
Hallo! Willkommen zum Talk über DevOps on Steroids


Hallo! Willkommen zum Talk über DevOps on Steroids


Das hier ist der Original-Eröffnungsslide eines Talks von 2009, in dem John Allspaw - später auch Buchautor in dem Bereich - und Paul Hammond beschreiben, wie sie bei Flickr arbeiten.

# Wie machen die das?

Damals hat jeder gefragt: wie machen die das?

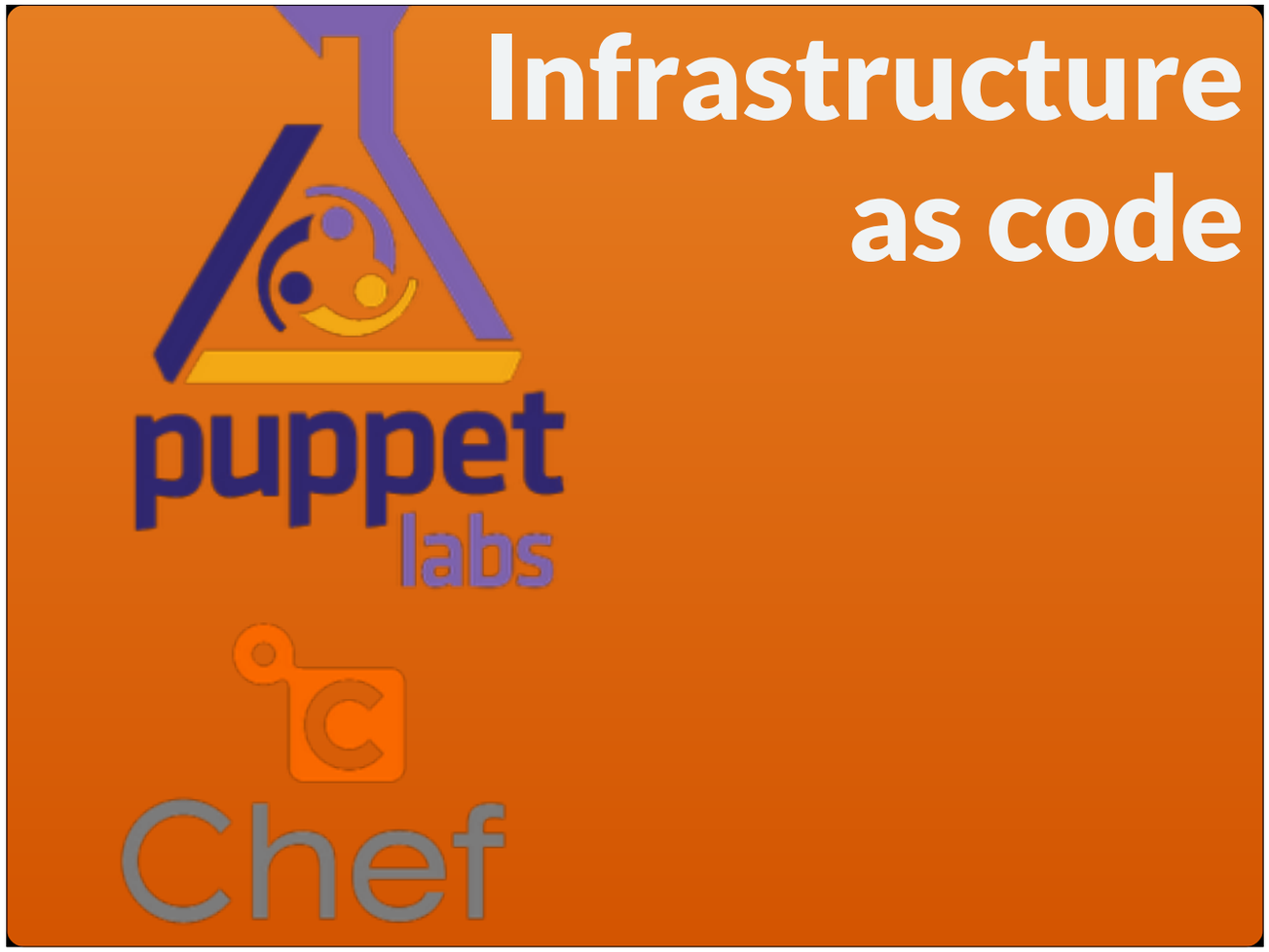

Die Antwort, die die Kollegen von Flickr damals gegeben haben ist folgende: Dev & Ops kooperieren. Und nutzen Automatisierung dabei.

Und sie automatisierten wirklich alles.

**„Hmm, bei mir hat es aber funktoniert.“**

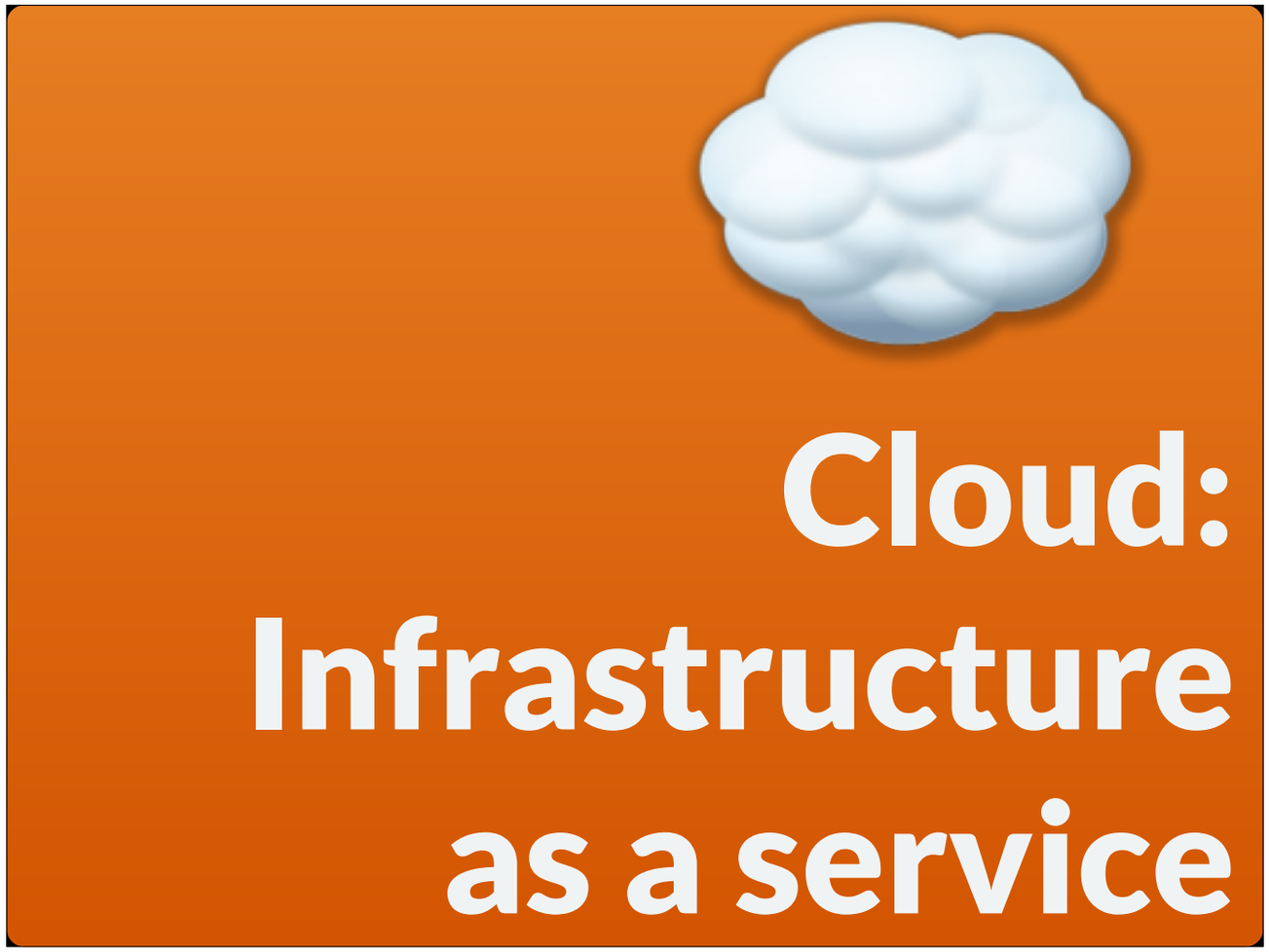
Das haben wir Developer über 1000 Jahre lang gesagt. Und wir kamen damit durch, weil unsere Entwicklungsumgebung immer etwas anderes war als unsere Produktion, und die Differenz bot uns jede Ausrede.


Das ist vorbei, seitdem wir Entwicklungsumgebungen können, die identisch zur Produktionsumgebung aussehen.
Das Development-Environment ist automatisch Konfiguriert, die Setupkosten sind kleiner.




Damit beide wirklich identische sind sollen sie gleich konfiguriert sein. Und Dinge zwei mal gleich machen - das können nur Rechner, wir Menschen sind da nicht so gut drin. Also haben wir das einrichten der Infrastruktur selbst auch noch vollständig automatisiert.
Und damit passierte etwas sehr cooles - plötzlich war unsere Infrastruktur aber keine Hardware mehr, sondern eine Software. es war alles Code, und wir mussten gar nicht mehr wissen, ob es eine wirklich Hardware dazu gibt.


Alle Konfiguration und Software ist in code - und geniesst damit die Vorteile von Versionierung. Alle Konfigurationen sind damit dokumentiert, nachvollziehbar und wiederherstellbar. Jede Änderung kann mit Datum und Grund nachvollzogen werden. Sie kann, wenn die neue Variante nicht funktionieren sollte, einfach wiederhergestellt werden.



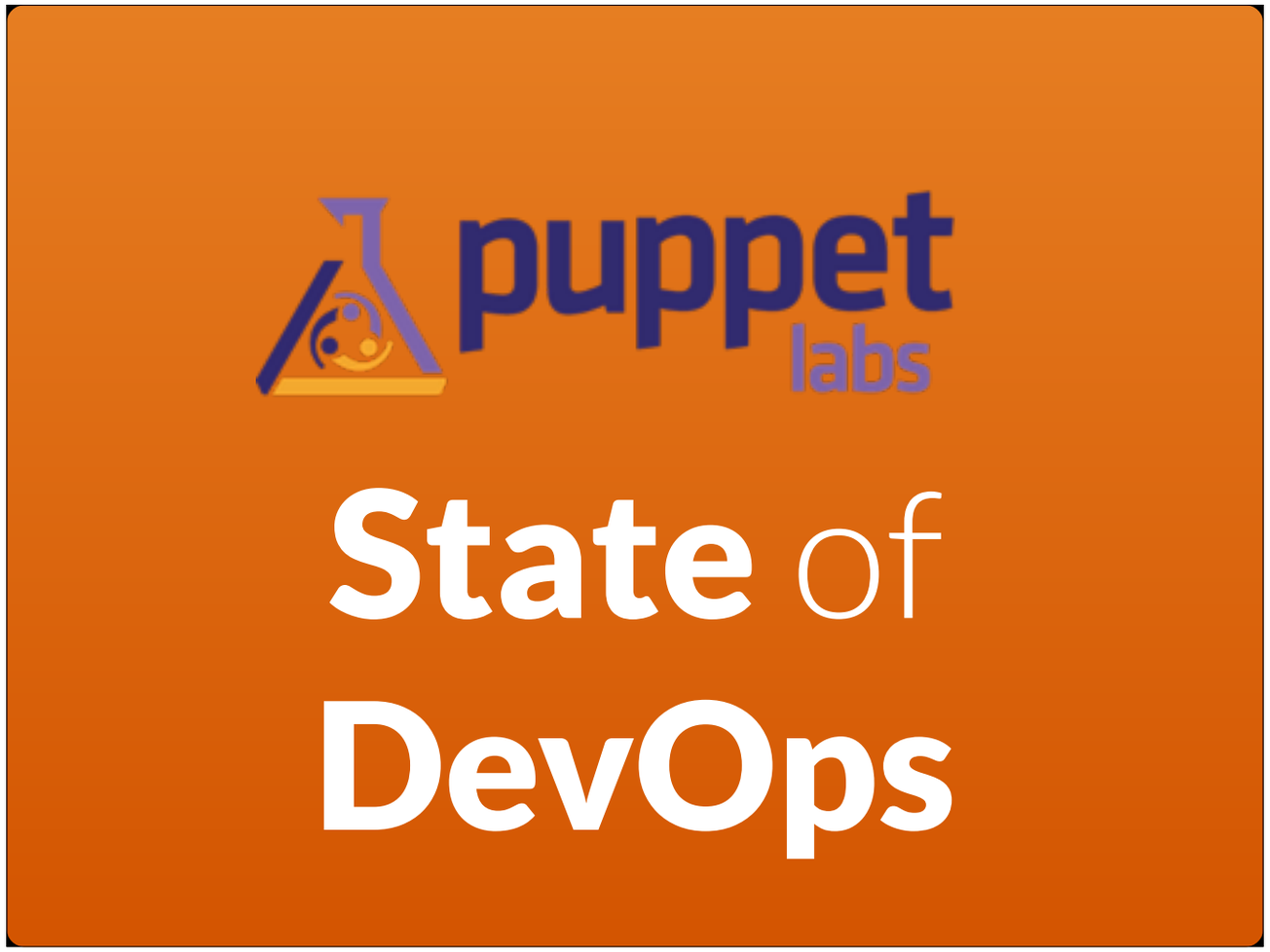

Und Infrastructure as Code hatte das grosse Glück, mit einem anderen Konzept zusammenzufallen - Cloud, oder, mal ganz ehrlich gesprochen, Amazon AWS. Das ist die Killerapplikation für DevOps-Tools gewesen, und jeder hat es gemacht.


Und als dann nicht nur Devops und Infrastructure as Code da war, sondern auch Cloud, da waren die Opser endlich glücklich - weil sie Scotty spielen konnten.


Und man entdeckte damit noch sehr viele Dinge, die man auch auf einmal machen konnte, und die nur wenig Zusatzaufwand kosteten. Sekundärdienste wie Monitoring, Alerting, Reporting etc erlauben das kontinuierliche Verstehen der aktuellen - und zukünftigen - Daten der Plattform.
Bei uns wird das in einer Kombination mit Puppet eingesetzt. Wenn eine neue Maschine entsteht, geht sie automatisch ins monitoring und alerting - die Daten sind nachvollziehbar, recherchierbar und man weiß auch Bescheid, wenn ein Server ausfällt - ohne dass hier irgendetwas zusätzlich konfiguriert werden muss.


So sieht das bei uns in den Projekten in der Praxis aus - praktisch überall wo wir zugange sind findet sich am Ende so ein Screen an der Wand, der das aktuelle Geschäft mit den massgeblichen und gerade relevanten Metriken abbildet. Wir haben Projekte mit 200-300 Graphen. Nicht weil wir sie brauchen, sondern weil es so preiswert geworden ist, on demand Analytics zu liefern.


Das hat für Operations von Webapplikationen praktisch alles geändert - während man vorher quasi im Blindflug operiert hat und man erst mit den Katastrophen selbst gemerkt hat das etwas nicht stimmt, war man jetzt im Instrumentengestützten Flug unterwegs, und wusste ganze genau, was jetzt geht und was nicht geht.




Puppet Labs, Hersteller der dicken Infrastructure as Code-Lösung Puppet, fragt einmal jährlich den State of DevOps ab.


Bei den Umfragen geht es ihnen nicht nur um den State, sondern auch wie sich die Highperformer von den Lobperformern unterscheiden.
Das sind die Unterschiede: es wird 30 mal häufiger deployed, und der Weg von Commit in die Produktion ist 200 mal schneller als bei den Low Performern.

… with **60 percent** fewer failed deployments and a **mean time to recover** (MTTR) that's **168 times faster.**

puppet labs

**Puppet Labs, 2015**

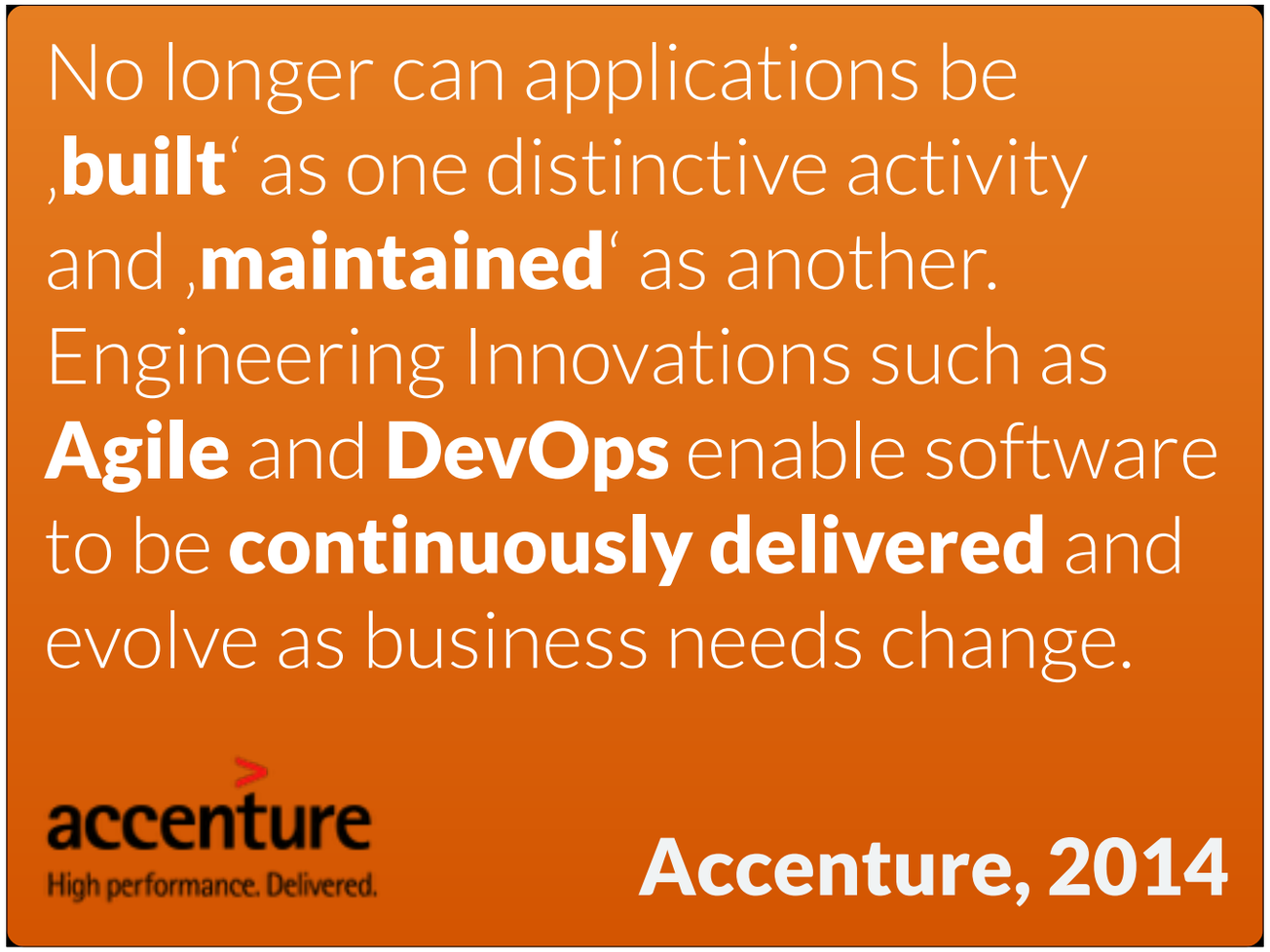
Und nicht nur das - die Deployments schlagen häufiger fehl, und wenn ein Fehler passiert, dann wird er viel schneller korrigiert.

It's their **use** of **DevOps practices** that sets these **top performers** apart from the pack.

puppet labs

**Puppet Labs, 2015**

Und wenn man nach der Ursache schaut, warum die das können - dann wird man genau bei den DevOps-Praktiken fündig.




Inzwischen haben es auch die grossen Unternehmensberatungen spitz bekommen. Tatsächlich kann man offensichtlich mit solchen Innovationen wie dem 14jährigem Teenager Agile und dem Grundschüler DevOps Continuous Delivery machen.

**Development** to **Operations** (**DevOps**) implementations will **increase significantly** during **2015-2016**.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

**Cap Gemini, 2014**

Auch Cap Gemini geht davon aus, dass DevOps genau jetzt immer wichtiger wird. Interessanterweise nennen sie DevOps hier „Development to Operations“.


Man kann  DevOps also als Erfolgsrezept betrachten - es hat sich durchgesetzt und bildet in unserer Welt den Standard. Leider ist aber eine Frage verlorengegangen auf dem Weg bis heute.

10 deploys per day
Dev & ops cooperation at Flickr

John Allspaw & Paul Hammond
Velocity 2009

# 10 Deploys am Tag?

# Warum nur?

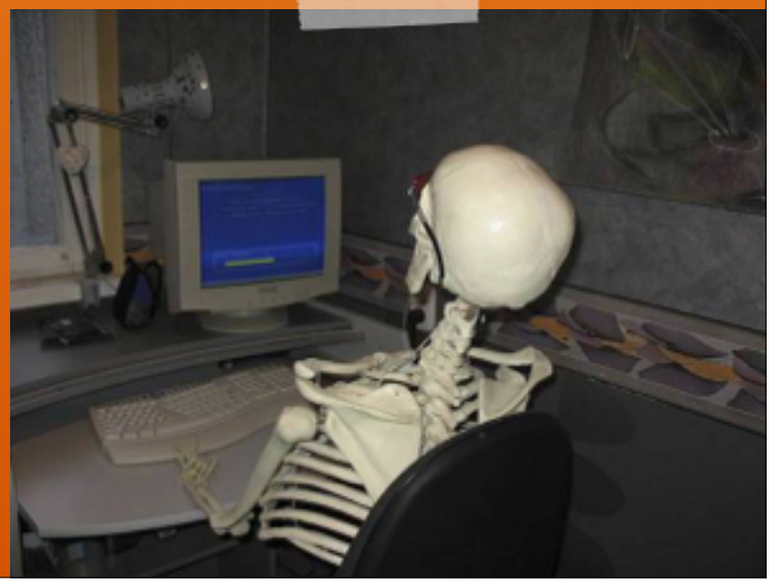
# **Features schneller** von der **Idee** zum **Kunden**


DevOps sollte Features schneller zum Kunden bringen, und damit das ganze Unternehmen beschleunigen.

# Performance- und User**feedback schneller** an **Ops, Dev & Biz**


Und nicht nur da soll es schneller werden - auch der Weg der Information zurück soll beschleunigt werden.

# Weniger **Wartezeiten** auf **Dev, QA, Ops**, andere Teams, **Requirements**



Ein sicherer Weg zu mehr Geschwindigkeit ist natürlich auch weniger Warten.

# Die Verlässlichkeit der Applikation erhöhen.

Gleichzeitig soll es die Verlässlichkeit erhöhen. Weniger Ausfälle, weniger seltsames Verhalten, besseres Skalieren etc.

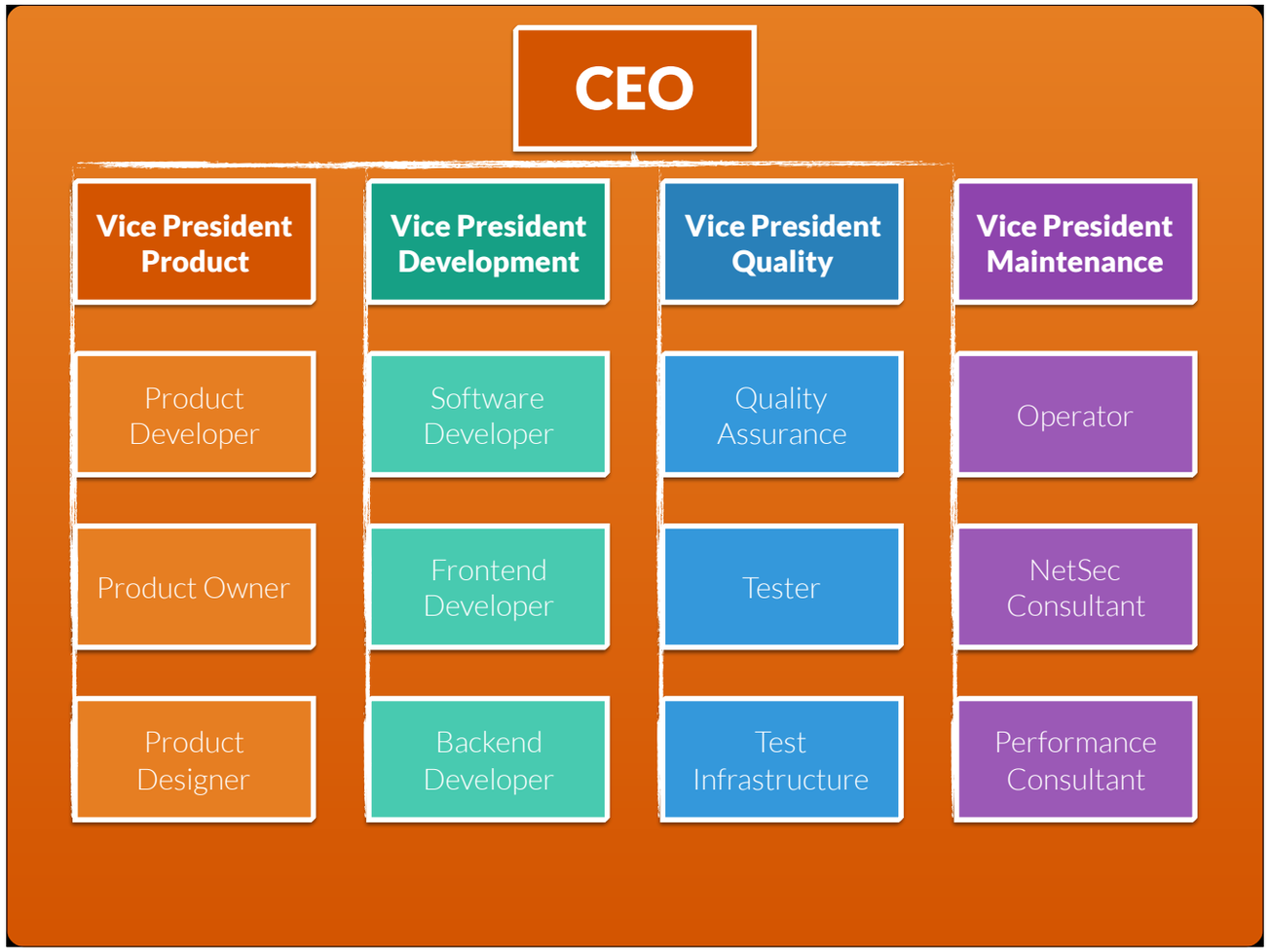

Damit ich all das genannte optimieren kann, reicht es nicht, wenn ich DevOps nur zwischen Dev und Ops mache - denn damit könnte ich nur einen kleinen Teil verbessern. DevOps bedeutet in Wahrheit DevProductSuportNetSecBizOps. Es geht über alle Bereiche. Warum bestehen die dadrauf?

**Das ganze Unternehmen soll schneller werden.**

**Nicht die Abteilung.**

# Silo-Effekt


Wer kennt den Silo-Effekt? Der tritt, wie auch seine kleine Schwester Silo-Denke, vor allem in grossen Unternehmen auf. Dort wird wenig über Abteilungsgrenzen hinweg kommuniziert, aber das gibt es auch in kleineren Unternehmen - bei uns gibt es zB Team-Silos, die wenig mit der Aussenwelt kommunizieren, oder Abteilungsilos wie Sales.




Und es ist Absicht, dass diese Silos entstanden sind. Dahinter steht die Idee der Funktionalen Organisation. Ich trenne die Spezialisten nach Abteilungen, und die haben jeweils auch einen Chef, der sich damit auskennt.


Diese Abteilungen haben an der Spitze jemanden, der sich gut mit dem Thema auskennt, und darunter auch Spezialisten - die sich gut mit Ihrer Materie auskennen.
Sie übernehmen die Aufgabe Product Development für das ganze Unternehmen, und sind dafür auch Accountable. Dafür haben sie vom CEO Ziele bekommen, die sie zu erfüllen haben, und definieren ihre eigenen Prozesse und planen das auch selbst. Die sind pfiffig, also sind die Prozesse reif und die Arbeit ist gut.


Dummerweise gibt es an den Aussenkanten immer Ärger. Und da kommt die DevOps-Kultur her.
Development will, vom Business damit beauftragt, schnelle neue Features in kurzer Zeit. Dafür bekommt sowohl der Vice President Product als auch der Vice President Development Ihren Jahresbonus.
Auf der anderen Seite spielt Stabilität die entscheidende Rolle - es soll wenig Bugs geben, wenig Reklamationen, wenig Ausfälle und eigentlichen einen 100% stabilen & kontinuierlichen Regelbetrieb.

# Fingerpointing


Das kann natürlich nicht klappen, und im Resultat erzeugt man Fingerpointing - die Administration wird als Diktator gesehen, das Development als unfähige Bugschleudern.

# Agile

löst die **Silo-Effekte** zwischen

- **Requirements** und
- **Development** und
- **Qualität**

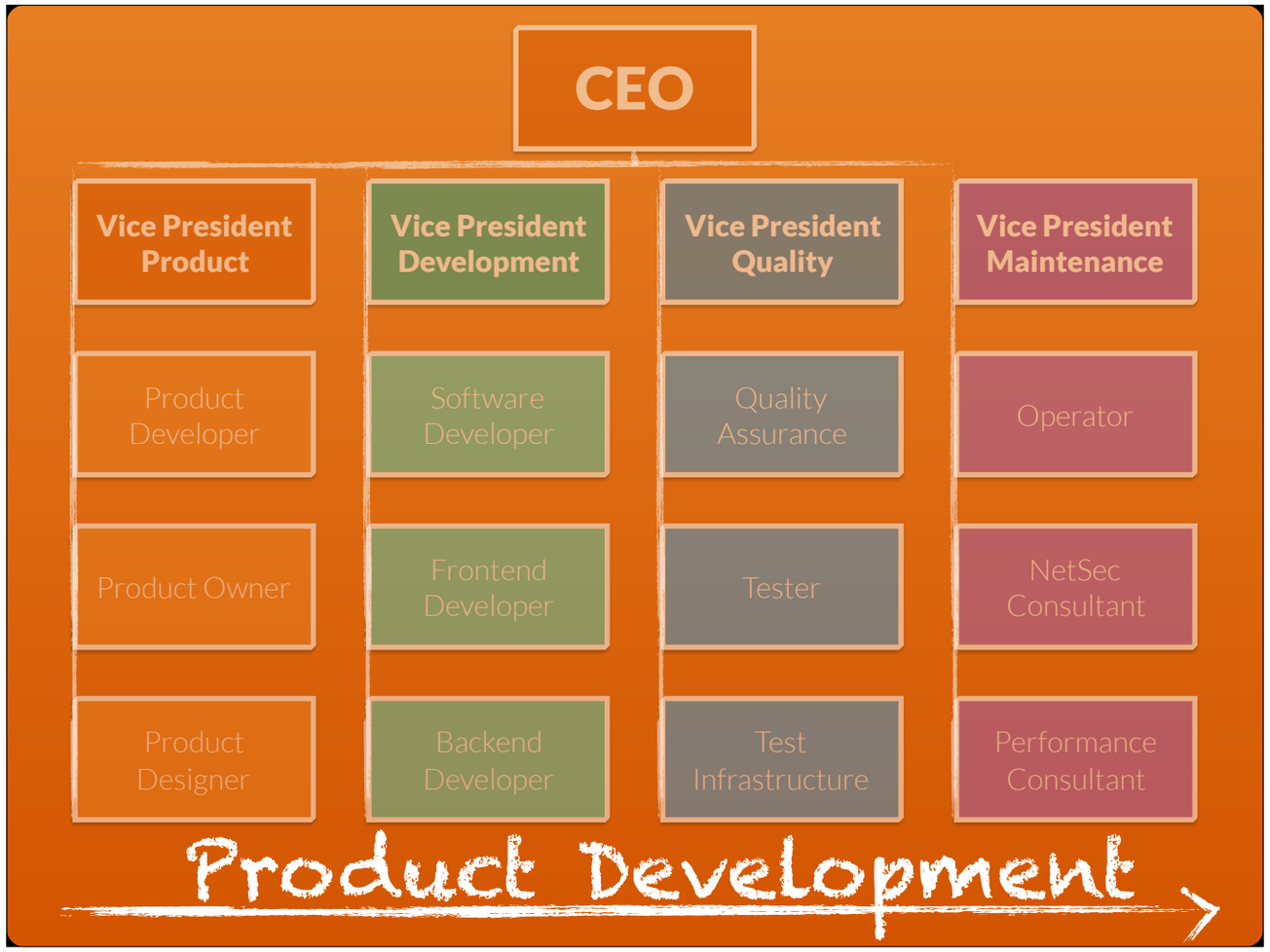
Agil hat schon Silo-Brecher-Effekte gehabt - Requirements, Development und Qualität greifen bei agilen Methoden hart ineinander.

# DevOps

löst die **Silo-Effekte** zwischen

- **Requirements** und
- **Development** und
- **Qualität** und
- **Deployment** und
- **Maintenance** und
- **Operations**

DevOps spannt diesen Bogen noch deutlich weiter - und nimmt Deployment, Maintenance und Operations ebenfalls mit dazu.




In der Praxis ergibt das natürlich deutlich Sinn - denn die Produktentwicklung betrifft im Regelfall alle diese Bereiche.

# Silos abbauen

Wie reisse ich ein Silo ab?

Direkte **Kooperation**
statt **Abteilungsgrenzen**

**Diskussionen &** Debatten
statt **Handovers & Prozessen**

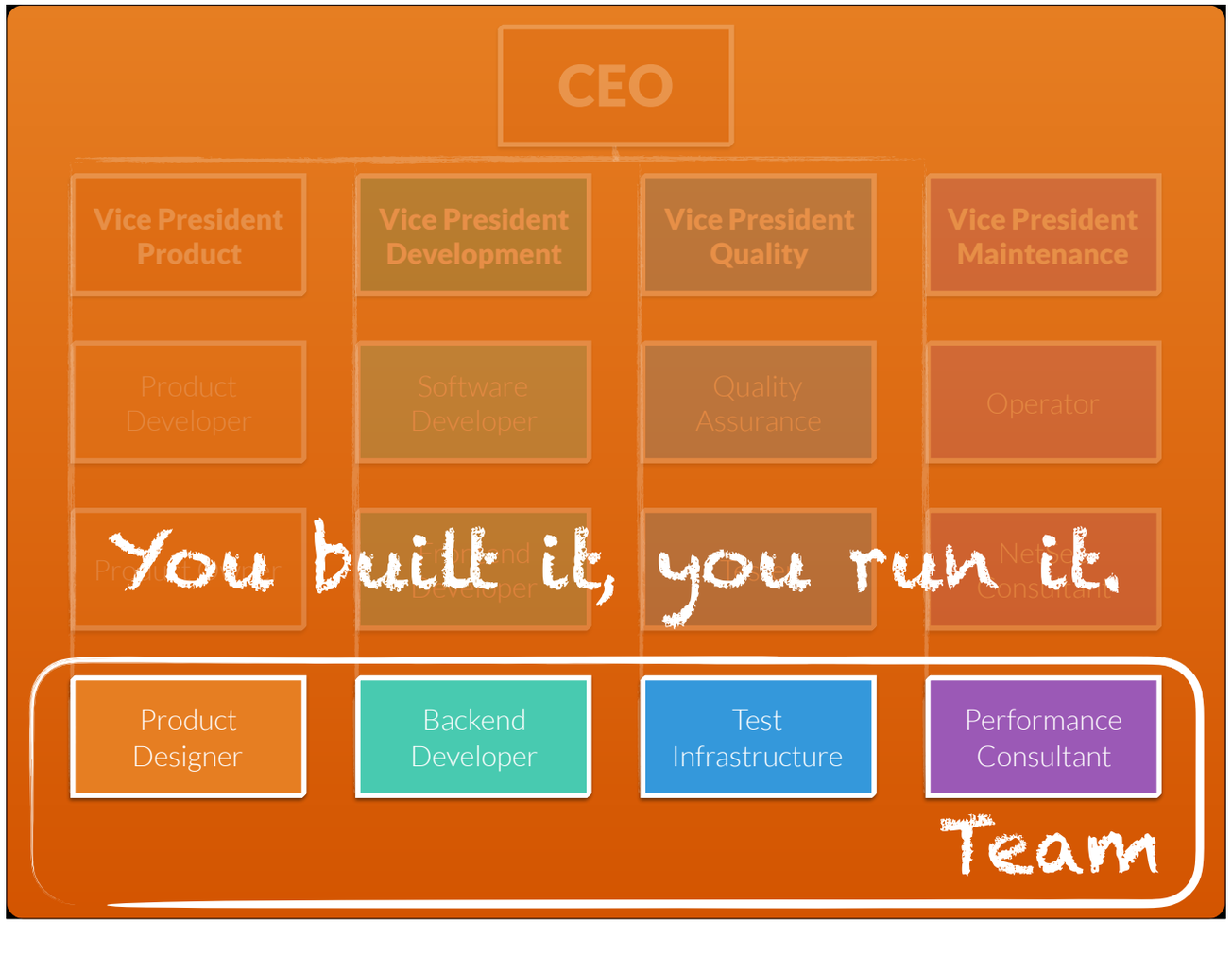
Die einfaches Variante ein Silo zu sprengen ist es zu ignorieren. Ich arbeite nicht in den Grenzen meiner Abteilung, sondern kooperiere direkt. Alle Diskussionen und Debatten werden direkt mit den anderen Ansprechpartnern geführt, und nicht nur innerhalb meines Development-Departments. Wer war gestern in Judiths Talk? Genau, diese Art von direkter Kommunikation ist gefragt.

# Gemeinsame Themen

- Requirements
- Businessmetriken
- Zeitplan / Releaseplanung
- technische Ressourcen
- Architektur


Und dort werden alle Themen diskutiert, die bereichsübergreifende Effekte haben. Das bedeutet nicht nur Requirements, sondern auch die Metriken um sie zu messen. Das bedeutet Release-Planung, Architekturplanung, die technischen Ressourcen, die eingesetzten werden sollen.

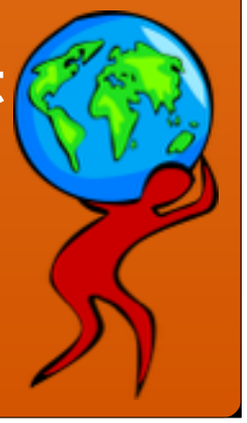

Nukleus dieser Zusammenarbeit ist das autonome Crossfunktionale Team, das minimal alle Development, QA- und Ops-Aspekte als Kompetenz und Entscheidungsfähigkeit enthält. Sie können selbstständig agieren, und brauche nicht externe Rückfragen zu stellen.




Die Idee dazu kommt - natürlich - aus der agilen Softwareentwicklung, dahinter stehen grossfunktionale Teams. Bei DevOps wird das aber noch erweitert - in „You built it, you run it.“

# Geteilte Verantwortung

- **Verantwortung** für das **Produkt** statt für die **Abteilung**
- **Dokumentation & Ticketing** ist ein **Werkzeug, kein Vertrag**
- **Handover** ist **implizit,** nicht **explizit**

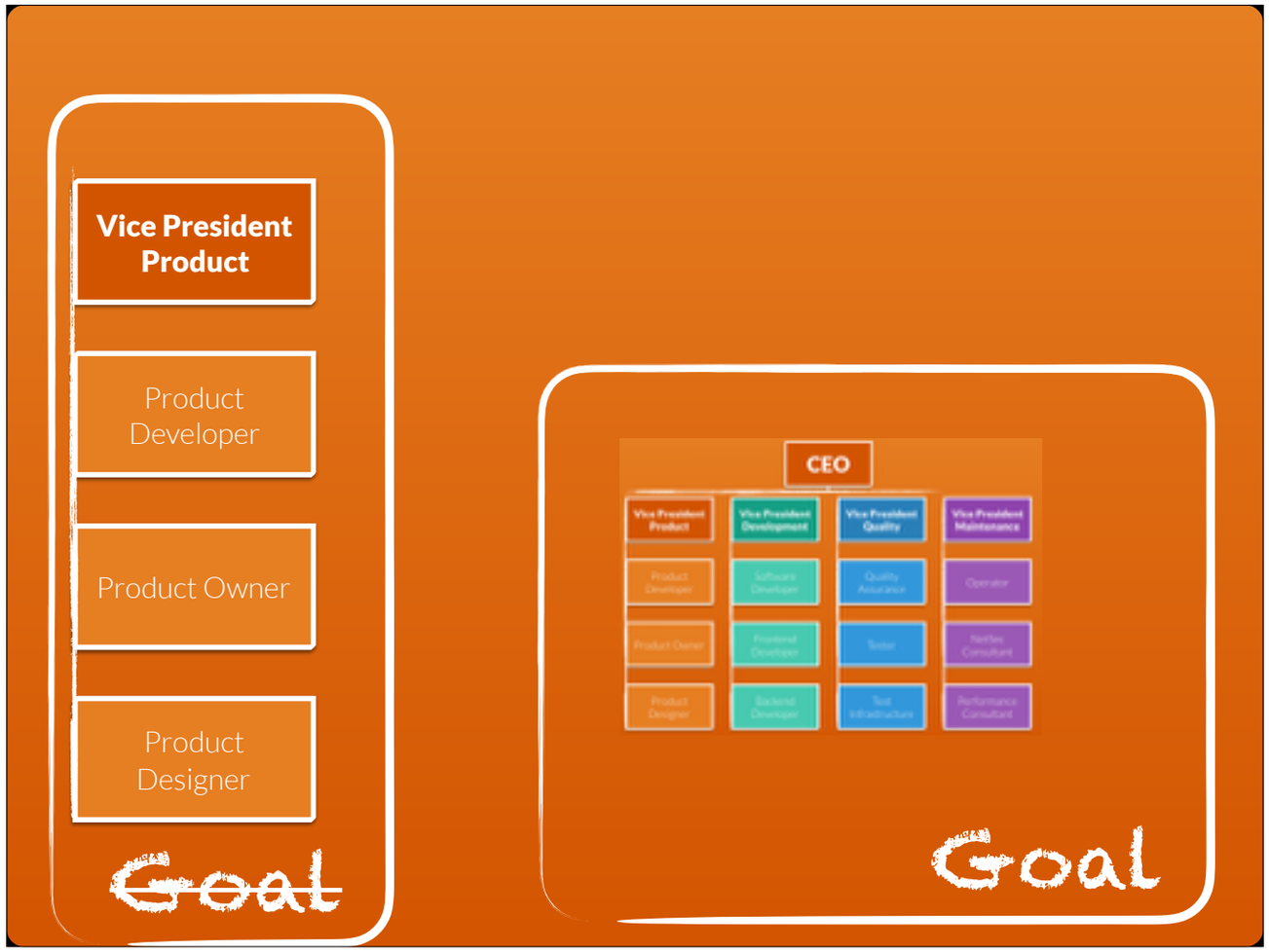


DevOps-Kultur zielt mit der Verantwortung nicht auf lokale Verantwortung - sondern auf shared accountability für das Gesamtsystem.

# Geteilte Ziele

- Fokus auf
  - Produkt
  - Gesamtprozess
- Gemeinsame Metriken
  - Produktivdaten
  - Nutzungsdaten
  - Build- & Qualitätsdaten

Damit das funktioniert brauche ich gemeinsame Ziele - mit dem Fokus auf das Produkt und auf den Gesamtprozess. Und ich brauche gemeinsame Metriken, um die Bewegung in Richtung dieser Ziele zu verstehen.




Abteilungsziele stehen DevOps unmittelbar im Weg - denn sie erlauben nicht, dass man das Gesamtsystem optimiert.
Der Fokus bei DevOps ist das Gesamtsystem, sprich: das Produkt selbst und alle Prozesse im Unternehmen.

# Respekt & Vertrauen

(erwiesenermaßen am schwersten)

**„Product Design** baut das **richtige Produkt“**
**„Dev** baut das Produkt auf **die richtige Art.“**
**„Ops** liefert **echte, relevante Metriken.“**

Da ist insbesondere die Grenze in andere, nicht-technische Domains schwierig, denn ich verstehe zu wenig, um Anlass zu vertrauen zu haben. Wie man das trotzdem macht zeige ich später.

# Automatisierung

- **Gründe für Automatisierung:**
  - es ist **„State of the Art „**
    - ich las einen **Blogartikel**
    - ich hörte einen **Konferenzvortrag**
    - ich will es auch **ausprobieren**
- **weil ich es kann!**
- **was mit Cloud drin**

Und natürlich ist Automatisierung ein wichtiger Teil auch der DevOps-Kultur.
Wichtig ist hier aber das Ziel. Wenn man mit anderen Entwicklern redet, dann hört man heimlich oft diese Gründe.

# Automatisierung

- weil es dem **Kunden** nutzt!
  - höhere **äussere Qualität**
  - **weniger** Fehler
  - hohe **Verlässlichkeit**
  - hohe **Stabilität**
  - schnelles **Feedback**
  - schnelle **Featureentwicklung**
  - schnelle **Anpassung**

Das eigentlich Ziel ist aber ein ganz anderes. Automatisierung soll das Produkt selbst, das Verhalten, den Gesamtprozess und das Feedback verbessern.

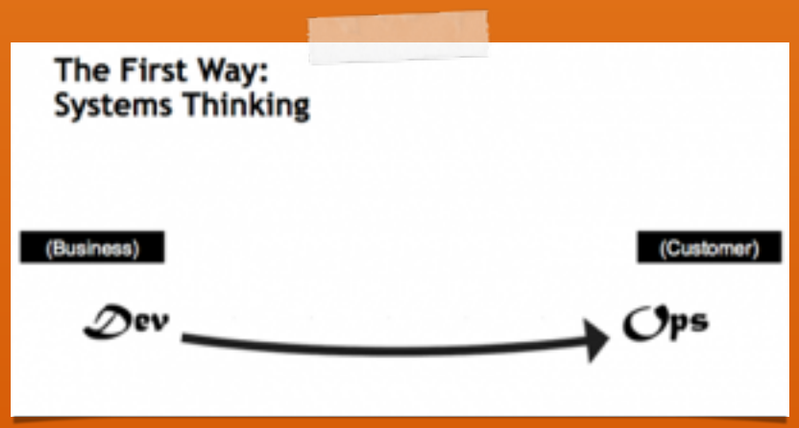

# Widersprüche

| Klassisch | DevOps |
| --- | --- |
| Klare Accountability | Globale Verantwortung |
| Abteilungsziele | Gemeinsame Ziele |
| Persönliche Ziele | Gemeinsame Ziele |
| Eigenen Prozess optimieren | Globalen Prozess optimieren |

Der Haken an der Sache ist: das ganze steht oft im Widerspruch zur bisherigen Unternehmenskultur. Klare Accountability „jemand hat den Hut auf“ widerspricht einer gemeinsamen Verantwortung. Wenn der mit dem Hut auf einen Fehler macht, ist das in DevOps auch mein Fehler - denn hier geht es um geteilte Verantwortung. Abteilungsziele und persönliche Ziele - jemand mit Jahreszielen anwesend? - stehen in Widerspruch zu gemeinsamen Zielen. Welche soll ich erfüllen wenn ich die Wahl habe? Die Verbesserung des eigenen Prozesses steht im Widerspruch zum gemeinsamen Prozess.


Netterweise gibt es da inzwischen gute Ansätze. Der bekannteste sind die 3 Ways of DevOps, die auch demnächst im DevOps cookbook herauskommen.

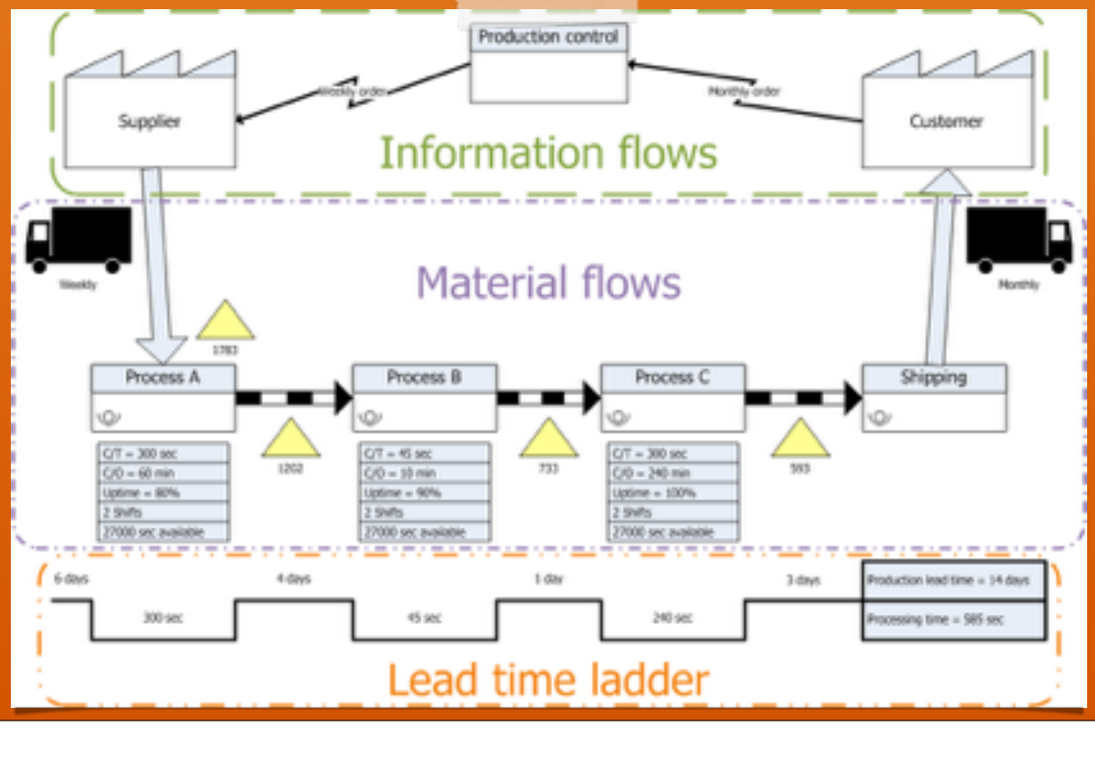
# 1 Systems Thinking




Der erste Ansatz ist Systems Thinking, also der gemeinsame Blick auf das Gesamtsystem, um es zu verstehen und verbessern zu können.


… eine andere Methode - „Draw how to make toast“. Da gibt es einen hervorragenden 10 Minute-TedTalk dazu, und eine Website auf der alles notwendige erklärt wird - es gibt sogar diverse Templates. Ich hätte den Film gerne heute gezeigt, aber die 10 Minuten bekommt ihr auch anders hin.

# 1 Systems Thinking




Eine professionelle Variante dazu ist Value Stream Mapping, bei dem die Prozesse eines Unternehmens in Folge betrachtet werden. Das ist eher nontrivial, deshalb empfehlen wir

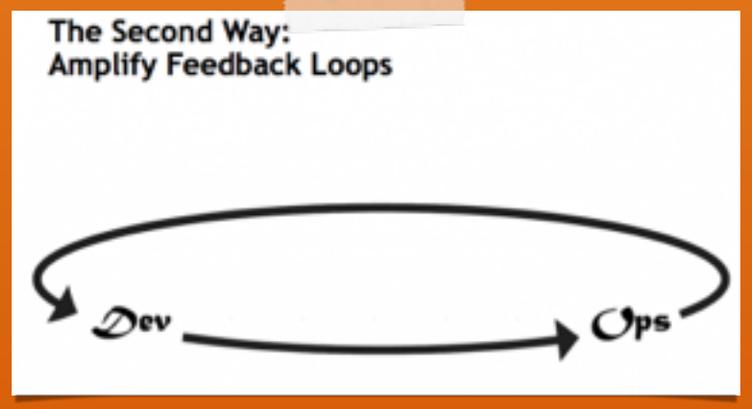
# 1 Systems Thinking

Sometimes it **hurts** to look in a **mirror**.

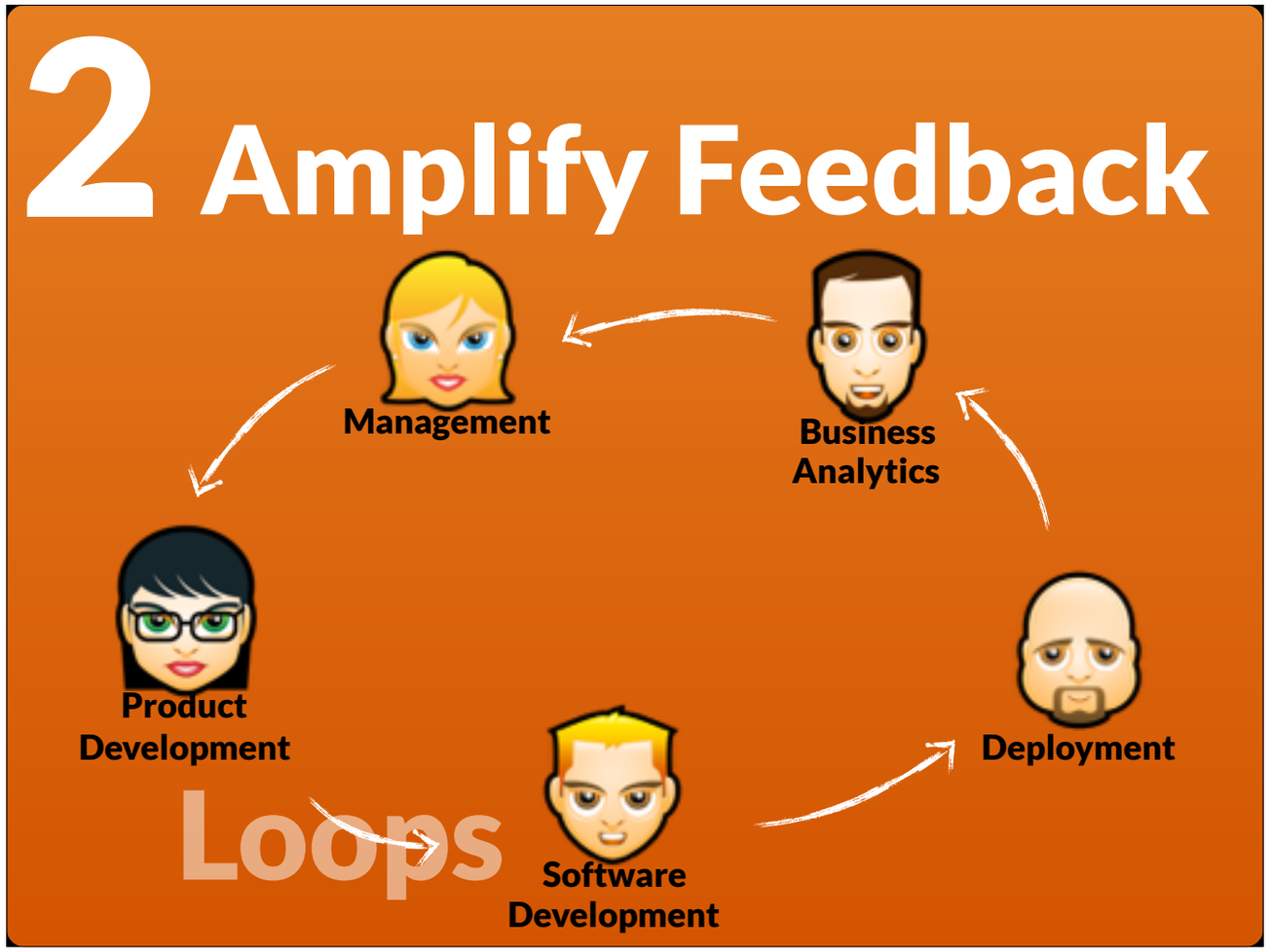

Viele Unternehmen wehren sich dagegen, weil sie diese Transparenz nicht ertragen - zB weil die eigenen Fehlfunktionen und Schwächen ebenfalls transparent gemacht werden. Ohne die Transparenz über die Abteilungsgrenzen hinweg kann ich aber kein DevOps haben.

Quelle: https://www.flickr.com/photos/jox1989/

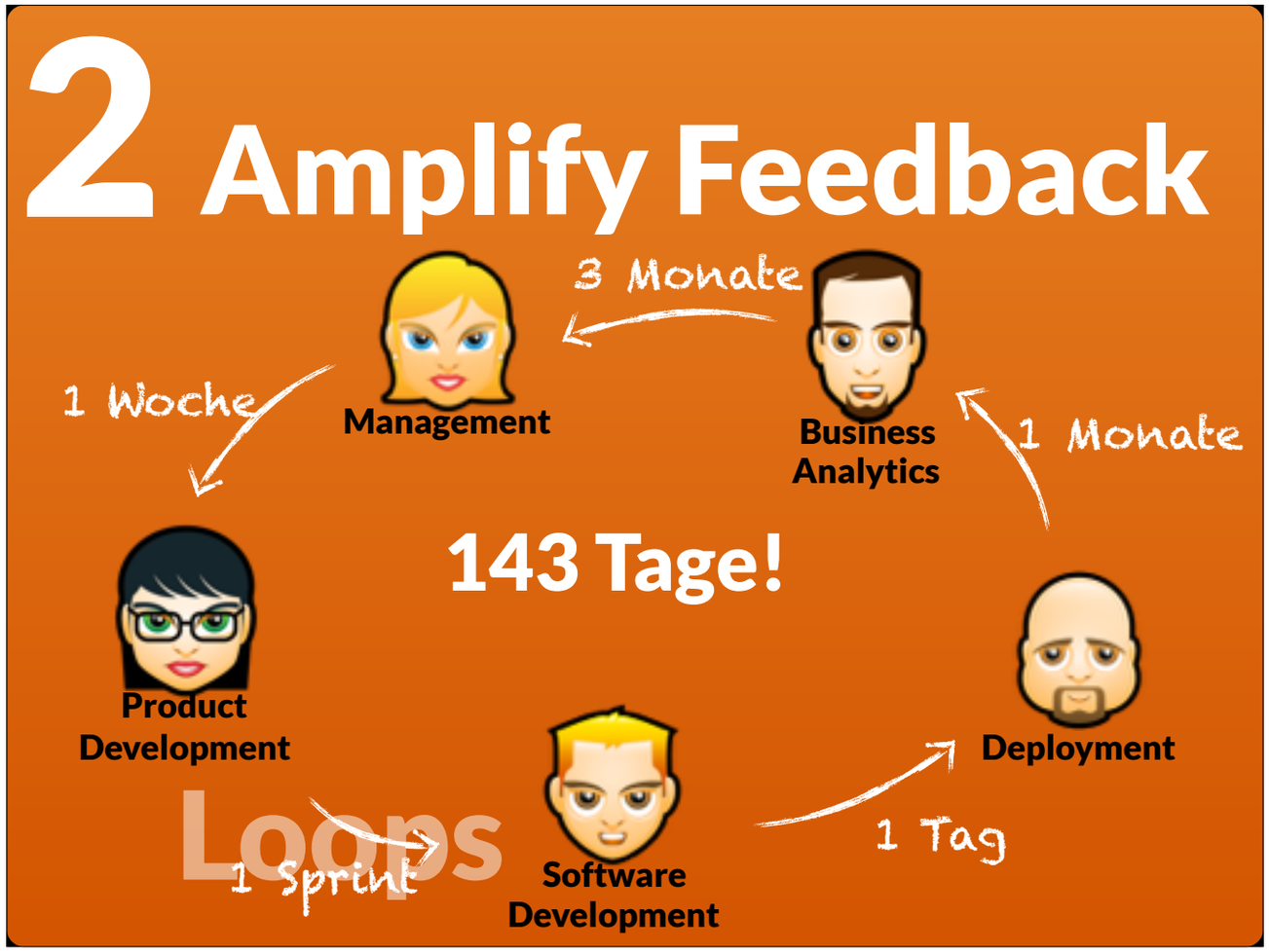
# 2 Amplify Feedback Loops



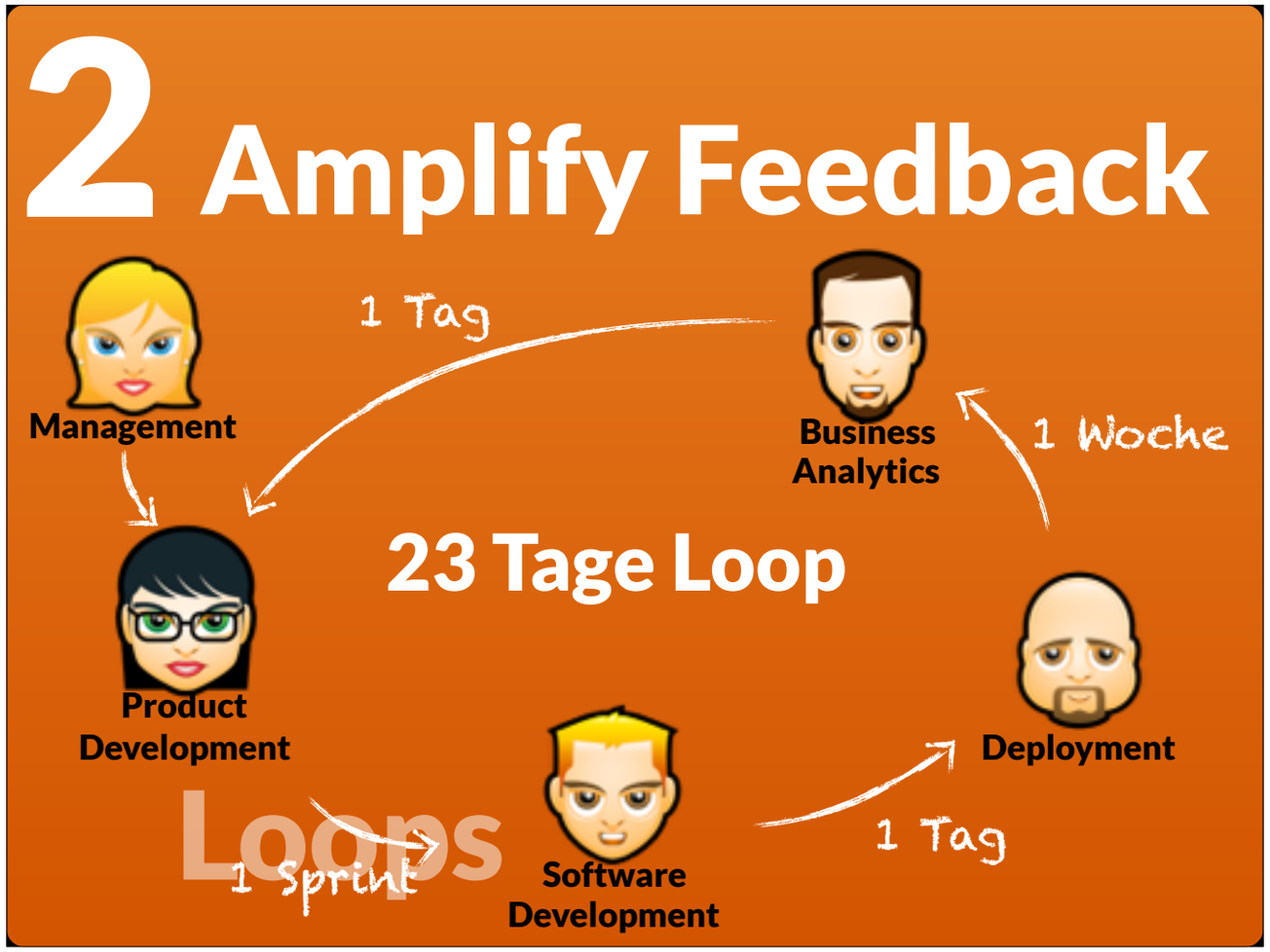

Der zweite Weg ist die Verstärkung von Feedback Loops. Dazu muss ich erst mal welche haben, und zwar über die gesamte Strecke von Rechts nach links.




Das ist ein typischer Feedback-Loop in einem IT-Unternehmen. Das Management entscheidet, wie das Produkt sich weiterentwickeln soll, das Product Development entwickelt und konzipiert es, das Development baut es, die Admins deployen es und der Business-Analytics-Mensch misst, ob das schlau war - und sagt dann dem Manager Bescheid.

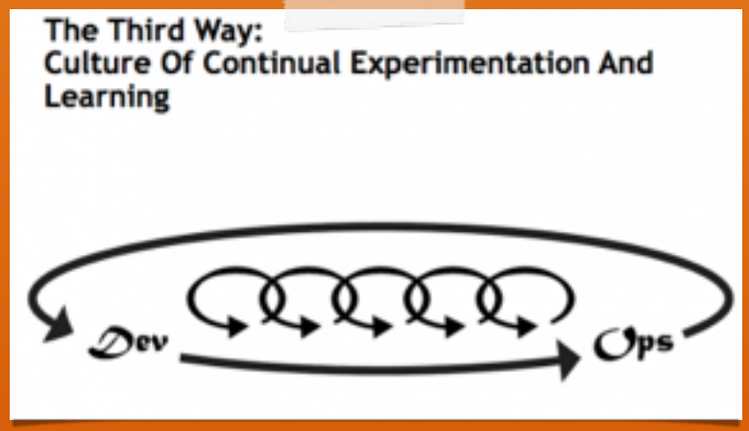



Der Nachteil an so einem Feedback-Loop ist das Tempo. Nach der Vorstandssitzung dauert es eine Woche, bis das Product Development was davon erfährt, das kann beim Entwickler immer nur pro Sprint einmal neue Dinge einkippen, der Deploy geht schnell - denn wir machen DevOps - dann braucht der Business-Analytics-Guy aber wieder einen Monat, um relevante Daten zu sammeln, und sein Bericht muss dann auch wieder bis zur nächsten Vorstandssitzung warten, damit das Produkt angepasst werden kann. Die beste Reaktionsgeschwindigkeit so eines Loops liegt in diesem Fall bei 143 Tagen - offensichtlich kann man so nicht schnell arbeiten und lernen oder den Prozess verbessern.




Der nächste Schritt das kürzen & beschleunigen dieses Loops. Wenn das Management zB nicht mehr im Detail mitmischt, das Business-Analytics-Feedback schneller kommt - dann wird der Feedback-Loop um mehr als Faktor 6 beschleunigt, und sowohl Produkt als auch Prozessverbesserungen finden schnell statt. Aber Prozessverbesserung ist nur die eine Hälfte - die andere hälfte zu guten Prozessen und guten Produkten ist Innovation. Und wie bekomme ich Innovation?

# 3 Culture of Continual




# Experimentation & Failure

Der letzte Schritt ist die „Culture of Continual Experimentation & Failure“.

# 3 Culture of Continual

# Fail cheap

# Fail often

## Experimentation & Failure

Dahinter steht eine alte Idee von uns Webleuten: fail cheap & fail often.
Wenn ich die Feedback-schleife kurz habe, und einen schnellen Prozess habe. dann kann ich auch preiswerte Fehler machen. Und ich lege es nicht mehr darauf an, das beste theoretische Produkt zu erzeugen - sondern das beste am Kunden selbst getestete & validierte.

# 3 Culture of Continual

# Sichtbarkeit

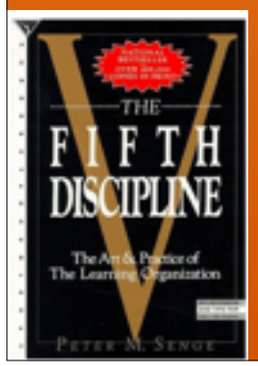
# Resilienz

# Experimentation & Failure

Damit ich das bekommen kann brauche ich hohe Transparenz und Sichtbarkeit - und das muss meine Firmenkultur ertragen können. Und netterweise passiert das auch, aber über Zeit. Um so mehr ich transparent aus Fehlern lernen kann, um so resilienter wird mein Unternehmen. Und um so besser verstehe ich die Kollegen.

# 3 Ways of DevOps

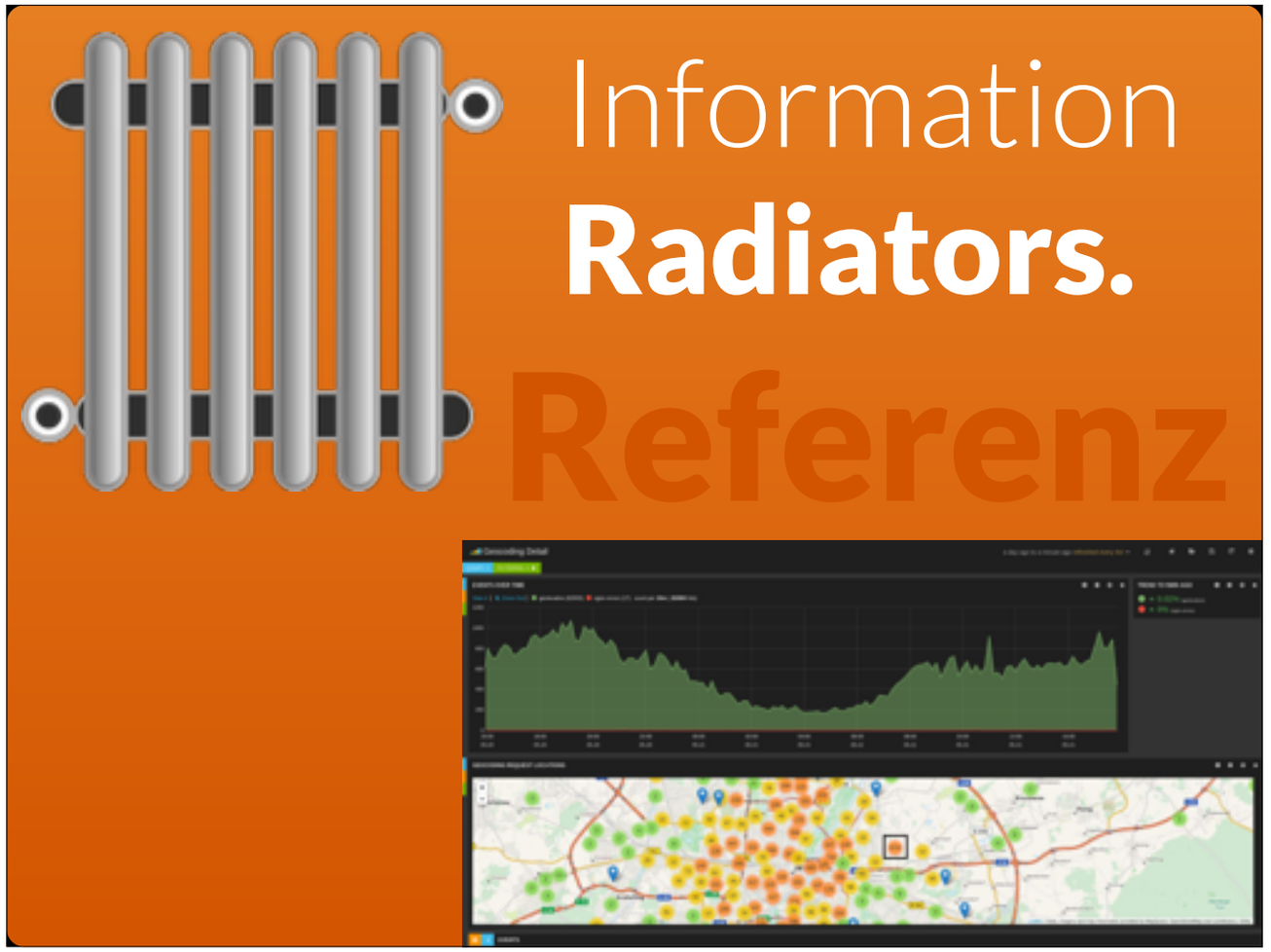
**Systems Thinking**
**Amplify Feedback Loops**
**Culture of Continual Experientation**




**DevOps-Culture ist das Outcome**

Eine DevOps-Kultur kann man - wie jede Kultur - nicht einfach machen - aber ich kann mit diesen drei Wegen dafür sorgen, dass sie eine gute Chance hat zu entstehen. Sie ist das Outcome aus dem gemeinsamen Verständnis, der Kooperation und dem Reflektieren.


Der Wechsel hatte aber noch organisatorische Folgen. Auf einmal war das Development-Team auch selbst für den Betrieb verantwortlich, man brauchte also derartige Kompetenzen, und auch alle Folgekompetenzen mussten selbst aufgebaut werden.

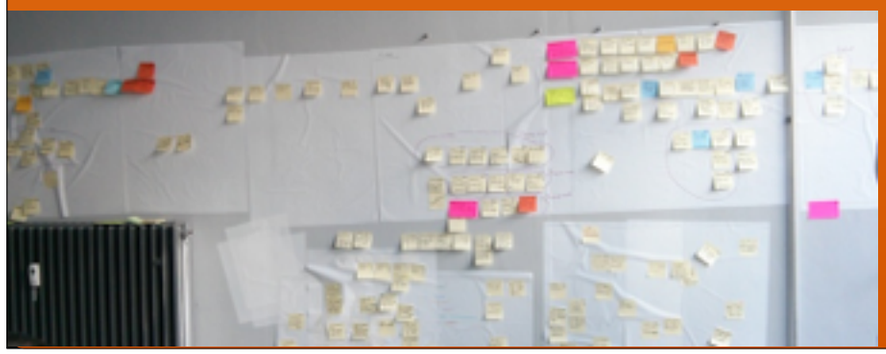



Folgekompetenzen - das sind so Sachen wie Backup oder Restore. Aber auch Monitoring, Alerting und Reporting.
Und da zeigt sich schon der erste Synergieeffekt. Es wurde einfach ein Monitor aufgebaut, der die aktuelle Situation in Operations darstellt. Alle Geschäftsrelevanten Metriken sind direkt im Flur sichtbar. Ohne viel Diskussion wurden diese Information Radiators zur Referenz für die aktuelle Situation - sogar die Unternehmensleitung guckt heute bei Ausfällen als ersten auf diesen Monitor. Und wenn es ein Problem gibt wird es direkt mit den Developern und den System Engineers besprochen - denn die sitzen neben dem Radiator.


Das darf man in der Praxis nicht unterschätzen - erst durch die transparenz gibt es gemeinsame Diskussionen und damit auch gemeinsame Verbesserungsmassnahmen.

# Shared Mental Models

Referenz



http://verraes.net/

Aber nicht nur so entstehen gemeinsame mentale Modelle - für den Ops-Bereich wurde tatsächlich „How to make Toast“ gemacht. In diesem Projekt wird DDD eingesetzt, vom hervorragenden Mathias Verraes betreut. Deshalb gab es einen Event Storming Workshop - und genau dieser hilft ebenfalls dabei, gemeinsames mentales Modell zu erzeugen.


Mit der Verantwortung für Produktion, Monitoring und Alerting wandert auch die Notwendigkeit für Bereitschaft in das Dev-Team. Das klingt für uns Entwickler erst mal wie eine Katastrophe - ich soll jetzt Nachts um 3 raus?

**Wenn man das hier macht …**

Incident Retrospectives
Incident Command System
Automated Tests
Continuous Deployment
Extensive Logging

**… gibt es das Problem gar nicht.**

Faktisch stellt sich dieses Problem in einer DevOps-Infrastruktur sogar noch seltener als in einer klassischen Struktur, bei der die Entwickler nur in GAU-Situationen ebenfalls in der Nacht heraus müssen. Wenn ich nicht nur gutes Logging habe, sondern auch automatische Tests, automatisches Deployment, und eine lernende Organisation bei Fehlern - dann passieren diese Ausfälle nicht nur seltener, sie werden auch schneller und schmerzfreier bearbeitet.

# Erfolge

- Bessere **Kooperation** zwischen den Bereichen
- Häufige **Deploys**
- Erfolgreiche **Launches**, on time
- Bessere **Planbarkeit**
- Höhere **Verlässlichkeit**

**… aber noch lange nicht fertig.**

All diese Massnahmen haben längst angefangen sich zu verzinsen - auch wenn noch lange nicht alles da ist. Die Kooperation zwischen den Bereichen verbessert sich, es kann praktisch jederzeit deployed werden. Die Launches sind deutlich verlässlicher geworden, sowohl in ihrer Planbarkeit als auch in der allgemeinen Qualität. Aber es gibt noch Silos, und die Schranke zwischen Business und Development ist auch noch nicht verschwunden - auch wenn die Organisation sich selbst inzwischen in der Richtung bewegt hat.


Also: DevOps wirkt, ist Mainstream, und man kann es in der Praxis machen. Und man kann damit das magische Dreieck des Projektmanagements brechen. Manchmal ist es schon cool, was wir ITler so alles für Dinge können.