andrena OBJECTS

Experts in agile software engineering

Continuous Delivery – aber wie?

André Kappes

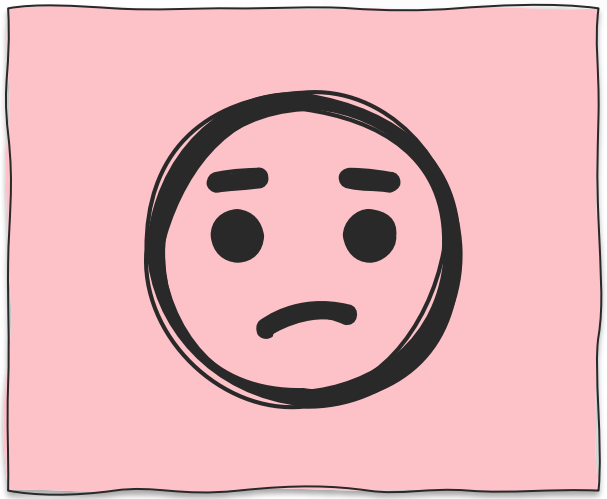
andre.kappes@andrena.de

23. September 2021

Java Forum Stuttgart

Wie ist das, wenn Ihr eine neue Software-Version ausrollt?

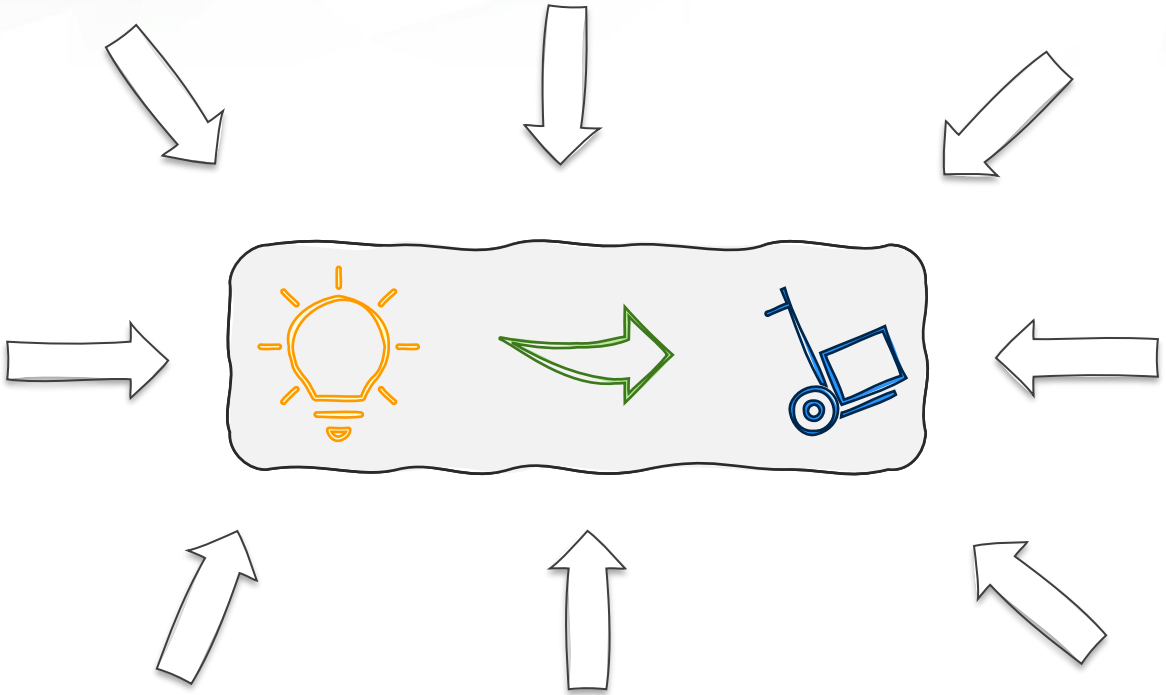
- Deployment dauert lange
- Selten
- Manuelle Schritte abarbeiten
- Mit Bereitschaft verbunden
- Es geht dann doch noch etwas schief



...oder so?

- Schnell
- Häufig
- Auf Knopfdruck, automatisiert
- Wenn etwas schiefgeht, haben wir einen Plan

Worum geht es heute?



Wie komme ich dazu?

Ich begeistere mich für…

- Software Craftsmanship
- Clean Code
- Automatisiertes Testen
- Domain Driven Design

… und eben Continuous Delivery

andre.kappes@andrena.de

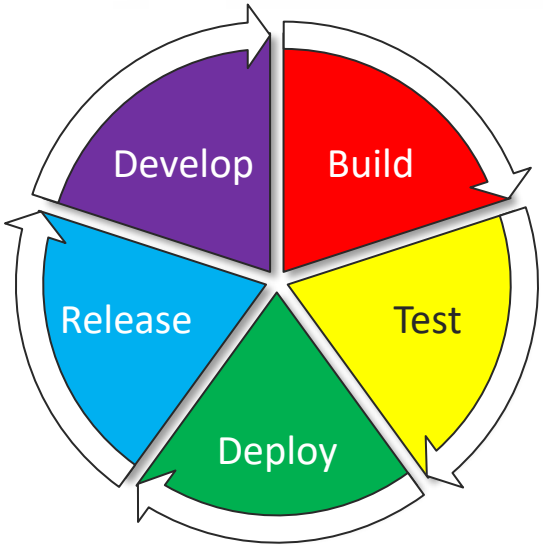
Continuous what?

Continuous Delivery

Nach Martin Fowler

You're doing continuous delivery when: [1]

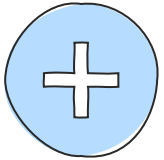
- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand



[1] https://martinfowler.com/bliki/ContinuousDelivery.html#footnote-when

Continuous Deployment

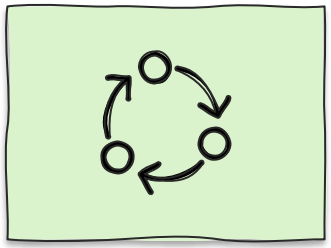
Continuous Delivery



automatische Deployments in Produktion von Commits auf dem Mainline-Branch

Wozu das Ganze?

- Reduzierte Cycle Time, kürzere Time to Market



- Überschaubare Releases, Reduktion von Komplexität
- Automatisierung führt zu weniger Fehlern

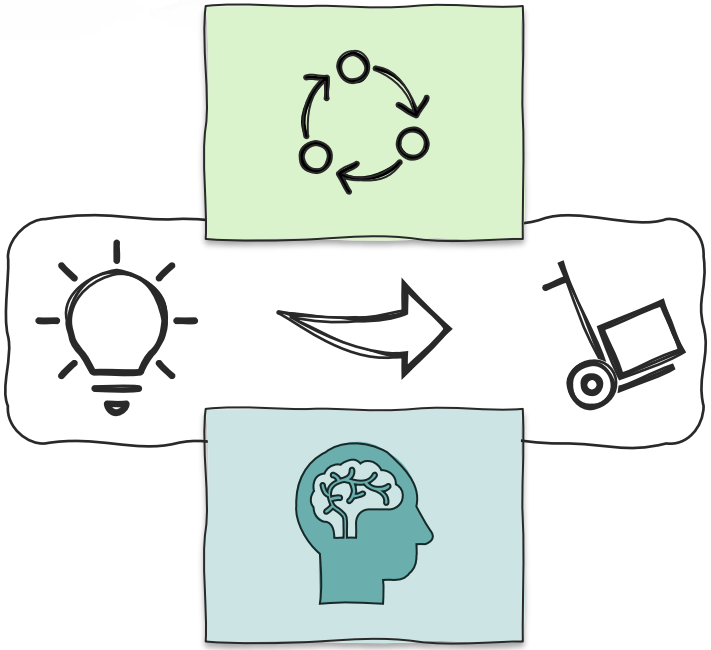
Strategische Überlegungen

Was beeinflusst, wie wir liefern?

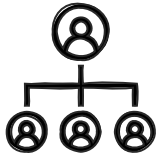
Stakeholder

Wohin?

Manuelle Schritte



Architektur



Organisation

Taktische Entscheidungen

Wie mache ich Code-Änderungen?

- Beim Entwickeln das Deployment mitdenken

- Inkrementell und evolutionär arbeiten
- Breaking Changes vermeiden
- Interface-Änderungen per Parallel Change / branch by abstraction

Wie gehe ich mit dem Versionskontrollsystem um?

Feature branches

- Schützt mainline
  - Build, Unit Test, evtl. auch Ausrollen
  - Merge als Quality Gate: Code review

Trunk-based development

- Frühe Integration vermeidet Merge-Hölle
- Verbergen unfertiger Features durch Feature Flags

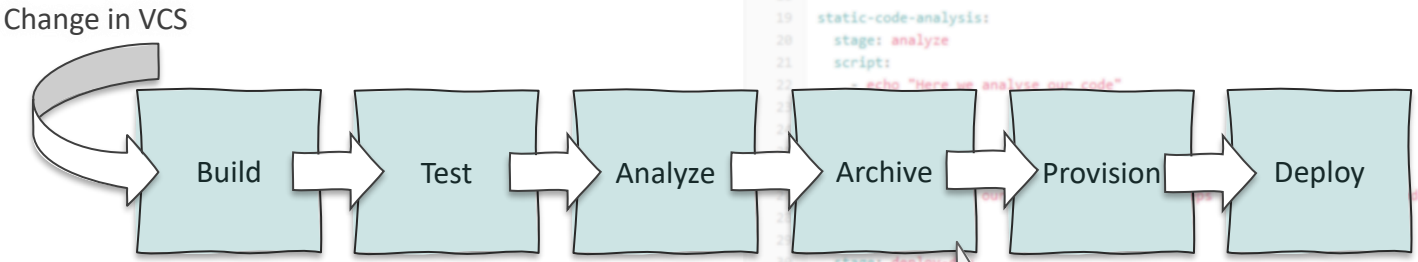
Near-trunk based development unterstützt Continuous Delivery

Wie sieht meine Continuous Delivery Pipeline aus?

Zentrales Software-Artefakt

zur Automatisierung des Release

Mitversioniert (Infrastructure as Code)



Trenne Code und Configuration

Wie viele Stages brauche ich und wozu?

Unterscheidung nach Funktion

- **DEV** Integration der Dev-Team-Arbeit; Ausführen der (autom.) Akzeptanztests
- **TEST** Für User Acceptance Test; entscheide, ob auf **PRODUCTION** ausgerollt wird
- **PRODUCTION**

Ggf. weitere Stages für

- Performancetests (Isolation)
- Demos/Kundentests

Je mehr Stages, desto automatisierter muss das Aufsetzen einer einzelnen Stage geschehen

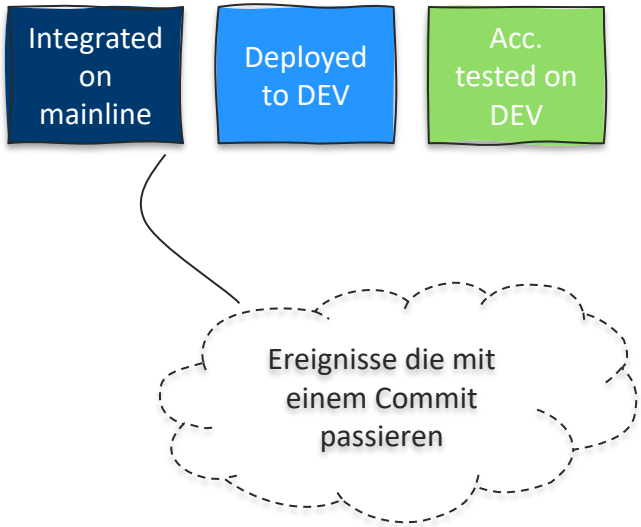
Paradigmen für Stages

„Build on foundations that are known to be good.“ [2]

- Stages sollen möglichst gleich wie **PRODUCTION** sein
- Deployments auf die verschiedenen Stages sollten möglichst gleich sein
- Alle Änderungen von **PRODUCTION** durchlaufen alle Stages

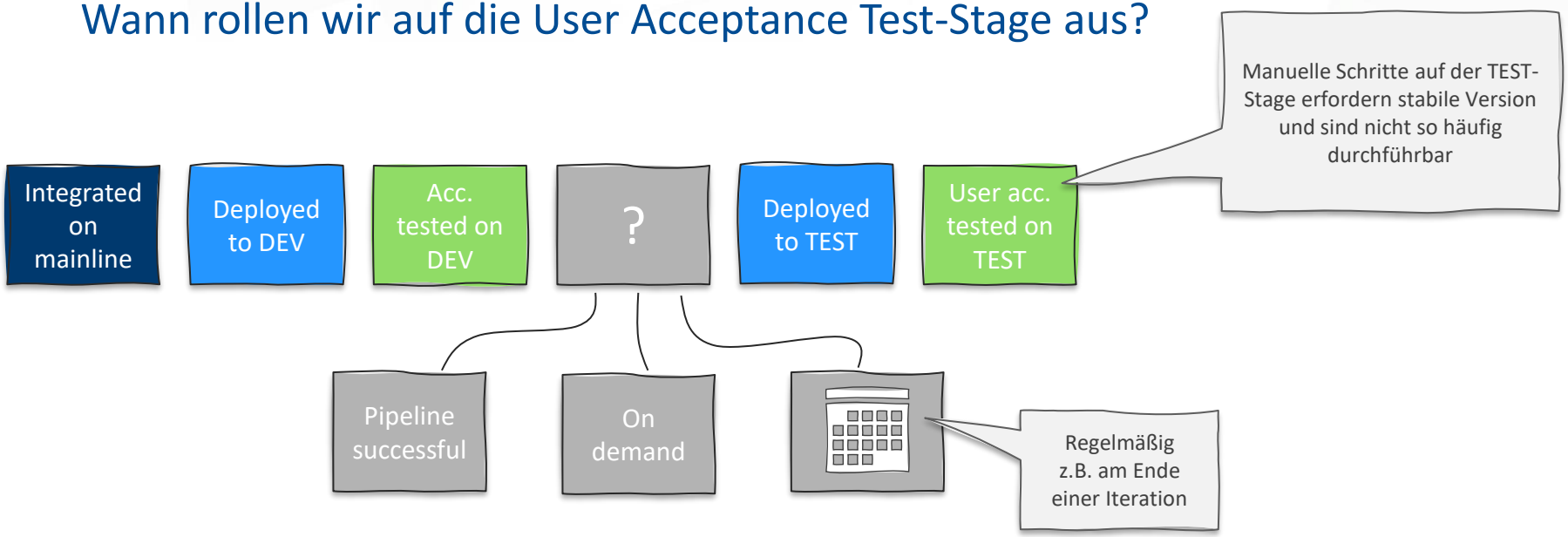
[2] Jez Humble, David Farley, „Continuous Delivery“

Deployment auf die DEV-Stage



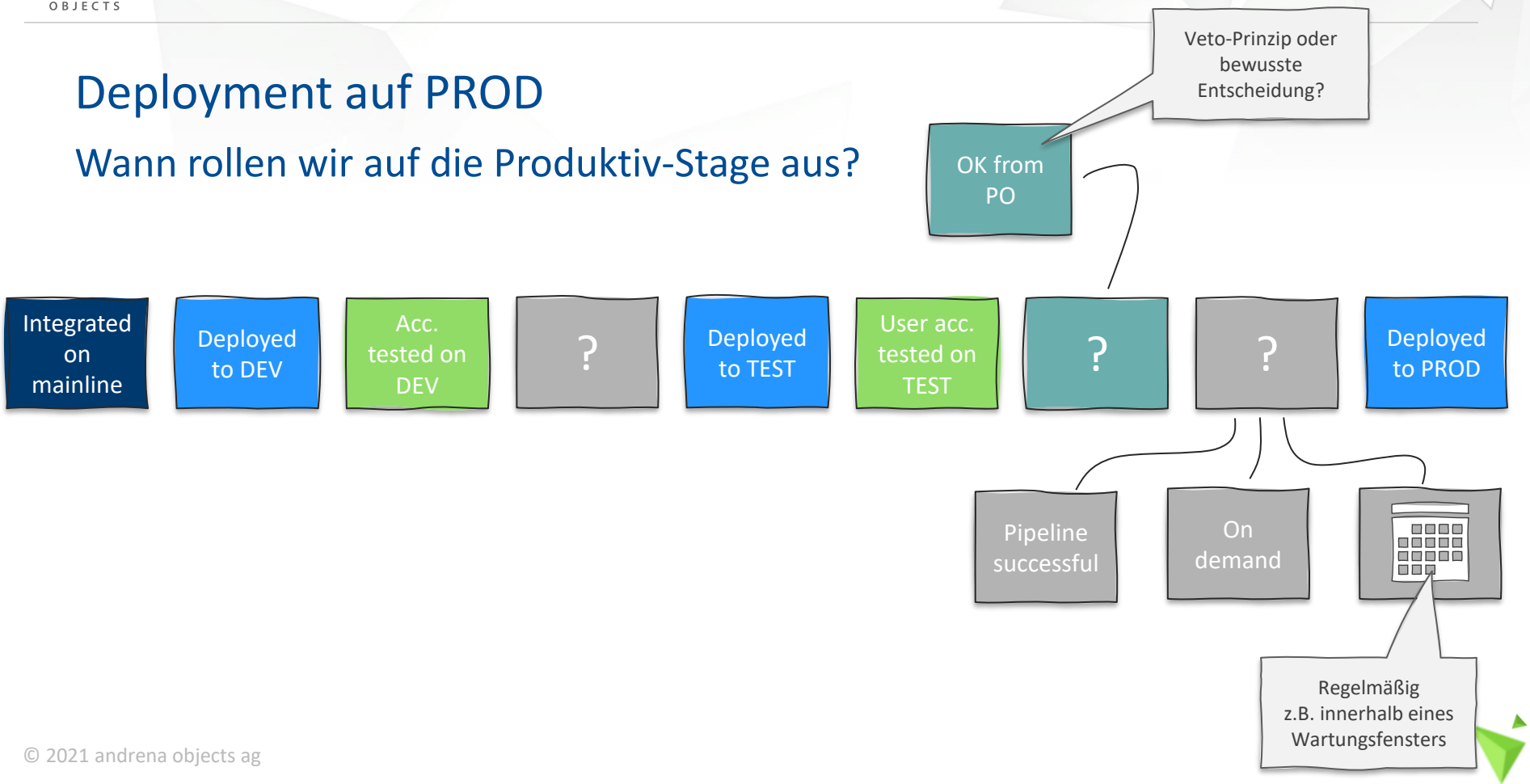
Deployment auf TEST

Wann rollen wir auf die User Acceptance Test-Stage aus?



Deployment auf PROD

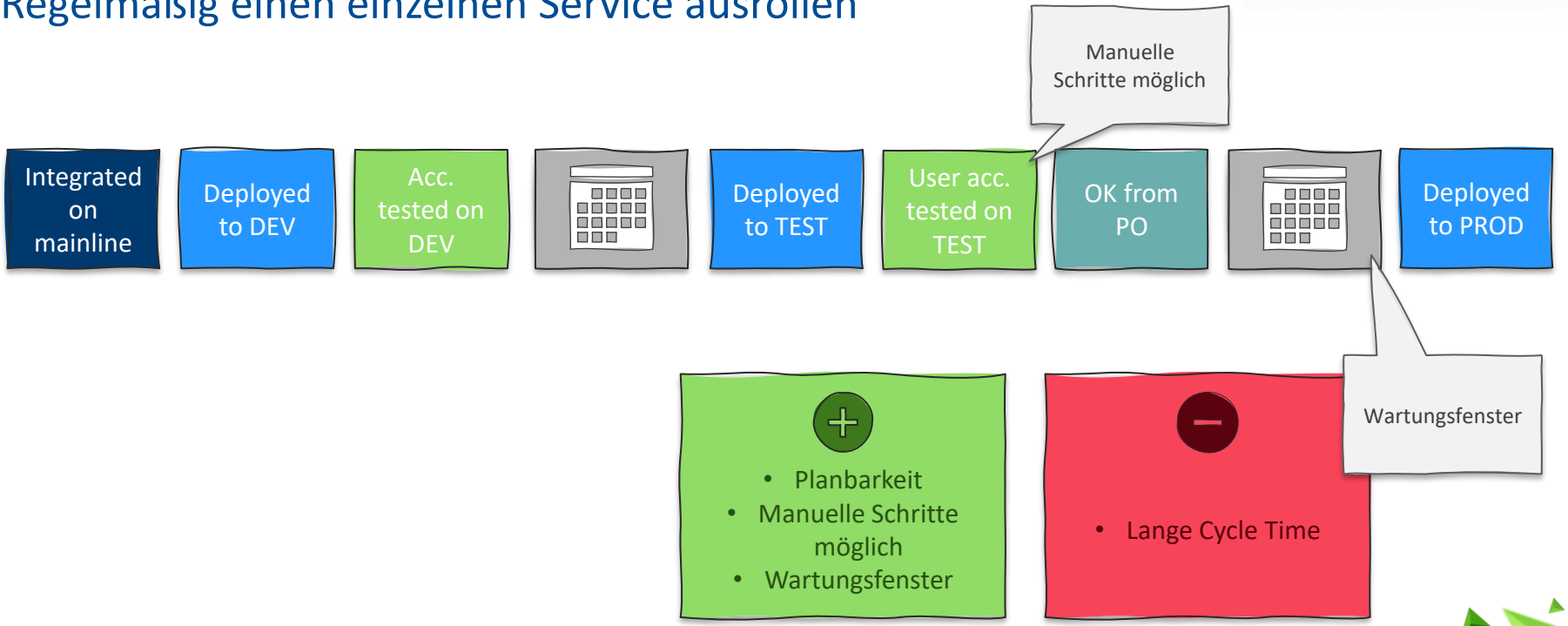
Wann rollen wir auf die Produktiv-Stage aus?



Deployment-Szenarien

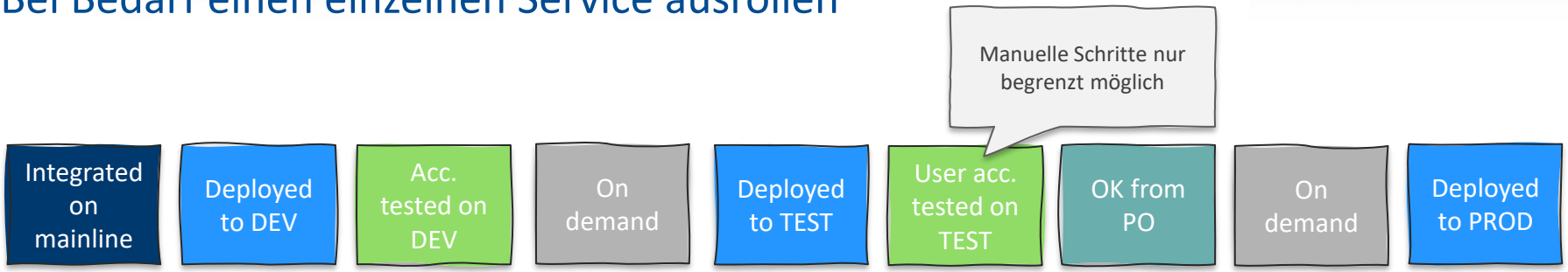
Release Train Pattern

Regelmäßig einen einzelnen Service ausrollen



Deployment On Demand

Bei Bedarf einen einzelnen Service ausrollen



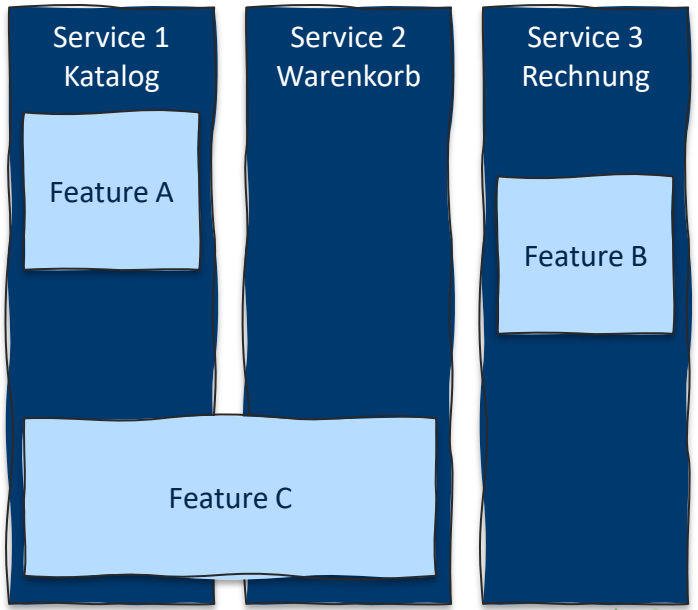
- Kürzere Cycle Time

- Volatiler Stand der Stages
- Zero Downtime-Strategie nötig

#Services per Feature

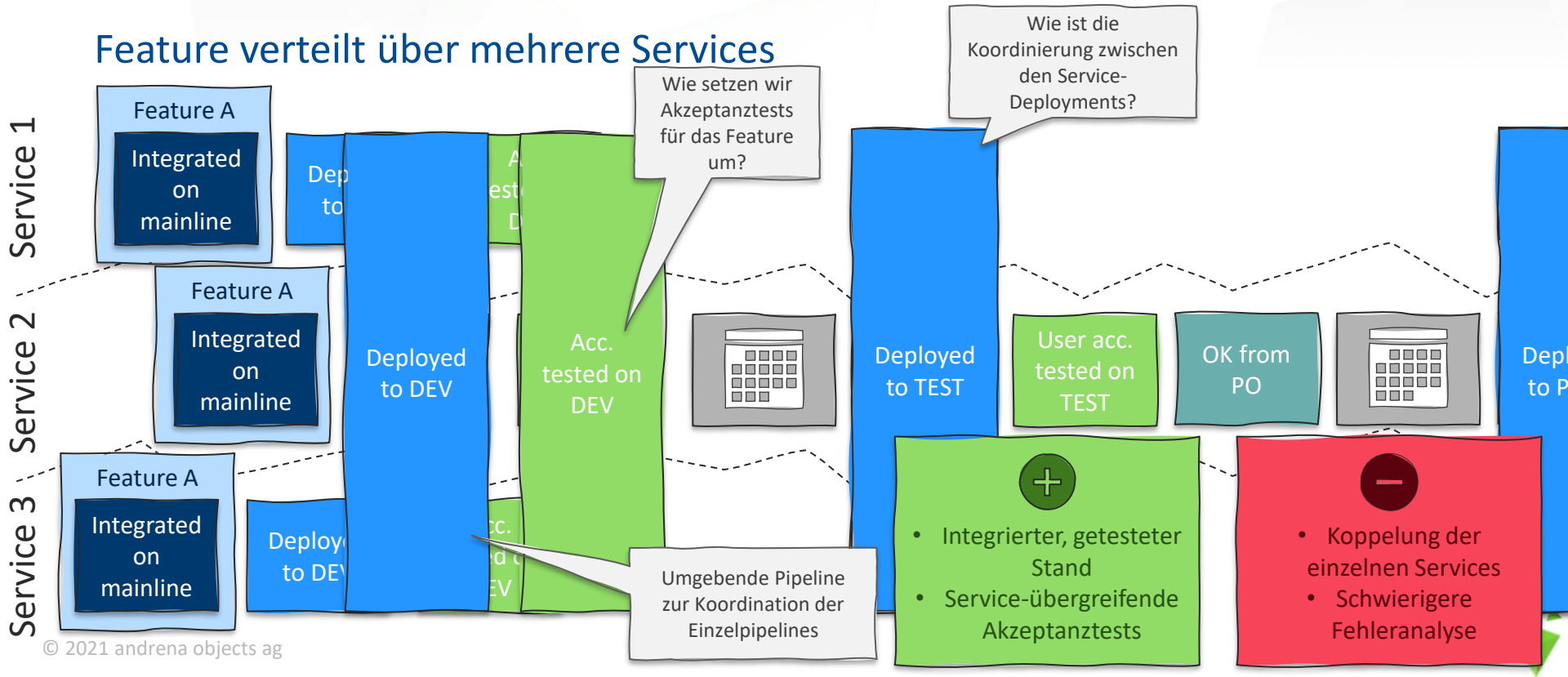
Ein Feature erfordert Änderungen in mehreren Services

- Service = unabhängig deploybar
- Idealerweise: Ein Feature = Änderungen in einem Service → pro Deployment wird ein Service neu ausgerollt
- Was wenn wir nicht in einer idealen Welt sind?
  - Option 1: Zusammenausführen der Einzel-Deployments
  - Option 2: Entkoppeln der Deployments (Schnittstellen-Versionierung, Feature Flags)



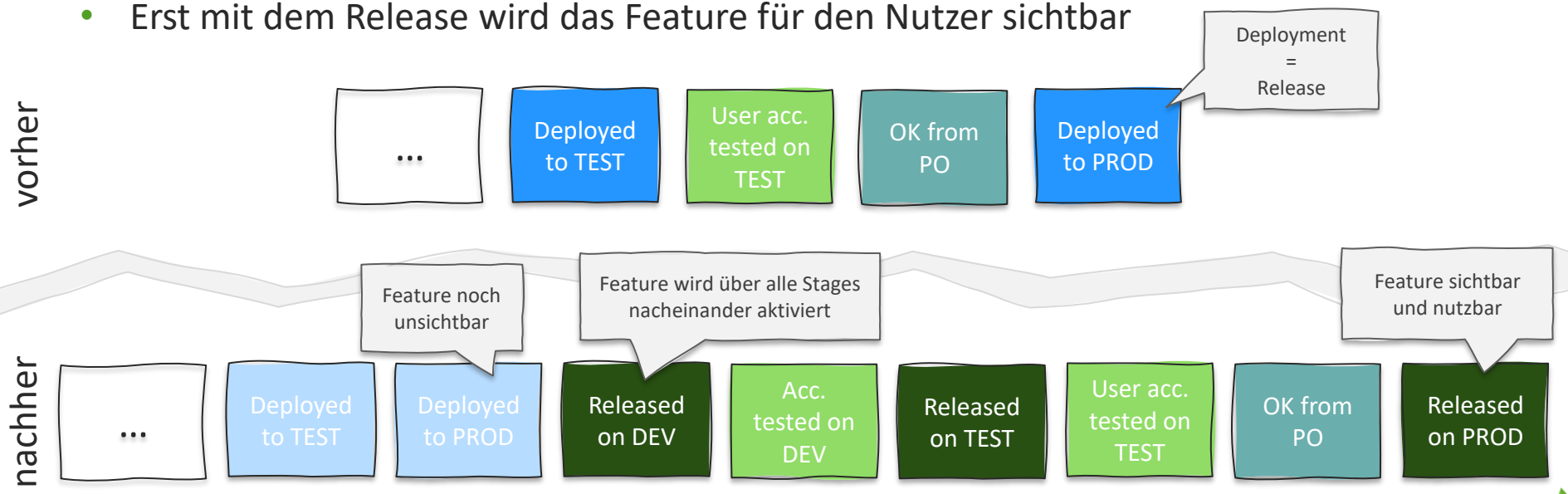
Multi Service Release Train

Feature verteilt über mehrere Services



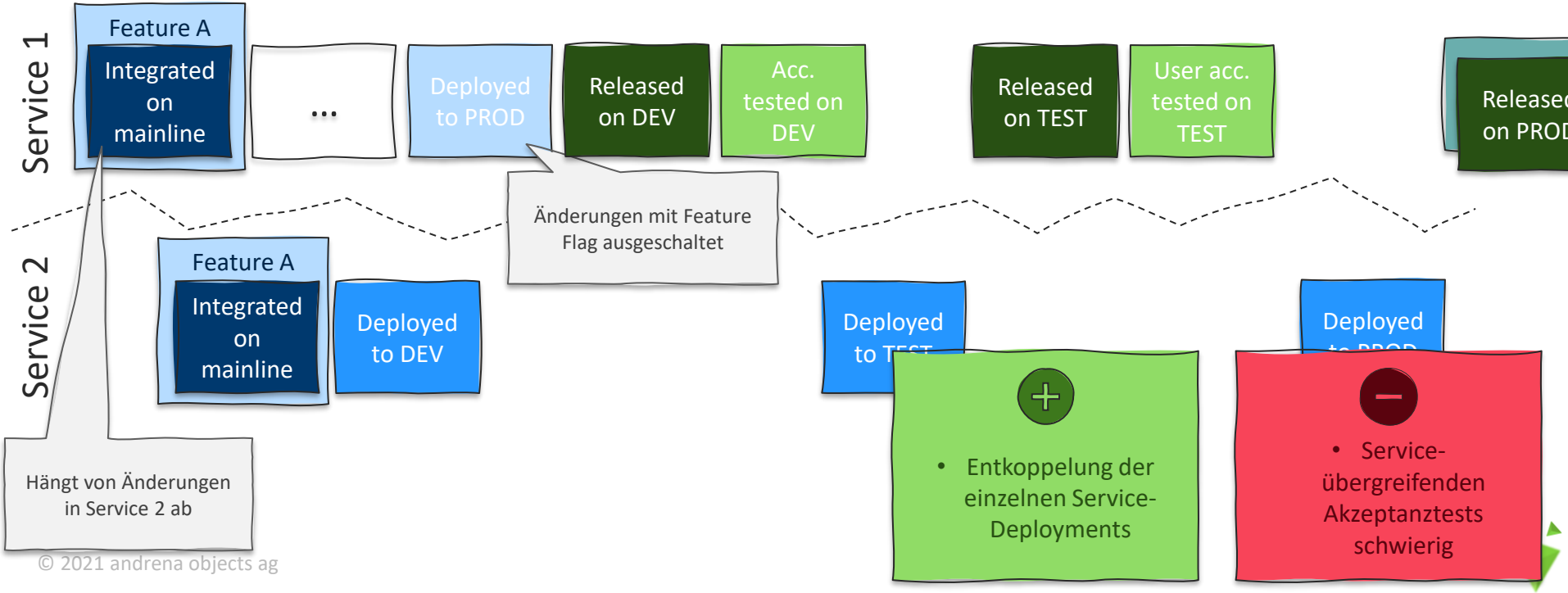
Entkoppelung von Deployment und Release mit Feature Flags

- Deployment ist nur noch technisches Ereignis
- Erst mit dem Release wird das Feature für den Nutzer sichtbar



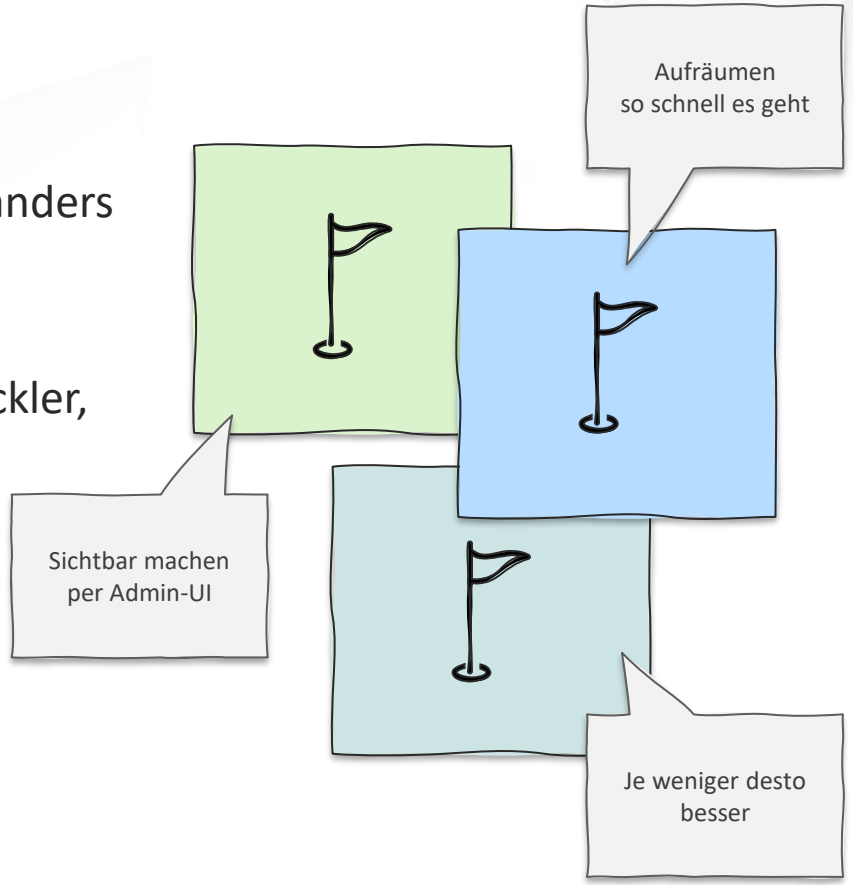
Deployment On Demand

Feature verteilt über mehrere Services

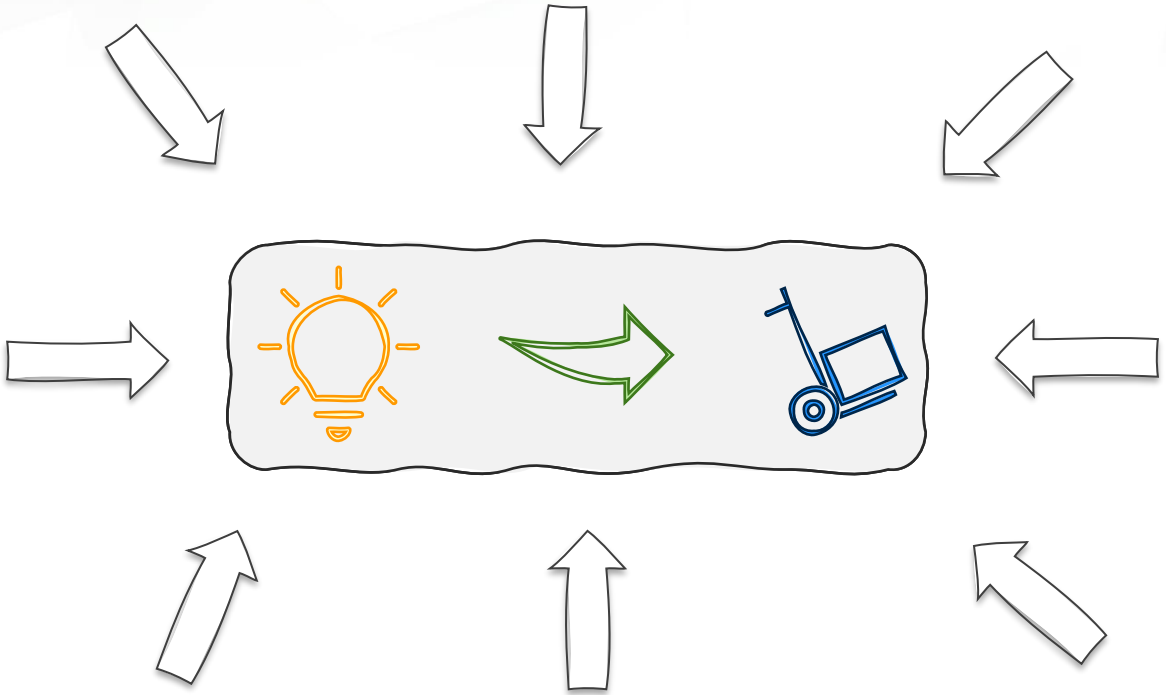


Feature Flags: Die Rettung!?

- Komplexität bleibt erhalten, wird nur anders gemanagt
- Feature Branches sehen nur die Entwickler, Feature Flags die ganze Organisation



Wrap up



Vielen Dank!

Feedback? Fragen?