INSTITUT FÜR INFORMATIK

# Treiber und Hindernisse für die Einführung von Microservices in der deutschen Softwareindustrie

Holger Knoche, Wilhelm Hasselbring

Bericht Nr. 1702
Juni 2017
ISSN 2192-6247


CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

# Treiber und Hindernisse für die Einführung von Microservices in der deutschen Softwareindustrie

Holger Knoche, Wilhelm Hasselbring

Bericht Nr. 1702
Juni 2017
ISSN 2192-6247

e-mail: hkn@informatik.uni-kiel.de,
wha@informatik.uni-kiel.de

**Zusammenfassung**

Microservices sind ein Architekturstil für Software, dem derzeit sowohl in der Industrie als auch der akademischen Forschung große Aufmerksamkeit zuteil wird. Viele Unternehmen setzen Microservices bereits mit großem Erfolg ein, und die vermeintlichen Vorteile dieses Architekturstils werden in zahlreichen Blogbeiträgen diskutiert. Insbesondere sogenannte „Internet-Scale-Systeme“ nutzen Microservices, um ihre immensen Skalierbarkeitsanforderungen zu erfüllen und neue Funktionen rasch an ihre Nutzer auszuliefern.

Microservices sind jedoch nicht nur bei Unternehmen mit Internet-Scale-Systemen beliebt. Viele „traditionelle“ Unternehmen prüfen derzeit, ob dieser Stil auch für ihre Anwendungen eine praktikable Option ist. Allerdings könnten für diese Unternehmen andere Gründe als beispielsweise Skalierbarkeit ausschlaggebend sein, und sie könnten sich anderen Herausforderungen gegenüber sehen. Zudem könnten sich die Gründe für und gegen Microservices je nach Branche unterscheiden.

In diesem Bericht präsentieren wir die Ergebnisse einer Umfrage zu den Treibern und Hindernissen für die Einführung von Microservices, die unter professionellen Softwareentwicklern in der deutschen Softwareindustrie durchgeführt wurde. Zusätzlich zu den allgemeinen Treibern und Hindernissen legen wir einen besonderen Schwerpunkt auf die Nutzung von Microservices zur Modernisierung von bestehenden Softwaresystemen, wobei wir insbesondere Auswirkungen auf Laufzeitperformance und Transaktionalität hervorheben. Wir konnten interessante Unterschiede zwischen „Early Adopters“ und traditionellen Unternehmen feststellen. Erstere legen besonderen Wert auf die *Skalierbarkeit* ihrer Systeme, während für letztere vor allem die *Wartbarkeit* von Bedeutung ist.

# Inhaltsverzeichnis

# 1 Einführung

Microservices [22; 18] sind ein Architekturstil, der in den letzten Jahren stark in den Fokus der Aufmerksamkeit gerückt ist. Laut Google Trends sind Microservices Anfang 2014 erstmals auf breites Interesse gestoßen, und dieses ist seitdem stetig gewachsen [2]. Der Begriff selbst wurde 2012 geprägt [18], allerdings bestehen Umsetzungen dieses Architekturstils bereits deutlich länger. Beispielsweise begann der Videostreaming-Anbieter Netflix, einer der wohl bestbekannten „Early Adopter", im Jahre 2008 mit der Einführung einer Microservice-Architektur, um die Vorteile des Cloud-Computing bestmöglich ausnutzen zu können [20]. Stand heute nutzen viele bekannte Unternehmen Microservices, beispielsweise Amazon, Spotify, Linkedin und Uber. Mittels dieses Architekturstils haben diese Unternehmen nach eigenen Angaben erstaunliche Skalierbarkeit erreicht und sich in hart umkämpften Geschäftsfeldern behaupten können. Neben Skalierbarkeit [9] können durch Microservices inbesondere auch Agilität und Zuverlässigkeit erreicht werden [10].

Besonders bemerkenswert an Microservices ist die Tatsache, dass viele Unternehmen ihr Wissen und ihre Technologien öffentlich machen. Beispielsweise unterhalten Netflix[1] und Otto[2] Technologie-Blogs, auf denen aktuelle Ideen und Erfahrungen diskutiert werden. Zudem werden viele Programmbibliotheken und Infrastrukturkomponenten, die zur Entwicklung und zum Betrieb von Microservices benötigt werden, kostenfrei als Open-Source-Software veröffentlicht.[3] Weiterhin erlauben Cloud-Anbieter wie Amazon Web Services, die notwendigen Ressourcen schnell und mit geringem Aufwand zu beschaffen. Infolgedessen sind die Einstiegshürden für Microservices verhältnismäßig gering.

Vor diesem Hintergrund stellen viele Unternehmen Überlegungen an, ob Microservices ein gangbarer Weg für ihre Softwaresysteme sind. Jedoch sind viele dieser Systeme nicht „Internet-Scale" und keinem weltweiten Wettbewerb ausgesetzt. Aus diesem Grund könnten diese Unternehmen Microservices aus anderen Gründen in Erwägung ziehen als die „Early Adopter". Noch interessanter als die Gründe für die Einführung von Microservices sind die Hindernisse, die aus Sicht der Unternehmen der Einführung von Microservices entgegenstehen. Zudem könnten die Gründe für und wider Microservices sich je nach Branche unterscheiden.

Obwohl die Einführung von Microservices umfangreich in zahlreichen Blog-Posts und anderen Online-Medien diskutiert wird, existieren derzeit nur wenige belastbare empirische Daten zu diesem Thema. Um zu ergründen, warum „traditionelle" Unternehmen die Einführugn von Microservices in Erwägung ziehen, haben wir eine Umfrage unter professionellen Softwareentwicklern in Deutschland durchgeführt. Da viele Unternehmen bereits über bestehende Softwaresysteme verfügen, haben wir besonderen Wert auf die Frage gelegt, in welchem Umfang Microservices als Mittel zur Modernisierung von Software wahrgenommen werden, welche Modernisierungsziele verfolgt werden, und wie der potentielle Einfluss auf Laufzeitperformance und Transaktionalität eingeschätzt wird.

[1] https://techblog.netflix.com

[2] https://dev.otto.de

[3] https://github.com/Netflix, https://github.com/otto-de

Dieser Bericht ist wie folgt aufgebaut. In Abschnitt 2 werden Grundlagen und wesentliche Eigenschaften von Microservices kurz erläutert. Die Methodik der Umfrage wird in Abschnitt 3 beschrieben. Die Vorstellung der Ergebnisse erfolgt in Abschnitt 4; eine Diskussion der Validität der Ergebnisse sowie verwandter Arbeiten folgt in Abschnitt 5. Ein Fazit in Abschnitt 6 schließt den Bericht.

# 2 Grundlagen

## 2.1 Eigenschaften von Microservice-Architekturen

Es existiert keine allgemeingültige Definition von Microservice-Architekturen. Stattdessen werden sie über eine Reihe üblicher Eigenschaften charakterisiert [8], die im Detail in [18] beschrieben sind. Diese Eigenschaften werden im Folgenden kurz zusammengefasst.

Wie der Name bereits andeutet, bilden *Services* die wesentlichen Bausteine einer Microservice-Architektur, und bilden zugleich das Hauptmittel zur Modularisierung. Die Services laufen in separaten Prozesskontexten, und können individuell bereitgestellt, ersetzt oder abgekündigt werden. Services werden nach fachlichen Gesichtspunkten geschnitten und von Teams betreut, die für den gesamten Lebenszyklus des Service von der Entwicklung bis zum Betrieb verantwortlich sind. Die Teams behalten diese Verantwortung üblicherweise für die gesamte Lebensdauer des Service (*„products, not projects“*).

Im Gegenzug erhalten die Teams ein hohes Maß an Eigenständigkeit; insbesondere werden zentralisierte Steuerung und Datenhaltung so weit wie möglich reduziert. Somit ist es den Teams möglich, beispielsweise die aus ihrer Sicht am besten geeignete Programmiersprache oder Datenbank für den jeweiligen Service zu wählen. Zudem erlaubt es diese Autonomie, Entscheidungen bei Bedarf zu revidieren oder anzupassen, ohne Einfluss auf andere Teams zu nehmen. Ein hohes Maß an Automatisierung erlaubt es den Teams, neue Versionen ihrer Services jederzeit nach eigenem Ermessen in Produktion zu bringen.

Diese organisatorische Eigenständigkeit ist jedoch nur durch strikte technische Isolation und lose Kopplung möglich. Inter-Service-Kommunikation erfolgt ausschließlich über definierte Schnittstellen, die auf plattformunabhängigen Datenformaten und Technologien basieren. Microservices bevorzugen bewährte und leichtgewichtige Komponenten zur Kommunikation. Dazu gehören Web-Technologien wie HTTP und REST sowie Messaging-Komponenten wie Apache Kafka und RabbitMQ. Komplexe Lösungen wie beispielsweise Enterprise Service Buses werden eher gemieden, da sie dazu verleiten können, Geschäftslogik aus den Services in die Kommunikationsinfrastruktur zu verlagern. In Microservice-Architekturen wird hingegen großer Wert darauf gelegt, die Geschäftslogik ausschließlich in den Services zu implementieren (*„smart endpoints and dumb pipes“*).

Als hochgradig verteilte Architektur sind Microservices in besonderem Maße anfällig für Störungen und Teilausfälle. Aus diesem Grund müssen diese Architekturen so gestaltet sein, dass die mögliche Nichtverfügbarkeit benötigter Services so weit wie möglich kompensiert wird, um eine unkontrollierte Ausbreitung der Störung (*„cascading failure“*) zu vermeiden. Zu diesem Zweck sind diverse Muster entstanden [32], und bewährte

Implementierungen sind in frei verfügbaren Bibliotheken wie beispielsweise Hystrix[4] zur Implementierung des Circuit-Breaker-Musters verfügbar. Der Name dieses Musters ist vom englischen Begriff „circuit breaker“ für eine Sicherung in einem Stromkreis abgeleitet. Hierbei werden Abhängigkeiten zu anderen Services in sogenannten Circuit-Breaker-Objekten gekapselt, die als Proxies für diese Ressourcen dienen und deren Verfügbarkeit überwachen. Fällt die betreffende Ressource aus oder überschreitet sie beispielsweise einen gegebenen Schwellwert für die Antwortzeit, „springt die Sicherung heraus“ und liefert Werte aus Caches oder Standardwerte zurück. In diesem Modus prüft die Sicherung üblicherweise in regelmäßigen Abständen, ob die Ressource wieder verfügbar ist, und leitet ggf. wieder Anfragen an diese weiter. Es ist jedoch anzumerken, dass diese Muster von den Entwicklern sorgfältig umgesetzt werden müssen, um die gewünschte Wirkung zu entfalten. Insbesondere das Verhalten im Fehlerfall ist oftmals eine Einzelfallentscheidung.

Um sicherzustellen, dass ihre Microservices ausreichend gegen Teilausfälle gewappnet sind, erzeugen einige Unternehmen sogar absichtlich Störungen in ihren produktiven Umgebungen um zu Testen ob diese Störungen durch das System ohne Systemausfall geeignet behandelt werden. Die von Netflix entwickelte *Simian Army*[5] besteht aus einer Reihe von Werkzeugen, die verschiedene Arten von Störungen hervorrufen können. Das wohl bekannteste dieser Werkzeuge ist der *Chaos Monkey*, der zufällig ausgewählte virtuelle Maschinen oder Container ohne Vorwarnung abschaltet.

## 2.2 Microservices, DevOps und Continuous Delivery

Aus den beschriebenen Eigenschaften wird bereits deutlich, dass es sich bei Microservices um keinen rein technischen Ansatz handelt. Die Forderung nach weitgehend autonomen, funktionsübergreifenden Teams und der Fähigkeit, Services schnell und automatisiert bei Bedarf bereitzustellen, zeigt die enge Verbindung von Microservices mit DevOps [12; 4; 34] und Contiuous Delivery / Deployment [11]. Es existieren jedoch verschiedene Meinungen darüber, wie diese Konzepte genau miteinander in Verbindung stehen. Fowler betrachtet eine DevOps-Kultur als eine Voraussetzung für den Einsatz von Microservices [7], während Wolff den Standpunkt vertritt, dass Microservices nicht zwangsweise zu einer Umsetzung von DevOps führen [35]. Bass et al. sehen Continuous Delivery als Teil von DevOps und merken an, dass viele Unternehmen, die bereits Continuous Delivery praktizieren, nun auf Microservice-Architekturen setzen [3].

## 2.3 Bereitstellung und Betrieb von Microservices

Die Populariät von Microservices hat auch zu einigen bedeutenden technologischen Entwicklungen im Bereich Bereitstellung und Betrieb geführt. Eine dieser Entwicklungen ist die containerbasierte Virtualisierung, ein Konzept, das auch unter dem Namen seiner bekanntesten Implementierung, *Docker*[6], bekannt ist. Im Gegensatz zu klassischen virtuellen Maschinen nutzen Container keinen Hypervisor, sondern Isolationsmechanismen des Kernels des Wirtssystems, der auch die Ressourcenverwaltung für die Container

[4] https://github.com/Netflix/Hystrix
[5] https://github.com/Netflix/SimianArmy
[6] https://www.docker.com/

übernimmt. Damit sind Container zwar weniger stark voneinander isoliert als virtuelle Maschinen, können aber in sehr kurzer Zeit bereitgestellt, gestartet und gestoppt werden. Diese Eigenschaft macht containerbasierte Virtualisierung zu einem wichtigen Werkzeug, um eine hohe Elastizität zu erreichen.

Um diese Elastizität ausschöpfen zu können, wurden Werkzeuge zum automatisierten Management von Clustern entwickelt. Bekannte Werkzeuge sind Kubernetes[7], Docker Swarm, das inzwischen Teil der Docker Engine ist, und DC/OS[8]. Diese Werkzeuge bieten Funktionen wie automatische Skalierung, Service Discovery und automatisches Redeployment, falls ein Knoten ausfallen sollte. Damit wird der Betrieb Microservice-basierter Anwendungen stark vereinfacht.

## 2.4 Microservices und Persistenz

Wie bereits erwähnt, umfasst die Autonomie der Teams auch die Entscheidung, wie die Datenhaltung eines Service realisiert wird. Zwar erlaubt diese Freiheit, die Stärken verschiedener Datenbanksysteme (z.B. relationale Datenbanken oder Graphdatenbanken) gezielt einzusetzen, jedoch bringt sie auch Nachteile mit sich, insbesondere bezüglich Datenkonsistenz.

Vor allem konsistente Datenänderungen über mehrere Services hinweg sind oftmals schwierig zu realisieren. Üblicherweise werden zu diesen Zweck Datenbanktransaktionen eingesetzt. Jedoch bietet nicht jedes Datenbanksystem Transaktionen an, ganz zu schweigen von der Fähigkeit, an verteilten Transaktionen teilzunehmen. Zudem können verteilte Transaktionen die Skalierbarkeit erheblich behindern. Aus diesem Grund wird in Microservice-Architekturen eine transaktionslose Koordination bevorzugt [18]. Ansätze dazu sind explizite Kompensation oder der Try-Cancel-Confirm-Ansatz [25].

Ein weiterer Nachteil, der vor allem für den Betrieb relevant ist, besteht darin, dass konsistente Backups durch verteilte Datenhaltung stark erschwert werden. Daher müssen geeignete Maßnahmen getroffen werden, um die Sicherheit der Daten gegen Verlust zu gewährleisten.

## 2.5 Microservices als Mittel zur Softwaremodernisierung

Ein besonders interessanter Aspekt von Microservices besteht darin, dass verschiedene Autoren sie als geeignetes Mittel zur Modernisierung bestehender, monolithischer Softwaresysteme ansehen [22; 35]. Zudem existieren eine Reihe von Erfahrungsberichten von Unternehmen, die erfolgreich Teile ihrer Softwaresysteme durch Microservices ersetzen oder ersetzt haben. Obwohl einige Unternehmen, darunter das Versandhaus Otto[9], ihre Software von Grund auf neu entwickelt haben [10], empfehlen die meisten Autoren ein inkrementelles Vorgehen. Stine [31] skizziert einen Prozess mit drei wesentlichen Phasen:

1. Neue Funktionen werden ausschließlich als Microservices implementiert. Dem bestehenden Monolithen werden keine neuen Funktionen mehr hinzugefügt.

[7] http://kubernetes.io/
[8] https://dcos.io/
[9] https://dev.otto.de/2015/09/30/on-monoliths-and-microservices/

2. Eine Schnittstellenschicht wird erstellt, die den neu erstellten Microservices Zugriff auf die Funktionen des Monolithen ermöglicht. Diese Schicht dient als „Anti-Corruption Layer“ [6], um das neue und alte Domänenmodell sauber voneinander zu trennen.

3. Bestehende Funktionalität wird schrittweise aus dem Monolithen entfernt und als Microservices reimplementiert. Stine schlägt vor, dabei mit den Funktionen zu beginnen, die den dringendsten Änderungsbedarf haben. Dieser Prozess wird fortgeführt, bis der Monolith entweder vollständig abgelöst ist oder nur noch stabile Funktionalität enthält, die den Migrationsaufwand nicht rechtfertigt.

Es ist wichtig, festzustellen, dass mit Beginn der dritten Phase der Monolith selbst auf die neu geschaffenen Microservices zugreift. Dies kann bedeutende Konsequenzen haben:

- Die Schnittstellenschicht wird nun bidirektional genutzt.
- Vormals native Funktionsaufrufe innerhalb des Monolithen müssen unter Umständen durch Serviceaufrufe ersetzt werden.
- Datenbankzugriffe innerhalb des Monolithen müssen gegebenenfalls restrukturiert oder durch Serviceaufrufe ersetzt werden.

Zusätzlich treffen die zuvor diskutierten Einschränkungen bezüglich Transaktionalität und Konsistenz nun auch auf den Monolithen zu.

# 3 Forschungsdesign

## 3.1 Vorgehensweise

Um einen Einblick in die Treiber und Hindernisse für die Adoption von Microservices zu gewinnen, haben wir eine quantitative Umfrage unter professionellen Softwareentwicklern in der deutschen Softwareindustrie durchgeführt. Zur Steigerung der Relevanz der Antworten wurde die Entscheidung getroffen, primär Entwickler zu befragen, die sich bereits mit dem Thema auseinandergesetzt haben. Aus diesem Grund besuchten wir Konferenzen, die Microservices zum Thema hatten, um dort Teilnehmer persönlich anzusprechen. Auch kontaktierten wir Sprecher auf derartigen Konferenzen per E-Mail und über berufliche soziale Netzwerke. Zudem bewarben wir die Umfrage auf einer bekannten deutschen Entwickler-Webseite. Der Versuch, Teilnehmer über mit Microservices befasste Gruppen in sozialen Netzwerken zu gewinnen, blieb aufgrund schlechter Rücklaufquoten erfolglos.

Die Umfrage erfolgte sowohl mit papiernen als auch mit webbasierten Fragebögen. Die Papierfragebögen wurden genutzt, wenn Teilnehmer persönlich angesprochen wurden, während die webbasierte Variante zum Einsatz kam, wenn Teilnehmer über elektronische Medien kontaktiert wurden. Die Fragebögen umfassten insgesamt 19 Fragen und waren so angelegt, dass sie in 10 bis 15 Minuten ausgefüllt werden konnten. Nach interner Prüfung und Tests pilotierten wir die Umfrage mit 21 Teilnehmern eines Treffens zu Microservices, die großteils in der Finanzdienstleistungsbranche tätig waren. Da die Pilotierung keinen Änderungsbedarf für den Fragebogen ergab und die Methodik

| | Anzahl Teilnehmer | Prozentualer Anteil |
|---|---|---|
| Bis zu 6 Monate | 17 (6) | 23,94% (12,00%) |
| 6 bis 12 Monate | 14 (11) | 17,92% (22,00%) |
| 12 bis 24 Monate | 20 (15) | 28,17% (30,00%) |
| Mehr als 24 Months | 12 (12) | 16,90% (24,00%) |
| Ungültige / keine Antwort | 8 (6) | 11,27% (12,00%) |

Tabelle 1: Befassungsdauer mit Microservices

| | Anzahl Teilnehmer | Prozentualer Anteil |
|---|---|---|
| Architekt | 50 (36) | 70,42% (72,00%) |
| Berater | 9 (9) | 12,68% (18,00%) |
| Entwickler | 40 (30) | 56,34% (60,00%) |
| Führungskraft | 11 (8) | 15,49% (16,00%) |

Tabelle 2: Berufliche Positionen der Teilnehmer

der der Hauptstudie entsprach, konnten wir gemäß [33] die Antworten der Pilotstudie mit der der Hauptstudie zusammenführen. Dies war von besonderem Vorteil, da wir eingeladen wurden, die Ergebnisse der Umfrage beim nächsten Treffen dieser Gruppe zu präsentieren (siehe auch Abschnitt 5.2). Zum Vergleich sind die Ergebnisse exklusive der Daten der Pilotstudie in Tabelle 1 in Klammern mit angegeben.

## 3.2 Demographische Information

Insgesamt nahmen 71 Personen an unserer Umfrage teil. Wie in Tabelle 1 dargestellt, befasste sich etwa die Hälfte der Befragten seit weniger als einem Jahr mit Microservices, ein Drittel seit ein bis zwei Jahren, und ein Fünftel seit mehr als zwei Jahren. Die übrigen Teilnehmer machten keine oder ungültige Angaben.

Bezüglich ihrer beruflichen Position gaben 70% der Teilnehmer an, als Softwarearchitekten tätig zu sein, 13% als Berater, 56% als Entwickler und 15% als Führungskräfte. Die große Mehrheit der Teilnehmer, fast 96%, war in Entwicklungsabteilungen beschäftigt, 21% in Betriebsabteilungen. Die genauen Zahlen sind in den Tabellen 2 und 3 dargestellt. Bei beiden Fragen war es zulässig, mehr als eine Antwort zu geben.

Die Teilnehmer wurden darüber hinaus gebeten, die Branche zu benennen, in der sie tätig sind. Die Antworten wurden manuell in fünf übergeordnete Branchen gruppiert; die Daten sind in Tabelle 4 dargestellt.

| | Anzahl Teilnehmer | Prozentualer Anteil |
|---|---|---|
| Entwicklung | 68 (47) | 95.77% (94.00%) |
| Betrieb | 15 (9) | 21.13% (18.00%) |

Tabelle 3: Abteilungen, in denen die Teilnehmer tätig waren

| | Anzahl Teilnehmer | Prozentualer Anteil |
|---|---|---|
| Beratung / Entwicklung | 16 (15) | 22,54% (30,00%) |
| Energie / Industrie | 11 (9) | 15,49% (18,00%) |
| Finanzdienstleistungen | 20 (6) | 28,17% (12,00%) |
| Handel / E-Commerce | 6 (6) | 8,45% (12,00%) |
| Sonstige / keine Angabe | 18 (14) | 25,35% (28,00%) |

Tabelle 4: Branchen, in denen die Teilnehmer tätig waren

### 3.3 Anmerkungen

Einige Fragen des Fragebogens forderten die Teilnehmer dazu auf, die Bedeutung gegebener Aspekte mittels Gruppen von Likert-Items zu bewerten. Um die Bewertungen der verschiedenen Aspekte vergleichbar zu machen, wurde eine Punktzahl als gewichteter Durchschnitt der Antworten pro Item berechnet, womit diese implizit als intervallskaliert angenommen wurden. Diese Interpretation von Likert-Items ist umstritten [14], weswegen wir die Punktzahl nur zur groben Sortierung der Aspekte nutzen. Zudem mussten in den Balkendiagrammen einige Werte aufgrund von Rundungsfehlern um einen Prozentpunkt angepasst werden, damit die Summe exakt 100% ergab.

## 4 Ergebnisse

### 4.1 Bestehende Nutzung von Microservices

Zu Beginn unseres Fragebogens stand die Frage, in welchem Umfang Microservices in den Unternehmen bereits eingesetzt wurden. Wie in Abbildung 1 dargestellt, gab fast ein Drittel (27%) der Befragten an, dass ihr Unternehmen oder ihre Kunden Microservices bereits in großem Umfang einsetzen. Die übrigen Teilnehmer gaben zu gleichen Teilen (37%) an, Microservices in geringem Umfang oder gar nicht einzusetzen.

Ein Blick auf die branchenspezifischen Zahlen offenbart hingegen deutliche Unterschiede. Während Microservices im Bereich Handel und E-Commerce offenbar sehr intensiv genutzt werden (Median: Einsatz in großem Umfang), kommen sie bei Finanzdienstleistern nur selten zum Einsatz (Median: Kein Einsatz). Zudem ist bemerkenswert, dass der Anteil der Intensivnutzer im Bereich Energie und Industrie deutlich unter dem Durchschnitt liegt.

**Zusammenfassung:** *Microservices werden bereits in nennenswertem Umfang in der Praxis eingesetzt, der Grad der Nutzung schwankt jedoch deutlich zwischen den Branchen. Die intensivste Nutzung wurde im Bereich E-Commerce festgestellt, die geringste bei Finanzdienstleistern.*

### 4.2 Treiber für die Einführung von Microservices

Um wichtige Treiber für die Einführung von Microservices zu identifizieren, wurden die Teilnehmer aufgefordert, neun Eigenschaften, die Microservices gemeinhin zugeschrieben werden, mit Hinblick auf ihre Bedeutung für die Entscheidung für den Einsatz von

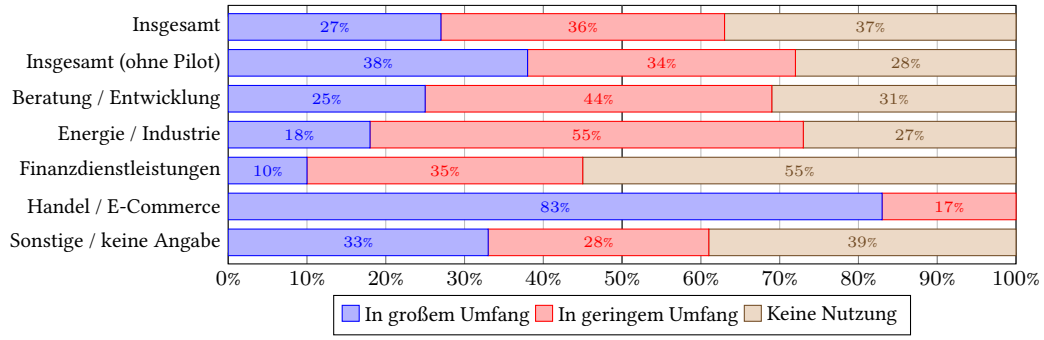


Abbildung 1: Bestehende Nutzung von Microservices in verschiedenen Branchen

Microservices zu bewerten. Die Auswahl der Treiber geht im Wesentlichen auf die Eigenschaften zurück, die in [18], [22] und [35] beschrieben werden. Die folgenden potentiellen Treiber waren zu bewerten:

1. *Einfache und gezielte Skalierbarkeit / Elastizität* – Microservices können aufgrund ihrer individuellen Deploybarkeit durch Hinzufügen und Entfernen von Instanzen gezielt skaliert werden.
2. *Kurze „Time to Market“* – Neue Funktionen können schnell und nach eigenem Ermessen der Teams in Produktion gebracht werden.
3. *Gute Wartbarkeit* – Als lose gekoppelte und einfach ersetzbare Einheiten sind Microservices weniger anfällig dafür, stark verwobene Codestrukturen auszubilden.
4. *Wahlfreiheit der Implementierungssprache („Polyglot Programming“)* – Die Implementierungssprache(n) können für jeden Service frei gewählt werden.
5. *Wahlfreiheit der Persistenz („Polyglot Persistence“)* – Die zugrundeliegende Datenhaltung kann für jeden Service frei gewählt werden.
6. *Wegbereiter für Continuous Delivery und DevOps* – Als bevorzugte Architektur für Continuous Delivery und DevOps können Microservices als Wegbereiter für diese Ansätze dienen.
7. *Eignung für Cloud-Deployments und Container* – Microservices erlauben es, die Vorteile der Cloud und containerbasierter Virtualisierung auszunutzen.
8. *Organisatorische Verbesserungen* – Die Ausrichtung der Teams an den Servicegrenzen kann zu einer Verbesserung der organisatorischen Effizienz führen.
9. *Attraktivität als Arbeitgeber* – Die Nutzung moderner Architekturen und Technologien kann die Attraktivität eines Unternehmens für potentielle Bewerber erhöhen.

Die Bewertung dieser Treiber erfolgte mittels einer Likert-Skala mit den Optionen *ausschlaggebend*, *bedeutsam*, *wenig bedeutsam* und *unbedeutend*. Die Option *ausschlaggebend* sollte von den Teilnehmern für Treiber gewählt werden, die aus ihrer Sicht allein bereits den Einsatz von Microservices rechtfertigen.

| | Insgesamt | | Beratung / Entwicklung | | Energie / Industrie | | Finanzdienst-leistungen | | Handel / E-Commerce | | Sonstige / keine Angabe | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | % | Wertung | % | Wertung | % | Wertung | % | Wertung | % | Wertung | % | Wertung |
| Skalierbarkeit und Elastizität | 34 / 46<br>20 / 0 | 2,14<br>(2,18) | 44 / 31<br>25 / 0 | 2,19 | 36 / 45<br>18 / 0 | 2,16 | 20 / 75<br>5 / 0 | 2,15 | 67 / 33<br>0 / 0 | 2,67 | 28 / 33<br>39 / 0 | 1,89 |
| Wartbarkeit | 28 / 56<br>11 / 3 | 2,12<br>(2,08) | 31 / 69<br>0 / 0 | 2,31 | 40 / 40<br>10 / 10 | 2,10 | 25 / 65<br>10 / 0 | 2,15 | 33 / 50<br>17 / 0 | 2,16 | 22 / 50<br>22 / 6 | 1,88 |
| Time to Market | 31 / 51<br>13 / 6 | 2,08<br>(1,98) | 25 / 62<br>12 / 0 | 2,11 | 9 / 55<br>9 / 27 | 1,46 | 25 / 65<br>5 / 5 | 2,10 | 67 / 33<br>0 / 0 | 2,67 | 44 / 28<br>28 / 0 | 2,16 |
| Wegbereiter CD und DevOps | 14 / 45<br>28 / 13 | 1,60<br>(1,56) | 19 / 38<br>38 / 6 | 1,71 | 18 / 27<br>18 / 36 | 1,26 | 10 / 55<br>30 / 5 | 1,70 | 17 / 50<br>33 / 0 | 1,84 | 11 / 50<br>22 / 17 | 1,55 |
| Eignung für Virtualisierung | 15 / 35<br>38 / 11 | 1,53<br>(1,68) | 31 / 38<br>31 / 0 | 2,00 | 9 / 9<br>64 / 18 | 1,09 | 5 / 45<br>30 / 20 | 1,35 | 0 / 67<br>17 / 17 | 1,51 | 22 / 28<br>44 / 6 | 1,66 |
| Organisatorische Verbesserung | 6 / 41<br>38 / 15 | 1,38<br>(1,44) | 6 / 38<br>38 / 19 | 1,32 | 9 / 36<br>18 / 36 | 1,17 | 0 / 55<br>40 / 5 | 1,50 | 17 / 50<br>33 / 0 | 1,84 | 6 / 28<br>50 / 17 | 1,24 |
| Polyglot Persistence | 6 / 34<br>42 / 18 | 1,28<br>(1,20) | 12 / 19<br>31 / 38 | 1,05 | 0 / 36<br>36 / 27 | 1,08 | 5 / 40<br>45 / 10 | 1,40 | 0 / 33<br>67 / 0 | 1,33 | 6 / 39<br>44 / 11 | 1,40 |
| Polyglot Programming | 10 / 28<br>42 / 20 | 1,28<br>(1,40) | 6 / 19<br>44 / 31 | 1,00 | 18 / 55<br>18 / 9 | 1,82 | 5 / 20<br>60 / 15 | 1,15 | 33 / 33<br>33 / 0 | 1,98 | 6 / 28<br>39 / 28 | 1,13 |
| Attraktivität als Arbeitgeber | 3 / 23<br>34 / 41 | 0,89<br>(0,94) | 0 / 38<br>25 / 38 | 1,01 | 9 / 0<br>27 / 64 | 0,54 | 0 / 15<br>55 / 30 | 0,85 | 17 / 50<br>17 / 17 | 1,68 | 0 / 22<br>28 / 50 | 0,72 |

Tabelle 5: Treiber für den Einsatz von Microservices in verschiedenen Branchen

Die Ergebnisse der Antworten aus den einzelnen Branchen sind in Tabelle 5 dargestellt. Die Einträge sind absteigend nach der branchenübergreifenden Bewertung sortiert. Es sind jeweils die prozentualen Anteile der vier Optionen (von *ausschlaggebend* oben links bis *unbedeutend* unten rechts) und die Wertung als gewichtetes Mittel aufgeführt, wobei die Gewichtung von 3 (*ausschlaggebend*) bis 0 (*unbedeutend*) reicht.

Wie aus der Tabelle ersichtlich, wurden Skalierbarkeit, Wartbarkeit und Time to Market als primäre Treiber für die Einführung von Microservices angegeben, die jeweils von etwa einem Drittel der Teilnehmer als ausschlaggebend eingestuft wurden. Weitere bedeutende Treiber sind die Rolle als Wegbereiter für Continuous Delivery und DevOps sowie die Eignung für Cloud und Container. Die Verbesserung der Organisationsstruktur sowie die Wahlfreiheit von Programmiersprache und Persistenz werden als weniger bedeutend erachtet, die Erhöhung der Attraktivität als Arbeitgeber wird mit deutlichem Abstand als am wenigsten relevant eingeschätzt.

Die Betrachtung der branchenspezifischen Zahlen zeigt, dass im Bereich Beratung und Entwicklung insbesondere die Wartbarkeit als Treiber für den Einsatz von Microservices wahrgenommen wird. Auch die Eignung für Virtualisierung ist in dieser Branche im Vergleich stark ausgeprägt. Die polyglotte Programmierung ist in den Bereichen Energie und Industrie sowie E-Commerce überdurchschnittlich wichtig. Im Bereich E-Commerce wurde zudem der Aspekt der Attraktivität als Arbeitgeber deutlich höher gewertet als in den übrigen Branchen, was insbesondere aufgrund des hohen Nutzungsgrads in dieser Branche interessant ist.

Einige Treiber wurden auch bemerkenswert niedrig eingestuft. Besonders auffällig sind die Aspekte der kurzen Time to Market und die Eignung für Virtualisierung im Bereich Energie und Industrie, die in dieser Branche sehr viel weniger bedeutsam zu sein scheinen.

**Zusammenfassung:** *Skalierbarkeit, Wartbarkeit und Time to Market sind die primären Treiber für den Einsatz von Microservices. In einigen Branchen sind aber auch polyglotte Programmierung und die Eignung für Virtualisierung von hoher Bedeutung.*

## 4.3 Hindernisse für den Einsatz von Microservices

Die Indentifikation wesentlicher Hindernisse für den Einsatz von Microservices erfolgte analog zu den Treibern. Allerdings werden derartige Hindernisse deutlich weniger in der Literatur behandelt als die Treiber. Die für die Umfrage genutzte Auswahl der Hindernisse basiert auf [35], ergänzt um Aspekte, die in unseren Gesprächen mit Entwicklern zur Sprache kamen. Die Hindernisse umfassen sowohl abstrakte Hindernisse wie Widerstände gegenüber Veränderungen als auch konkrete Herausforderungen der Implementierung. Zur besseren Strukturierung wurden diese in zwei separate Fragen aufgeteilt. Folgende abstrakte Hindernisse waren durch die Teilnehmer zu bewerten:

1. *Widerstände durch Entwickler* – Die Entwickler könnten nicht *willens* sein, die notwendigen Veränderungen für den Einsatz von Microservices mitzutragen.

2. *Unzureichende Fähigkeiten der Entwickler* – Die Entwickler könnten nicht *fähig* sein, die notwendigen Veränderungen für den Einsatz von Microservices mitzutragen.

3. *Widerstände durch den Betrieb* – Die für den Betrieb der Anwendungen zuständigen Mitarbeiter könnten nicht *willens* sein, die notwendigen Veränderungen für den Einsatz von Microservices mitzutragen.

4. *Unzureichende Fähigkeiten des Betriebs* – Die für den Betrieb der Anwendungen zuständigen Mitarbeiter könnten nicht *fähig* sein, die notwendigen Veränderungen für den Einsatz von Microservices mitzutragen.

5. *Lizenzen und Supportverträge* – Die weitgehend freie Wahl der Werkzeuge und Bibliotheken durch die Teams kann den Aufwand erhöhen, ausreichende Lizenzen und Supportverträge für den Betrieb sicherzustellen.

6. *Komplexität des Deployments* – Als hochgradig verteilte Architektur bedingen Microservices komplexere Deployments als traditionelle, monolithische Anwendungen.

7. *Compliance und Vorschriften* – Die Prozesse und organisatorischen Veränderungen, die mit dem Einsatz von Microservices einhergehen, können schwer mit Complianceregelungen oder gesetzlichen Vorschriften vereinbar sein, etwa verpflichtenden Codereviews oder Sicherheits- und Haftungsfragen [5].

8. *Unzureichende Reife der Technologien* – Da Microservices erst kürzlich in den Fokus der Aufmerksamkeit gerückt sind, können die zugrundeliegenden Werkzeuge und Technologien als noch nicht reif für den produktiven Einsatz wahrgenommen werden.

9. *Konsistenz von Backups* – Polyglotte und dezentrale Persistenz erschwert das Erstellen konsistenter Backups, was zu Konflikten mit bestehenden Prozessen und Praktiken des Datenbankbetriebs führen kann.

10. *Mangelnde Kompatibilität* – Die neuen Technologien, die mit Microservices eingeführt werden, könnten mit bestehenden Technologien inkompatibel sein.

Die Einstufung dieser Hindernisse erfolgte analog zu den Treibern, jedoch war in diesem Fall die Relevanz für einen *Verzicht* auf den Einsatz von Microservices zu bewerten. Daher stand anstelle der Einstufung *ausschlaggebend* die Einstufung *Showstopper* zur Verfügung.

Wie aus Tabelle 6 ersichtlich, wurden unzureichende Fähigkeiten des Betriebs und der Entwickler sowie Widerstände des Betriebs als vorrangige Hindernisse für die Einführung von Microservices eingestuft, gefolgt von der Komplexität des Deployments. Die übrigen Punkte sind insgesamt recht niedrig eingestuft, jedoch werden die Verträglichkeit mit Compliance und Vorschriften sowie die Widerstände durch Entwickler in verschiedenen Branchen sehr unterschiedlich eingeschätzt.

Die branchenspezifischen Zahlen legen nahe, dass die Verträglichkeit mit Compliance insbesondere im Bereich Beratung und Entwicklung und der Finanzdienstleistungsbranche zu Bedenken führt. In ersterer wurde dieser Aspekt von 25% der Befragten als *Showstopper* eingestuft, in letzterer erhielt dieser die höchste Wertung aller Branchen (Median: *bedeutend*). Die Schwierigkeit konsistenter Backups ist insbesondere für die Finanzdienstleistungsbranche von Bedeutung und erhielt die dritthöchste Bewertung; 60% der Befragten bewerteten diesen Aspekt als *bedeutend*.

Die rollen- und abteilungsspezifischen Zahlen, die aus Platzgründen nicht dargestellt sind, zeigen, dass Widerstände und fehlende Fähigkeiten insbesondere von Beratern und Führungskräften als besonders bedeutend eingeschätzt werden. So bewerteten 44% der Berater unzureichende Fähigkeiten des Betriebs als *Showstopper* (Wertung: 2,09). Führungskräfte waren besonders skeptisch bezüglich unzureichender Fähigkeiten der Entwickler, die von 36% der Befragten als *Showstopper* eingestuft wurden und die höchste Wertung dieser Rolle erhielten (1,80). Bemerkenswert bei der Betrachtung der abteilungsspezifischen Daten ist die Tatsache, dass sowohl Entwicklungs- als auch Betriebsabteilung sich darin einig sind, dass unzureichende Fähigkeiten das größte Hindernis darstellen, und diese Aspekte auch sehr ähnlich bewertet haben.

**Zusammenfassung:** *Unzureichende Fähigkeiten der Entwickler und des Betriebs und Widerstände des Betriebs werden als die bedeutendsten Hindernisse für die Einführung von Microservices angesehen. Dies trifft insbesondere auf Berater und Führungskräfte zu. Abhängig von der Branche sind auch Widerstände der Entwickler und Unverträglichkeit mit Compliance und Vorschriften von Bedeutung.*

Die konkreten Implementierungsherausforderungen wurden ebenfalls auf einer vierstufigen Skala bewertet. Da die Befragten die Schwierigkeit der Umsetzung bewerten sollten, waren die Stufen *einfach*, *mittel*, *schwierig* und *unrealisierbar*. Folgende Herausforderungen waren zu bewerten:

1. *Automatisierte Deployment-Pipelines* – Automatisierte Deployment-Pipelines sind für die gewünschte Autonomie der Teams notwendig, sind in der Praxis jedoch oftmals schwierig zu realisieren [17].

2. *Ad-hoc-Provisionierung von Ressourcen* – Die kurzfristige Provisionierung von Ressourcen wie Maschinen oder Containern ist sowohl für die Deployment-Pipelines als auch die Elastizität in Produktion vonnöten.

| | Insgesamt | | Beratung / Entwicklung | | Energie / Industrie | | Finanzdienst-leistungen | | Handel / E-Commerce | | Sonstige / keine Angabe | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung |
| Unzureichende Fähigkeiten (Ops) | 16 / 46<br>33 / 6 | 1,73<br>(1,80) | 25 / 62<br>6 / 6 | 2,05 | 0 / 36<br>55 / 9 | 1,27 | 5 / 42<br>53 / 0 | 1,52 | 17 / 50<br>17 / 17 | 1,68 | 28 / 39<br>28 / 6 | 1,90 |
| Widerstand durch Ops | 14 / 47<br>28 / 10 | 1,65<br>(1,72) | 25 / 50<br>12 / 12 | 1,87 | 9 / 55<br>27 / 9 | 1,64 | 5 / 57<br>37 / 0 | 1,68 | 0 / 50<br>33 / 17 | 1,33 | 22 / 28<br>33 / 17 | 1,55 |
| Unzureichende Fähigkeiten (Dev) | 20 / 34<br>34 / 11 | 1,62<br>(1,48) | 12 / 50<br>31 / 6 | 1,67 | 9 / 18<br>36 / 36 | 0,99 | 21 / 37<br>42 / 0 | 1,79 | 33 / 17<br>33 / 17 | 1,66 | 28 / 33<br>28 / 11 | 1,78 |
| Komplexität des Deployments | 4 / 46<br>35 / 14 | 1,39<br>(1,48) | 0 / 69<br>25 / 6 | 1,63 | 0 / 27<br>18 / 55 | 0,72 | 5 / 30<br>60 / 5 | 1,35 | 0 / 33<br>33 / 33 | 0,99 | 11 / 61<br>28 / 0 | 1,83 |
| Compliance und Vorschriften | 17 / 23<br>25 / 35 | 1,22<br>(1,32) | 25 / 19<br>25 / 31 | 1,38 | 0 / 18<br>9 / 73 | 0,45 | 15 / 35<br>35 / 15 | 1,50 | 0 / 33<br>50 / 17 | 1,16 | 28 / 11<br>17 / 44 | 1,23 |
| Mangelnde Kompatibilität | 9 / 27<br>33 / 31 | 1,14<br>(1,08) | 6 / 19<br>50 / 25 | 1,06 | 9 / 27<br>27 / 36 | 1,08 | 0 / 32<br>42 / 26 | 1,06 | 0 / 0<br>50 / 50 | 0,50 | 22 / 39<br>6 / 33 | 1,50 |
| Konsistente Backups | 1 / 41<br>27 / 31 | 1,12<br>(1,08) | 0 / 38<br>25 / 38 | 1,01 | 0 / 18<br>36 / 45 | 0,72 | 5 / 60<br>25 / 10 | 1,60 | 0 / 33<br>33 / 33 | 0,99 | 0 / 39<br>22 / 39 | 1,00 |
| Unzureichende technolog. Reife | 3 / 20<br>49 / 28 | 0,98<br>(0,96) | 6 / 25<br>38 / 31 | 1,06 | 0 / 9<br>55 / 36 | 0,73 | 0 / 21<br>63 / 16 | 1,05 | 0 / 33<br>17 / 50 | 0,83 | 6 / 17<br>50 / 28 | 1,02 |
| Widerstand durch Dev | 11 / 20<br>24 / 45 | 0,97<br>(0,80) | 12 / 19<br>31 / 38 | 1,05 | 27 / 9<br>18 / 45 | 1,17 | 10 / 25<br>30 / 35 | 1,10 | 0 / 17<br>17 / 67 | 0,51 | 6 / 22<br>17 / 56 | 0,79 |
| Supportverträge und Lizenzen | 3 / 15<br>49 / 32 | 0,88<br>(0,86) | 0 / 0<br>56 / 44 | 0,56 | 0 / 9<br>55 / 36 | 0,73 | 5 / 25<br>55 / 15 | 1,20 | 0 / 17<br>17 / 67 | 0,51 | 6 / 22<br>44 / 28 | 1,06 |

Tabelle 6: Hindernisse für den Einsatz von Microservices in verschiedenen Branchen

3. *Umsetzung dezentraler Persistenz* – Die Realisierung dezentraler und polyglotter Persistenz kann im Unternehmensumfeld schwer zu realisieren sein, da insbesondere hier oftmals zentrale Persistenz bevorzugt wird.

4. *Betriebsaufgaben in Entwicklungsteams* – Der Gedanke funktionsübergreifender Teams, die für Microservices gewünscht werden, bedingt, dass Betriebsaufgaben durch Entwicklungsteams übernommen werden. Dies schließt auch wenig beliebte Aufgaben wie beispielsweise regelmäßige Rufbereitschaft mit ein.

5. *Aufgabenverlagerung für Betriebsteams* – Betriebsteams sehen sich einer Veränderung ihres Aufgabenfelds ausgesetzt, da Teile ihrer täglichen Arbeit in den Deployment-Pipelines automatisiert oder von den Entwicklungsteams übernommen werden. An diese Stelle treten neue Aufgaben wie beispielsweise Unterstützung und Weiterentwicklung der Pipelines.

6. *Betrieb verteilter Anwendungen* – Der zuverlässige Betrieb verteilter Anwendungen ist eine Herausforderung und unterscheidet sich deutlich vom Betrieb traditioneller, monolithischer Anwendungen, den viele Betriebsteams gewohnt sind.

7. *Automatisierung von Integrationstests und weiterer Teststufen* – Automatisierte Deployments erfordern einen hohen Automationsgrad der Tests. Nach unserer Erfahrung werden insbesondere Integrationstests und spätere Teststufen häufig fast ausschließlich manuell getestet. Dieser manuelle Testaufwand muss zum Erreichen kurzfristiger Deployments reduziert werden.

8. *Infrastructure as Code* – Um die kurzfristige Bereitstellung der notwendigen Infrastruktur zu gewährleisten, ist eine formale Beschreibung dieser Infrastruktur vonnöten, was oftmals als „Infrastructure as Code“ bezeichnet wird. Dieser Ansatz muss unter Umständen zunächst etabliert werden.

9. *Umsetzung polyglotter Programmierung* – Die Realisierung polyglotter Programmierung kann zusätzliche Werkzeuge, Prozesse und Kenntnisse erfordern, die ggf. erst aufgebaut werden müssen.

| | Insgesamt | | Beratung / Entwicklung | | Energie / Industrie | | Finanzdienst-leistungen | | Handel / E-Commerce | | Sonstige / keine Angabe | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung |
| Betrieb verteilter Anwendungen | 0 / 56<br>37 / 7 | 1,49<br>(1,54) | 0 / 69<br>25 / 6 | 1,63 | 0 / 36<br>64 / 0 | 1,36 | 0 / 50<br>35 / 15 | 1,35 | 0 / 50<br>50 / 0 | 1,50 | 0 / 67<br>28 / 6 | 1,62 |
| Aufgabenverlagerung des Betriebs | 3 / 52<br>34 / 11 | 1,47<br>(1,52) | 6 / 62<br>31 / 0 | 1,73 | 0 / 36<br>36 / 27 | 1,08 | 0 / 50<br>45 / 5 | 1,45 | 0 / 83<br>0 / 17 | 1,66 | 6 / 44<br>33 / 17 | 1,39 |
| Einrichtung eines Monitoring | 1 / 42<br>49 / 7 | 1,36<br>(1,36) | 0 / 50<br>38 / 12 | 1,38 | 0 / 9<br>82 / 9 | 1,00 | 0 / 45<br>50 / 5 | 1,40 | 0 / 17<br>83 / 0 | 1,17 | 6 / 61<br>28 / 6 | 1,68 |
| Ressourcenprovisionierung | 4 / 37<br>45 / 14 | 1,31<br>(1,20) | 0 / 19<br>50 / 31 | 0,88 | 0 / 27<br>73 / 0 | 1,27 | 0 / 55<br>45 / 0 | 1,55 | 0 / 33<br>33 / 33 | 0,99 | 17 / 39<br>28 / 17 | 1,57 |
| Dezentrale Persistenz | 3 / 38<br>46 / 13 | 1,31<br>(1,24) | 0 / 38<br>50 / 12 | 1,26 | 0 / 18<br>64 / 18 | 1,00 | 0 / 40<br>50 / 10 | 1,30 | 0 / 67<br>17 / 17 | 1,51 | 11 / 39<br>39 / 11 | 1,50 |
| Testautomatisierung | 0 / 34<br>54 / 13 | 1,22<br>(1,22) | 0 / 44<br>44 / 12 | 1,32 | 0 / 18<br>73 / 9 | 1,09 | 0 / 30<br>55 / 15 | 1,15 | 0 / 17<br>83 / 0 | 1,17 | 0 / 44<br>39 / 17 | 1,27 |
| Betrieb durch Entwicklungsteams | 1 / 33<br>43 / 23 | 1,12<br>(1,18) | 6 / 25<br>56 / 12 | 1,24 | 0 / 36<br>27 / 36 | 0,99 | 0 / 32<br>53 / 16 | 1,17 | 0 / 50<br>17 / 33 | 1,17 | 0 / 33<br>39 / 28 | 1,05 |
| Einrichtung von Pipelines | 1 / 28<br>49 / 21 | 1,08<br>(1,10) | 0 / 6<br>69 / 25 | 0,81 | 0 / 45<br>36 / 18 | 1,26 | 0 / 30<br>55 / 15 | 1,15 | 0 / 33<br>17 / 50 | 0,83 | 6 / 33<br>44 / 17 | 1,28 |
| Infrastructure as Code | 4 / 20<br>51 / 24 | 1,03<br>(1,02) | 0 / 12<br>56 / 31 | 0,80 | 0 / 18<br>45 / 36 | 0,81 | 0 / 25<br>60 / 15 | 1,10 | 0 / 17<br>50 / 33 | 0,84 | 18 / 24<br>41 / 18 | 1,43 |
| Polyglotte Programmierung | 1 / 25<br>32 / 41 | 0,85<br>(0,72) | 6 / 6<br>38 / 50 | 0,68 | 0 / 27<br>18 / 55 | 0,72 | 0 / 55<br>35 / 10 | 1,45 | 0 / 0<br>33 / 67 | 0,33 | 0 / 17<br>33 / 50 | 0,67 |

Tabelle 7: Konkrete Implementierungsherausforderungen in verschiedenen Branchen

10. *Einrichtung eines geeigneten Monitoring* – Da Microservices anfällig für partielle Ausfälle sind, muss ein geeignetes Monitoring etabliert werden, so dass Probleme in Produktion schnell erkannt und lokalisiert werden können.

Wie aus Tabelle 7 ersichtlich, erhielten der Betrieb verteilter Anwendungen sowie die Aufgabenverlagerung für Betriebsteams die höchste Wertung von den Befragten, gefolgt von der Einrichtung eines Monitoring, kurzfristiger Ressourcenprovisionierung und dezentraler Persistenz. Jedoch scheint keine dieser Herausforderungen als Showstopper eingeordnet zu werden, da keine von einer nennenswerten Anzahl Teilnehmer *unrealisierbar* bewertet wurde. Es bestehen jedoch bemerkenswerte Unterschiede zwischen den Branchen mit Hinblick auf bestimmte Herausforderungen. So wurde beispielsweise die kurzfristige Provisionierung von Ressourcen in der Finanzdienstleistungsbranche deutlich schwieriger eingeschätzt als im Bereich Beratung und Entwicklung. Auch die Unterstützung polyglotter Programmierung wurde durch erstere deutlich schwieriger eingeschätzt als in den übrigen Branchen.

**Zusammenfassung:** *Der Betrieb verteilter Anwendungen sowie die Aufgabenverlagerung des Betriebs werden als schwierigste konkrete Herausforderungen eingeschätzt. Jedoch gibt es keine echten Showstopper.*

## 4.4 Modernisierungsziele für bestehende Anwendungen

Wie bereits zuvor erwähnt, werden Microservices auch als vielversprechende Option zur Modernisierung bestehender Anwendungen betrachtet. Zwei Drittel der Befragten gaben an, dass es Pläne oder sogar Projekte gäbe, Microservices in bestehende Anwendungen einzuführen, wie Abbildung 2 zeigt.

Bemerkenswerterweise bejahten 92% der geringfügigen Microservice-Nutzer diese Frage, verglichen mit 79% der intensiven Nutzer bzw. 32% der Nicht-Nutzer. Diese Zahlen deuten an, dass Unternehmen, die bereits Microservices einsetzen, die Intensivierung der Nutzung anstreben, während Unternehmen, die bislang keine Microservices nutzen, dies wahrscheinlich in naher Zukunft auch nicht tun werden. Auf die Frage, ob nur neue

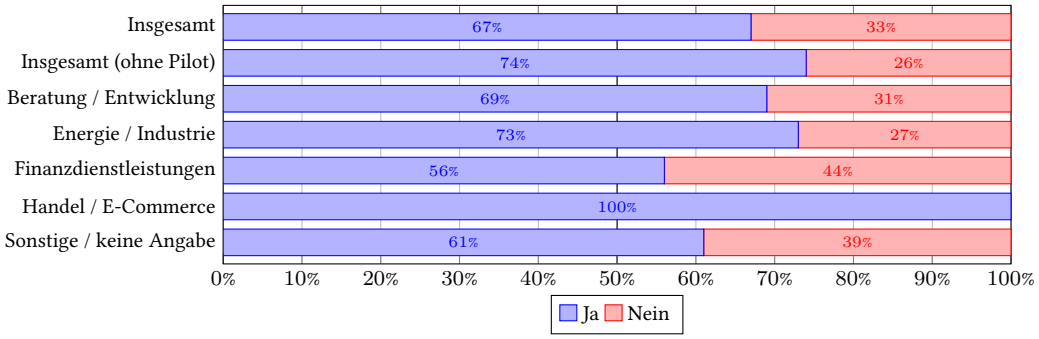


Abbildung 2: Pläne zur Einführung von Microservices in bestehende Anwendungen in verschiedenen Branchen

Funktionalität ergänzt oder auch bestehende ersetzt werden solle, gaben 85% der Befragten an, dass sie auch bestehende Funktionalität ersetzen würden.

Ähnlich zu den Treibern für die Einführung von Microservices sollten auch wesentliche Modernisierungsziele identifiziert werden, die von Unternehmen verfolgt werden. Zu diesem Zweck wurden die Teilnehmer befragt, ob sie die folgenden potentiellen Ziele *primär*, *sekundär* oder *gar nicht* mit Microservices verfolgen würden:

1. Verbesserung der Wartbarkeit
2. Verbesserung der „Time to Market“
3. Verbesserung der Skalierbarkeit
4. Verbesserung der Qualität
5. Vorbereitungen für Continuous Delivery oder DevOps
6. Einführung neuer Technologien
7. Erhöhung der Teammotivation

Die Antworten sind in Tabelle 8 dargestellt. Die Gewichtungen für die Antworten reichen von 2 (*primär*) bis 0 (*gar nicht*). Wie aus der Tabelle ersichtlich, ist die Verbesserung der Wartbarkeit das primäre Modernisierungsziel, gefolgt von der Verbesserung der Time to Market sowie Skalierbarkeit. Die Verbesserung der Qualität sowie Vorbereitungen für Continuous Delivery und DevOps werden ebenfalls als wichtig eingestuft. Die Einführung neuer Technologien und die Verbesserung der Teammotivation folgen mit deutlichem Abstand.

**Zusammenfassung:** *Zwei Drittel der Befragten gaben an, dass Pläne oder Projekte zur Einführung von Microservices in bestehende Anwendungen existieren. Eine große Mehrheit würde zudem bestehende Anwendungsteile durch Microservices ersetzen. Die Verbesserung der Wartbarkeit ist das primäre Modernisierungsziel, gefolgt von kürzerer Time to Market, Skalierbarkeit und Qualität.*

| | Insgesamt | | Beratung / Entwicklung | | Energie / Industrie | | Finanzdienst-leistungen | | Handel / E-Commerce | | Sonstige / keine Angabe | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung | % | Wer-tung |
| Verbesserung Wartbarkeit | 82 / 15 3 | 1,79 (1,70) | 81 / 19 0 | 1,81 | 100 / 0 0 | 2,00 | 85 / 15 0 | 1,85 | 50 / 50 0 | 1,50 | 78 / 11 11 | 1,67 |
| Verbesserung Time to Market | 61 / 32 7 | 1,54 (1,52) | 88 / 12 0 | 1,88 | 36 / 36 27 | 1,08 | 60 / 40 0 | 1,60 | 83 / 17 0 | 1,83 | 44 / 44 11 | 1,32 |
| Verbesserung Skalierbarkeit | 51 / 45 4 | 1,47 (1,46) | 50 / 44 6 | 1,44 | 73 / 18 9 | 1,64 | 60 / 40 0 | 1,60 | 50 / 50 0 | 1,50 | 28 / 67 6 | 1,23 |
| Verbesserung Qualität | 48 / 44 8 | 1,40 (1,38) | 25 / 56 19 | 1,06 | 73 / 18 9 | 1,64 | 50 / 50 0 | 1,50 | 50 / 50 0 | 1,50 | 50 / 39 11 | 1,39 |
| Vorbereitung CD und DevOps | 51 / 38 11 | 1,40 (1,36) | 62 / 31 6 | 1,55 | 45 / 55 0 | 1,45 | 45 / 35 20 | 1,25 | 67 / 33 0 | 1,67 | 44 / 39 17 | 1,27 |
| Einführung neuer Technologie | 30 / 45 25 | 1,05 (1,08) | 19 / 50 31 | 0,88 | 36 / 36 27 | 1,08 | 20 / 50 30 | 0,90 | 50 / 50 0 | 1,50 | 39 / 39 22 | 1,17 |
| Verbesserung Teammotivation | 21 / 55 24 | 0,97 (0,90) | 19 / 56 25 | 0,94 | 18 / 45 36 | 0,81 | 20 / 70 10 | 1,10 | 50 / 50 0 | 1,50 | 17 / 44 39 | 0,78 |

Tabelle 8: Modernisierungsziele für bestehende Anwendungen

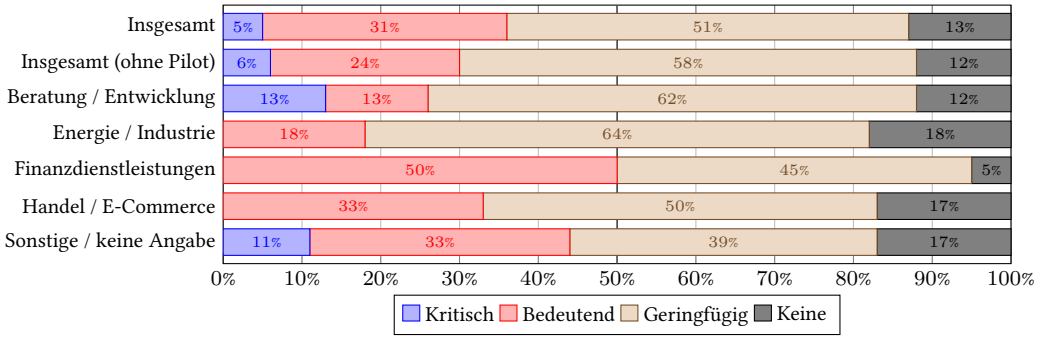


Abbildung 3: Erwartete Performanceverschlechterung aufgrund von Interprozess- und Netzwerkkommunikation

## 4.5 Performance und Transaktionalität

Eine fundamentale Eigenschaft von Microservices besteht darin, dass jede Instanz in einem eigenen Prozesskontext ausgeführt wird. Damit führt jeder Serviceaufruf zumindest zu Interprozesskommunikation, in der Regel sogar zu Netzwerkkommunikation. Diese Kommunikationsformen sind bekanntermaßen deutlich langsamer als native Methodenaufrufe. Aus diesem Grund kann das Auslagern bestehender Funktionalität in Microservices eine deutliche Verschlechterung der Performance hervorrufen.

Allerdings sahen nur 36% der Befragten die Gefahr einer bedeutenden oder gar kritischen Verschlechterung. 51% erwarteten eine geringfügige, 13% sogar gar keine Performanceauswirkung. Die detaillierten Zahlen sind in Abbildung 3 dargestellt.

Die Einführung von Serviceaufrufen kann die Performance auch mittelbar beeinflussen. Wie in [15] gezeigt, kann die zeitliche Verlängerung existierender Transaktionen den Grad der Konflikte („Lock Contention“) in der Datenbank erhöhen und somit den Transaktionsdurchsatz verringern. Eine derartige Verlängerung kann auch dann auftreten, wenn der Serviceaufruf selbst nicht an der Transaktion teilnimmt.

Der Großteil der Befragten (66%) sah hier ein mögliches Problem, erwartete jedoch nur in Einzelfällen spürbare Auswirkungen. Immerhin 18% der Teilnehmer bewerteten dies als ein ernsthaftes Problem; insbesondere die Teilnehmer aus der Finanzdienstleistungsbranche, von denen 30% diese Einstufung vornahmen. Die detaillierten Zahlen zeigt Abbildung 4.

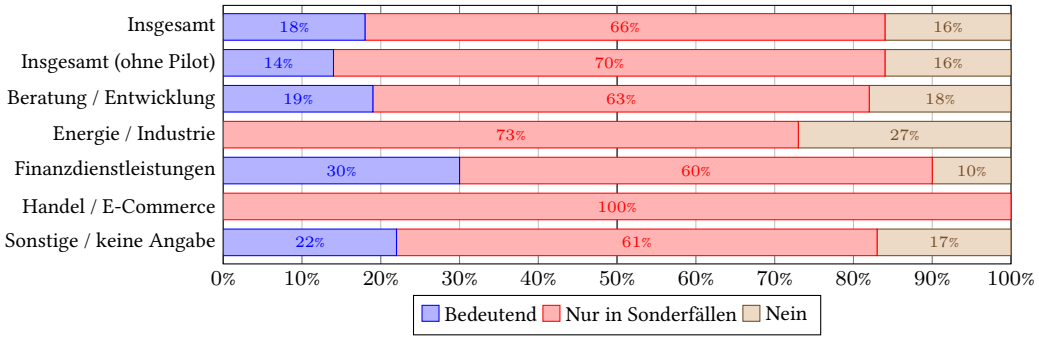


Abbildung 4: Erwarteter Performanceverlust in bestehenden Transaktionen

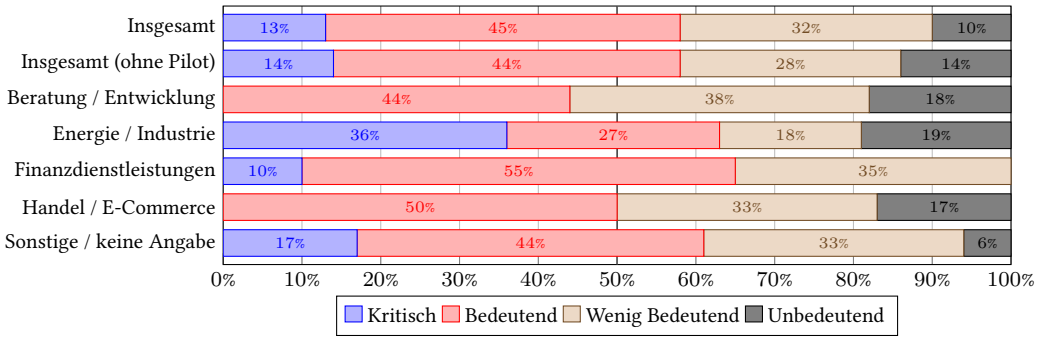


Abbildung 5: Bedeutung der Nichtverfügbarkeit von ACID-Transaktionen

**Zusammenfassung:** *Die Mehrheit der Befragten erwartet nur geringe oder keine Performanceeinbußen aufgrund von Interprozess- und Netzwerkkommunikation. Performanceeinbußen wegen erhöhter Lock Contention werden als potentielles Problem wahrgenommen, jedoch nur in Sonderfällen erwartet.*

Wie bereits zuvor diskutiert, ist serviceübergreifende ACID-Transaktionalität in Microservice-Architekturen üblicherweise nicht verfügbar. Da derartige Transaktionen in betrieblicher Software allgegenwärtig sind, kann dies als Verlust wahrgenommen werden, insbesondere im Kontext von Modernisierungsprojekten. Insgesamt bewerteten 13% der Befragten dies als kritisch, und 45% als bedeutend. Für 32% der Teilnehmer betrachten dies als wenig bedeutend, 10% gar als unerheblich. Wie aus Abbildung 5 ersichtlich, wird diese Einschränkung insbesondere im Bereich Energie und Industrie als kritisch eingestuft.

Im Gegensatz zum serviceübergreifenden Fall ist die Verwendung von Transaktionen innerhalb eines Service nicht beschränkt. 79% der Befragten bewerteten es als sehr wichtig (27%) oder wichtig (52%), die benötigten Transaktionsgrenzen im Servicedesign zu berücksichtigen. Keiner der Befragten betrachtete diesen Aspekt als unwichtig. Wie aus Abbildung 6 ersichtlich, bestehen deutliche Unterschiede zwischen den Branchen, wobei dieser Aspekt im Bereich Handel und E-Commerce auffällig niedrig bewertet wird.

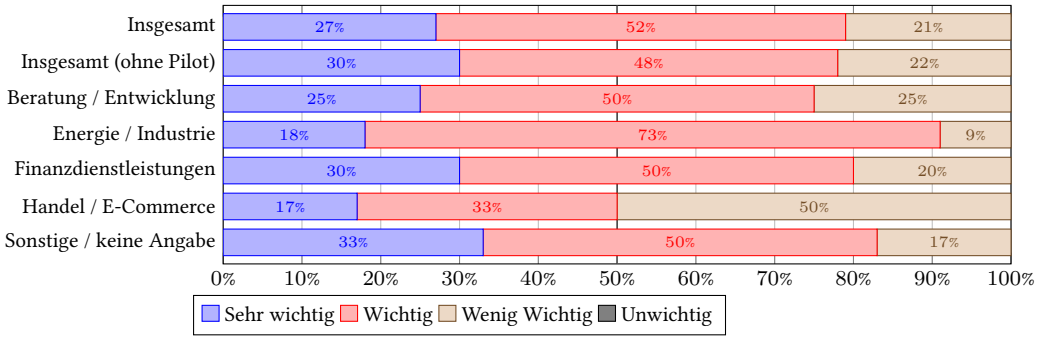


Abbildung 6: Bedeutung von Transaktionsgrenzen für das Servicedesign

**Zusammenfassung:** *Das Fehlen von serviceübergreifenden ACID-Transaktionen wird von der Mehrheit der Befragten als bedeutende Einschränkung eingestuft. Mehr als drei Viertel der Befragten schätzen als wichtig ein, Transaktionsgrenzen bereits im Servicedesign zu berücksichtigen.*

# 5 Diskussion

## 5.1 Validität der Ergebnisse

Um potentielle Gefahren für die Validität der Ergebnisse dieser Umfrage zu betrachten, nutzen wir das in [27] dargestellte Schema. Wir sehen die größte Gefahr für die *Konstruktvalidität* darin, dass die Teilnehmer die Fragen und Antwortmöglichkeiten unterschiedlich verstehen konnten. Insbesondere bei Themen wie Microservices, die derzeit einem gewissen Hype unterliegen, sind oftmals Begrifflichkeiten nicht eindeutig belegt. Dennoch war es aus unserer Sicht angemessen, diese Begrifflichkeiten zu verwenden, da diese den Teilnehmern am ehesten geläufig waren. Dieser Gefahr wurde durch sorgfältige Gestaltung des Fragebogens, internen Reviews und Tests entgegengewirkt.

Die wesentliche Einschränkung bezüglich *externer Validität* ist aus unserer Sicht durch die Struktur der Stichprobe gegeben. Da diese lediglich Personen aus Deutschland umfasste, ist die Übertragbarkeit der Ergebnisse auf andere Länder möglicherweise begrenzt. Zudem war die Stichprobe nicht repräsentativ, da die Teilnehmer mittels *Convenience Sampling* gewonnen wurden. Eine weitere Gefahr resultiert aus der Entscheidung, primär Teilnehmer anzusprechen, die sich bereits mit Microservices auseinandersetzen. Während deren bestehendes Wissen das Risiko von Missverständnissen reduzieren und zu fundierteren Antworten führen kann, ist eine positive Voreingenommenheit gegenüber diesem Architekturstil nicht auszuschließen. Wir haben daher auch Antworten von sieben Teilnehmern aufgenommen, die sich nach eigenen Angaben noch nicht mit Microservices befasst haben. Da Microservices jedoch derzeit ein Hypethema darstellen, ist eine gewisse positive Voreingenommenheit gegenüber diesem Stil zu erwarten.

Eine mögliche Gefahr für *Reliabilität* der Ergebnisse kann aus dem Umstand erwachsen, dass eine nennenswerte Zahl an Teilnehmern durch persönliche Ansprache gewon-

nen wurde. Obwohl wir uns sehr bemüht haben, keine der Fragen im persönlichen Gespräch vorwegzunehmen, kann eine indirekte Beeinflussung nicht völlig ausgeschlossen werden.

## 5.2 Beobachtungen und Feedback zu den Ergebnissen

Wie zuvor berichtet, wurden Teile der Befragten beim Besuch von Veranstaltungen zum Thema Microservices gewonnen. Im Rahmen dieser Veranstaltungen bot sich uns die Gelegenheit, Präsentationen und Diskussionen zu diesem Thema beizuwohnen. Eine bedeutende Beobachtung hierbei war, dass viele Microservice-Architekturen offenbar „unbeabsichtigt“ entstehen. Die betroffenen Unternehmen waren mit einer spezifischen Herausforderung, etwa mangelnder Skalierbarkeit, konfrontiert, entwickelten ihre Architektur entsprechend weiter, und endeten letztlich bei Microservices. Die initialen Herausforderungen, die zu dieser Entwicklung führten, waren hingegen oftmals verschieden, und umfassten Skalierbarkeit, Sicherheitsrichtlinien und die Integration heterogener Technologien.

Zudem wurde uns die Möglichkeit geboten, die Ergebnisse der Umfrage beim Folgetreffen unserer Pilotgruppe zu präsentieren. Obwohl die Teilnehmer den Ergebnissen zustimmten, kam es zu einer lebhaften Diskussion. Aus dieser ergab sich, dass den Teilnehmern mit Hinblick auf Wartbarkeit insbesondere die Fähigkeit zur effizienten Testbarkeit von einzelnen Microservices wichtig ist. Mit Hinblick auf Widerstände und unzureichende Fähigkeiten waren mehrere Teilnehmer besorgt, dass insbesondere ältere Mitarbeiter nicht in der Lage sein könnten, die neuen Paradigmen und Muster zu verinnerlichen, die mit Microservice-Architekturen einhergehen.

## 5.3 Verwandte Arbeiten

Da Microservices erst kürzlich in den Fokus der akademischen Forschung gerückt sind, existieren derzeit nur wenige empirische Arbeiten, die sich mit Microservices befassen. Systematische Literaturrecherchen werden beispielsweise in [24] und [1] durchgeführt. Letztere enthält auch eine Aufstellung bedeutender Herausforderungen in Microservice-Architekturen, und nennt hierbei insbesondere Betrieb, Integration und Performance.

Schermann et al. [28] präsentieren die Ergebnisse einer Umfrage mit 42 Teilnehmern, die primär auf Implementierungsspezifika wie genutzte Protokolle, Datenformate, Monitoring-Daten und bevorzugte Programmiersprachen abzielt. Zudem existieren industrielle Umfragen, die sich zumindest in Teilen mit Microservices befassen. Eine Umfrage von NGINX [23] ergab, dass je etwa ein Drittel der Teilnehmer Microservices produktiv nutzt, erprobt, oder nicht nutzt. Vergleichbare Zahlen werden auch in [19] genannt. Diese Zahlen decken sich bemerkenswert genau mit unseren Ergebnissen zur bestehenden Nutzung von Microservices.

Letztere Studie nennt Deployment-Agilität, Entwicklungsgeschwindigkeit und Elastizität als die wichtigsten Treiber für die Einführung von Microservices. Zudem wird herausgestellt, dass Microservices als Mittel zur Modernisierung insbesondere für große Unternehmen von besonderer Bedeutung sind. Des Weiteren wird die Veränderung der Entwicklungskultur als wesentliches Hindernis bezeichnet; weitere Hindernisse werden nicht benannt. Empirische Arbeiten zur Einführung von DevOps [30; 26] und Erfah-

rungsberichte zur Einführung von Continuous Delivery [21] legen ebenfalls nahe, dass die Veränderung der Kultur eine besondere Herausforderung darstellt. Dies deckt sich mit unseren Ergebnissen, dass Widerstände und mangelnde Fähigkeiten die vorrangigen Hindernisse zur Einführung von Microservices darstellen.

# 6 Fazit

In diesem Bericht wurden wichtige Treiber und Hindernisse für die Einführung von Microservices in der deutschen Softwareindustrie identifiziert, und wesentliche Unterschiede zwischen verschiedenen Branchen benannt. Als primäre Treiber wurden Skalierbarkeit, Wartbarkeit und Verkürzung der Time to Market erkannt, während mangelnde Fähigkeiten von Betrieb und Entwicklung die vorrangige Hindernisse darstellen. Obwohl eine Reihe von konkreten Implementierungshürden als schwierig eingeschätzt wurden, stellte jedoch keine einen Showstopper dar. Wir möchten allerdings hervorheben, dass dies keinesfalls zur Annahme verleiten darf, dass Microservices für jedes Softwaresystem geeignet wären, da eine Reihe von Vor- und Nachteilen zu berücksichtigen sind [13; 29].

Bezüglich der Treiber konnten wir bemerkenswerte Unterschiede zwischen „Early Adopters“, die vorrangig auf die *Skalierbarkeit* ihrer „Internet-Scale“-Systeme abzielen, und „traditionellen“ Unternehmen feststellen, die primär an der *Wartbarkeit* ihrer Systeme interessiert sind. Mit Hinblick auf die Modernisierung bestehender Softwaresysteme scheint Wartbarkeit der wichtigste Treiber zu sein. Performanceeinbußen aufgrund von Remote-Aufrufen werden nicht in wesentlichem Maße befürchtet. Die Nichtverfügbarkeit serviceübergreifender Transaktionen wird hingegen deutlich skeptisch gesehen.

Aufbauend auf den Ergebnissen dieser Studie wären vergleichbare Studien in anderen Ländern und ein Vergleich mit der Situation in Deutschland wertvoll. Um derartige Studien zu erleichtern, sind die Materialien dieser Studie unter [16] veröffentlicht. Auch eine qualitative Vertiefung der Ergebnisse wäre von großem Wert, um ein besseres Verständnis insbesondere der Hindernisse zu entwickeln und mögliche Gegenmaßnahmen entwerfen zu können.

# Literatur


[1] Alshuqayran, N. ; Ali, N. ; Evans, R. : A Systematic Mapping Study in Microservice Architecture. In: *Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA 2016)*, 2016

[2] Balalaie, A. ; Heydarnoori, A. ; Jamshidi, P. : Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In: *IEEE Software* 33 (2016), Nr. 3, S. 42–52

[3] Bass, L. ; Weber, I. ; Zhu, L. : *DevOps: A Software Architect's Perspective.* New York : Addison-Wesley, 2015

[4] Brunnert, A. ; van Hoorn, A. ; Willnecker, F. ; Danciu, A. ; Hasselbring, W. ; Heger, C. ; Herbst, N. ; Jamshidi, P. ; Jung, R. ; Kistowski, J. von ; Koziolek, A.

; Kross, J. ; Spinner, S. ; Vögele, C. ; Walter, J. ; Wert, A. : Performance-oriented DevOps: A Research Agenda. 2015. – Forschungsbericht

[5] Esposito, C. ; Castiglione, A. ; Choo, K.-K. R.: Challenges in Delivering Software in the Cloud as Microservices. In: *IEEE Cloud Computing* 3 (2016), Nr. 5, S. 10–14

[6] Evans, E. : *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Upper Saddle River, NJ : Addison-Wesley, 2007

[7] Fowler, M. : *Microservice Prerequisites*. 2014. – http://martinfowler.com/bliki/MicroservicePrerequisites.html

[8] Fowler, M. : *Microservices*. 2014. – Vortrag bei der GOTO 2014, https://www.youtube.com/watch?v=wgdBVIX9ifA

[9] Hasselbring, W. : Microservices for Scalability: Keynote Talk Abstract. In: *Proceedings of the International Conference on Performance Engineering (ICPE 2016)*, 2016

[10] Hasselbring, W. ; Steinacker, G. : Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. In: *Proceedings of the International Conference on Software Architecture (ICSA 2017) Workshops*, 2017

[11] Humble, J. ; Farley, D. : *Continuous Delivery*. Upper Saddle River, NJ : Addison-Wesley, 2011

[12] Hüttermann, M. : *DevOps for Developers*. New York, NY : Apress, 2012

[13] Killalea, T. : The Hidden Dividends of Microservices. In: *Communications of the ACM* 59 (2016), Nr. 8, S. 42–45

[14] Knapp, T. R.: Treating Ordinal Scales as Interval Scales: An Attempt To Resolve the Controversy. In: *Nursing Research* 39 (1990), Nr. 2, S. 121–123

[15] Knoche, H. : Combining Application-Level and Database-Level Monitoring to Analyze the Performance Impact of Database Lock Contention. In: *Proceedings of the Symposium on Software Performance (SSP 2016)*, 2016

[16] Knoche, H. ; Hasselbring, W. : *Data for: Drivers and Barriers for Microservice Adoption in the German Software Industry*. 2017. – http://dx.doi.org/10.5281/zenodo.820146

[17] Leppänen, M. ; Mäkinen, S. ; Pagels, M. ; Eloranta, V. P. ; Itkonen, J. ; Mäntylä, M. V. ; Männistö, T. : The highways and country roads to continuous deployment. In: *IEEE Software* 32 (2015), Nr. 2, S. 64–72

[18] Lewis, J. ; Fowler, M. : *Microservices*. 2014. – http://martinfowler.com/articles/microservices.html

[19] Lightbend, Inc.: *Enterprise Development Trends 2016.* 2016. – https://info.lightbend.com/COLL-20XX-Enterprise-Development-Trends-2016-Report_RES-LP.html

[20] Meshenberg, R. : *Microservices at Netflix Scale: Principles, Tradeoffs & Lessons Learned.* 2016. – Vortrag bei der GOTO 2016, https://www.youtube.com/watch?v=57UK46qfBLY

[21] Neely, S. ; Stolt, S. : Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In: *2013 Agile Conference*, 2013

[22] Newman, S. : *Building Microservices.* Sebastopol, CA : O'Reilly, 2015

[23] NGINX: *The Future of Application Development and Delivery Is Now.* 2016. – https://www.nginx.com/resources/library/app-dev-survey/

[24] Pahl, C. ; Jamshidi, P. : Microservices: A Systematic Mapping Study. In: *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2016)*, 2016

[25] Pardon, G. ; Pautasso, C. : Atomic Distributed Transactions: A RESTful Design. In: *Proceedings of the International Conference on World Wide Web*, 2014

[26] Riungu-Kalliosaari, L. ; Mäkinen, S. ; Lwakatare, L. E. ; Tiihonen, J. ; Männistö, T. : DevOps Adoption Benefits and Challenges in Practice: A Case Study. In: *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2016).* Springer, 2016, S. 590–597

[27] Runeson, P. ; Höst, M. : Guidelines for conducting and reporting case study research in software engineering. In: *Empirical Software Engineering* 14 (2008), Nr. 2, S. 131

[28] Schermann, G. ; Cito, J. ; Leitner, P. : All the Services Large and Micro: Revisiting Industrial Practice in Services Computing. In: *Proceedings of the International Conference on Service-Oriented Computing (ICSOC 2015) Workshops.* Berlin : Springer, 2016, S. 36–47

[29] Singleton, A. : The Economics of Microservices. In: *IEEE Cloud Computing* 3 (2016), Nr. 5, S. 16–20

[30] Smeds, J. ; Nybom, K. ; Porres, I. : DevOps: A Definition and Perceived Adoption Impediments. In: *Proceedings of the International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2015).* Springer, 2015, S. 166–177

[31] Stine, M. : *Migrating to Cloud-Native Application Architectures.* Beijing : O'Reilly, 2015

[32] T. Nygard Michael: *Release It! – Design and Deploy Production-Ready Software.* Raleigh, NC : The Pragmatic Bookshelf, 2007

[33] Thabane, L. ; Ma, J. ; Chu, R. ; Cheng, J. ; Ismaila, A. ; Rios, L. P. ; Robson, R. ; Thabane, M. ; Giangregorio, L. ; Goldsmith, C. : A tutorial on pilot studies: the what, why and how. In: *BMC Medical Research Methodology* 10 (2010), Nr. 1

[34] Waller, J. ; Ehmke, N. C. ; Hasselbring, W. : Including Performance Benchmarks into Continuous Integration to Enable DevOps. In: *SIGSOFT Software Engineering Notes* 40 (2015), Nr. 2, S. 1–4

[35] Wolff, E. : *Microservices – Flexible Software Architectures*. Leanpub, 2016