
DevOps

Tearing Down the Walls



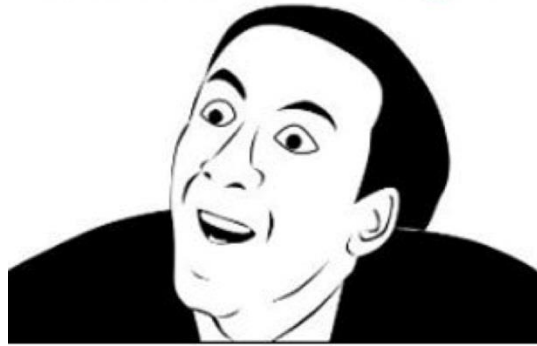
Aufbau

1. Was ist DevOps?
2. Die 5 DevOps-Prinzipien
3. Kritik
4. Fazit
5. Ausblick

Was ist DevOps?

Development + **O**perations = **DevOps**

You don't say?



Was ist DevOps?

DevOps ist eine **Bewegung**, die das Ziel hat, die **Time-To-Market** einer **Änderungseinheit** zu **reduzieren** - bei gleichzeitiger Gewährleistung **hoher Qualität**.

Dies soll durch die konsequente Anwendung von **Lean-Prinzipien** auf den **gesamten Software-Wertstrom** erreicht werden.

Was ist DevOps?



To make error is human. To propagate error to all server in automatic way is [#devops](#).

Relevanz

“Gartner Says By 2016, DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations”

Press Release, 05.03.2015, <http://www.gartner.com/newsroom/id/2999017>

“The number of DevopsDays conferences have increased from 1 in 2009 to 18 all over the world in 2014.”

Effective DevOps, O'Reilly Media, Early Release May 2015

Relevanz



Google Trends, abgerufen am 05.02.2016

<https://www.google.de/trends/explore#q=DevOps&cmpt=q&tz=Etc%2FGMT-1>

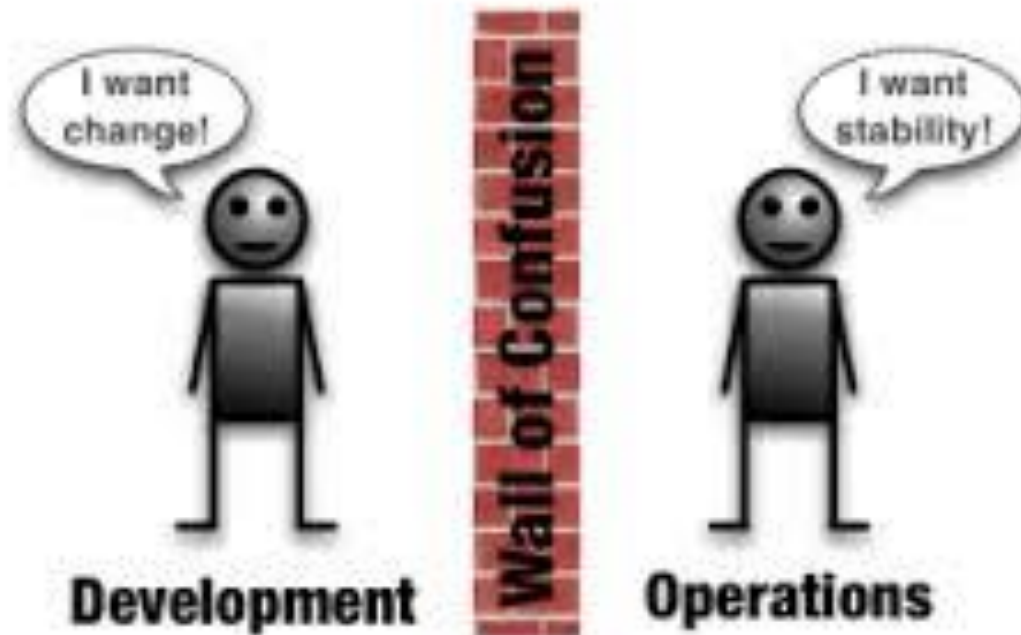
Begriff „DevOps“

- Wurde geprägt von Patrick Debois durch Ausrichtung der „DevOpsDays“ 2009 in Belgien
- Hat seine Wurzeln in „Agile Infrastructure“ (Deboir and Shafer) und „Lean Startup“

Das Grundproblem: Warum DevOps

- Entwickler (Dev) sollen **schnell Veränderungen** umsetzen
- Administratoren (Ops) sollen **Sicherheit und Stabilität** der Systeme gewährleisten
- Die unterschiedlichen Ziele führen zu „**Silo-Denken**“ und bilden eine imaginäre „**Mauer**“ zwischen Dev und Ops
- Agile Methoden erhöhen Dev-Geschwindigkeit und damit Druck auf Ops

Der Zielkonflikt zwischen Dev und Ops



Conway's Law

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

**Teams, die nicht gut zusammenarbeiten
produzieren Lösungen, die nicht gut
zusammenarbeiten.**

Silo-Denken

Stereotypen

Dev aus Sicht von Ops:

- Ungeduldig
- Unbeherrscht
- Leichtsinnig
- „Prima Donna“-Künstler
- Keine Ahnung von stabilem Service

Ops aus Sicht von Dev:

- Langsam
- Unbeherrscht
- Zu restriktiv
- Feinde des Fortschritts
- Keine Ahnung von zeitgemäßen Anwendungen

Silo-Denken



@rahuldighe

Silo-Denken



Auswirkungen von Silo-Denken

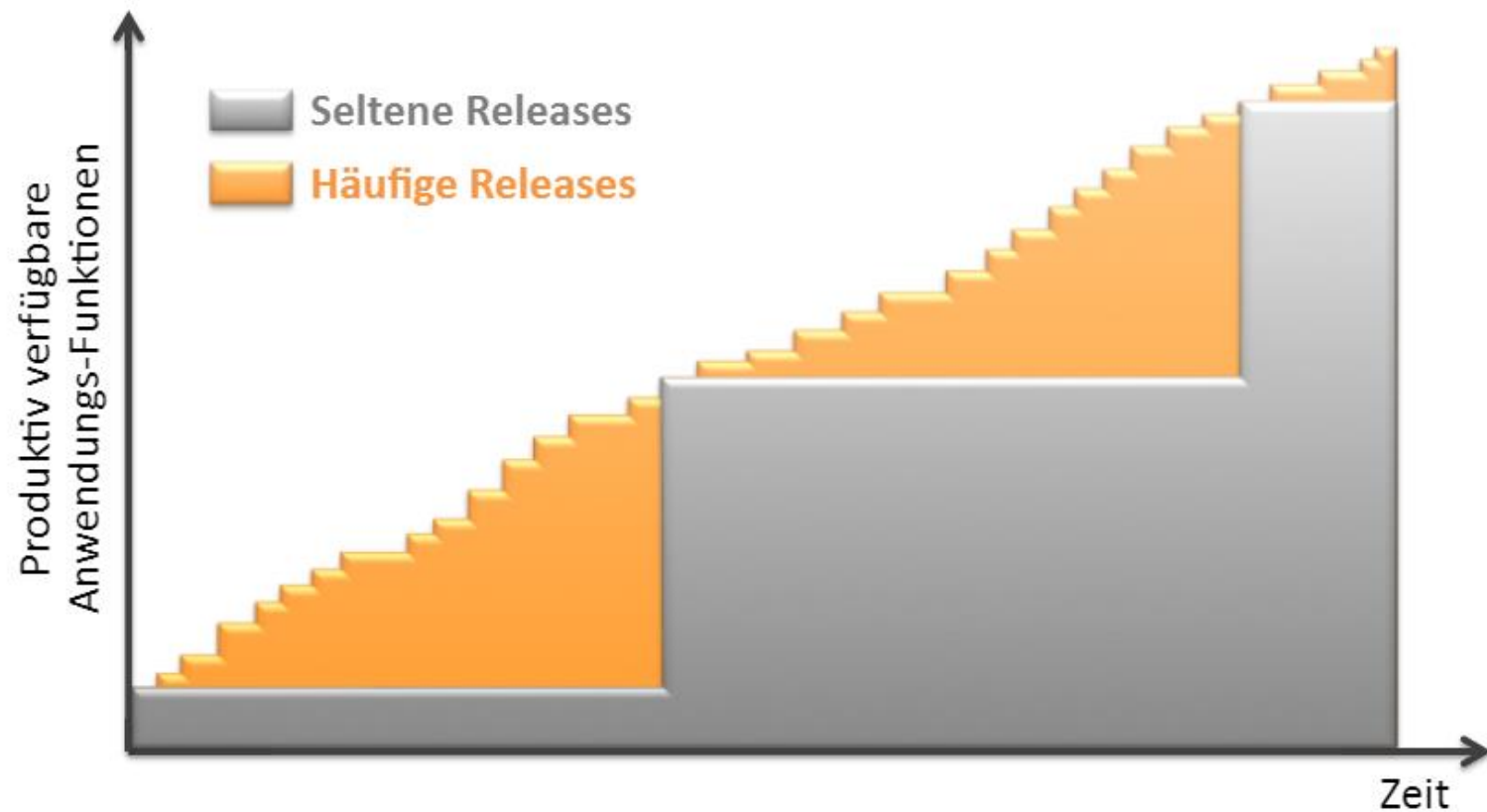
- mehr und länger andauernde System-Ausfälle
- Mehr Fehler
- Höheres Risiko (=weniger Gewissheit)
- Angst vor Veränderung -> Lähmung
- Längere Time-to-Market, schlechtere Qualität



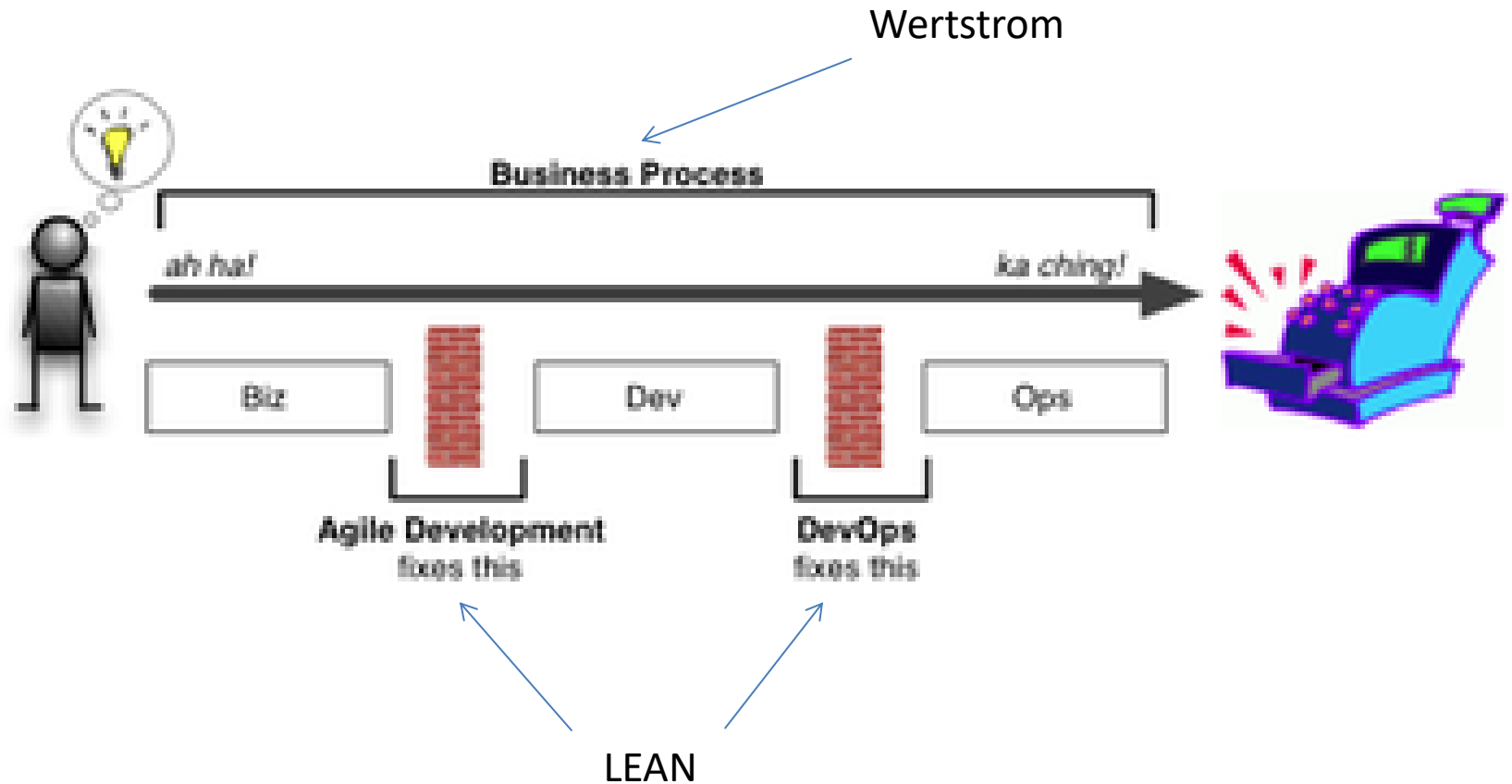
Konsequenz: Wertstrom stoppt an „Wall of Confusion“

- Der Wertstrom „stoppt“ an der Mauer zu Operations
- Viel Zeit geht verloren (Verschwendung) - Features sind fertig, aber nicht produktiv verfügbar

Silo-Denken: Konsequenzen



DevOps to the Rescue!



CALMS – Die 5 DevOps-Prinzipien

- CULTURE
- AUTOMATION
- LEAN
- METRICS
- SHARE



**KEEP
CALMS
AND
DO
DEVOPS**

Unsere Dramaturgie

- LEAN
- AUTOMATION
- METRICS
- CULTURE
- SHARE

[illegible]

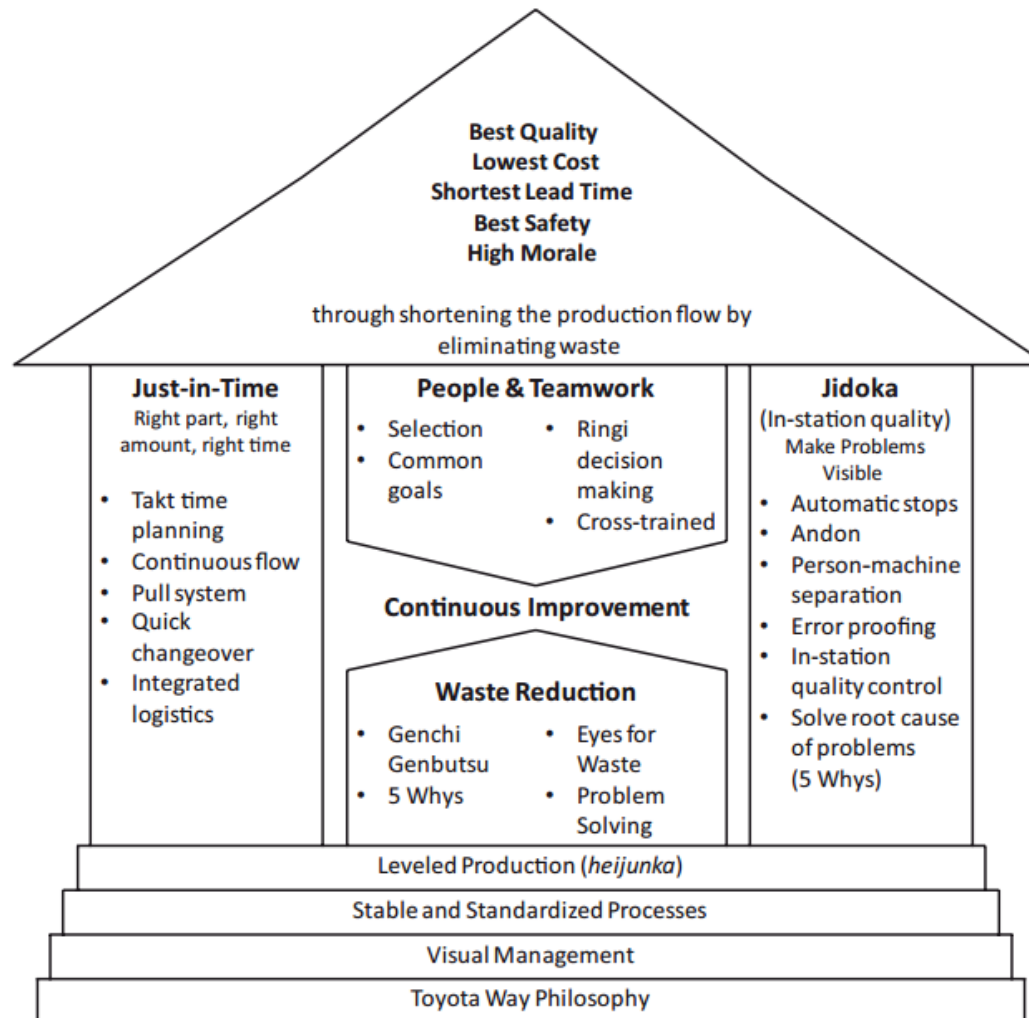
Was ist „Lean“?



Was ist „Lean“?

- Eine **Philosophie**, die das Ziel hat, einen **Prozess** durch die **Eliminierung von Verschwendung kontinuierlich zu verbessern** und dabei die **Bedürfnisse der Kunden** als **Ausgangspunkt allen Handelns** sieht

Ursprung: Toyota Produktions-System



Die drei Arten der Verschwendung

Muda:

„... jede menschliche Aktivität, die Ressourcen verbraucht, aber keinen Wert erzeugt.“(Wallace J. Hopp)

Mura:

Unausgeglichenheit, Unausgewogenheit, Inkonsistenz

Muri:

Überlastung von Menschen (psychische/physisch) oder Maschinen

Ausprägungen von Muda

„TIMWOOD“

- Materialbewegungen (*TRANSPORTATION*)
- Bestände (*INVENTORY*)
- Bewegungen (*MOTION*)
- Wartezeiten (*WAITING*)
- Verarbeitung (*OVER-PROCESSING*)
- Überproduktion (*OVER-PRODUCTION*)
- Korrekturen und Fehler (*DEFECTS*)

Die 5 Lean-Prinzipien

1. Den (zu produzierenden) Wert aus Sicht des Kunden definieren
2. Den Wertstrom identifizieren
3. Das Fluss-Prinzip umsetzen
4. Das Pull-Prinzip einführen
5. Perfektion anstreben

1. Wert aus Sicht des Kunden definieren

Was ist „Wert“?

- Eine Leistung, die den Anforderungen des Kunden hinsichtlich
 - Qualität
 - Verfügbarkeit (im Sinne von „da, wenn der Kunde es will“) und
 - Preisentspricht
- „Wert“ ist, was der Kunde kauft

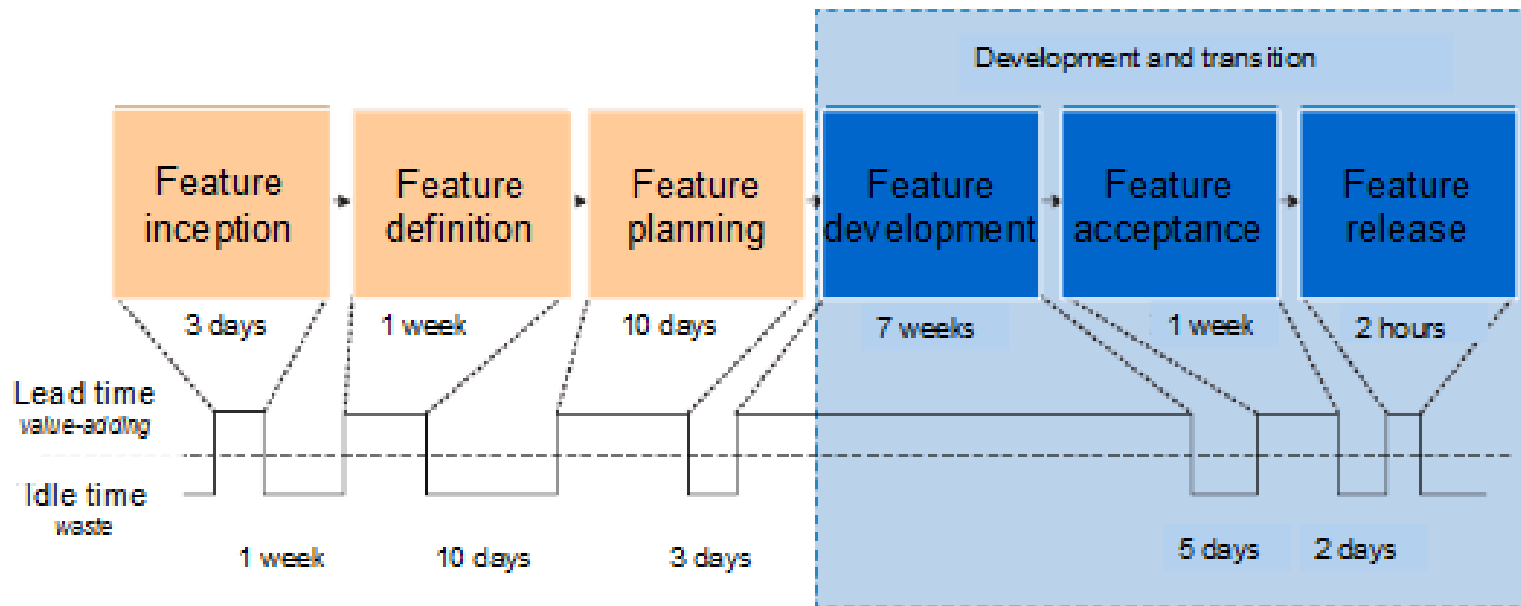
2. Wertstrom identifizieren

- engl. *Value Stream*
- alle wertschöpfenden und nicht-wertschöpfenden Aktivitäten, die notwendig sind, um ein Produkt bzw. eine Dienstleistung herzustellen und anzubieten
- Ziel des Wertstroms: Kunden zufriedenstellen
- Jeder Wertstrom umfasst einen oder mehrere Geschäftsprozesse
- Unterscheidung dreier Arten von Tätigkeiten:
 - Wertschöpfende Tätigkeiten
 - Nicht-wertschöpfende Tätigkeiten, die unvermeidbar sind
 - Nicht-wertschöpfende Tätigkeiten, die vermeidbar sind (**Verschwendung**)

2. Wertstrom identifizieren

- dient hauptsächlich der Aufdeckung von Verschwendung
- forciert ganzheitliche Betrachtung aller am Wertstrom beteiligten Prozesse
- trägt zu silo-übergreifenden Optimierung des Gesamtprozesses bei

Beispiel Wertstrom



Beispiel Wertstrom

- I asked him “When you did the Value Stream Mapping, did you do it across engineering and operations?” After about 15 seconds of dead silence he sheepishly answered, “Shit, we never thought about that.”

<http://itrevolution.com/devops-culture-part-1/>

3. Fluss-Prinzip umsetzen

Das Produkt oder die Leistung soll:

- kontinuierlich
- mit möglichst wenigen Unterbrechungen, Wartezeiten und Umwegen
- über alle Stationen/Abteilungen hinweg
- in kleinen Änderungs-Einheiten (Features, Bugfixes, ...)

zum Kunden **“fließen”**.

Dies wird durch die Beseitigung der identifizierten Verschwendung erreicht.

Kleine Änderungseinheiten

- Geringere TTM
- Geringere MTTR
- <http://www.startuplessonslearned.com/2009/02/work-in-small-batches.html>

4. Pull-Prinzip einführen

- Es wird nur produziert:
 - **Was** der Kunde will
 - **Wenn** der Kunde etwas will

5. Perfektion anstreben

- Die vier vorgenannten Schritte werden kontinuierlich gemessen, geprüft und kritisch hinterfragt
- Dadurch werden immer weitere Verbesserungspotentiale aufgezeigt und umgesetzt
- Ziel: jede Tätigkeit und jedes Gut trägt zum Wert für den Kunden bei

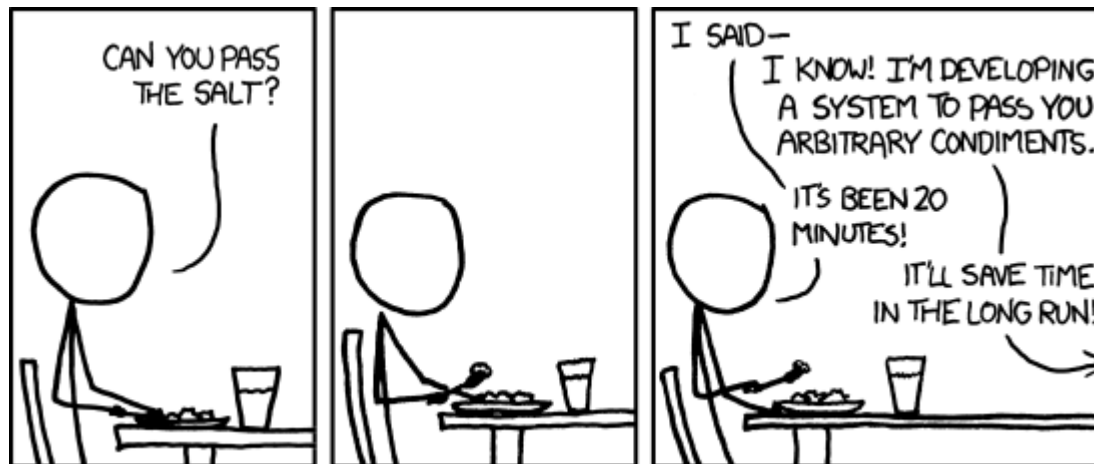


Kaizen

PAUSE



Automation



Automatisierung

- Das Ziel von DevOps ist **NICHT** die komplette Automatisierung des Wertstromes
- Automatisierung ist aber ein **wesentliches Hilfsmittel** zur **Optimierung** des Wertstromes durch die **Eliminierung von Verschwendung** und die Einführung eines **definierten Arbeitsflusses**
- Wichtig ist, zu wissen, wann es sich **lohnt**, etwas zu automatisieren – und wann **nicht**

Automatisierungsstufen des Software-Entwicklung-Prozesses

- (0) Automatisierte Entwicklungsumgebung
 - Setup (IDE, Virtualisierung, ...)
 - build, test, check
- 1. Continuous Integration
- 2. Continuous Delivery
- 3. Continuous Deployment

Continuous Integration

Continuous Integration:

- Automatisierter Bauvorgang aller Artefakte („deliverables“)
 - Automatisierte Tests
 - Automatisierte statische Code-Analyse
- > „Software kann immer gebaut werden und erfüllt bestimmte Qualitätskriterien“

Continuous Delivery

Continuous Delivery: CI + Staging

- Automatisierte Installation und Konfiguration der verschiedenen Zielumgebungen
- Automatisierte Akzeptanztests
- Automatisierte Kapazitätstests
- Automatisiertes Deployment in verschiedene weitere Test-Umgebungen:
 - User Acceptance Testing (UAT)
 - Exploratory Testing
 - ...
- **Aber: manuelles Auslösen des Deployments in Produktion!**
(=Deployment erfolgt automatisiert, muss aber manuell gestartet werden)

->“Software **könnte** jederzeit in Produktion deployed werden“

Continuous Deployment

Continuous Deployment:

- Konsequente Weiterentwicklung von CDelivery
- -> „Jede Änderung, die alle Stages erfolgreich durchlaufen hat, wird automatisch in Produktion deployed“

DevOps und Automatisierung

- bereichsübergreifende Automatisierung zwischen Entwicklung und Produktivbetrieb erfordert Zusammenarbeit und gemeinsame Verantwortung für Installation und Konfiguration der verschiedenen Zielumgebungen (UAT, Staging, Produktion usw.)
- Lösung: **Infrastructure as Code**

Infrastructure as Code

Grundidee:

- Infrastruktur wie Software behandeln („as Code“ eben)
- Anweisungen für das Aufsetzen und Konfigurieren der Umgebungen werden Teil des Quellcodes
- Änderung an Infrastruktur erfordert Änderung des Quellcodes

Infrastructure as Code

„...Infrastructure as Code [liefert] keine Blaupause für jedes Unternehmen[...]. Vielmehr ist es das Ziel einer Reise, auf der Unternehmen ihre Entwicklungs- und Betriebsteams zusammenbringen, um verschiedene Bereiche in ihren Rechenzentren und Produkt-Pipelines anzupassen und zu automatisieren. So führt Infrastructure as Code schließlich zu einer gemeinsam designten Infrastruktur und gemeinsam definierten Abläufen, die stark an die Prozesse zur Entwicklung von Anwendungscode erinnern. Dadurch wird der Betrieb der Unternehmens-IT automatisier- und programmierbar.“

Beispiel: Ansible

```
---  
- hosts: webservers  
  remote_user: root  
  
  tasks:  
    - name: ensure apache is at the latest version  
      yum: name=httpd state=latest  
    - name: write the apache config file  
      template: src=/srv/httpd.j2 dest=/etc/httpd.conf  
  
- hosts: databases  
  remote_user: root  
  
  tasks:  
    - name: ensure postgresql is at the latest version  
      yum: name=postgresql state=latest  
    - name: ensure that postgresql is started  
      service: name=postgresql state=running
```

Ansible-Playbook (yaml)

Beispiel: Puppet

```
node 'host2' {  
  class { 'apache': }           # use apache module  
  apache::vhost { 'example.com': # define vhost resource  
    port    => '80',  
    docroot => '/var/www/html'  
  }  
}
```

Puppet-Manifest

Infrastructure as Code: Tools



Vorteile von Infrastructure as Code

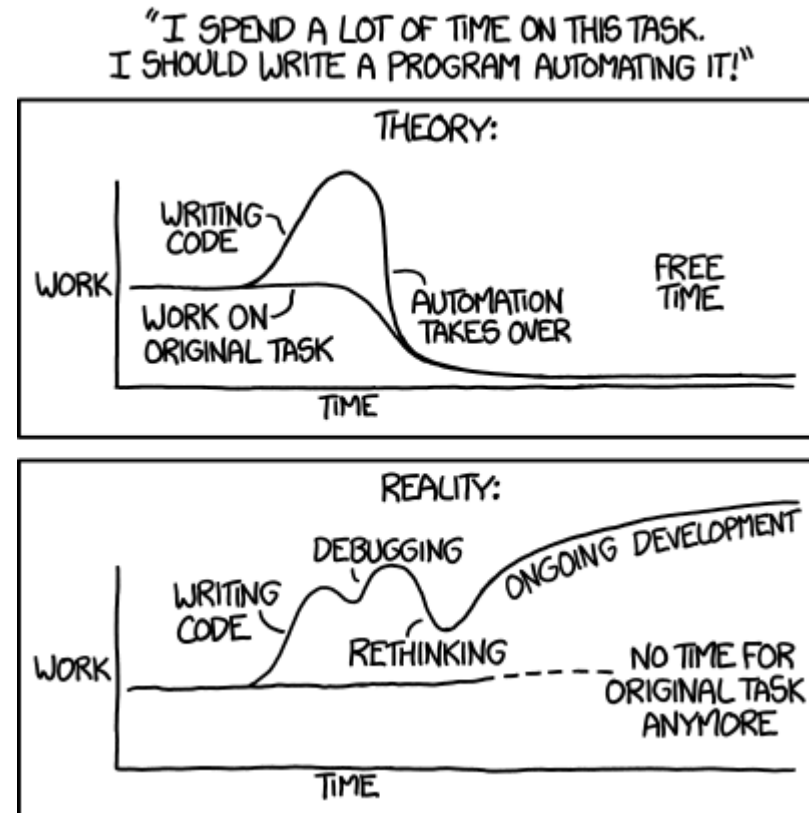
- Keine Shadow-IT
- Erhöhung der Effizienz
- Infrastruktur wird mit Code kontinuierlich getestet und verwaltet
- Änderungen an Infrastruktur werden in Repository versioniert
- Zusammenarbeit von Dev und Ops wird forciert

Grad der Automatisierung



...wirklich?

Wann lohnt sich Automatisierung?



Wann lohnt sich Automatisierung?

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	2 WEEKS	1 DAY
	1 DAY					8 WEEKS	5 DAYS

Wann lohnt sich Automatisierung?

- wenn in relevantem Ausmaß Zeit gespart wird
 - Einbindung von Legacy-Systemen in automatisierte Prozesse kann beispielsweise unverhältnismäßig viel Zeit kosten
- Zeitinvestitionen sind nicht immer vergleichbar
 - Zuverlässiges Rollback-Script ist vermutlich „mehr wert“ als beispielsweise die automatische Generierung eines Change-Logs
- Zeitersparnis kann auch anderen zugutekommen und multipliziert sich dann

You are not Netflix

„ While DevOps tool providers may sell automation products that promise to make you the next Facebook or Netflix, the reality is that your organization isn't like Facebook or Netflix, and it never will be. If you try to replicate their automation solutions, you'll quickly find yourself in a world of hurt. The investment you have made in DevOps and automation will blow up in your face—not because you automated, but because you attempted to automate too much.”

Automatisierungs-Paradoxon

- Je höher der Grad der Automatisierung, desto kompetenter muss der Mensch sein, der den automatischen Prozess verwaltet
- “Efficient Automation makes humans more important, not less”

Automatisierungs-Ironie

- Wenn zu viel automatisiert wird, wird der Operator zum „Beobachter“ des Systems
- Er verlernt die Grundlagen der Prozesse, die automatisiert wurden
- Er kann daher im Fehlerfall nicht mehr angemessen und schnell reagieren
- Je hochgezüchteter die Automatisierung, desto gravierender die Effekte der Automatisierungs-Ironie
- Es kann daher sinnvoll sein, bewusst manche Teilprozesse oder Teilschritte nicht zu automatisieren

Metrics



I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.

(Lord Kelvin)

izquotes.com

DevOps messen

- damit der Erfolg von DevOps überprüft und kontinuierlich verbessert werden kann, muss er **messbar** gemacht werden.
- Neben **klassischen** Metriken für Software-Entwicklung und Betrieb wird bei DevOps großer Wert auf Metriken gelegt, welche die **Leistungsfähigkeit des gesamten Wertstromes** aufdecken

„According to the lean philosophy, it is essential to optimize globally, not locally.“

„klassische“ Software-Metriken

- Betreffen nur das Silo „Entwicklung“
- Beispiele:
 - Lines of Code
 - zyklomatische Komplexität
 - Stil
 - Testabdeckung
 - Technische Schuld

„klassische“ Operations-Metriken

- Betreffen nur das Silo „Operations“
- Beispiele:
 - Uptime
 - Antwortzeiten
 - Ressourcen-Auslastung

DevOps-Metriken

- Betreffen den **gesamten Wertstrom**
- Die vier wichtigsten Metriken:

Durchsatz	Stabilität
Häufigkeit von Deployments in Produktion	mittlere Reparaturzeit
Änderungs-Durchlaufzeit	Fehlerrate in Produktion

Häufigkeit von Deployments in Produktion

- „Deployment Frequency“
- Wie oft wird in das Produktivsystem deployed
- Relativ einfach messbar
- Ggf. Unterscheidung in # erfolgreicher/nicht erfolgreicher Deployments

Änderungs-Durchlaufzeit

- „Change Lead Time“
- Wie lange benötigt eine Änderungseinheit vom Beginn der Arbeit bis zum Deployment in Produktion
- Problem: wann „beginnt“ die Arbeit an einer Änderungseinheit?
 - Erfassen der User Story?
 - Schreiben der ersten Code-Zeile?

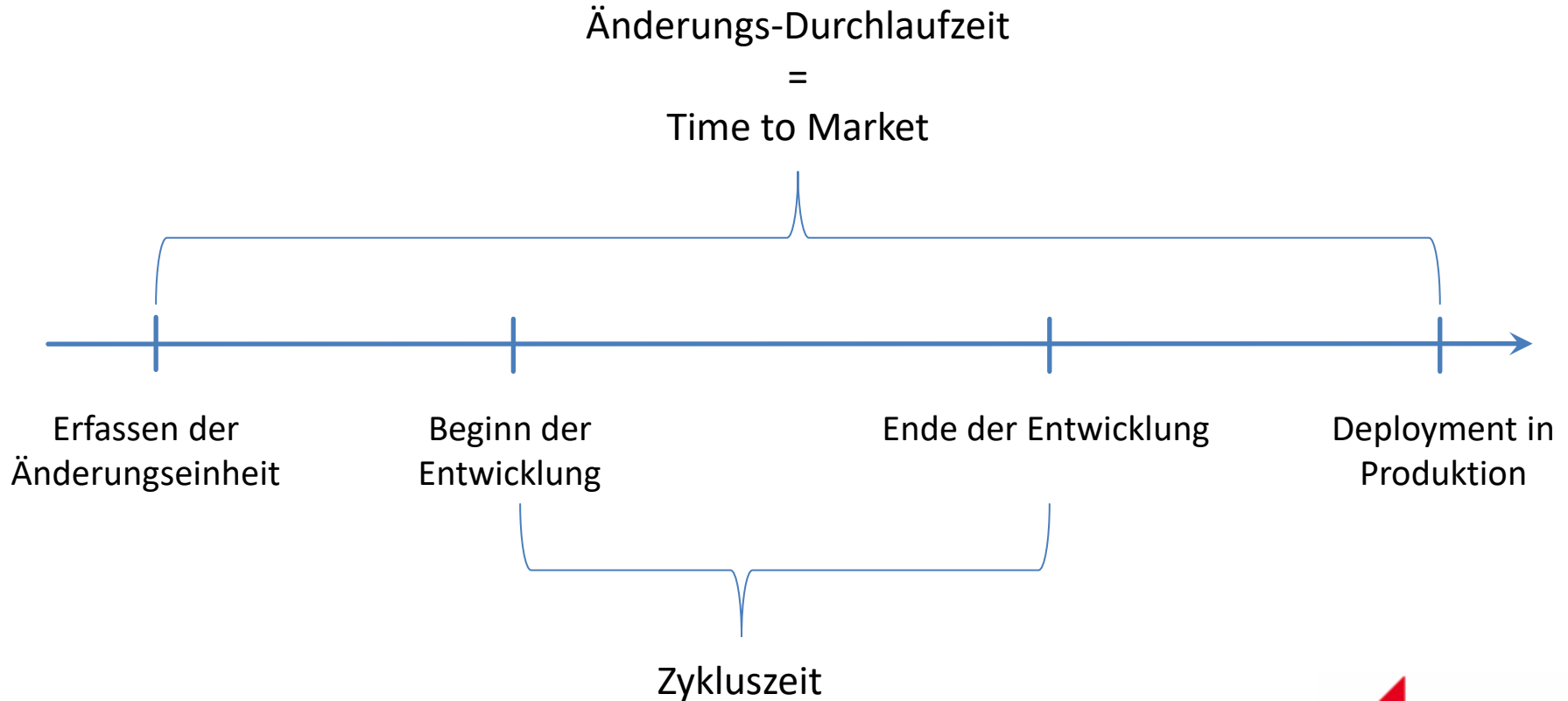
Änderungs-Durchlaufzeit

Beispiel:



Änderungs-Durchlaufzeit

Mögliche Verfeinerung:



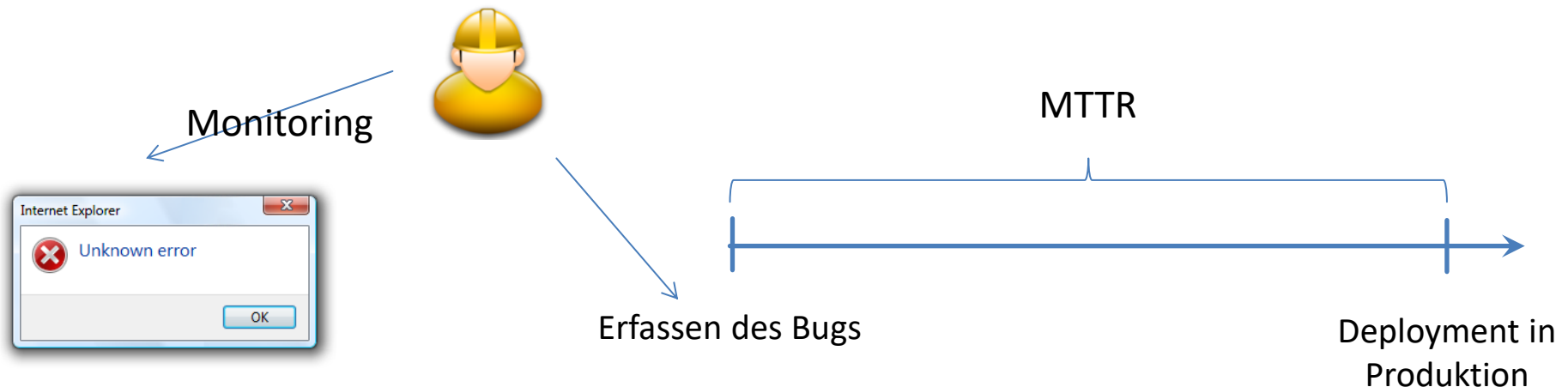
Änderungs-Durchlaufzeit

Einige Hinweise:

- Lead Time und Cycle Time werden teilweise synonym benutzt
- Begriffe kommen aber aus der Produktionsorganisation und sind unterschiedlich und klar definiert
 - <https://hbswk.hbs.edu/archive/hbs-toolkit-basic-operations-self-instructional-workbook>
- Starten Sie mit dem, was Sie am einfachsten messen können

Mittlere Reparaturzeit

- „Mean Time to Recover/Repair“ (MTTR)
- Zeit zwischen Feststellung eines Fehlers in Produktion und dessen Beseitigung
- Beispielsweise messbar als Durchlaufzeit eines Issues vom Typ „Bug“



Fehlerrate in Produktion

- „Change Failure Rate“
- wie viele Bugs treten in Produktion auf
- Einfach messbare Metrik
- Ggf. Unterscheidung nach Schwere
- Evtl. Messung von Reaktionszeit

Zusammenhang zwischen Durchsatz/Stabilität

Häufige Deployments und kurze Durchlaufzeiten
können Stabilität, Fehlerrate und
Reaktionszeiten extrem verbessern.

Diskussion

Weitere mögliche DevOps-Metriken

- Work in Progress (WIP)
- Verfügbarkeit
- Performance
- Mean Time Between Failures (MTBF)
- Kulturelle Metriken
 - Akzeptanz DevOps, Zufriedenheit, ...
- Business-Metriken
 - Conversion Rate, Umsatz, ...



MTBF

Exkurs: statsd und Graphite

- <https://github.com/etsy/statsd>
- <http://graphite.wikidot.com/>
- <http://www.aosabook.org/en/graphite.html>
- <http://de.slideshare.net/jeremyq/statsd-and-graphite>

statsd

- Node.js-basierter Network-Daemon
- Lauscht per UDP oder TCP auf gesendete Messungen („stats“)
- Sammelt Messungen in „buckets“
- Aggregiert zeitgesteuert (10 sec/default) alle Messungen und sendet diese an irgendeinen Backend-Service, z.Bsp. Graphite

```
echo „myapp.auth.login:1|c“ | nc -u -w0 127.0.0.1 8125
```

Graphite

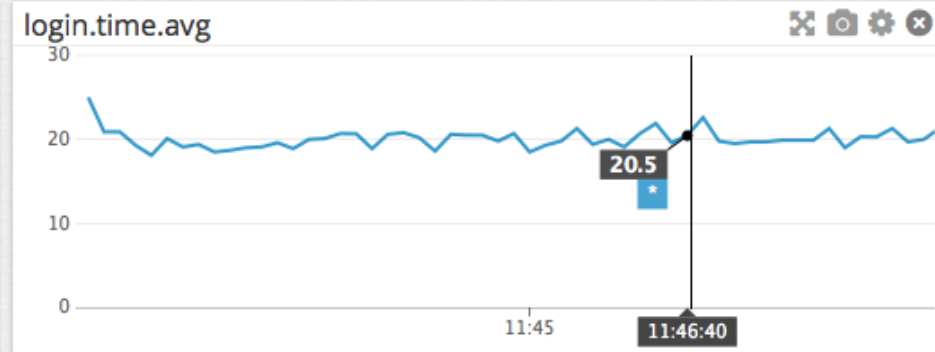
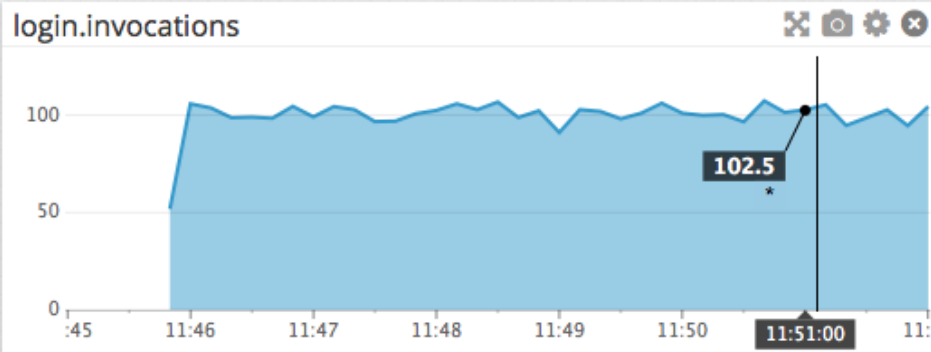
Monitoring-Tool



Exkurs: statsd und graphite

```
import statsd
statsd_client = statsd.StatsClient('localhost', 8125)
```

```
@statsd_client.timer('login.time')
def login(username, password):
    statsd_client.incr('login.invocations')
    if password_valid(username, password):
        render_welcome_page()
    else:
        render_error(403)
```



Pause

MISSION
PAUSED...

PRESS START WHEN READY

Culture

YOU **FOOL!**
It's not about the tools, it's about the **CULTURE!**



<http://leandrasmls.files.wordpress.com/2011/07/arguments-cartoon.jpg>

DevOps-Kultur

Das **Hauptziel** von DevOps ist eine Veränderung der **Organisationskultur** von:

- **Silos** zu **Kollaboration** und
- **Schuldzuweisungen** zu **gemeinsamer Verantwortung** und **Vertrauen**

für die erstellten digitalen **Produkte, Leistungen** und der dazu notwendigen **Prozesse**.

Was ist „Organisationskultur“?

Drei ausgewählte Definitionen:

„Konzept und Begriff der Kultur wurden aus der anthropologischen Forschung entlehnt. Es existiert kein Konsens über die Bedeutung des Begriffes, so dass auch in der Anwendung im Rahmen der Organisationstheorie teilweise beträchtliche Unterschiede bestehen.“

L. Smircich (1983), Concepts of Culture and Organizational Analysis; Administrative Science Quarterly 28, S. 339–358

„Organisationskultur ist die Sammlung von Traditionen, Werten, Regeln, Glaubenssätzen und Haltungen, die einen durchgehenden Kontext für alles bilden, was wir in dieser Organisation tun und denken.“

J. Marshall und A. McLean (1985): *Exploring Organisation Culture as a Route to Organisational Change*, in Hammond V. (ed), Current Research in Management, pp. 2-20, Francis Pinter, London.

„So machen wir das hier.“

D. Bright und B. Parkin: *Human Resource Management – Concepts and Practices*. Business Education Publishers Ltd., 1997, S. 13.

Vgl. <https://de.wikipedia.org/wiki/Organisationskultur>, abgerufen am 06.02.2016

Was ist „Organisationskultur“?

- Kultur ist ein abstraktes und komplexes Konzept
- Definitionen meist „vage und allumfassend“
- Schwer messbar
- Aber:
 - **Es ist allgemein anerkannt, dass die Kultur einer Organisation Einfluss auf das Erreichen der Organisationsziele hat**

DevOps-Kulturmerkmale

- Gegenseitiger Respekt
- Vertrauen
- No-blame
- Growth mindset
- Gemeinsame Anreize (Incentives)
- Kollaboration

„Organizational culture is one of the strongest predictors of both IT performance and overall performance of the organization. “

State of DevOps Report, 2014, PuppetLabs

Blame-Kultur

- „name, blame, shame“
- Bei Fehlern wird so lange gesucht, bis ein Sündenbock gefunden wurde
- Resultat:
 - Fehler werden verschleiert
 - Menschen haben Angst, Hemmungen und sichern sich ab
 - -> **Verschwendung**

No-Blame-Kultur

- Nicht Individuen führen zu Fehlern, sondern **Situationen**
- Vermeidung der Wiederholung ähnlicher Situationen steht im Vordergrund
- Keine Suche nach einem Sündenbock sondern **Suche nach Ursachen.**



<https://www.youtube.com/watch?v=2PjVuTzA2lk>

5-Why-Methode

- Methode zur Bestimmung von Ursache und Wirkung
- So lange „Warum“ fragen, bis die Ursache klar ist (5 ist symbolisch)

Problemstellung: Das Fahrzeug startet nicht.

1. Warum startet das Fahrzeug nicht?
Die Starterbatterie ist leer.
2. Warum ist die Starterbatterie leer?
Die Lichtmaschine funktioniert nicht.
3. Warum funktioniert die Lichtmaschine nicht?
Der Treibriemen ist gerissen.
4. Warum ist der Treibriemen gerissen?
Der Treibriemen wurde nie ausgewechselt.
5. Warum wurde der Treibriemen nie ausgewechselt?
Das Fahrzeug wurde bisher nie gewartet.

Fixed Mindset vs. Growth Mindset

Mindest-Theorie nach Carol Dweck

What Kind of Mindset Do You Have?



I can learn anything I want to.
When I'm frustrated, I persevere.
I want to challenge myself.
When I fail, I learn.
Tell me I try hard.
If you succeed, I'm inspired.
My effort and attitude determine everything.



I'm either good at it, or I'm not.
When I'm frustrated, I give up.
I don't like to be challenged.
When I fail, I'm no good.
Tell me I'm smart.
If you succeed, I feel threatened.
My abilities determine everything.

Fixed Mindest vs. Growth Mindset

Fixed Mindset:

- Intelligenz, Talente, Fähigkeiten sind gegeben und können nicht verbessert werden
- „Fehler sind Resultat ungenügender Fähigkeiten“
- Tendenz, sich nicht blamieren zu wollen

Fixed Mindest vs. Growth Mindset

Growth Mindset:

- Intelligenz, Talente, Fähigkeiten können verbessert werden
- „Fehler sind eine Gelegenheit, sich zu verbessern“
- Tendenz, auch nach Rückschlägen weiter hart zu arbeiten

Gemeinsame Anreize

- Entwickler werden nicht belohnt, wenn sie Tonnen von Code produzieren
- Operator werden nicht belohnt, wenn alle Server immer verfügbar sind
- Das **Team** wird belohnt, wenn Wert für den Kunden geliefert wird und alle gemeinsam daran arbeiten, den Fluss von Wert zum Kunden zu verbessern

Incentives Gone Wrong

OUR GOAL IS TO WRITE
BUG-FREE SOFTWARE.
I'LL PAY A TEN-DOLLAR
BONUS FOR EVERY BUG
YOU FIND AND FIX.



S. Adams E-mail: SCOTTADAMS@AOL.COM

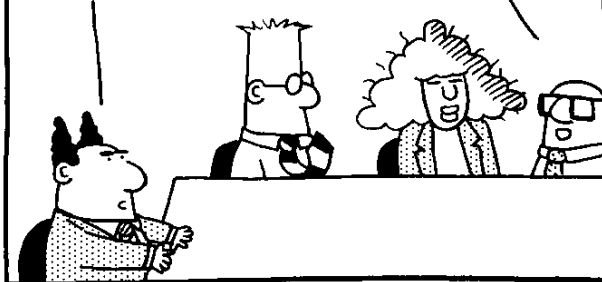
YAHOO!
WE'RE
RICH



11/13 © 1995 United Feature Syndicate, Inc. (NYC)

I HOPE
THIS
DRIVES
THE RIGHT
BEHAVIOR.

I'M GONNA
WRITE ME A
NEW MINIVAN
THIS AFTER-
NOON!



Kollaboration

- Art der möglichen Zusammenarbeit zwischen Dev und Ops hängt stark vom Unternehmen und dessen Leistungen ab
 - Bereichsübergreifende Teams
 - Temporärer(!) DevOps-Team
 - Austausch von Experten
 - „Infrastructure as a Team“

Share



Teilen

- Verantwortung
- Ziele
- Tools
- Information
- Wissen & Erfahrung
- Code
- Workflow
- Training

Fazit

„Branchengrößen wie Facebook, Flickr, Twitter oder Amazon gelten als Wegweiser für DevOps. Doch deren Vorgehen dürfen andere Unternehmen nicht einfach nachahmen, denn sie besitzen eine andere Organisation, Firmenkultur oder Softwareausstattung. Stattdessen sollten sie bei der Planung von DevOps-Prozessen mit dem wichtigsten Ziel beginnen, der schnelleren Auslieferung von Software, ohne dabei die Qualität zu gefährden. Das müssen die Entwicklungs-, Test- und Betriebsteams gemeinsam verinnerlichen. Tools können sie dabei unterstützen, die Aufgaben entlang der Prozesskette effizienter zu automatisieren.“

- <http://www.heise.de/developer/artikel/Mehr-Qualitaet-und-Geschwindigkeit-bei-DevOps-2859707.html>, abgerufen am 15.01.2016

Fazit

„Die größte Aufgabe für IT-Verantwortliche ist es herauszufinden, in welchen Teilen des Unternehmens ein DevOps-Einsatz am sinnvollsten wäre. Dies wird vermutlich dort sein, wo Geschwindigkeit eine wichtige Rolle spielt und das Unternehmen Möglichkeiten sieht, seinen Kunden ein besseres Erlebnis zu bieten als seine Wettbewerber (z.B. ein Einzelhändler, der den Bezahlvorgang auf seiner Website mit Hilfe von DevOps verbessert, oder eine Bank, die auf ihrer Website neue Möglichkeiten anbietet, um die Entwicklung von Fonds zu verfolgen).“

- <http://www.cio.de/a/devops-in-einer-two-speed-architektur,3251208,3>, abgerufen am 17.01.2016

Fazit

„Der Wechsel zu einer DevOps-Umgebung kann sowohl die Produktivität als auch die Markteinführungszeit von Softwareprodukten deutlich verbessern. Mit der Einführung einer neuen IT-Methode ist es dabei allerdings nicht getan. Gefragt ist eine unternehmensweite Transformation, bei der Prozess- und Governance-Fragen genauso Rechnung getragen wird wie Technologiethemen.“

- <http://www.cio.de/a/devops-in-einer-two-speed-architektur,3251208,4>, abgerufen am 06.02.2016

Fazit

- Ihr Fazit?

Ausblick

- DevSec
- IT der zwei Geschwindigkeiten



DevSec

- Agile Methoden führen zur Vernachlässigung der Sicherheit aufgrund schnellerer Release-Zyklen
- NSA-Skandal usw. hat Problembewusstsein geschärft
- **„bei DevSec [geht es] darum, das Sicherheitsdenken und entsprechende Praktiken fest in der Entwicklung und den zugehörigen Prozessen zu verankern.“**

IT der zwei Geschwindigkeiten

- „bimodale IT“, „Hybrid-IT“
- Die Interne IT gilt in Unternehmen oft als Innovationshemmer
- Unternehmen brauchen zwei verschiedene Arten von IT:
 - Traditionelle IT (Stabilität für Tagesgeschäft)
 - Agile IT (Innovation und Geschwindigkeit)

IT der zwei Geschwindigkeiten

Bimodal IT = Marathon Runners + Sprinters. Deeply Different, Both Essential

Think
Marathon
Runner



Mode 1	Goal	Mode 2
Reliability		Agility
Price for performance	Value	Revenue, brand, customer experience
Waterfall, V-Model, high-ceremony IID	Approach	Agile, kanban, low-ceremony IID
Plan-driven, approval-based	Governance	Empirical, continuous, process-based
Enterprise suppliers, long-term deals	Sourcing	Small, new vendors, short-term deals
Good at conventional process, projects	Talent	Good at new and uncertain projects
IT-centric, removed from customer	Culture	Business-centric, close to customer
Long (months)	Cycle Times	Short (days, weeks)

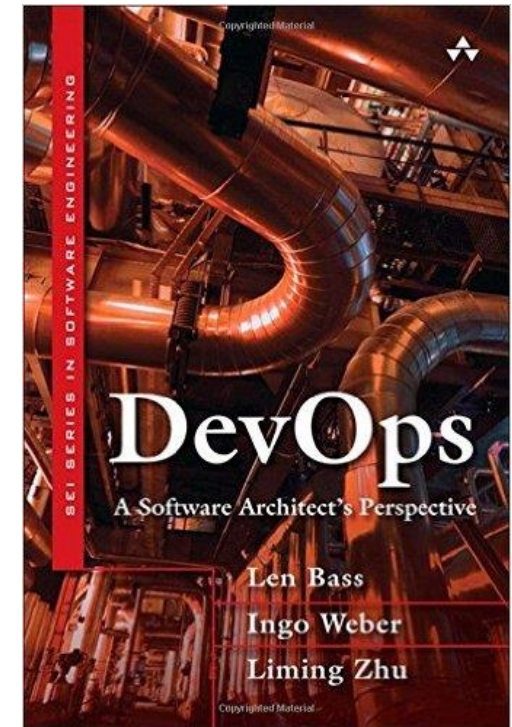
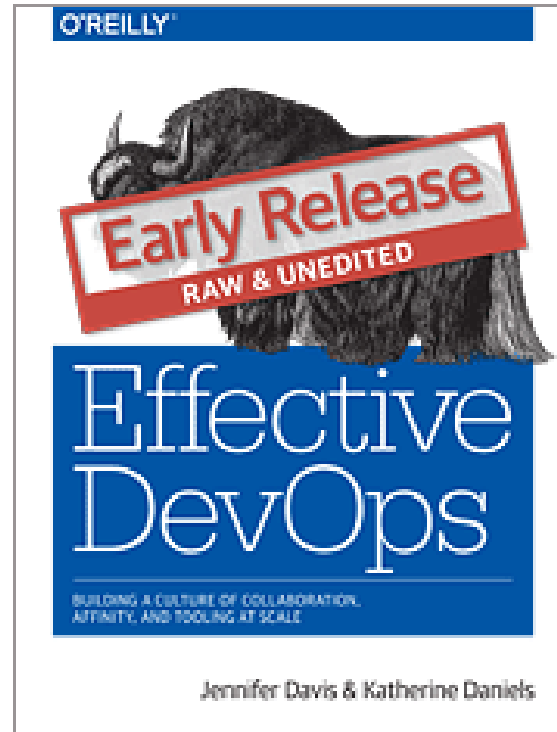
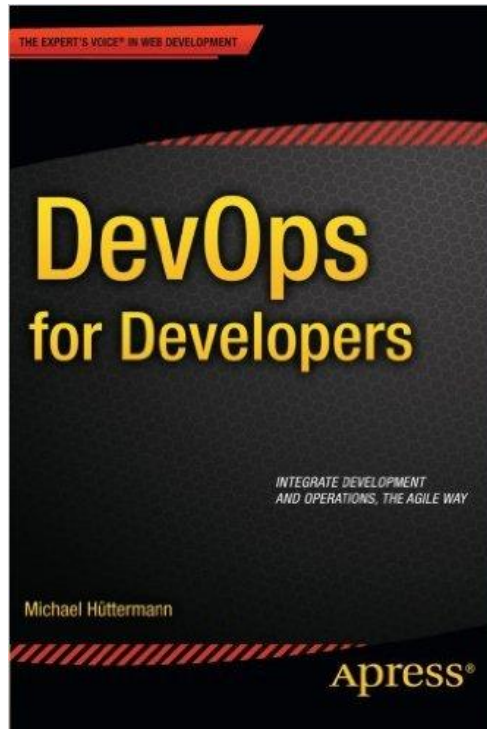
Think
Sprinter



Gartner.

© 2014 Gartner, Inc. and/or its affiliates. All rights reserved.

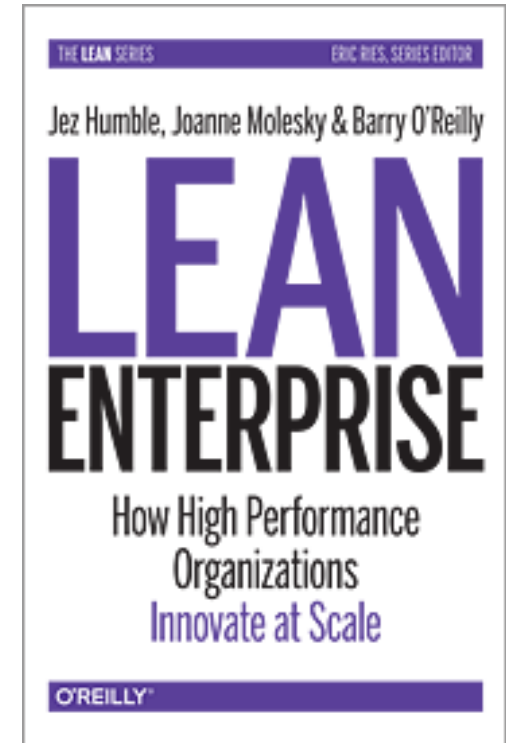
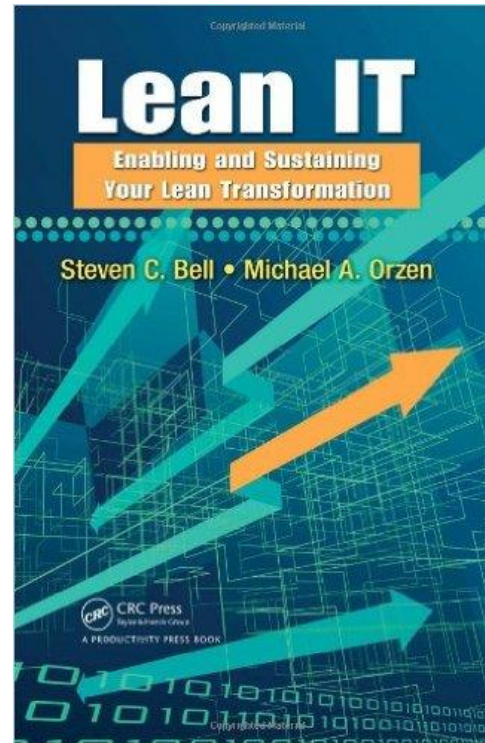
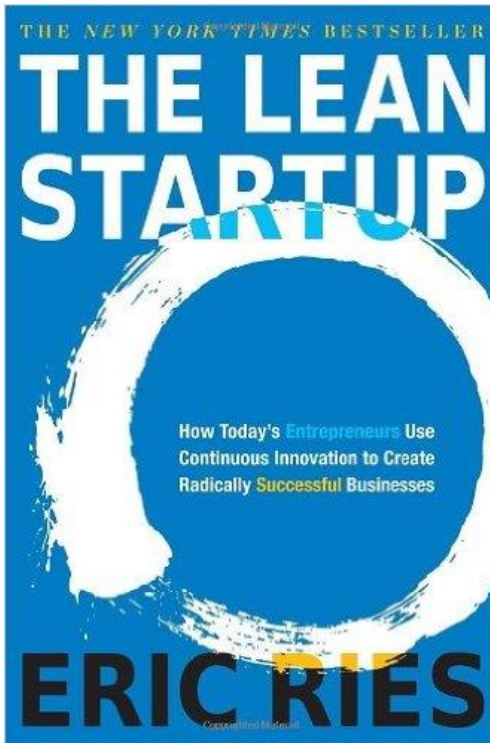
Literatur



Literatur



Literatur



BestOf: DevOps Borat

- To make error is human. To propagate error to all server in automatic way is [#devops](#).
- In Agile is all about fail quick. If you have success in company you do Agile wrong.
- If shit is never hit fan you are not get enough shit done.
- Pair programming is good concept but is involve too many programmer.
- In startup we are allow all developer use 20% of time for write code with not TDD so they can able of actual get shit done.
- In startup we are hire only rockstar, ninja or samurai. Vague familiar with computer is big plus.
- Is not in production until you are receive critical alert of it.
- For minimum viable product we are recommend of start with Hello world. [#leanstartup](#)
- In startup we are gamify site outage. Three outage and ops team is out.
- Best decision we ever make in startup was outsource all site outage to Cloud provider.
- Goal of sysadmin is replace itself with small shell script. Goal of devops is replace itself with small REST API.
- Hello World in cloud is involve 1 load balancer, 3 web server and 2 database server.
- In startup we are practice Outage Driven Infrastructure.