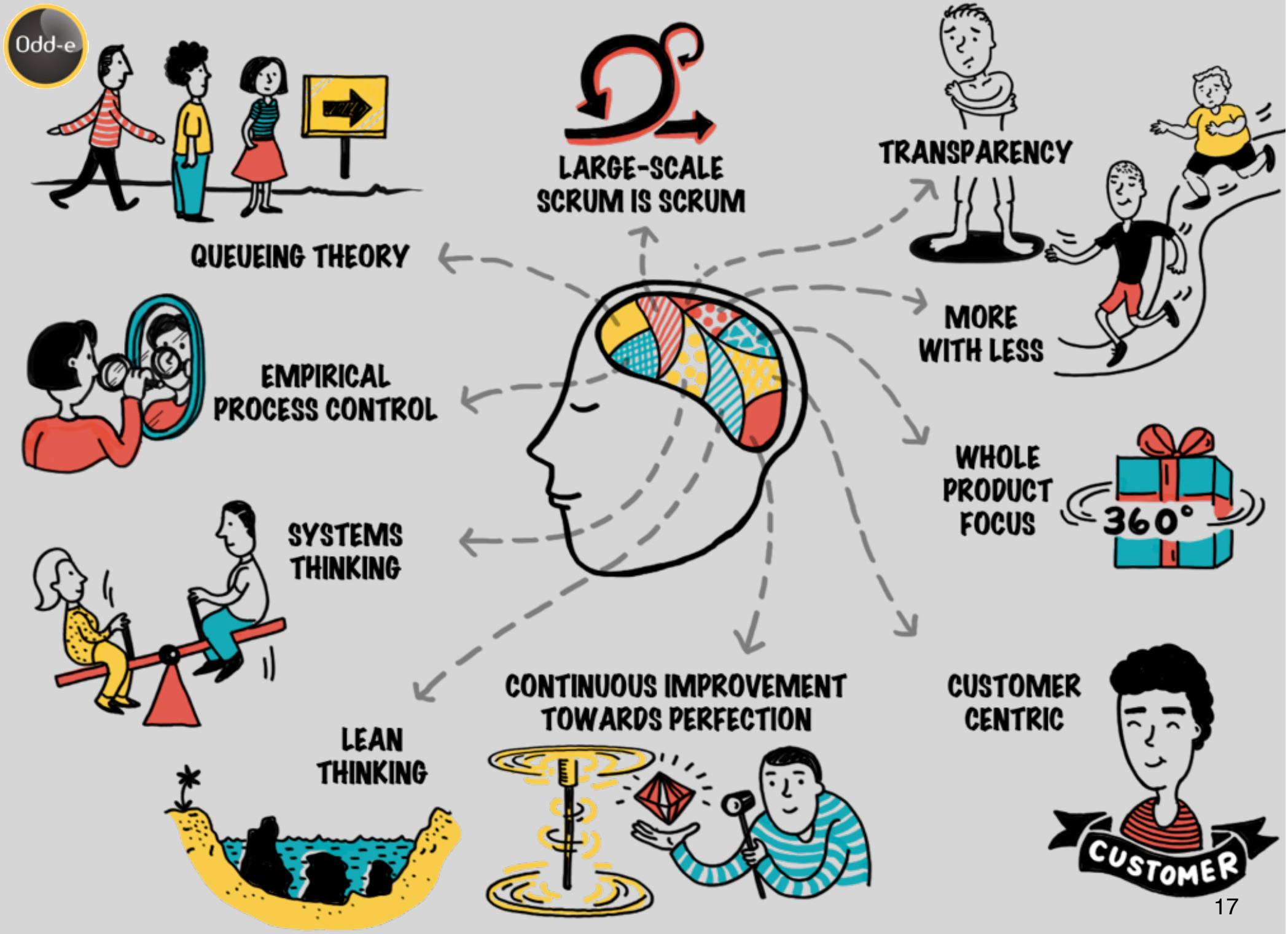
Product Developer

Odd-e

LeSS



BY-NC-ND


Odd-e
QUEUEING THEORY
LARGE-SCALE
SCRUM IS SCRUM
TRANSPARENCY
MORE
WITH LESS
EMPIRICAL
PROCESS CONTROL
WHOLE
PRODUCT
FOCUS
360°
SYSTEMS
THINKING
CONTINUOUS IMPROVEMENT
TOWARDS PERFECTION
CUSTOMER
CENTRIC
CUSTOMER
LEAN
THINKING

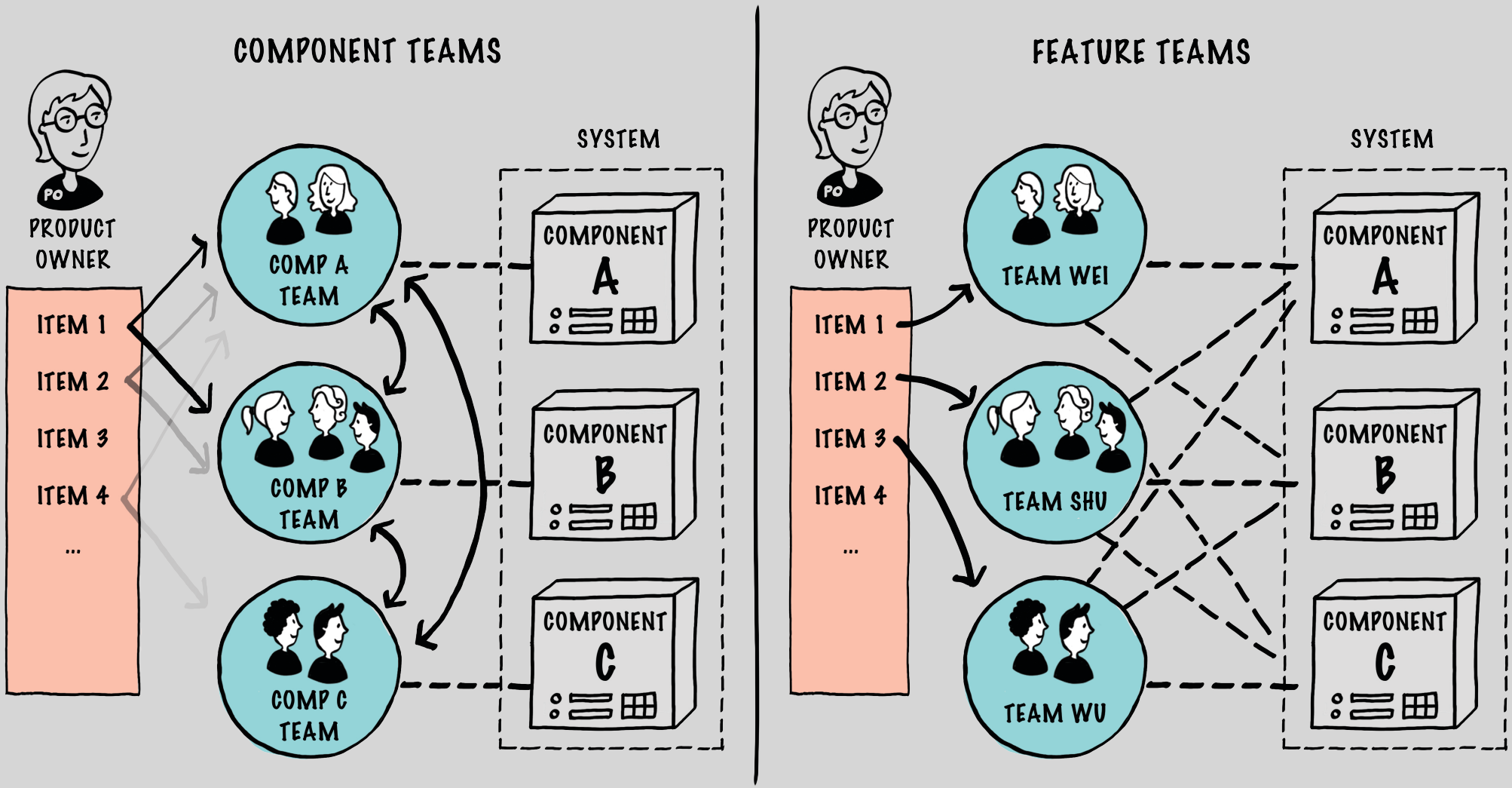
Whole Product Focus



Customer Centric



THE RITZ-CARLTON


COMPONENT TEAMS
PO
PRODUCT OWNER
ITEM 1
ITEM 2
ITEM 3
ITEM 4
...
COMP A TEAM
COMP B TEAM
COMP C TEAM
SYSTEM
COMPONENT A
COMPONENT B
COMPONENT C
FEATURE TEAMS
PO
PRODUCT OWNER
ITEM 1
ITEM 2
ITEM 3
ITEM 4
...
TEAM WEI
TEAM SHU
TEAM WU
SYSTEM
COMPONENT A
COMPONENT B
COMPONENT C

Types of Dependencies



By Bundesarchiv, Bild 183-1990-0414-009 / Wolfried Pätzold / CC-BY-SA 3.0, CC BY-SA 3.0

Asynchronous Dependencies
Interrupt

vs

Synchronous Dependencies
Collaborate



Maximizing Dependencies



Dependent on

vs

Shared Work



Interdependent Teams



Independent
Isolated
Teams

vs

Cross-Team Shared Learning



Feature Teams together
chose which team works on
which feature
rather than
all work for a specific
component always **goes to**
a specific team

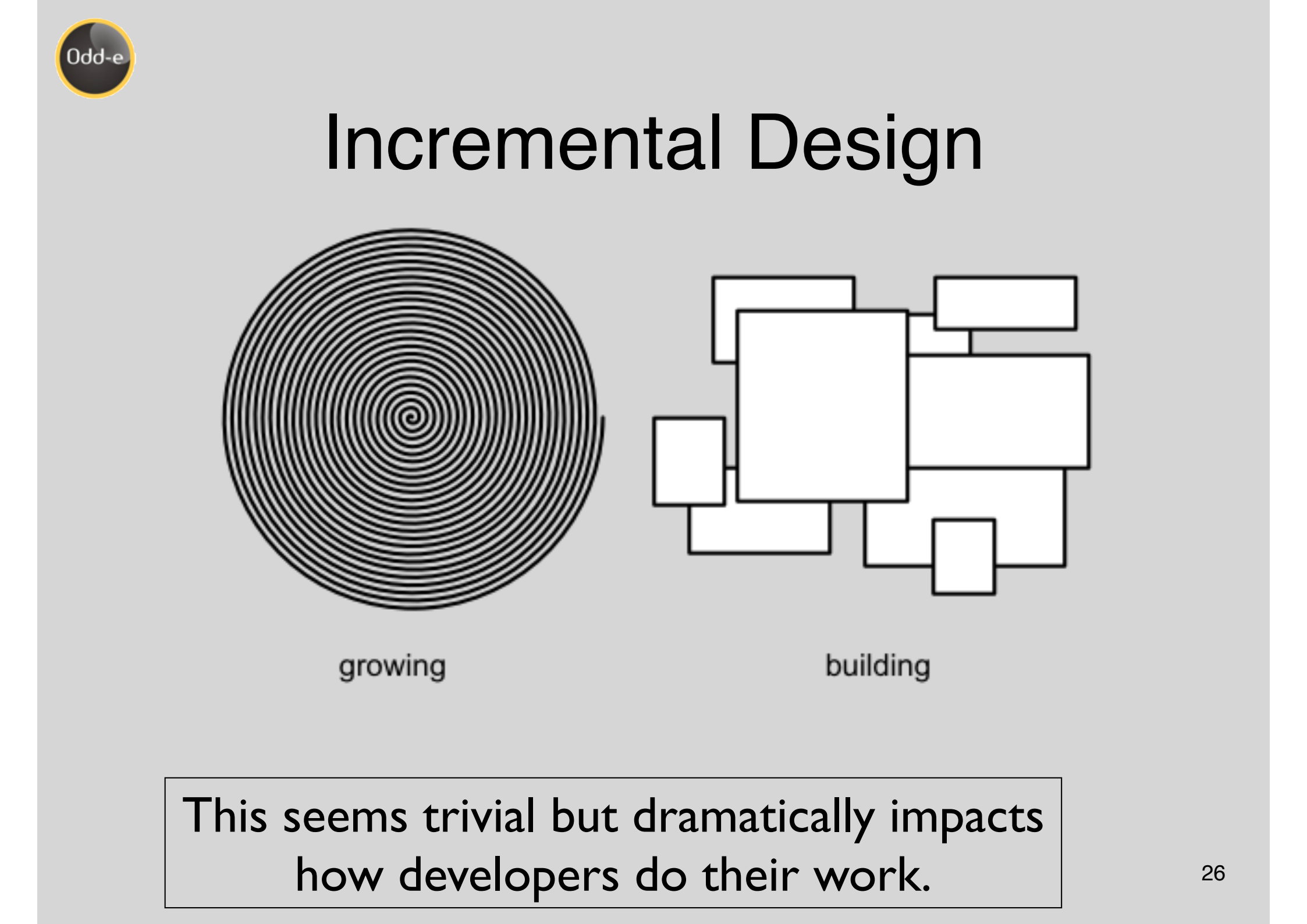
Accidental Specialization



Example

Team HUB likes:

- Features that are complex and touch many components
- Features that are used by lecturers and students
- Features that have a graphical UI



Incremental Design

growing

building

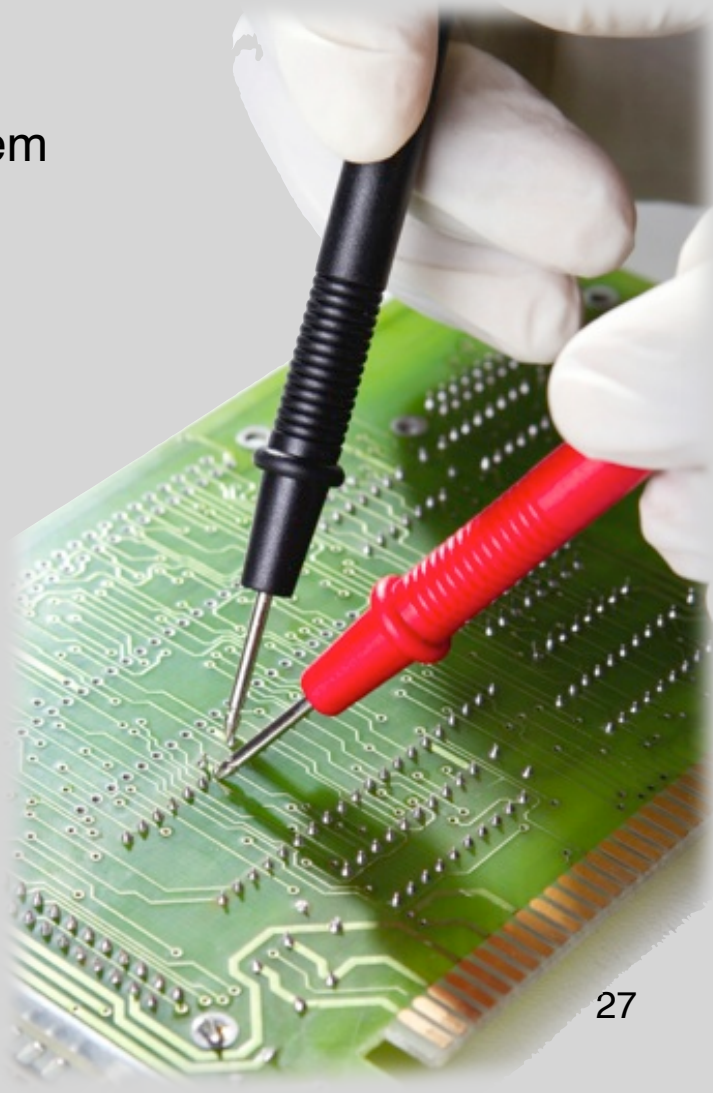
This seems trivial but dramatically impacts how developers do their work.



Continuous Integration

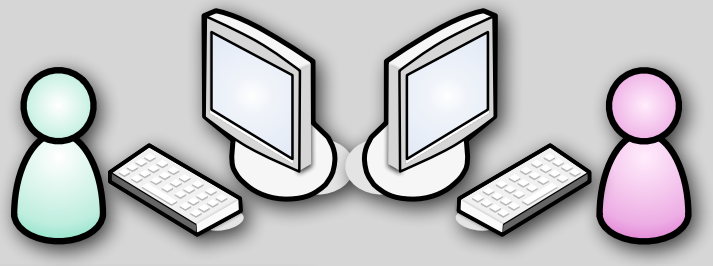
Continuous Integration is a **developer practice**
with the goal to always keep a **working system**
by making **small changes**, slowly growing the system
and **integrating** them at least **daily**
on the **mainline**
typically supported by a **CI system**
with lots of **automated tests**

- Increases transparency
- Increases cooperation and communication
- Enables people to work on same code



Odd-e

Scenario



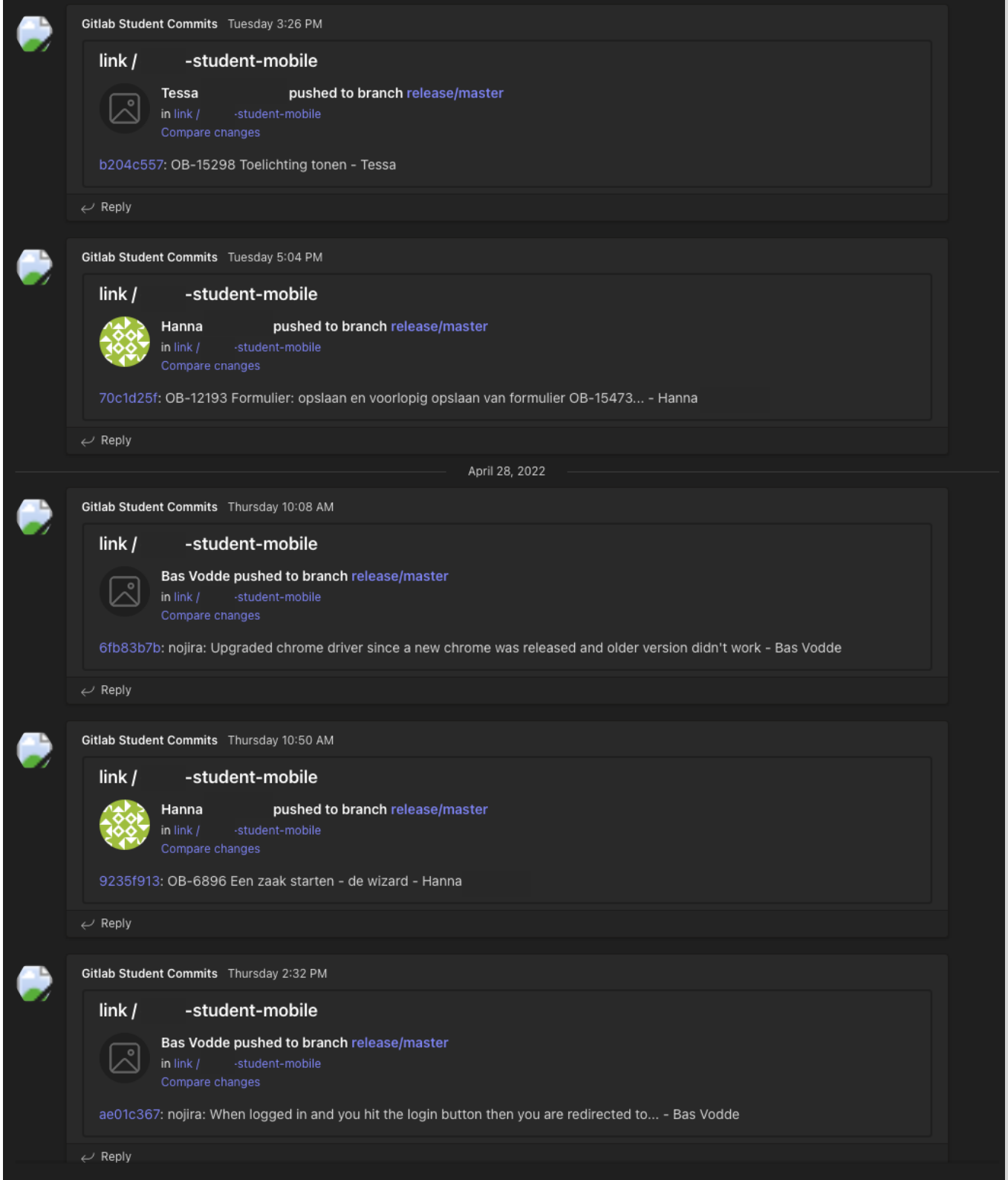
Hey! I'm changing the method that conflicts with your changes!

Roger!

... done and committed!

Ok! I'll update and check the merged changes!
... Looks good!

Great!



Communicate in Code



Avoid branching!



Mono Repository

contributed articles

DOI:10.1145/2854146

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

BY RACHEL POTVIN AND JOSH LEVENBERG

Why Google Stores Billions of Lines of Code in a Single Repository

This article ou
codebase and de
built monolithic
the reasons the
Google uses a ho
trol system to ho
visible to, and us
ware developers
centralized syste
many of Google'
Here, we provid
systems and wor
sible managing
tively with such
explain Google'
opment" strateg
tems that structu
Google's codeba
software for stati
up, and streamli

Google-Scale

Google's monol
tory, which is us
ware developer
the definition o
system, providi
gle-source repo
scaled successfu

The Google c
proximately one
a history of app
commits spanni
year existence. T
86TB[a] of data, in

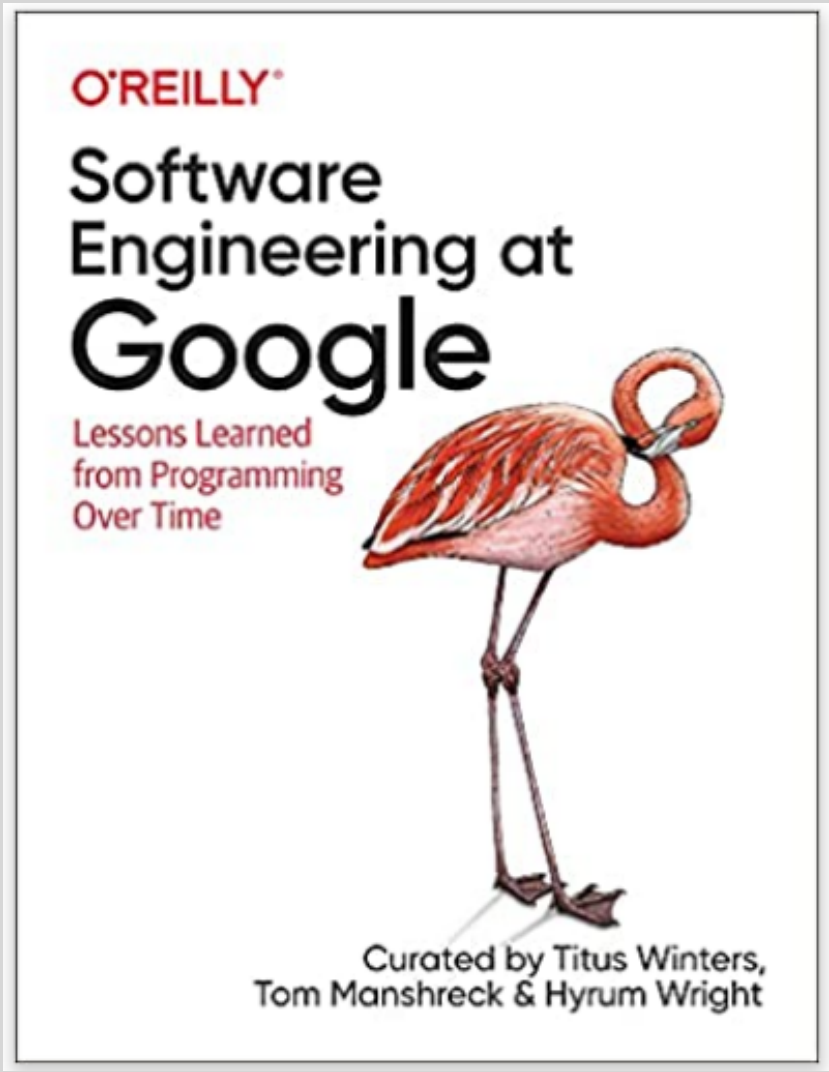
a Total size of uncon
release branches.

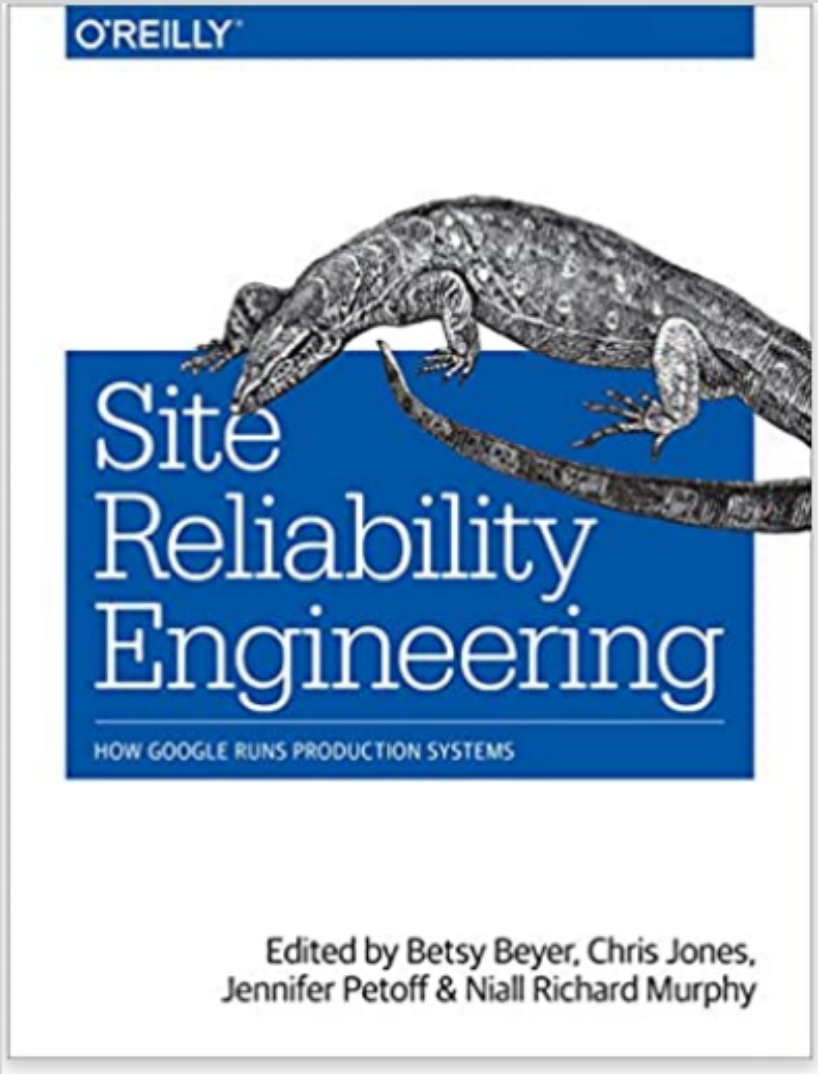
Maximizing Dependencies with Interdependent Teams



31 January 2024 - Tokyo

Google


O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
Curated by Titus Winters, Tom Manshreck & Hyrum Wright

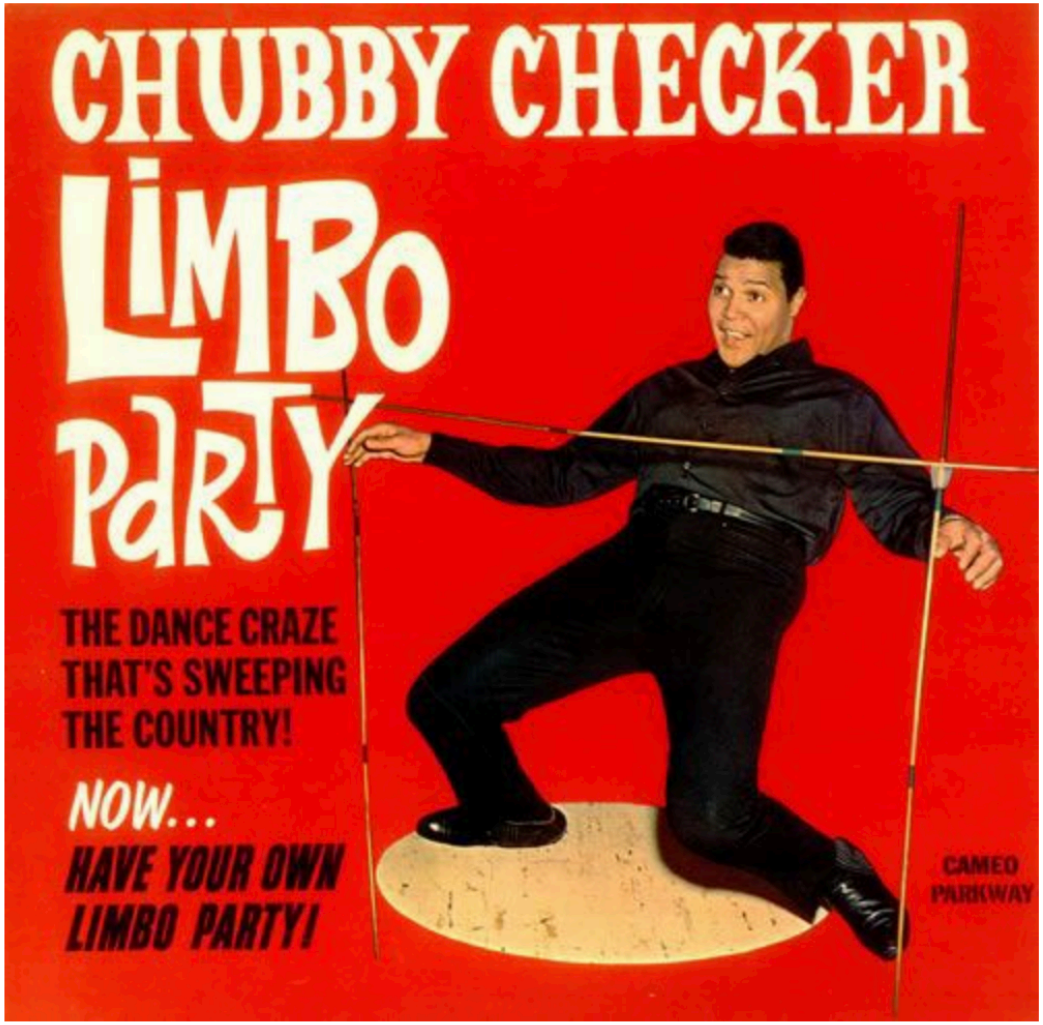

O'REILLY
Site Reliability Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS
Edited by Betsy Beyer, Chris Jones, Jennifer Petoff & Niall Richard Murphy

Limbo: Scaling Software Collaboration

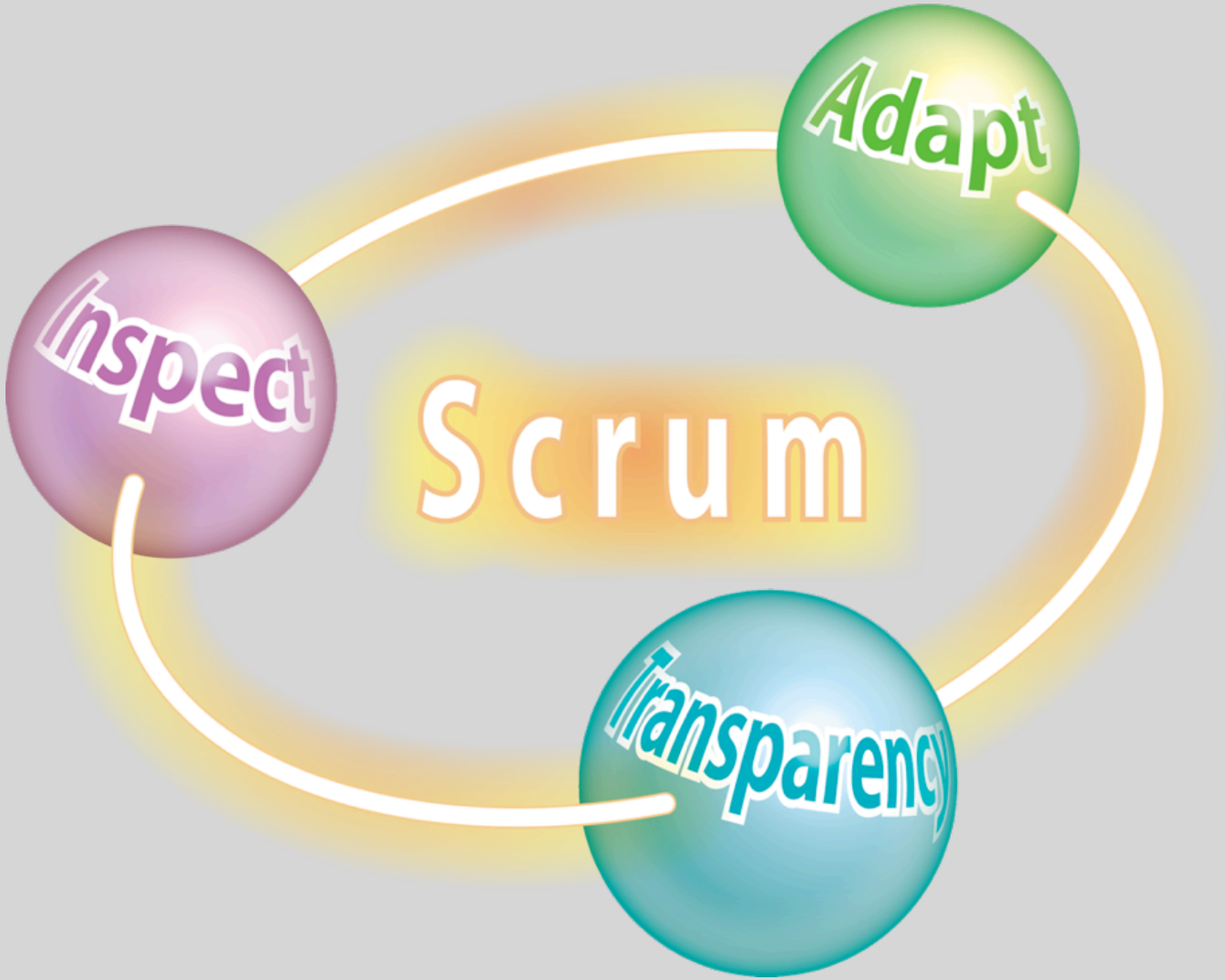


Kent Beck Jul 11, 2018 · 6 min read

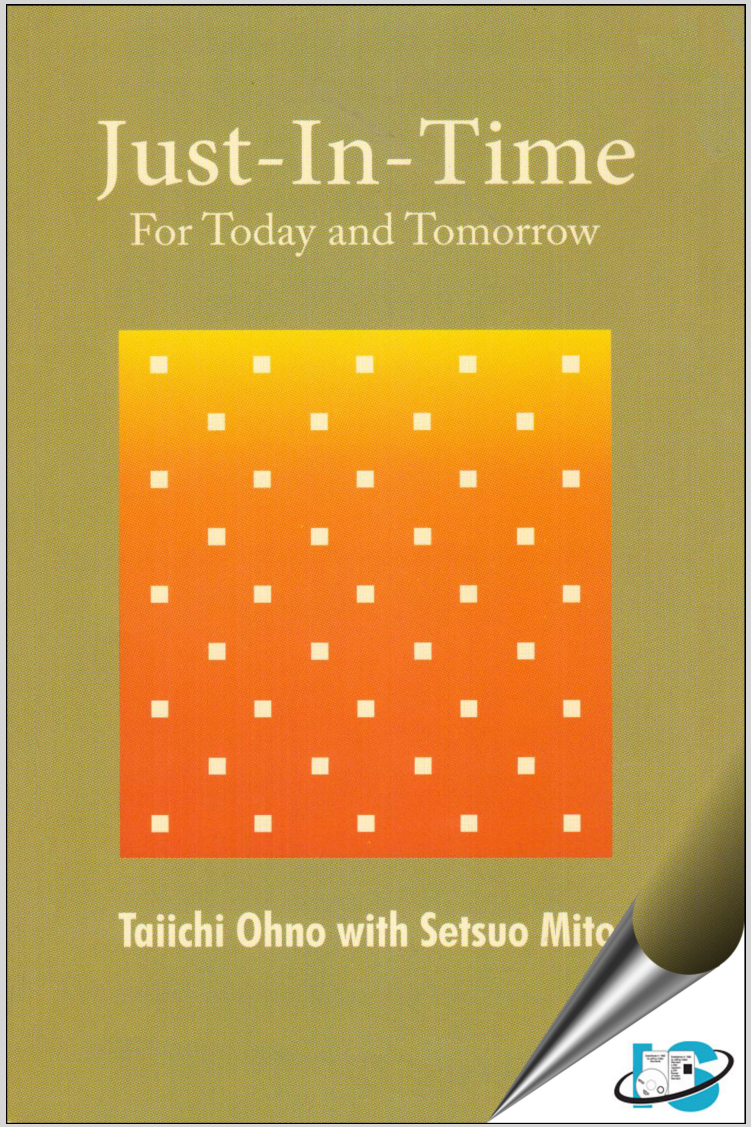




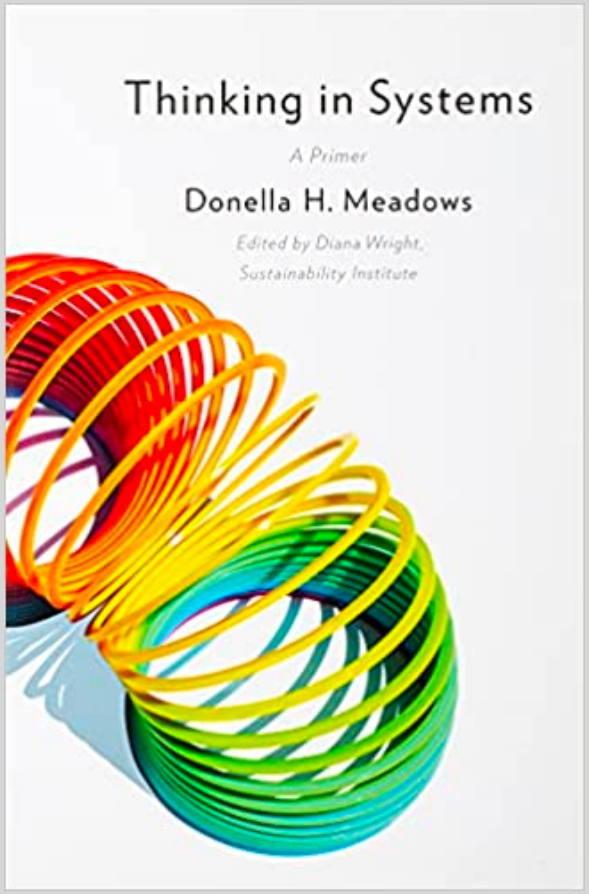
Large-Scale Scrum is Scrum



One piece flow



Avoiding Tragedy of the Commons



- *Educate and exhaust.*

  Help people to see the consequences of unrestrained use of the commons. Appeal to their morality

- *Privatize the commons.*

  Divide it up, so that each person reaps the consequences of his or her own actions.

- *Regulate the commons.*

  Mutual coercion, mutually agreed upon.

Micro-services ?

- An architectural style that decomposes the system in services that are independently deployable.

- When using micro-services (and LeSS), avoid:
  - Teams own services (also known as component teams)
  - Repository per services (also known as multi-repo)

- But be aware:
  - Service communication creates additional overhead
  - Additional network communication creates significant additional complexity



Example Incremental Upgrade (1)

- Refactor all usage of an API that is changing to one place
- Move all usage of changing API again into one component/class. For example this could be called “UpgradeService” and it only exists during the upgrade
- Use a bash script that does the upgrade. It does:
  - Simple renames of changes in the API. Using find and sed.
  - Renames files that are called <file>.upgrade to <file> so that it is possible to have two versions of a file without branching in repository. This is mostly used for configuration but also for e.g. “upgrade service”
  - Deletes all lines that are marked for deletion with a comment. For example “// ION3”
  - Runs an open source upgrade script that does more renames and changes



Example Incremental Upgrade (2)

- For all incompatabilities, first ask whether we can find a way that works in both versions, if so, refactor it to a different implementation that works for both implementations
- For things that do not exist in the new version, write a dummy wrapper so that we can compile it, it just doesn't do anything
- For things that do not exist in the old version, write an implementation that doesn't do anything in the old version, but does something in the new version
- An example of the above is best visible in the CSS where the changes are dramatic. We add CSS for the new version (which doesn't do anything in the old version, it is just dead code) and then for the old styles, we mark them as "// ION3" so that they will be deleted by the upgrade script



Example Incremental Upgrade (3)

- People work with 2 checked out versions (same code branch). One version that has run the script and one that hasn't. Try the change in the upgraded version, then manually copy it to the old version to see if it works there also, then we always only commit on the old version checkout (so we never commit the results of the upgrade script)

MORE *with* LESS.