

DevOps:

Softwarearchitektur an der Schnittstelle zwischen Entwicklung und Betrieb

Prof. Dr. Wilhelm (Willi) Hasselbring

<http://se.informatik.uni-kiel.de/>

<http://kosse-sh.de/>



Softwarearchitektur und Agilität

Neal Ford (ThoughtWorks Inc.), 2010:

- Architecture is the stuff that's **hard to change later**.
- And there should be as **little** of that stuff as possible.
- By deferring important architectural and design decisions **until the last responsible moment**, you can prevent unnecessary complexity from undermining your software projects.



User Group »Softwarearchitektur«

- 4. Arbeitstreffen 2012:
 - **Softwarearchitektur und Agilität**

Renaissance & Innovation in Architecture

- Organizations have accepted that "**cloud**" is the de-facto platform of the future, and the benefits and flexibility it brings have ushered in a **renaissance in software architecture**.
- The disposable infrastructure of cloud has enabled the first "cloud native" architecture, **microservices**.
- **Continuous Delivery**, a technique that is radically changing how tech-based businesses evolve, amplifies the impact of cloud as an architecture.
- We expect **architectural innovation** to continue, with trends such as containerization and software-defined networking providing even more technical options and capability.

Quelle: <http://www.thoughtworks.com/radar>

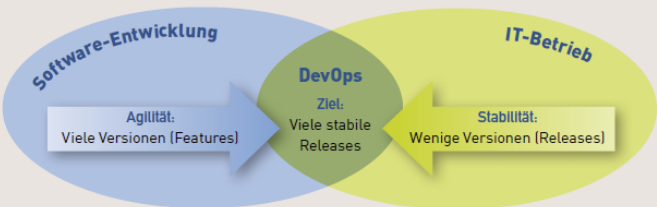


Warum nun eigentlich dieses Vortragsthema?

User Group »Softwarearchitektur«

- 6. Arbeitstreffen 2013:
 - **DevOps: Softwarearchitektur an der Schnittstelle zwischen Entwicklung und Betrieb**
- Message:
 - **Softwarearchitektur ist ein zentrales Artefakt an der Schnittstelle zwischen Entwicklung und Betrieb!**

Zusammenarbeit von Entwicklung und Betrieb



- Maßnahmen:**
- Entwicklung einer Kultur der (agilen) Zusammenarbeit zwischen Entwicklungsabteilung und Systembetrieb.
 - Qualitätssicherung und Effizienzsteigerung durch Automatisierung von Entwicklungs- und Betriebsaufgaben.

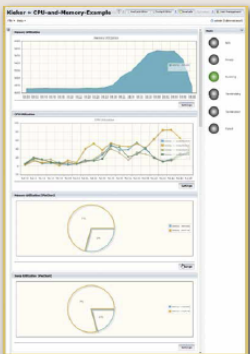
erfordert

liefert Feedback für

Monitoring und Messung

- Möglichst viel automatisiert messen
- Zur kontinuierlichen Überwachung (Monitoring, Darstellung in Dashboards)

- Instrumentierung zum Monitoring sollte schon Initial in die Software integriert werden, statt dies erst nach Problemen im Betrieb zu tun:
- So können die Anforderungen des Betriebs hinsichtlich Elastizität in der Cloud und für die Fehlerdiagnose direkt berücksichtigt werden
 - Performance-Abweichungen werden unmittelbar erkannt, bevor sie über die Deployment-Pipeline in den Betrieb propagiert werden



Ein Beispiel: Das Kieker Monitoring Framework wird im Kompetenzverbund Software-System-Engineering (KOSSE) erfolgreich in Technologieübernahmeprojekten eingesetzt.
<http://kieker-monitoring.net/>

überwacht durch

Automatisierung

- Automatisierung ist der Schlüssel zum DevOps-Erfolg:
- Automatisierter Bau von Systemen aus dem Versionsmanagement-Repository
 - Automatisierte Code-Analysen und Ausführung von Unit-, Integrations- und Systemtests
 - Automatisierte Installation in Test- und Produktionsumgebungen

z.B.: Jenkins: <http://jenkins-ci.org/>

- Neben fachlichen Akzeptanztests kommt automatisierten Tests nicht funktionaler Eigenschaften für den späteren Betrieb eine besonders wichtige Rolle zu:
- Lasttests für Performance und Verfügbarkeit
 - Regressions-Benchmarks um Performance-Abweichungen schnell zu erkennen

z.B.: JMeter: <http://jmeter.apache.org/>

z.B.: MoovBeck: <http://kieker-monitoring.net/moovbeck/>

- Continuous Integration und Continuous Delivery liefern die Werkzeuge, beispielsweise:
- Infrastructure-as-Code mit domänenspezifischen Sprachen zur Konfiguration und Installation, inkl. Versionierung dieses Infrastruktur-Codes
 - Entwicklungsmuster wie Feature-Toggles helfen dabei, Code früh an ausgewählte Nutzer zu liefern, auch wenn dieser noch nicht zur Verwendung durch alle gedacht ist.

z.B.: Puppet: <http://puppetlabs.com/>

z.B.: Toggle: <http://www.toggle.org/>

- Bestimmte Teilaufgaben müssen weiterhin manuell durchgeführt werden:
- Profiling zur Erkennung, Lokalisierung und Diagnose von Performance-Problemen
 - Gebrauchstauglichkeit von Benutzungsoberflächen

z.B.: JIP: <http://jipprof.sourceforge.net/>

Agile Vorgehensweisen, insbesondere testgetriebene Entwicklung, unterstützen DevOps. Fehler, die in der Deployment-Pipeline auftreten, dürfen nicht toleriert werden und müssen zeitnah korrigiert werden. Generell führt die kontinuierliche Integration von Qualitätssicherungsmaßnahmen zu einer kontinuierlich hohen Qualität und damit zu vielen stabilen Releases.

realisiert durch

Deployment Pipeline

Entwicklungs-Umgebung	Continuous Integration	Betriebsnahe Testumgebung	Life-System
Unit-Tests	Unit-Tests, Integrations und Akzeptanz-Tests	Nutzer-Tests	Monitoring
Debugging	Statische Analyse (Syntax Checks, Fehlermuster)	Lasttests	Fehlerdiagnose
Profiling	Dynamische Analyse (Lasttests, Benchmarks)	Monitoring	Elastizität
Versionsmanagement (Anwendungscode, Testdaten, Konfigurationsskripte, Infrastructure-as-Code, etc.)			

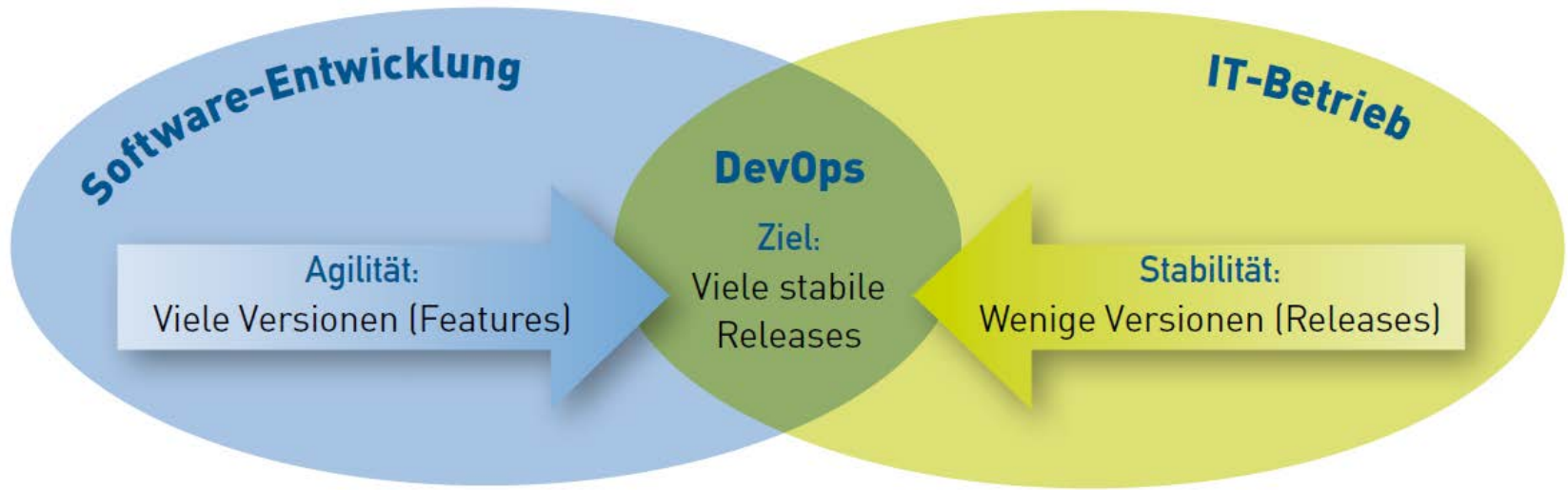
supported by



sponsored by



Zusammenarbeit von Entwicklung und Betrieb



Maßnahmen:

- Entwicklung einer Kultur der (agilen) Zusammenarbeit zwischen Entwicklungsabteilung und Systembetrieb.
- Qualitätssicherung und Effizienzsteigerung durch Automatisierung von Entwicklungs- und Betriebsaufgaben.

Automatisierung

Automatisierung ist der Schlüssel zum DevOps-Erfolg:

- Automatisierter Bau von Systemen aus dem Versionsmanagement-Repository
- Automatisierte Code-Analysen und Ausführung von Unit-, Integrations- und Systemtests
- Automatisierte Installation in Test- und Produktionsumgebungen

z.B.: Jenkins: <http://jenkins-ci.org/>

Neben fachlichen Akzeptanztests kommt automatisierten Tests nicht funktionaler Eigenschaften für den späteren Betrieb eine besonders wichtige Rolle zu:

- Lasttests für Performance und Verfügbarkeit
- Regressions-Benchmarks um Performance-Abweichungen schnell zu erkennen

z.B.: JMeter: <http://jmeter.apache.org/>

z.B.: MooBeuch: <http://kieker-monitoring.net/moobeuch/>

Continuous Integration und Continuous Delivery liefern die Werkzeuge, beispielsweise:

- Infrastructure-as-Code mit domänenspezifischen Sprachen zur Konfiguration und Installation, inkl. Versionierung dieses Infrastruktur-Codes
- Entwicklungsmuster wie Feature-Toggles helfen dabei, Code früh an ausgewählte Nutzer zu liefern, auch wenn dieser noch nicht zur Verwendung durch alle gedacht ist.

z.B.: Puppet: <http://puppetlabs.com/>

z.B.: Toggle: <http://www.toggle.org/>

Bestimmte Teilaufgaben müssen weiterhin manuell durchgeführt werden:

- Profiling zur Erkennung, Lokalisierung und Diagnose von Performance-Problemen
- Gebrauchstauglichkeit von Benutzungsoberflächen

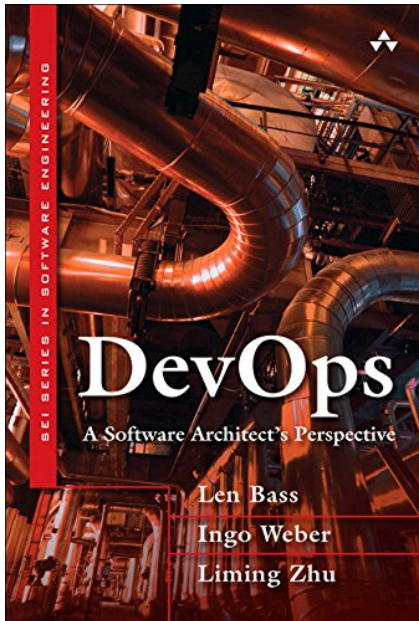
z.B.: JIP: <http://jiprot.sourceforge.net/>

Agile Vorgehensweisen, insbesondere testgetriebene Entwicklung, unterstützen DevOps.

Fehler, die in der Deployment-Pipeline auftreten, dürfen nicht toleriert werden und müssen zeitnah korrigiert werden.

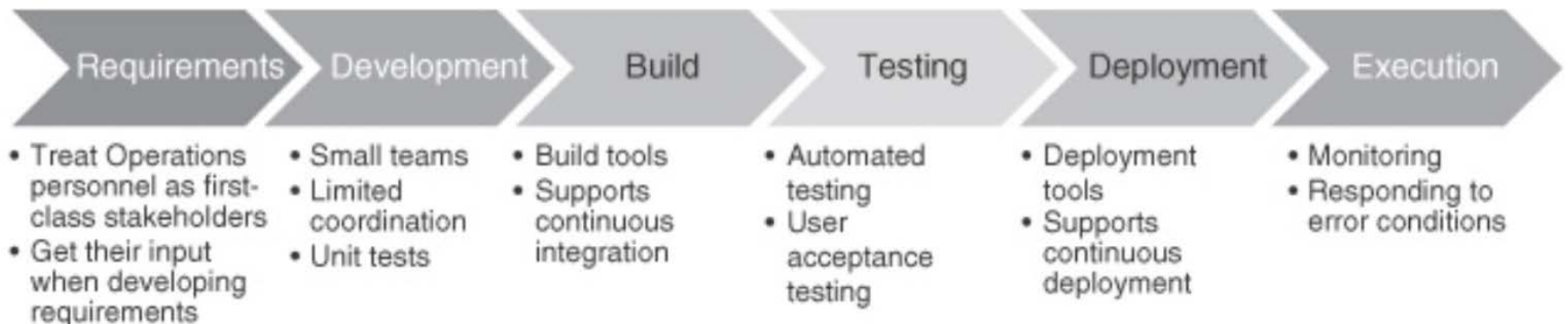
Generell führt die kontinuierliche Integration von Qualitätssicherungsmaßnahmen zu einer kontinuierlich hohen Qualität und damit zu vielen stabilen Releases.

DevOps & Softwarearchitektur



“The **deployment pipeline** is the place where the **architectural** aspects and the process aspects of DevOps intersect.”

[Bas et al. 2015]



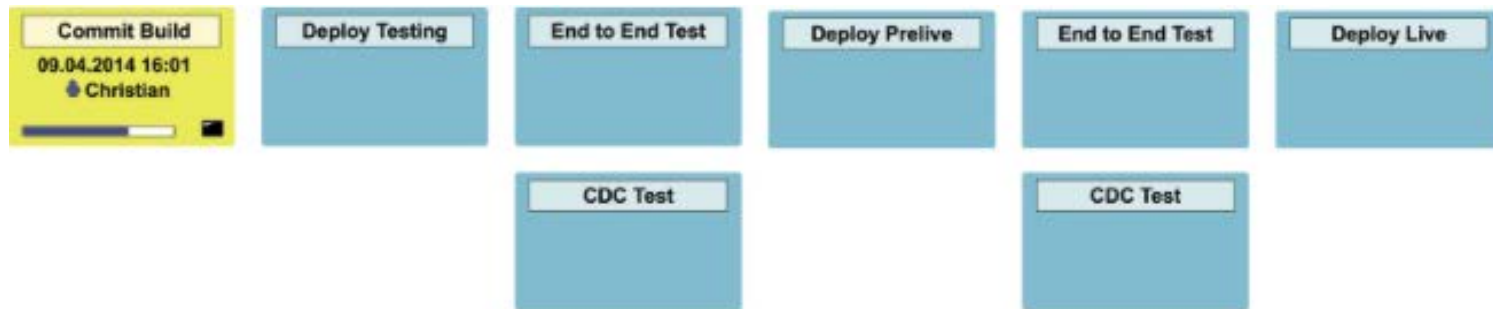
Deployment Pipelines für Continuous Deployment

Deployment Pipeline

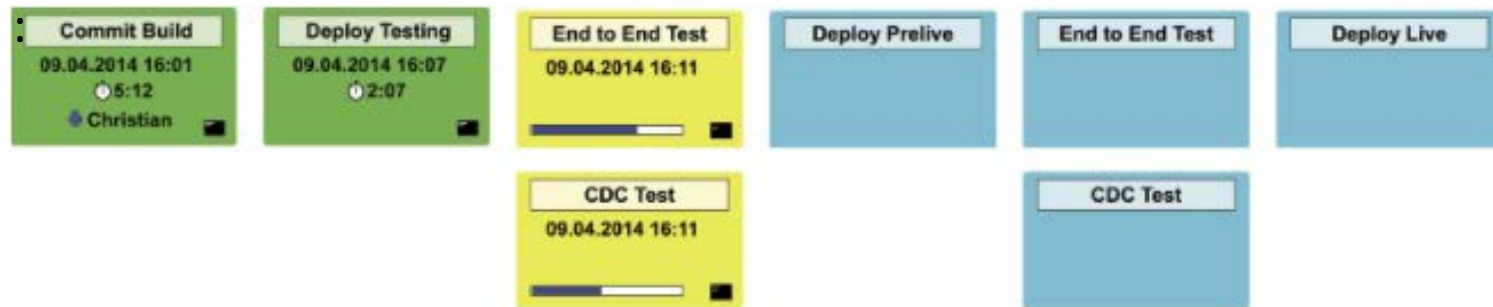


Beispiel: Deployment Pipelines @ Otto

Entwicklungs- und QA-Umgebung



Livestellung mit der parallelen Ausführung von Tests, inklusive Überprüfung von Consumer-Driven Contracts



Quelle: [Breetzmann et al. 2014]

Monitoring und Messung

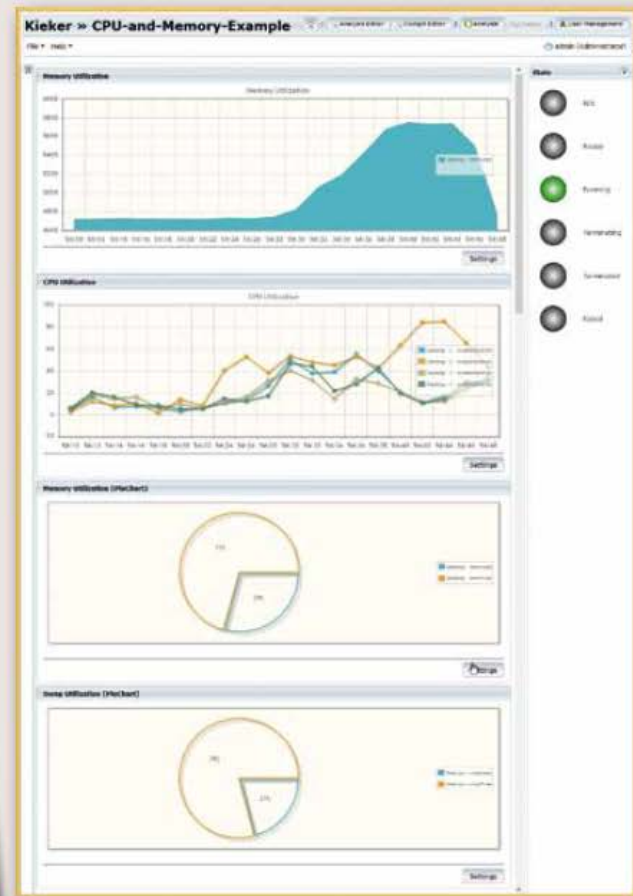
Möglichst viel automatisiert messen

- Zur kontinuierlichen Überwachung
(Monitoring, Darstellung in Dashboards)

Instrumentierung zum Monitoring sollte schon initial in die Software integriert werden, statt dies erst nach Problemen im Betrieb zu tun:

- So können die Anforderungen des Betriebs hinsichtlich Elastizität in der Cloud und für die Fehlerdiagnose direkt berücksichtigt werden
- Performance-Abweichungen werden unmittelbar erkannt, bevor sie über die Deployment-Pipeline in den Betrieb propagiert werden

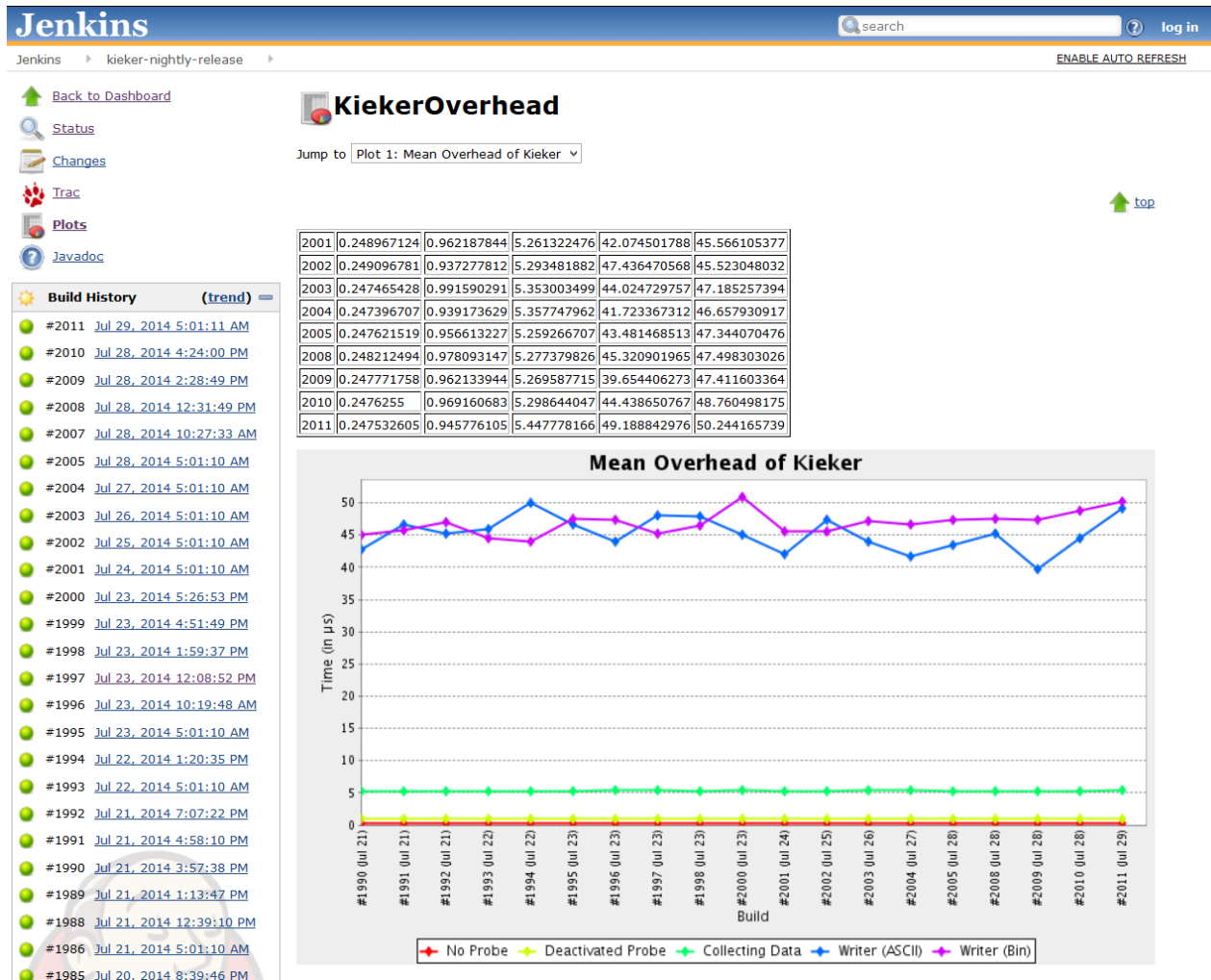
*Ein Beispiel: Das Kieker Monitoring Framework wird im Kompetenzverbund Software Systems Engineering (KoSSE) erfolgreich in Technologietransferprojekten eingesetzt.
<http://kieker-monitoring.net/>*



[van Hoorn et al. 2012]

Automatisierte Qualitätssicherung

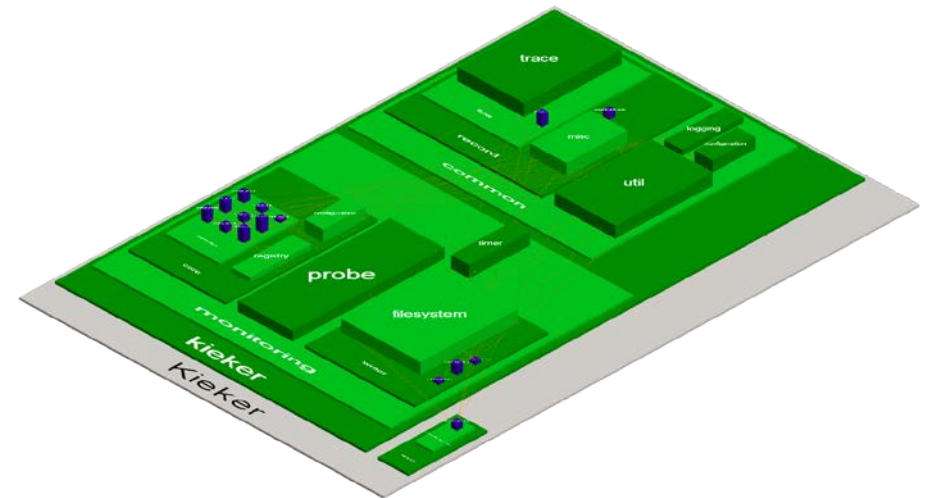
Beispiel: Regressions-Benchmarking



Integriert im
Continuous
Integration Setup
[Waller et al. 2015]

Möglichst mit
automatisierter
Anomalieerkennung
[Marwede et al. 2009,
Ehlers et al. 2011]

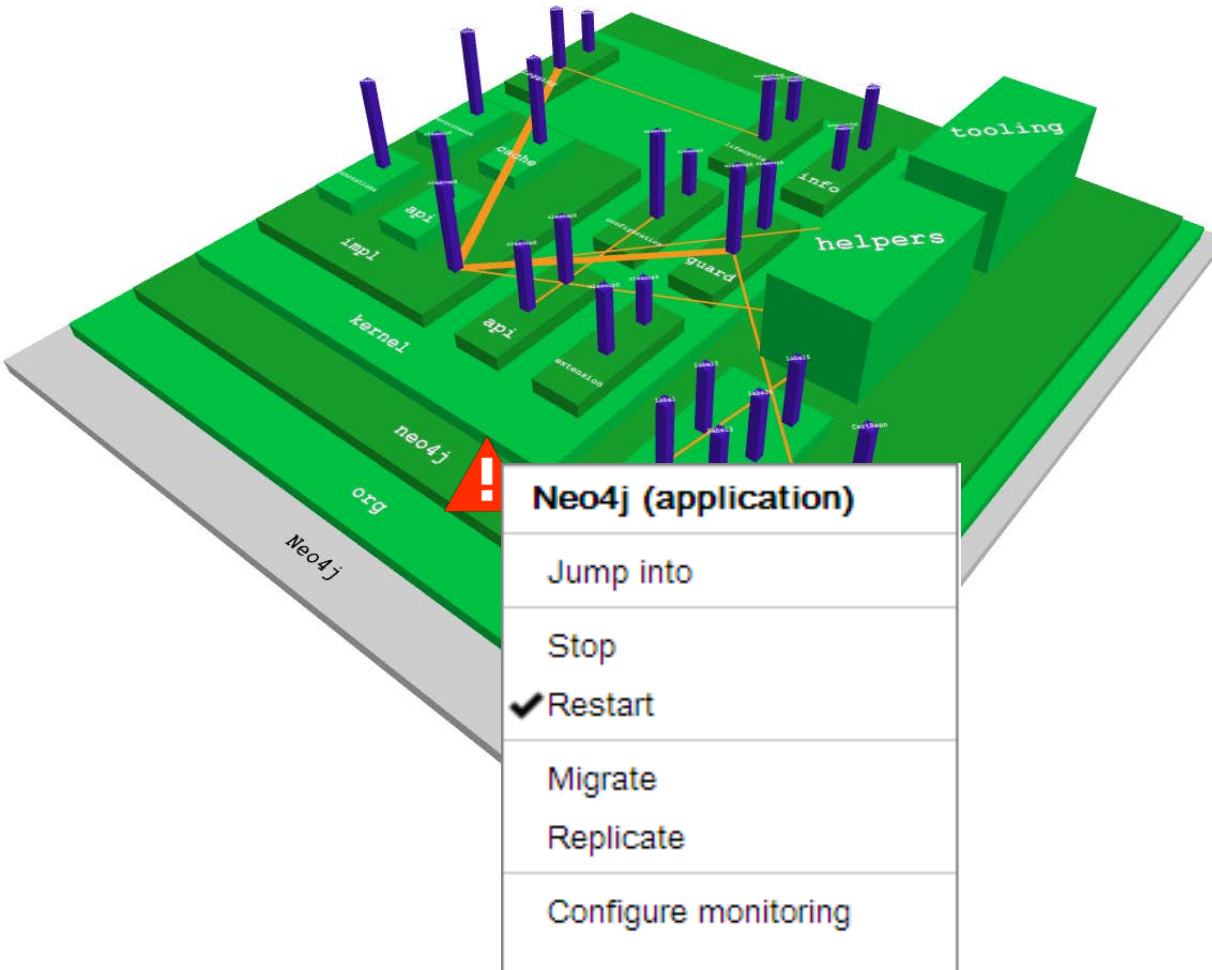
[Fittkau et al. 2013, 2015]



Software-Betriebs-Leitstand

[Giesecke et al. 2006, Fittkau et al. 2014]

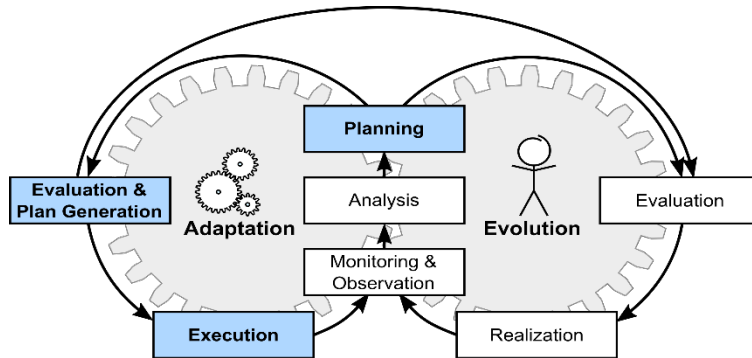
Software-Betriebsleitstände zur kontinuierlichen Überwachung



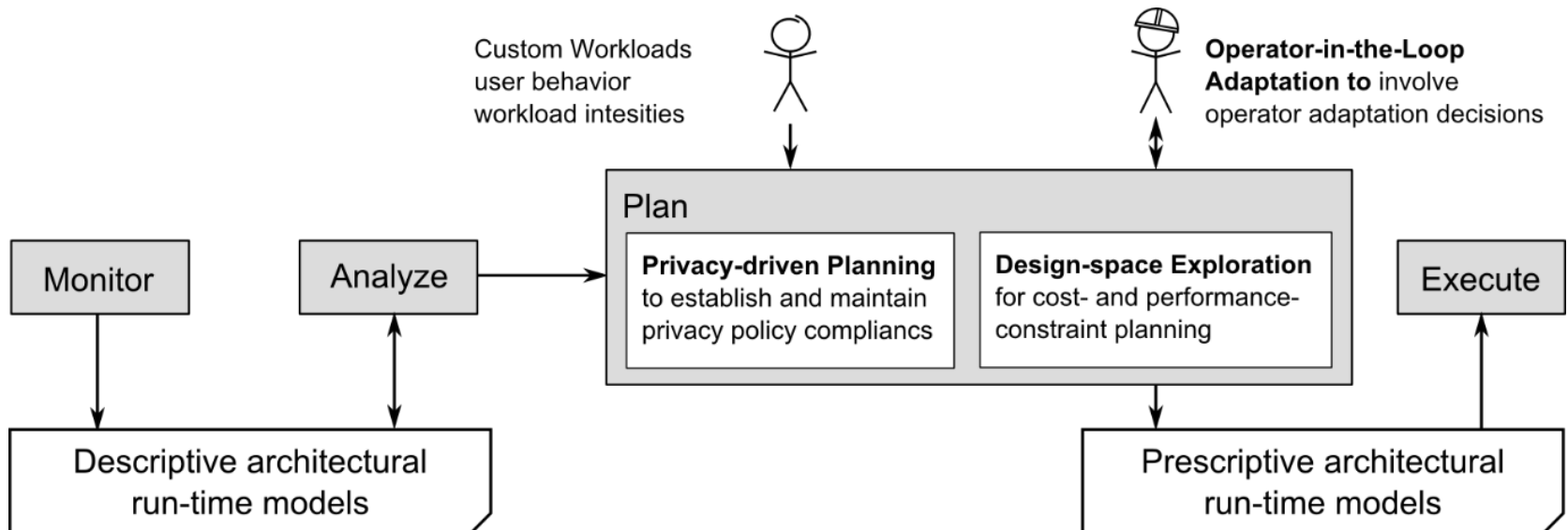
Betriebsleitstände stellen für die Operateure üblicherweise Architektursichten auf die überwachten Systeme dar

[Giesecke et al. 2006, Fittkau et al. 2013, 2014a, 2015]

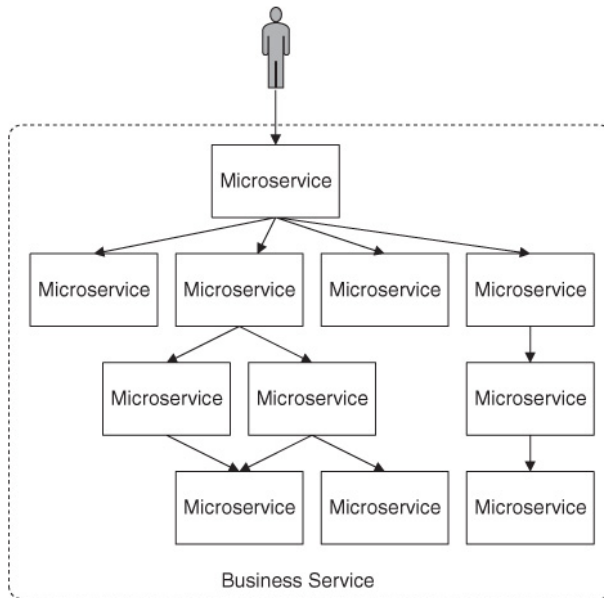
Models @ Runtime für Software-Betriebsleitstände



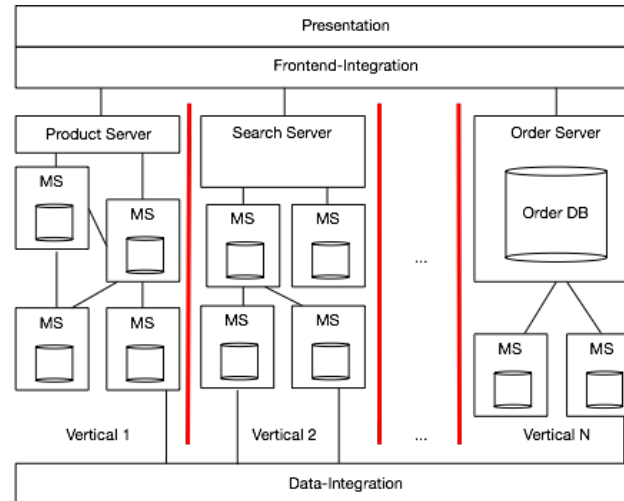
Models @ Runtime in iObserve
[Heinrich et al. 2014]
<http://www.dfg-spp1593.de/iobserve>



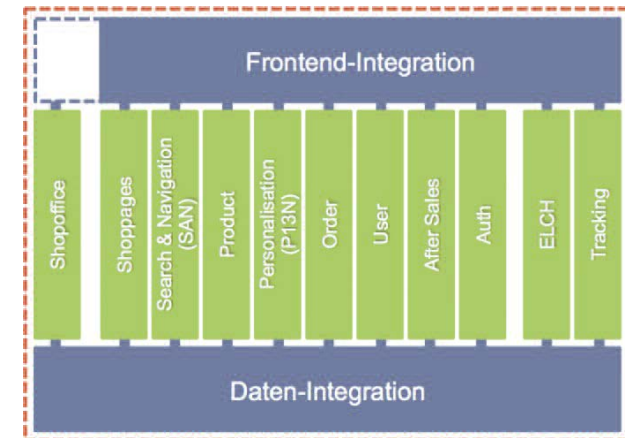
Microservice Architekturen als Enabler für DevOps



[Bas et al. 2015]



[Steinacker 2014]



[Kraus et al. 2013]

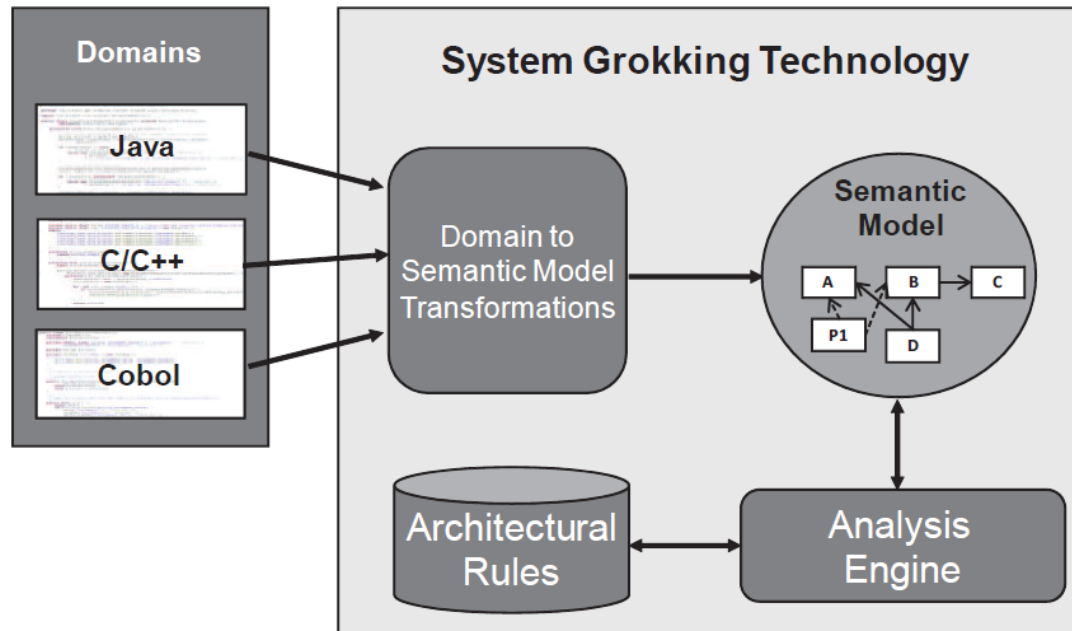
“Scalability is managed by each service individually and is included in its SLA in the form of a guaranteed response time given a particular load.”

[Bas et al. 2015, Chapter 4]

“The trade-off between many small components and a few large components must be considered in component and system design.”

[Hasselbring 2002]

Automatic and Continuous Software Architecture Validation im Continuous Integration



[Goldstein & Segall 2015]

See also [Frey et al. 2013, Fittkau et al. 2014b]

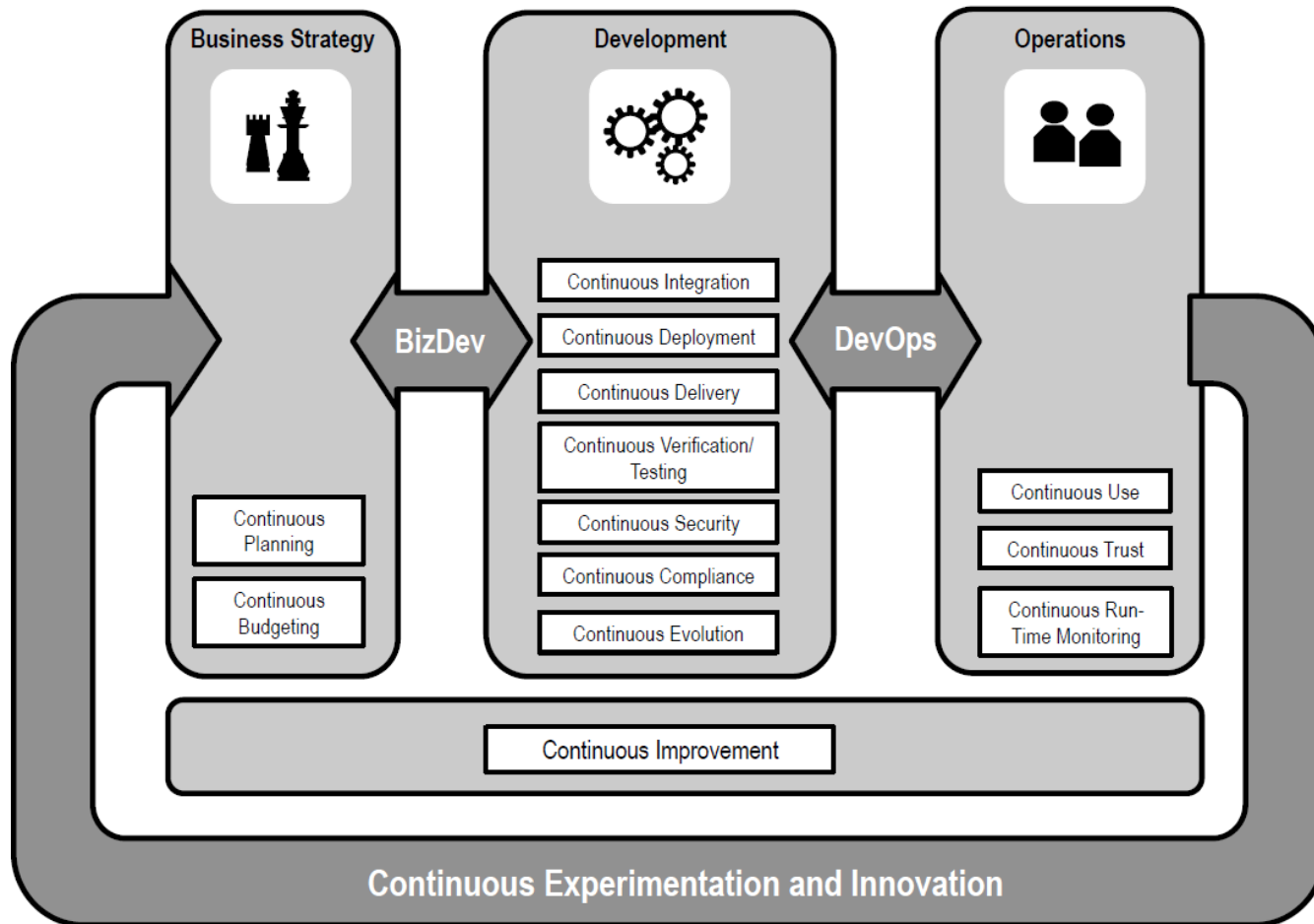


- Durch DevOps bekommt das Thema Softwarearchitektur auch in der agilen Softwareentwicklung eine stärkere Bedeutung und Würdigung
- Für DevOps ist es sinnvoll präskriptive und deskriptive (Architektur-) Modelle zu kombinieren
 - **Präskriptive Modelle** kommen aus der Softwareentwicklung (Forward Engineering)
 - **Deskriptive Modelle** kommen aus der Beobachtung der im Betrieb befindlichen Softwaredienste (Reverse Engineering durch dynamische Analyse)
- Die Integration von präskriptiven und deskriptiven Modellen bietet sich für Softwarebetriebsleitstände an
 - Operator-in-the-Loop Adaption in iObserve und ExplorViz

Softwarearchitektur ist ein zentrales Artefakt an der Schnittstelle zwischen Entwicklung und Betrieb!

Nächster Themenschwerpunkt beim 10. Arbeitstreffen am 17./18. November 2015:
Weiterentwicklung langlebiger Softwarearchitekturen – Design for Change
Weitere Informationen: <http://www.softwareforen.de/softwarearchitektur>

Biz ↔ Dev ↔ Ops



[Fitzgerald and Stol 2015]

References

- [Bas et al. 2015] Len Bass, Ingo Weber, Liming Zhu: “DevOps: A Software Architect’s Perspective”, Addison-Wesley 2015.
- [Breetzmann et al. 2014] Robert Breetzmann, Stephan Kraus, Christian Stamm: “Null Toleranz für Fehler: Wie wir auf otto.de die Qualität hoch halten”, OBJEKTSpektrum 4/2014, 18-23.
- [Ehlers et al. 2011] Jens Ehlers, André van Hoorn, Jan Waller, Wilhelm Hasselbring: “Self-Adaptive Software System Monitoring for Performance Anomaly Localization”, In: 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011).
- [Fittkau et al. 2013] Florian Fittkau, Jan Waller, Christian Wulf, Wilhelm Hasselbring: “Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach”, In: 1st IEEE International Working Conference on Software Visualization (VISOFT 2013).
- [Fittkau et al. 2014a] Florian Fittkau, André van Hoorn, Wilhelm Hasselbring: “Towards a Dependability Control Center for Large Software Landscapes”, In: 10th European Dependable Computing Conference (EDCC 2014).
- [Fittkau et al. 2014b] Florian Fittkau, Phil Stelzer, Wilhelm Hasselbring: “Live Visualization of Large Software Landscapes for Ensuring Architecture Conformance”, In: 2nd International Workshop on Software Engineering for Systems-of-Systems 2014 (SESoS 2014).
- [Fittkau et al. 2015] Florian Fittkau, Sascha Roth, Wilhelm Hasselbring: “ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes”, In: 23rd European Conference on Information Systems (ECIS 2015).
- [Fitzgerald and Stol] Brian Fitzgerald, Klaas-Jan Stol: “Continuous Software Engineering: A Roadmap and Agenda”, In: Journal of Systems and Software, July 2015.
- [Frey et al. 2013] Sören Frey, Wilhelm Hasselbring, Benjamin Schnoor: “Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms”, In: Journal of Software: Evolution and Process, 25(10): 1089-1115.
- [Giesecke et al. 2006] Simon Giesecke, Matthias Rohr, Wilhelm Hasselbring: “Software-Betriebs-Leitstände für Unternehmensanwendungslandschaften”, In: Informatik 2006.
- [Goldstein & Segall 2015] Maayan Goldstein, Itai Segall: “Automatic and Continuous Software Architecture Validation”, In: 37th International Conference on Software Engineering (ICSE 2015) Software Engineering In Practice Track.
- [Hasselbring 2002] Wilhelm Hasselbring: “Component-Based Software Engineering”, In: Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing, Singapore, pp. 289-305, 2002.
- [Heinrich et al. 2014] Robert Heinrich, Eric Schmieders, Reiner Jung, Kiana Rostami, Andreas Metzger, Wilhelm Hasselbring, Ralf Reussner, Klaus Pohl: “Integrating Run-Time Observations and Design Component Models for Cloud System Analysis”, In: 9th Workshop on Models@run.time 2014.
- [Kraus et al. 2013] Stephan Kraus, Guido Steinacker, Oliver Wegner: “Teile und Herrsche – Kleine Systeme für große Architekturen”, OBJEKTSpektrum 5/2013, 8-13.
- [Marwede et al. 2009] Nina Marwede, Matthias Rohr, André van Hoorn, Wilhelm Hasselbring: “Automatic Failure Diagnosis in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation”, In: 13th European Conference on Software Maintenance and Reengineering (CSMR 2009).
- [Steinacker 2014] Guido Steinacker: “Scaling with Microservices and Vertical Decomposition”, <http://dev.otto.de/2014/07/29/scaling-with-microservices-and-vertical-decomposition/>, 2014.
- [van Hoorn et al. 2012] André van Hoorn, Jan Waller, Wilhelm Hasselbring: “Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis”, In: 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012).
- [Waller et al. 2015] Jan Waller, Nils Ehmke, Wilhelm Hasselbring: “Including Performance Benchmarks into Continuous Integration to Enable DevOps”, In: ACM SIGSOFT Software Engineering Notes, 40(2).
- [Waterman et al. 2015] Michael Waterman, James Noble, George Allan: “How Much Up-Front? A Grounded theory of Agile Architecture”, In: 37th International Conference on Software Engineering (ICSE 2015).