codecentric

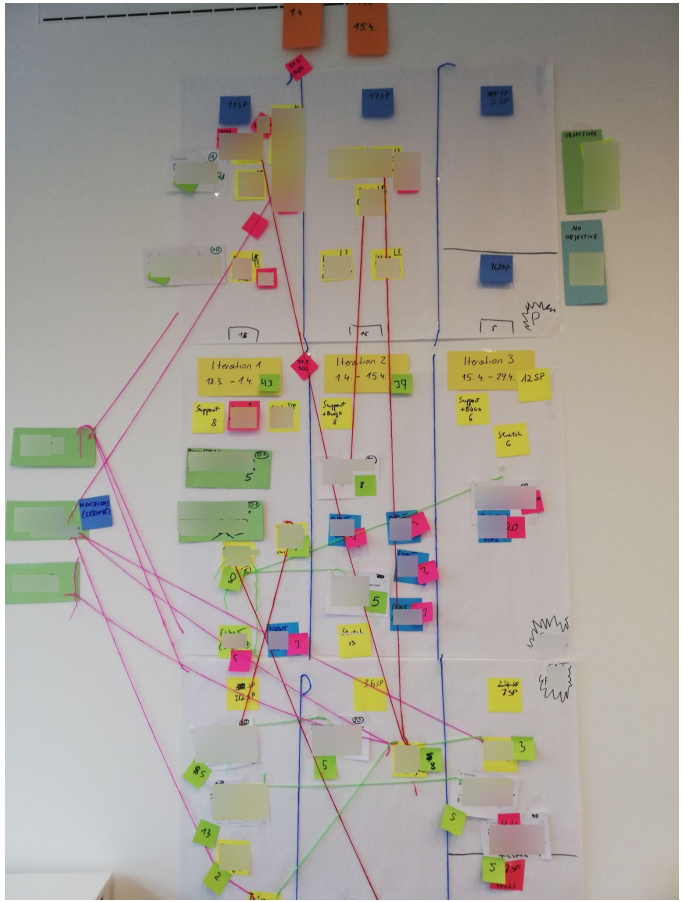
Escaping the dependency trap

Reducing dependencies

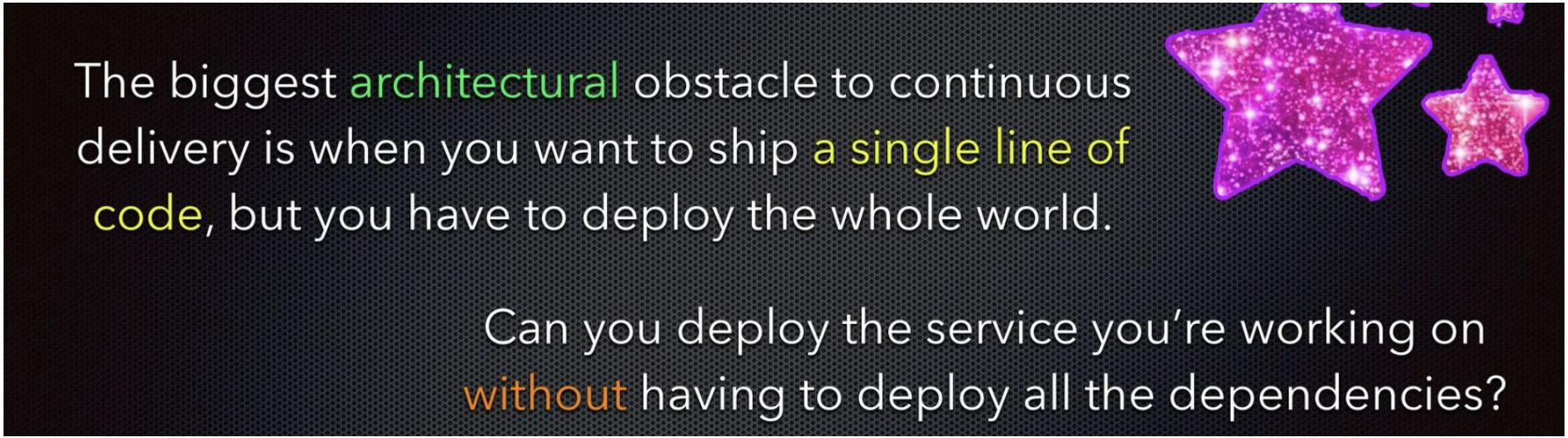
Thorsten Brunzendorf

XP Days Germany 2023-10-05

Dependencies at planning and design time

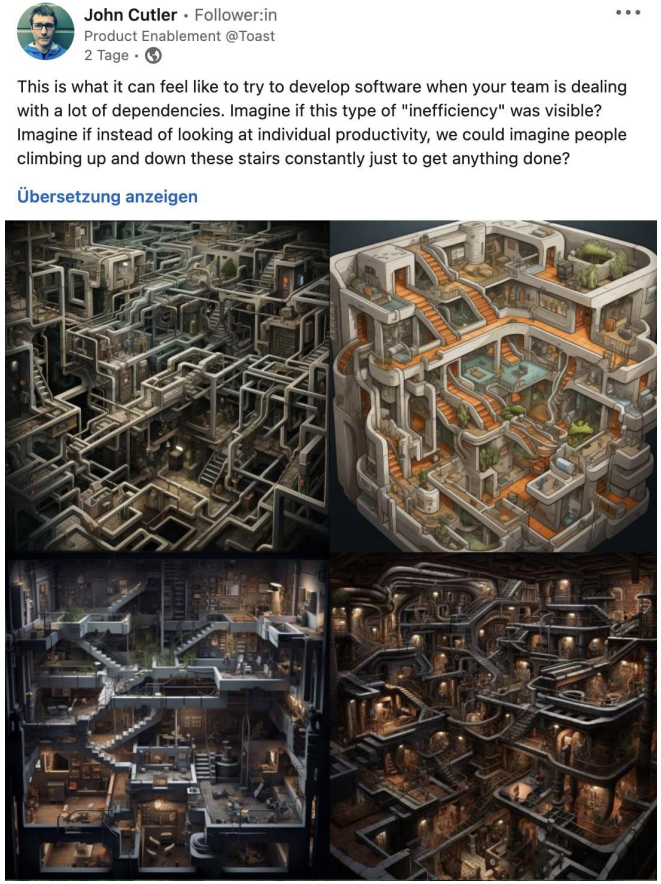


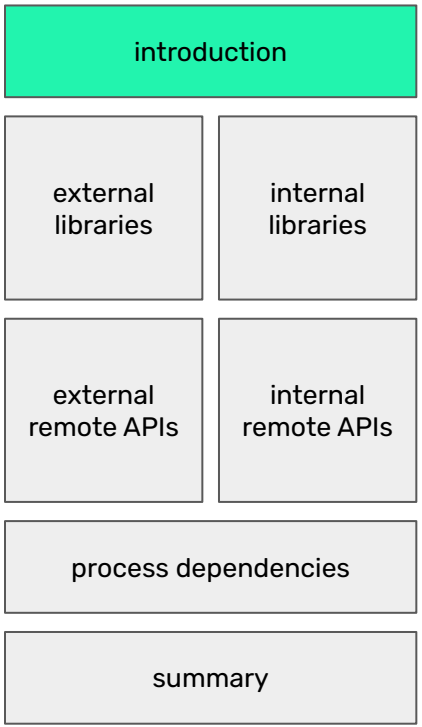
Dependencies at deployment and run time



Charity Majors @mipsytipsy

Dependencies as a measure of org dysfunction

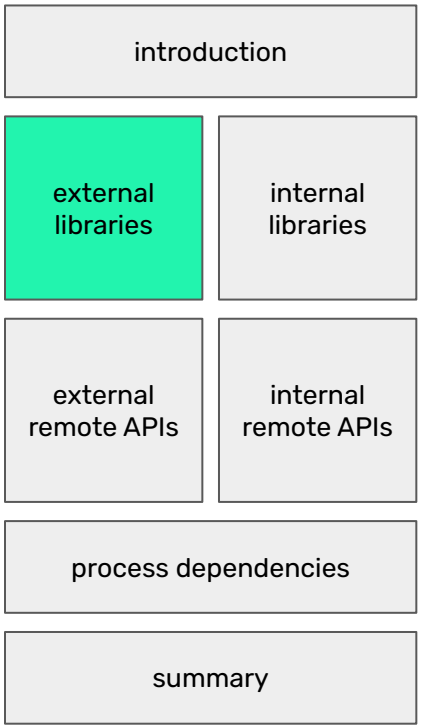


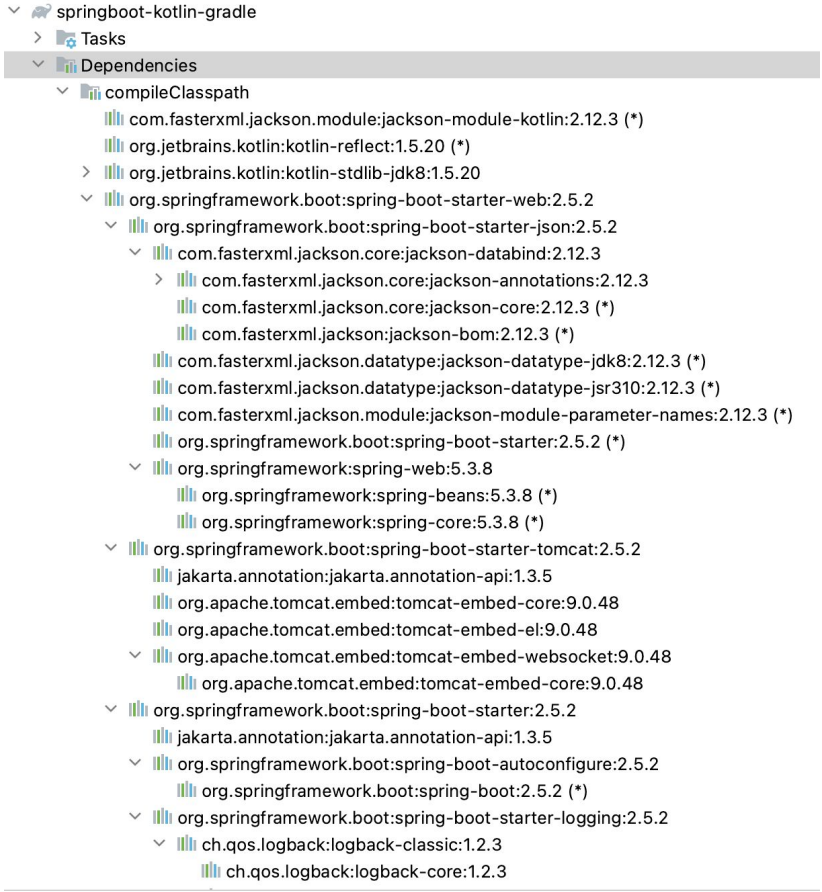


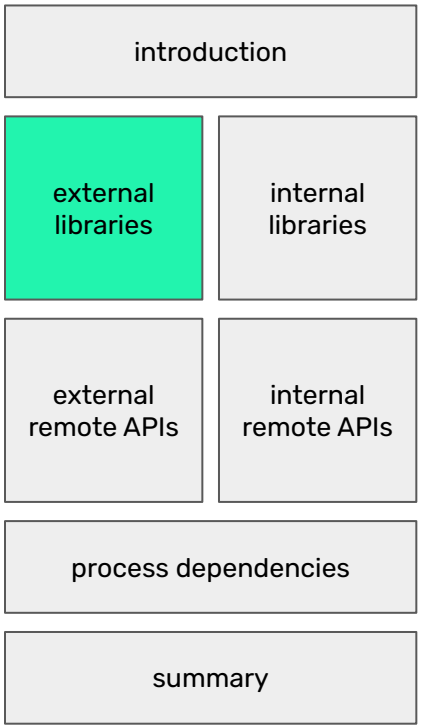
Structure

- **Team as unit of delivery**
- **Libraries vs Remote APIs**
- **Internal = same organization but another team vs External = another organization**

External libraries







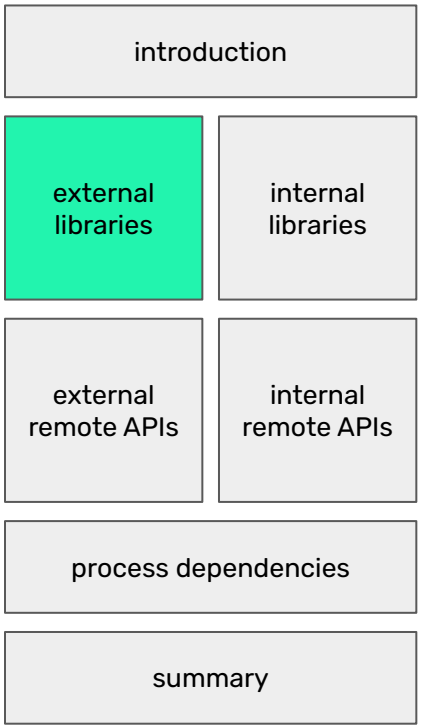
Dependencies on external libraries

- **mostly on open source libraries**

Using open source libraries is common sense and not much of a problem

… except for

Updating open source libraries

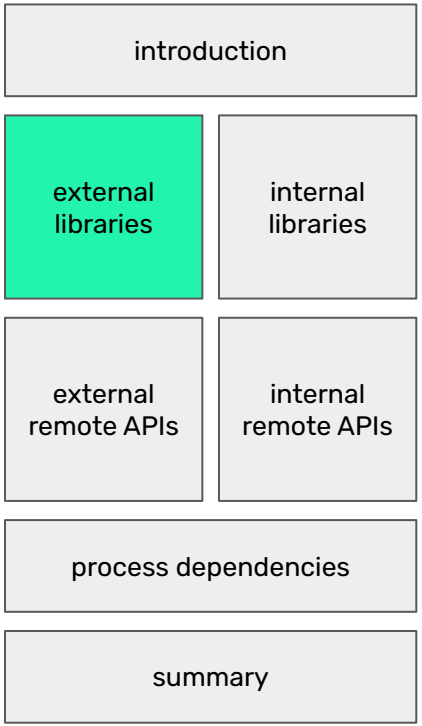


Reasons for wanting to update

- new features
- bugs
- security issues
- end-of-support for version used

Reasons for not updating

- breaking changes
- conflicting indirect dependencies
- unwanted license models
- no new releases



Challenges: breaking changes, conflicting indirect dependencies, unwanted license models

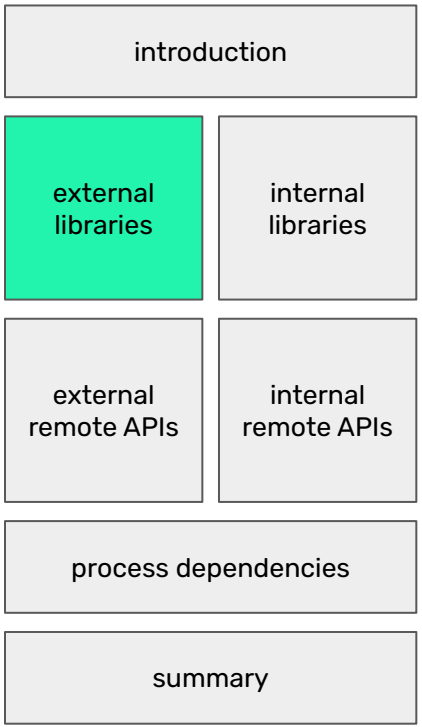
- Risk of updating to a new version

keep dependencies up to date automatically

- e.g. by using Renovate or Dependabot
- do not let risks pile up
- let your pipeline find issues

create new releases and put them into production

- continuous deployment or at least every iteration
- schedule releases at regular intervals - even if there is no customer demand



Challenge: no new releases

- **Risk of dependency on not so well-maintained library**

no new PRs? propose it yourself!

- **give back and propose a PR for your problem**

assess - isolate - remove

- **stay away from libraries that are seldom used or have infrequent releases**
- **don't let library-specific types contaminate your whole code - encapsulate them and build your own abstractions**
- **remove or replace "risky" libraries**

introduction

external libraries

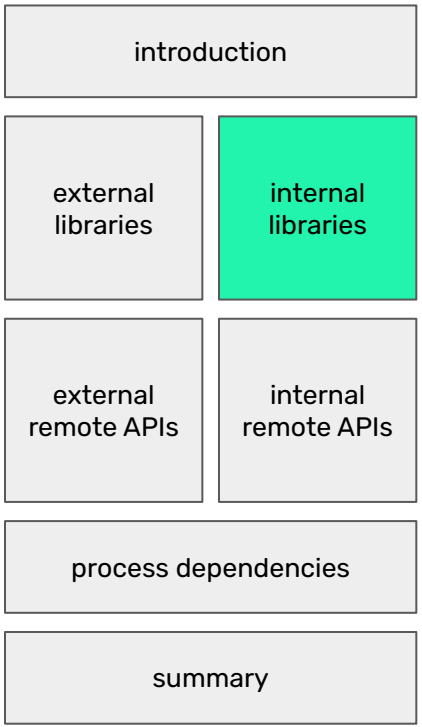
internal libraries

external remote APIs

internal remote APIs

process dependencies

summary



Dependencies on internal libraries

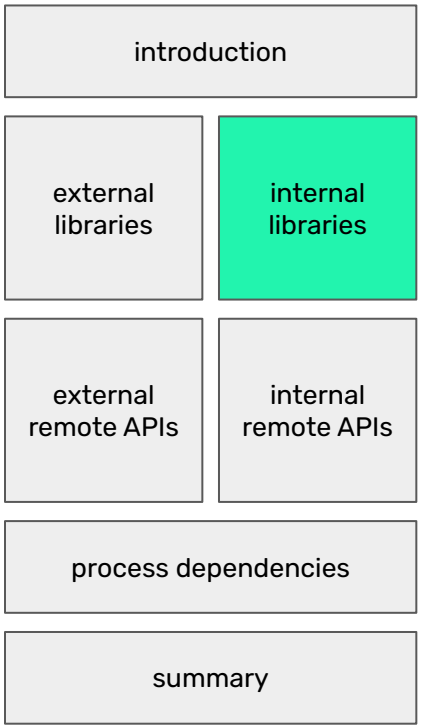
- **same org but maintained by other teams**

Shared domain model (and logic)

- **e.g. Shared Kernel**

Shared infrastructure abstractions

- **e.g. org specific core library**

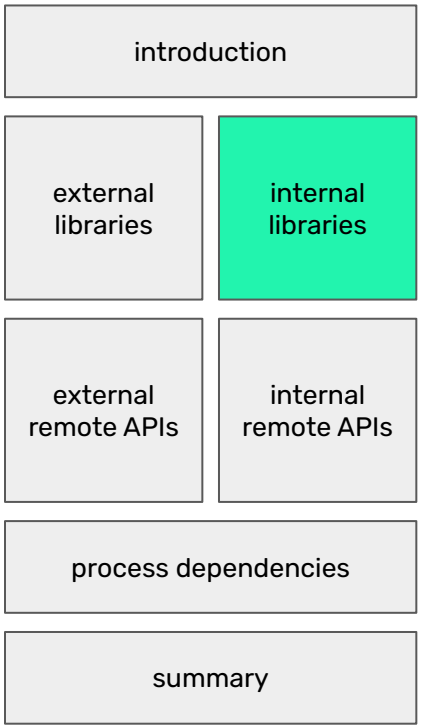


Challenge: multi-purpose org specific base lib

- **“not invented here” syndrome**
- **often comes with unintended coupling**
- **paves the way to distributed monolith**

stay away from multi-purpose org specific base libs

- **if really needed use smaller single-purpose libs with minimal dependencies**
- **never use it with APIs or as dependency for your own libs**

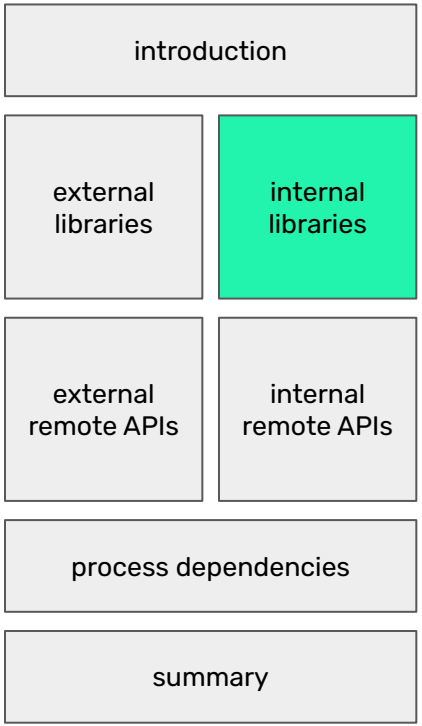


Challenge: unclear responsibilities for maintenance

- **shared library team has other priorities**
- **lifecycle often goes from mandatory use directly to unmaintained deprecation**

treat libraries as internal open source

- **usage is optional**
- **prepare (and accept) contributions**
- **assess - isolate - remove**
- **update and release frequently**



Libraries in general

create and use SBOMs (software bill of materials)

- **EU Cyber Resilience Act (proposed)**
- **CycloneDX (OWASP Foundation)**
- **SPDX (Linux Foundation)**

introduction

external libraries

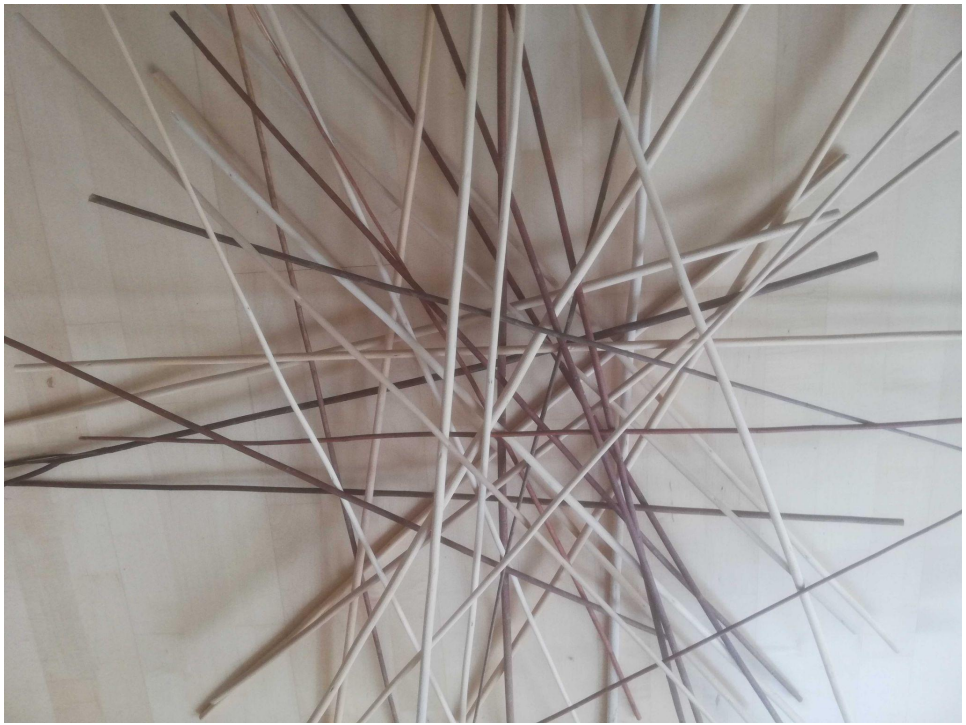
internal libraries

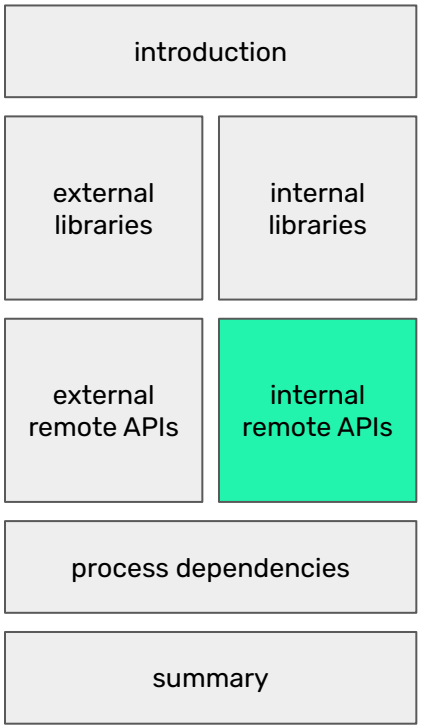
external remote APIs

internal remote APIs

process dependencies

summary





Dependencies on internal remote APIs

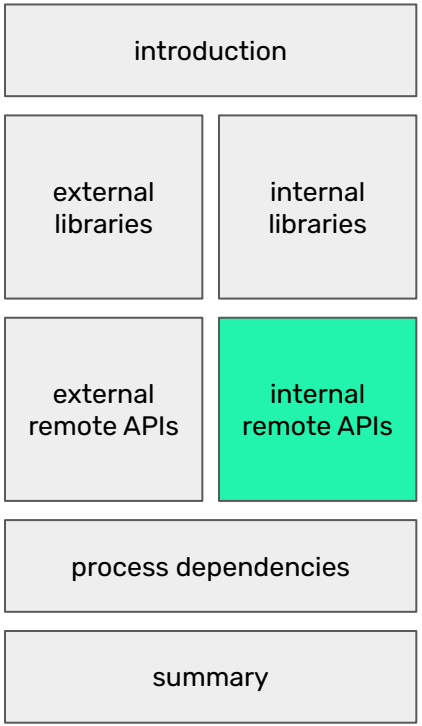
- **Same org but maintained by other teams**

Provider and consumer roles

- **Upstream - Downstream**

DDD context mapping

- **Different types of relationships: Partnership, Shared Kernel, Customer-Supplier, Conformist, Anti-Corruption Layer, Open Host Service, Published Language, Separate Ways**



Challenge as consumer: Need to integrate an API but it's not in production yet

- **Use mock API**
  when you have an API spec
  but provider is not implemented yet

- **Use feature toggle**
  when provider implemented but not deployed and released to production yet

introduction

external libraries

internal libraries

external remote APIs

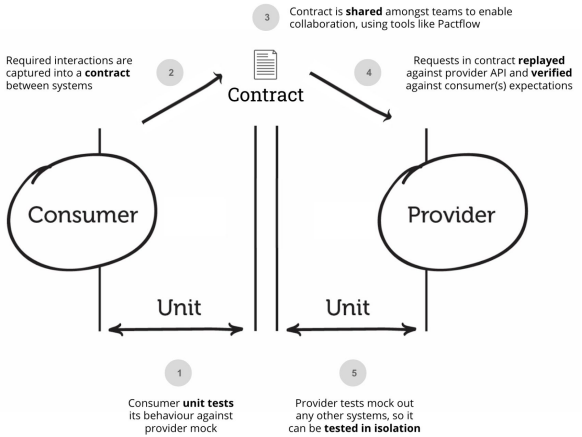
internal remote APIs

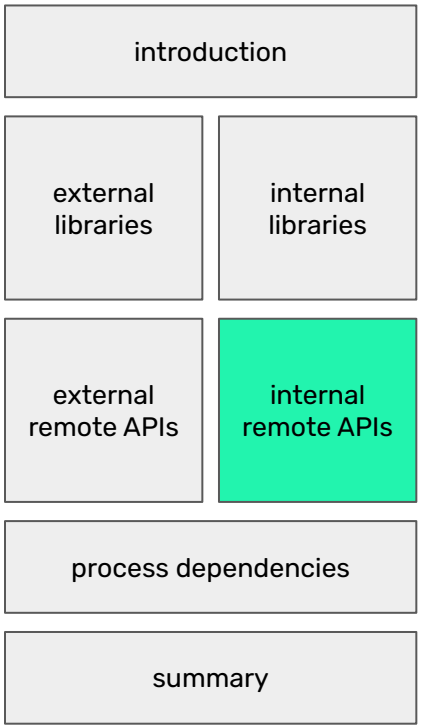
process dependencies

summary

Challenge as consumer: API changes unpredictably

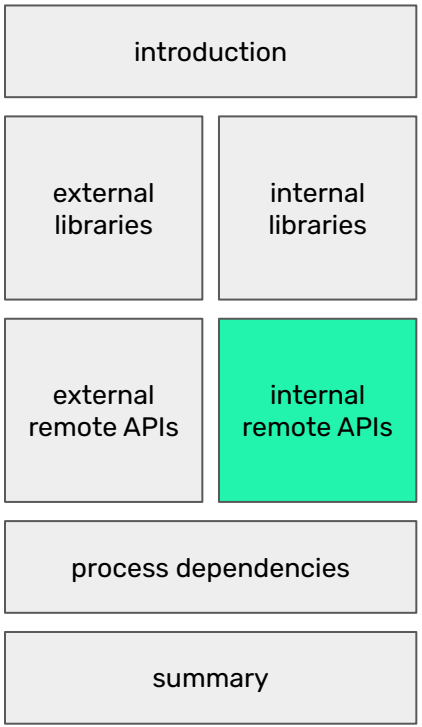
- **Implement Consumer Driven Contract Testing to verify that changes are compatible**
  - **Important: specify only attributes needed**





Challenge as consumer: API changes unpredictably

- **Use adapter as Anti Corruption Layer (ACL)**
  - **Variant: ACL as separate deployment**
- **Use feature toggle for switching between old and new API**
  - **Both client versions implemented (Branch by abstraction)**

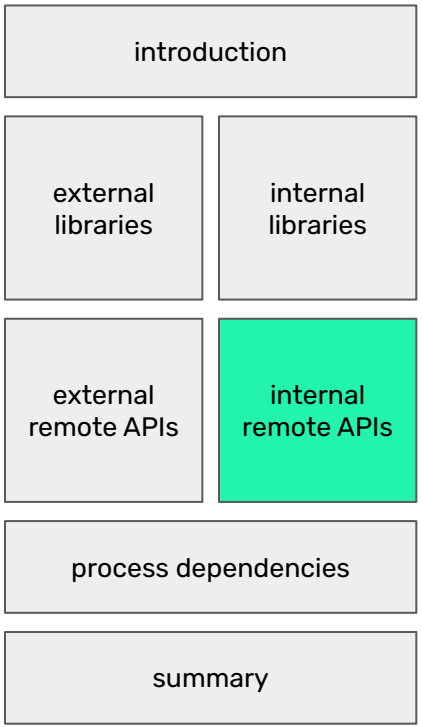


Challenge as provider: Need to serve different consumers

- **Consider consumer-specific interfaces**
  - instead of 1 general interface for all
  - can still use same implementation if appropriate
- **Provide query-based API**
  - e.g. GraphQL or OData

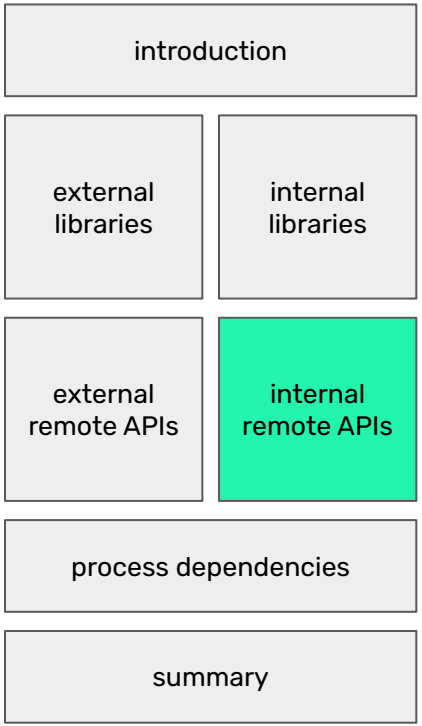
Challenge as provider: Need to serve different consumers

- **Beware: Generic APIs mean more coupling**
  - **Syntactic coupling is replaced with semantic coupling**
  - **Note: Also CRUD APIs can be considered quite generic**



Challenge as provider: Need to change API without impacting consumers

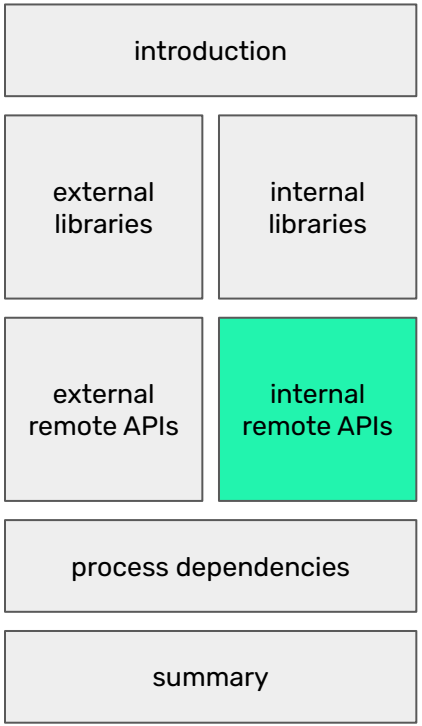
- **Make only compatible changes to API**
  - **and make incompatible changes only if CDC tests succeed**
- **Provide breaking changes as new interface**
  - **use versioning only as a last resort**



Challenges at runtime: Consumers depend on provider's operational quality and provider depends on consumers' load

Improve resilience

- **Reduce negative runtime effects of dependencies**
- **As consumers:**
  Learn about runtime characteristics of providers
  Leverage patterns like Retry and Circuit Breaker
- **As providers:**
  Learn about actual consumers and their load
  Implement load testing
- **Both: Build crucial business knowledge**



Challenges at runtime: Consumer depends on availability of provider

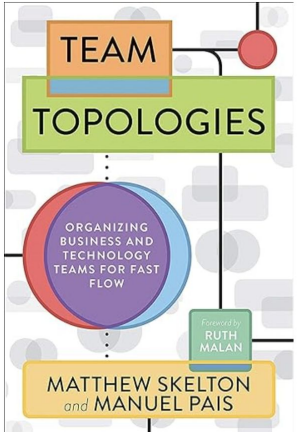
Switch to asynchronous interaction styles (using events or messages)

- If response is not needed immediately to continue processing
- Removes runtime coupling from consumer to provider
- Replaces it with runtime coupling of both provider and consumer to infrastructure (event or message broker)
- Changes direction of flow at runtime (provider plays active part)
- Reading data on demand is not possible - consumer needs to build read model from events received (alternative: caching)

Challenge: close collaboration with other teams is valuable but comes with high cognitive load

Define and publish team APIs

- **Make assumptions explicit for other teams and team members themselves**



Team API template

A template for defining a Team API. A Team API is a description and specification that a team can define that tells others how to interact with that team.

Based on some of the ideas in the book *Team Topologies* by Matthew Skelton @matthewskelton and Manuel Pais @manupaisable. See pages 47-49 of the book *Team Topologies*.

> See teamtopologies.com for more details about Team Topologies.

Overview

It can be useful to define a "Team API" for each team, to improve the team's clarity of purpose and help other groups understand how that team fits into the broader organization. Use this template to help teams think about how they would define their Team API.

How to use

Each team should answer the questions and fill in the details below. Remember that the answers and details will be a point-in-time snapshot of team relationships and team interactions.

Team API

Date:

- Team name and focus:
- Team type:
- Part of a Platform? (y/n) Details:
- Do we provide a service to other teams? (y/n) Details:
- What kind of Service Level Expectations do other teams have of us?
- Software owned and evolved by this team:
- Versioning approaches:
- Wiki search terms:
- Chat tool channels: #_____ #_____ #_____
- Time of daily sync meeting:

> Team type: (Stream-Aligned, Enabling, Complicated Subsystem, Platform)

What we're currently working on

- Our services and systems:
- Ways of working:
- Wider cross-team or organisational improvements:

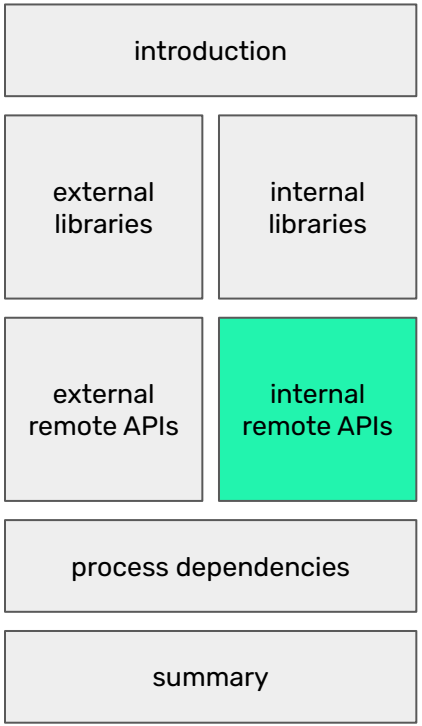
Teams we currently interact with

Team name/focus	Interaction Mode	Purpose	Duration
.			
.			
.			
.			
.			
.			

> Team Interaction Modes: (Collaboration, X-as-a-Service, Facilitating)

Teams we expect to interact with soon

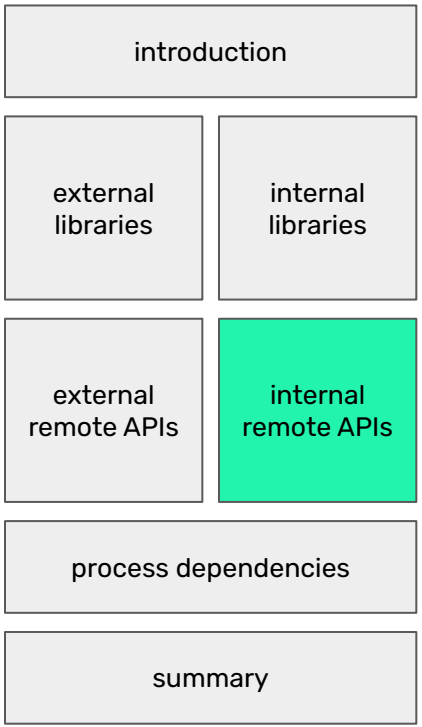
Team name/focus	Interaction Mode	Purpose	Duration
.			
.			
.			
.			
.			
.			



Challenge: close collaboration with other teams is valuable but comes with high cognitive load

Limit dependencies in progress to 1

*"When we had 2 or more items in process that needed coordinating with other teams everything in the process took longer. ... we had a fairly good handle on Work In Progress ... but it was Dependencies In Progress that we needed to start limiting. ... By adding a new policy – **don't start a new item with dependencies if there's already one in progress** – we kept our flow of work smoother and our predictions ... much more useful."*
- Neil Vass



Challenge: close collaboration with other teams is valuable but comes with high cognitive load

Don't change APIs in lockstep

- i.e. don’t change provider and consumer in same iteration
- it mostly does not work (other priorities or problems revealed only when testing in integration)
- exception: when ensemble programming together with other team on both sides

Examples for evolving APIs

Add new output attribute

1. Provider adds new attribute to response payload
   Consumers must be able to ignore additional data (Postel's law)
2. If needed consumers process new attribute

Remove input attribute

1. Provider removes attribute from request payload
   Consumers still includes this attribute in request
   Provider must be able to ignore additional data (Postel's law)
2. Eventually consumers refrains from sending this attribute

Examples for evolving APIs (2)

Add new input attribute

1. Provider adds new optional attribute to request payload
   but does not process any data yet or uses sensible default value
2. Later consumer sends new attribute - Provider does not change yet
3. When all consumers are ready, provider processes new attribute data
   and declares it mandatory if needed (2nd deployment)

Delete output attribute

1. Provider declares attribute optional and returns default value
2. Consumers need to ignore this output attribute
3. Provider sends null or removes attribute completely

Examples for evolving APIs (3)

Renaming an attribute

- Renaming is not an atomic operation
- Composition of adding a new attribute and removing an existing attribute

1. Add new attribute and copy content from old attribute
2. Deprecate old attribute and
   accept both new and old attribute, with precedence for new
3. Eventually ignore and then remove old attribute

introduction

external libraries

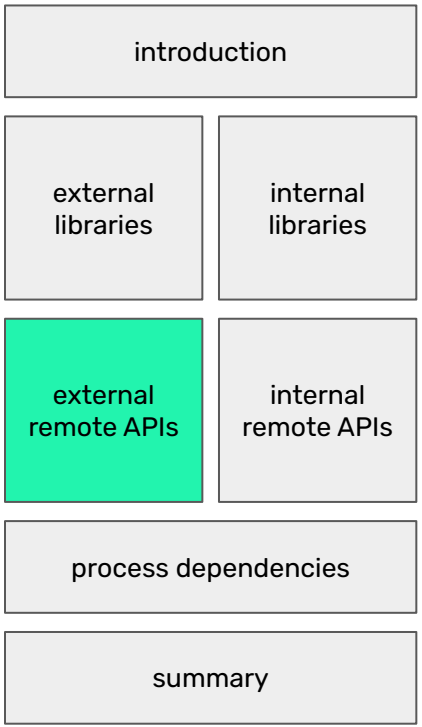
internal libraries

external remote APIs

internal remote APIs

process dependencies

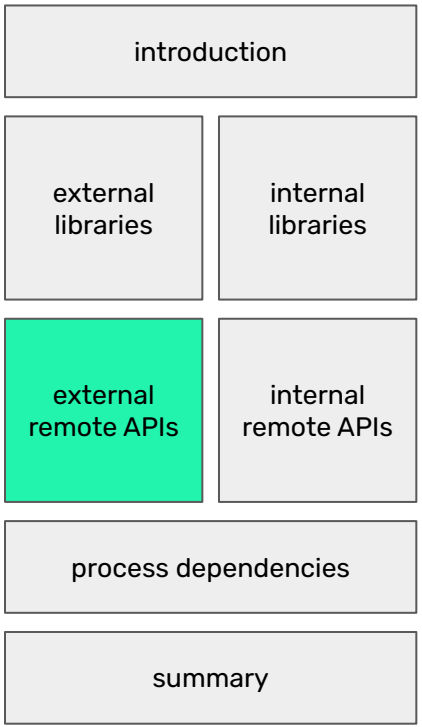
summary



External remote APIs (business partners or public)

What's different?

- **Consumers are less known to providers (and can be numerous)**
- **Providers are less known to consumers (and less approachable)**
- **Larger distance between consumers and providers => more coupling, higher cost of change**



Challenge: Larger distance between consumers and providers

- Consumer driven testing makes no sense here
- As providers: stabilize APIs
  - invest (even more) in API ops and API experience
- As consumers: use an adapter / ACL
- Improving resilience is even more important
  - invest in observability
- As providers: do not publish internal APIs to external consumers

introduction

external libraries

internal libraries

external remote APIs

internal remote APIs

process dependencies

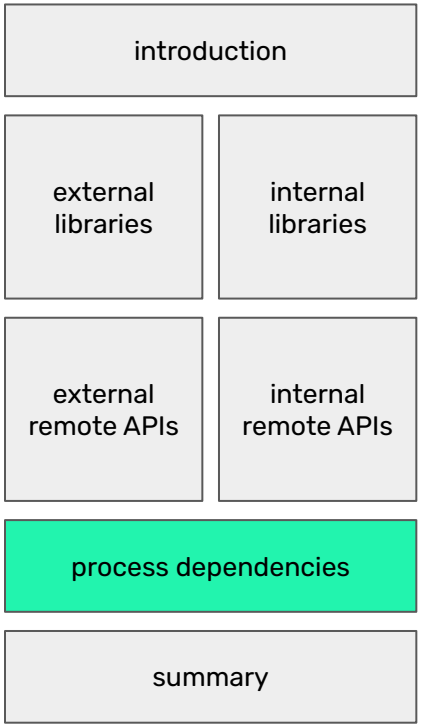
summary

Process dependencies - to other teams

Examples:

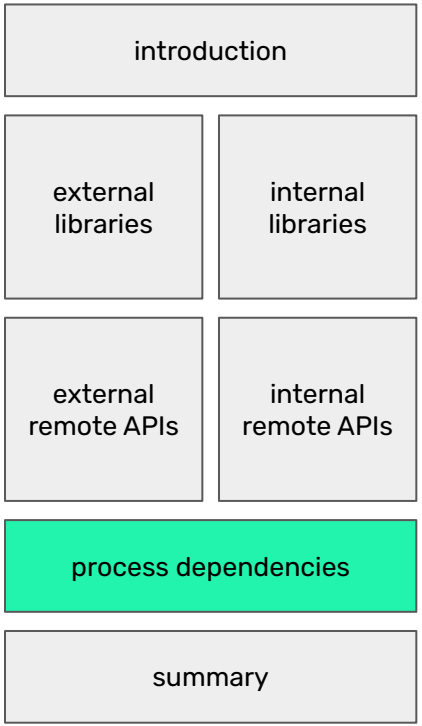
- **Hands-offs from/to other teams**
- **Sign-offs by other teams or stakeholders**

Some scenarios …



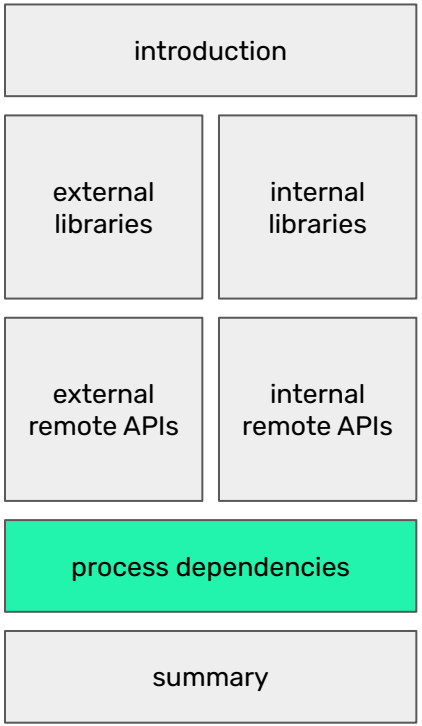
Scenario: separate approval gate
QA team or
User Acceptance Test by Subject Matter Experts

- **Include people or capability into your own team**
- **If not possible:**
- **Decouple deployment from release (toggles)**
- **Leverage QA team knowledge and let them lead exploratory testing (without formal gate)**



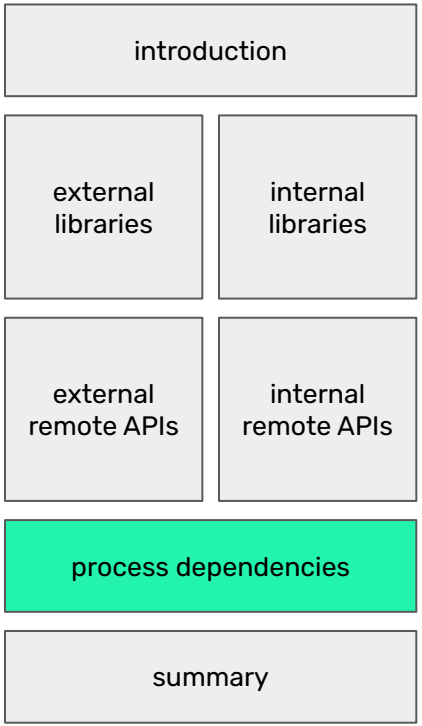
Scenario: separate infrastructure team, e.g. DBA team for central database

- Include people or capability into your own team
- If not possible:
- Tweak your architecture
  - e.g. use application specific database evolved together with application
  - e.g. use document-oriented database not maintained by DBA team



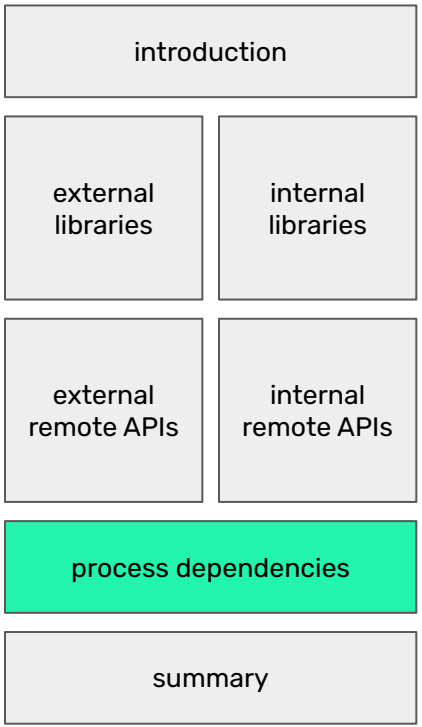
Scenario: release train - aka: we need to deploy to production together - it's the required process

- **Decouple yourself technically from the release train as much as possible**
- **Decouple deployment from release (again)**
- **Org hacks**
  - **different processes to deploy hotfixes or for critical vs non-critical changes**
  - **clarify what are non-critical changes and make most of your changes non-critical**



Scenario: "the business" says they do not need frequent releases

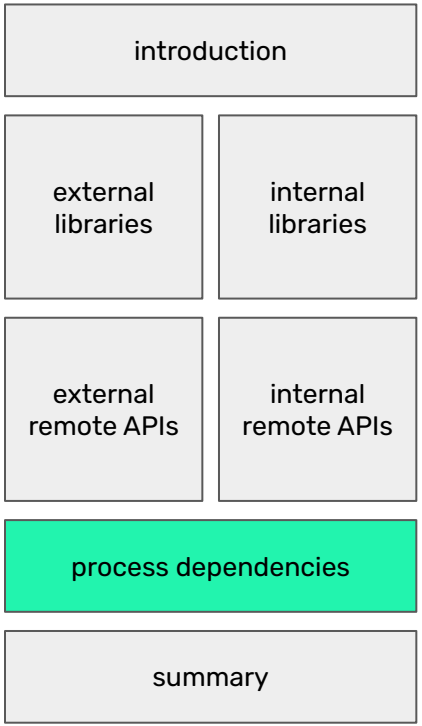
- **Continuous Delivery is a capability - even if biz says it's not needed**
  - **But: rollout security fixes, bug fixes, technical changes**
  - **Example: joint release of consumed API change**
  - **No finger pointing, be humble, build trust**



Reducing process dependencies

- **Goal: responsibility to deploy and/or release to users is within team**

"Autonomous teams are trusted by the organisation to "serve the customers to support the organisation" rather than just "serve the organisation". There is an expectation of judgement by the team. This is another way of saying they are expected to not just be order takers."
- Jason Yip

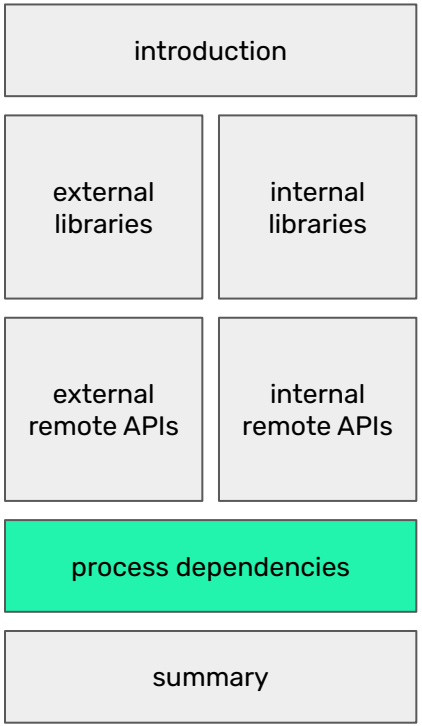


Systems thinking: Are we optimizing our subsystem (the team) and not the whole?

- **Teams should be the unit of delivery**
- **But sometimes team structures are at odds with your domain structure**

Use ideas from DDD (bounded contexts) and from Team Topologies (Stream-aligned teams)

"Knowing what changes together in the software is the key to determining how teams should be organised." (Nick Tune)

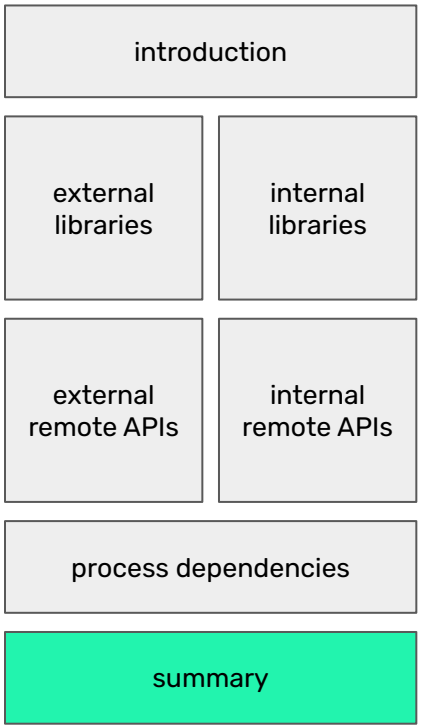


Systems thinking

- **And sometimes team structures are at odds with your current focus**

Make teams more fluid

- **Stability is important but stable teams are an illusion**
  => see Dynamic Reteaming
- **Decide what you want to invest in and structure the people around that**
- **Don't bind job positions to teams - common in traditional orgs**
- **Keep key people and allow for easy switching**



Summary

- **Question new dependencies**
- **Reduce existing dependencies**
- **Do not accept the status quo**
- **Dependencies are never just technical**
- **No finger pointing, be humble, build trust**

- **Eight flow accelerators –** Figure 15 illustrates how the eight flow accelerators help make value flow without interruption (Principle #6). These powerful accelerators of value are relevant to all Framework levels, but the challenges differ for each.

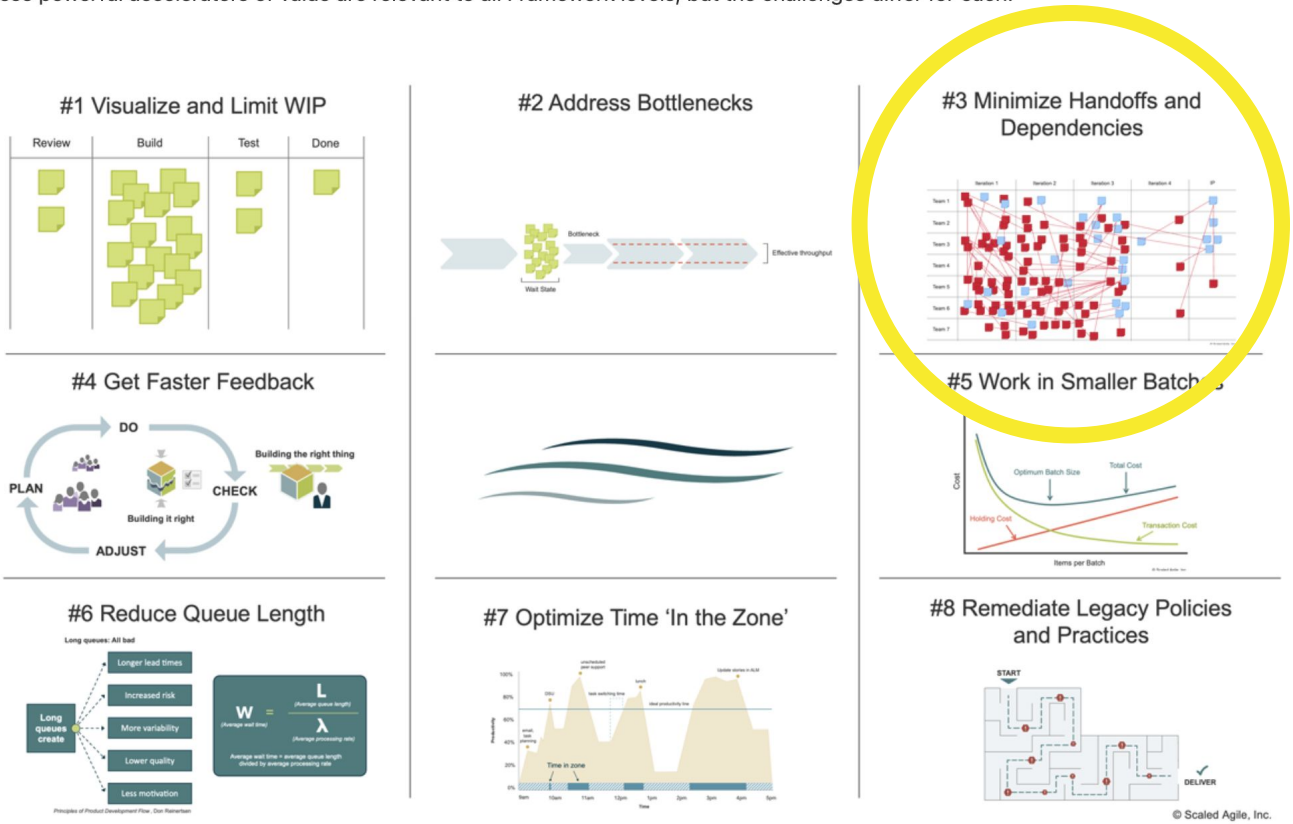


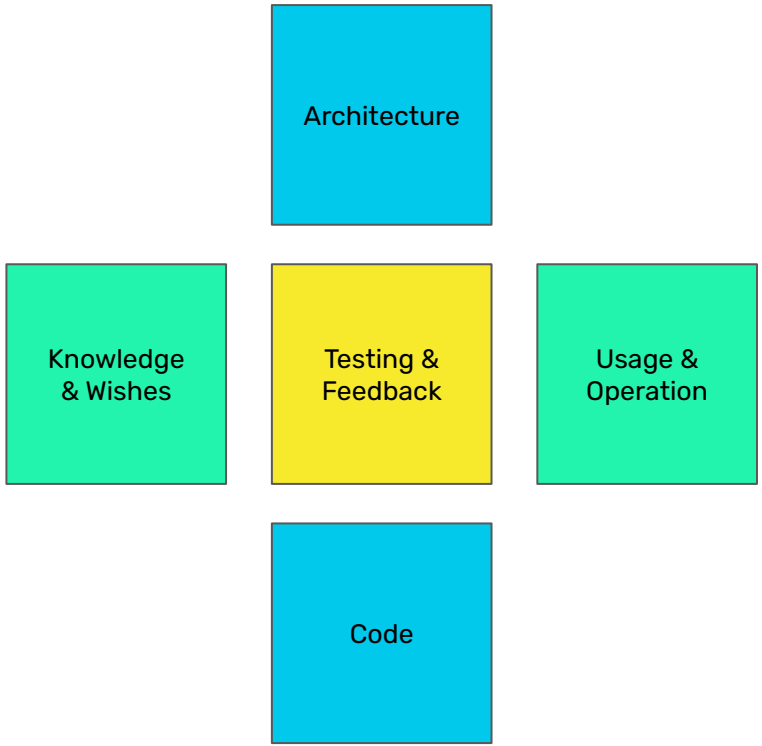
Figure 15. The eight flow accelerators

Thank you

Thorsten Brunzendorf
Senior IT Consultant

codecentric AG
Sophie-Germain-Straße 12
90443 Nürnberg

thorsten.brunzendorf@codecentric.de



Reference

Charity Majors https://speakerdeck.com/charity/compliance-and-regulatory-standards-are-not-incompatible-with-modern-development-best-practices
Consumer Driven Contract Testing https://docs.pact.io/
CycloneDX (OWASP Foundation) https://cyclonedx.org/
Daniel Terhorst-North https://dannorth.net/2023/03/02/but-what-about-the-bau-work/
Daniel Westheide https://www.innoq.com/en/blog/2016/11/the-perils-of-shared-code/
Gene Kim https://itrevolution.com/articles/on-coordination-costs-moving-a-couch-and-painting-a-room
Heidi Helfand https://www.heidihelfand.com/dynamic-reteaming/
Jason Yip https://jchyip.medium.com/key-practice-aligned-autonomous-cross-disciplinary-teams-d73c1cddc352
John Cutler https://www.linkedin.com/feed/update/urn:li:activity:7089500418510131200/
Martin Fowler https://martinfowler.com/articles/enterpriseREST.html
Neil Vass https://neil-vass.com/minimum-viable-estimation-part-2/
Nick Tune https://medium.com/nick-tune-tech-strategy-blog/visualising-sociotechnical-architecture-with-ddd-and-team-topologies-48c6be036c40
Oliver Drotbohm http://odrotbohm.de/2016/10/evolving-distributed-systems/
SAFe 6.0 https://scaledagileframework.com/whats-new-in-safe-6-0/
SPDX (Linux Foundation) https://spdx.dev/
Team API Template https://github.com/TeamTopologies/Team-API-template
Vladik Khononov https://speakerdeck.com/vladikk/balancing-coupling-in-distributed-systems

Cobweb photo by Pierre Bamin on Unsplash https://unsplash.com/de/fotos/QdNen0rPe4E
Jenga photo by Michał Parzuchowski on Unsplash https://unsplash.com/de/fotos/geNNFqfvw48
All other photos by Thorsten Brunzendorf