Continuous Delivery

Übernehmen einer Softwareentwicklungsdisziplin

Björn Beha und Suhay Sevinc

Zusammenfassung—Continuous Integration kommt heutzutage in den meisten Unternehmen zum Einsatz. Nach dem Commit des geänderten Codes, wird dieser automatisch gebuildet und getestet, wodurch der Entwickler sehr schnell Feedback zu Fehlern erhält. Dadurch hat man bereits einen stabilen Prozess, um nachhaltig qualitativ hochwertige Software in Produktion zu bringen. An dieser Stelle setzt nun Continuous Delivery an. Die Prinzipien, die bei Continuous Delivery verfolgt werden klingen im ersten Moment vielversprechend, denn es ermöglicht das Ausrollen von Software mit einer wesentlich höheren Geschwindigkeit sowie Zuverlässigkeit. Grundlage dafür ist die Continuous-Delivery-Pipeline, die das in Produktion bringen der Software weitgehend automatisiert und so einen reproduzierbaren, risikoarmen Prozess für die Bereitstellung neuer Releases darstellt [1].
Obwohl Continuous Delivery lediglich eine Sammlung von Techniken, Prozessen und Werkzeugen ist, kann sich das Aufsetzen dieser Disziplin sehr schwierig gestalten. In diesem wissenschaftlichen Artikel wird die Frage behandelt, was ein Unternehmen aufwenden muss, um diesen Schritt zu gehen und welche Auswirkungen diese Entscheidung auf das Unternehmen hat. Dabei wird gezeigt, welche Rolle DevOps bei diesem Schritt spielt, welche Komponenten ein Unternehmen dabei betrachten muss und welche Barrieren es gibt. Abschließend wird gezeigt, welche Änderungen in der Unternehmenskultur man dabei zu erwarten hat.

Index Terms—Computer Society, IEEE, IEEEtran, journal, LaTeX, paper, template.

1 Introduction

Continuous Integration wird bereits seit langem von vielen Unternehmen genutzt [8]. Dadurch haben Unternehmen bereits einen stabilen Prozess, um nachhaltig qualitativ hochwertige Software in Produktion zu bringen. An dieser Stelle setzt nun Continuous Delivery an. Die Prinzipien, die bei Continuous Delivery verfolgt werden klingen im ersten Moment vielversprechend, denn es ermöglicht das Ausrollen von Software mit einer wesentlich höheren Geschwindigkeit sowie Zuverlässigkeit. Auch wiederkehrende Herausforderungen zur Vorbeugung von Seiteneffekten und Regressionen lassen sich mit diesem Ansatz behandeln. Obwohl Continuous Delivery lediglich eine Sammlung von Techniken, Prozessen und Werkzeugen ist, stellt das Aufsetzen dieser Disziplin vor allem in Unternehmen mit unflexiblen Strukturen eine (zunehmende) Herausforderung dar [1].

> *„Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations. “* [1, S.5]

Laut dem Gesetz von Conway bedeutet Contiunous Delivery einen Bruch mit der bisherigen Organisation im Unternehmen. In dieser Arbeit wird ein möglicher Pfad beleuchtet, welchen ein Unternehmen gehen muss, um eine vollständige Continuous-Delivery-Pipeline aufzusetzen. Dabei gibt es unterschiedliche Voraussetzungen für Unternehmen. Beispielsweise ist es vom aktuellen Zustand der Organisation bzw. wann sie sich für diesen Schritt entscheidet abhängig. Ist das Projekt bereits fortgeschritten und muss die Bestehende Pipeline entsprechend erweitert werden oder kann das Projekt direkt von Anfang an darauf ausgelegt werden? Zu Beginn der Arbeit erfolgt somit eine kurze Erläuterung von DevOps, die Prämisse für die unbeschränkte Umsetzung von Continuous Delivery. Darauf aufbauend wird beschrieben, welche Teile einer Organisation bei der Entwicklung zur DevOps-Kultur betroffen sind und welche Barrieren es hier zu bewältigen gibt. Abschließend wird in der Diskussion erläutert, welche Änderungen ein Unternehmen zu erwarten hat, wenn es alle Vorteile von Continuious Delivery nutzen möchte.

2 Abgrenzung zu anderen Arbeiten

Diese Arbeit stützt sich auf die Inhalte der Bücher ”Continuous Delivery” [2] und ”Product-Focused Software Process” [3]. Diese beinhalten sowohl die benötigten Grundlagen als auch tiefer gehende Aspekte. Die hier vorgestellten Vorgehensweisen werden verwendet, um Umsetzungsstrategien für Unternehmen herauszuarbeiten.
Bei der Umsetzung dieser Strategien entstehen allerdings Probleme, die dazu führen, dass die Umsetzung von Continuous Delivery erschweren können. Hierfür ist eine Problemanalyse erforderlich. Für diese Analyse stützt sich diese Arbeit auf die Paper von Laukkanen et al [4] und Shahin et al [5]. Die beiden Paper betrachten eine Menge von Case-Studies und führen Literaturreviews aus. Diese Arbeiten haben alle eine Gemeinsamkeit: Die Frage nach der Adaption von Continuous Delivery wird recht im Allgemeinen beantwortet und ist somit weiter Weg für den Einsatz in der Praxis. Auswirkungen in Form von wie zum Beispiel Mitarbeiterschulungen, Kostenersparnis oder Prozessanpassungen wurden bisher nicht beleuchtet, welche allerdings eine wichtige Rolle einnehmen. Diese Fragen soll die vorliegende Arbeit von den anderen Arbeiten abgrenzen.

3 DevOps

DevOps ist eine Philosophie oder Organisationsform, mit der Continuous Delivery optimal funktioniert. Es stellt dabei eine Kollaboration zwischen Entwicklung (Developement) und Betrieb (Operations) in den Mittelpunkt, wodurch bereits das Wesentliche des Gedankens hervorgeht: Ein Zusammenwachsen der beiden Abteilungen zu einem Verband (siehe Abbildung 1). In klassischen Organisationsformen sind diese in mindestens zwei unterschiedliche Abteilungen getrennt und verfolgen verschiedene Ziele. Während bei dem Betrieb Kosteneffizienz in den Vordergrund rückt, stellt die Entwicklung neue Features bereit und wird an der Geschwindigkeit und Effizienz ihres Vorgehens gemessen. Bei der Einführung von Continuous Delivery sollten Betrieb und Entwicklung daher zusammenarbeiten, um das Notwendige Wissen und entsprechende Werkzeuge zu teilen. Jede Gruppe beherrscht durch die unterschiedliche Perspektive auf die Anwendungen einen Teil von Continuous Delivery besonders gut. Der Betrieb kennt beispielsweise Aspekte wie Monitoring, Security oder Netzinfrastrukturen. Ohne diese kann eine Anwendung kaum sinnvoll installiert oder betrieben werden kann. Die Entwicklung hingegen kennt den Code, die Entwicklungsinfrastruktur und Middleware wie Application Server etc. sehr genau. Gemeinsam lässt sich somit eine Continuous-Delivery-Pipeline aufbauen. Mit der Einführung von Continuous Delivery sollte allerdings auch die Einführung von DevOps in Betracht gezogen werden [1].

Bereits 2006 wurde bekannt, dass eines der großen IT-Unternehmen einen anderen Ansatz zur Erstellung seiner Dienste verfolgt. Die Trennung von Betrieb und Entwicklung wurde bis dato von allen IT-Organisationen umgesetzt. Zu diesem Zeitpunkt gab es bei Amazon bereits Teams, die sowohl die Entwicklung als auch den Betrieb vereinen. Jedes Team ist somit für eine spezifische Fachlichkeit zuständig und kann diese unabhängig von anderen Teams als Service implementieren. Der Dienst kann schließlich selbständig optimiert werden, um einen guten Betrieb zu ermöglichen. Des Weiteren kann ohne weitere Abstimmung eine Weiterentwicklung am Dienst vorgenommen werden. Auch der eingesetzte Technologie-Stack ist abhängig von der Entscheidung des Teams. Diese übernehmen die vollständige Verantwortung für ihre Komponenten. Daraus folgt eine geringere Koordination untereinander, wodurch wiederum Software schneller entwickelt und in Produktion gebracht werden kann. Auch Kunden wissen direkt, welcher Ansprechpartner für welchen Service zuständig ist (siehe Abbildung 1) [1].

Continuous Delivery wird häufig auch mit DevOps gleichgesetzt. Das trifft jedoch nicht zu. Durch DevOps lässt sich Continuous Delivery zwar drastisch vereinfachen, allerdings gibt es neben dieser (wesentlichen) Praktik im DevOps-Umfeld noch weitere Bereiche (Monitoring, Troubleshooting etc.), bei denen diese Organisationsform und eine damit einhergehende Zusammenarbeit sinnvoll sein kann [1].

4 Entwicklung zur DevOps-Kultur

Es ist denkbar, einen gewissen Teil von Continuous Delivery auch ohne DevOps einzuführen, denn hierfür muss es nicht zwingendermaßen gemischte Teams geben, sondern lediglich eine Zusammenarbeit an der Continuous-Delivery-Pipeline. Continuous Delivery verfolgt unterschiedliche Ziele von denen einige trotz einer reduzierten Pipeline erreicht werden können, ohne, dass eine völlige Umorganisation notwendig ist. Letztendlich bringen diese Ansätze allerdings weniger Vorteile mit sich. Wenn Continuous Delivery vollständig umgesetzt werden soll, gelingt dies nur mit der Einführung von DevOps. DevOps einzuführen, bedeutet jedoch eine fundamentale Änderung der Organisation. Eine solche Änderung ist mit einem erheblichen Aufwand verbunden und kann vor allem in großen Konzernen schwer umsetzbar sein. In diesem Kapitel wird darauf eingegangen, welche Aspekte dabei zu berücksichtigen sind und welche Folgen diese Transformation für Unternehmen hat [1].

4.1 Umsetzung von Continuous Delivery

Continuous Delivery ist vor allem eine technische Praktik. Trotzdem hat diese Disziplin verschiedene Auswirkungen auf die Organisation. Continuous Delivery steht und fällt mit der Pipeline. Das Umsetzen einer einfachen Delivery-Pipeline kann bereits mit simplen mitteln erfolgen. Vor allem zu Beginn eines Projekts kann der Aufbau einer solchen Pipeline verfolgt werden. Beispielsweise kann ein Unternehmen hier vorab mit den Phasen Commit und Deploy beginnen, da diese schließlich existieren müssen. Es ist dann möglich, die Pipeline im weiteren Verlauf schrittweise weiterzuentwickeln, indem entsprechende Phasen (z. B. automatisierte Tests) hinzugefügt werden. Wird Continuous Delivery zum Start eines Projekts eingeführt, kann dies auch bei der Auswahl der Technologien und Architekturen berücksichtigt werden [1].

Oft setzt Continuous Delivery allerdings an eine bestehende Codebasis an, die nicht dafür konzipiert wurde. In gewisser Weise besteht immer eine Delivery-Pipeline, da es stets einen Prozess zur Auslieferung der Software gibt. Nur kann dieser Prozess sehr komplex sein. Das Ziel von Continuous Delivery ist allerdings einen konstanten Fluss von Features und Codeänderungen durch die Pipeline zu erreichen. Um die bestehende Pipeline entsprechend anzupassen können verschiedene Ansätze wie beispielsweise Value Stream Mapping angewandt werden. Aus dieser Analysetechnik lassen sich dann Optimierungen ableiten. Zu weiteren Optimierungsmaßnahmen gehört auch DevOps [1]. Value Stream Mapping hingegen wird in dieser Arbeit nicht weiter beleuchtet.

Beim Übernehmen von Continupus Delivery reicht es allerdings oft nicht, schlicht entsprechende Werkzeuge zu verwenden. Solange die Prozesse innerhalb der Organisation nicht ablaufen, wie bei einer gut geölten Maschine, kann hier keine signifikante Steigerung

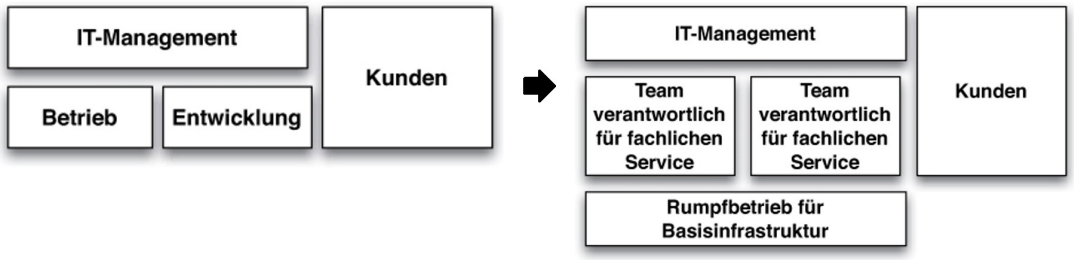

Abbildung 1. Wandel von klassischer Organisation zu DevOps-Teams

der Effizienz erreicht werden [5]. Der Schlüssel, um entsprechende Prozesse gut zum Laufen zu bringen ist dabei DevOps. Mit dieser Organisationsform funktioniert Continuous Delivery besonders gut, da hierfür das Wissen aus beiden Abteilungen benötigt wird. Während die Entwicklung die Anwendung, deren Konfiguration und Aufbau kennt, weiß der Betrieb über die Rahmenbedingungen bescheid [1].

4.2 Umsetzung von DevOps

Continuous Delivery ist hauptsächlich mit der DevOps-Bewegung verbunden [3], da DevOps eine wirkungsvolle Optimierungsmaßnahme darstellt. Dabei fokussiert sich dieser Ansatz auf die Liefergeschwindigkeit, das kontinuierliche Testen in Produktionsähnlichen Umgebungen, kontinuierliche Rückmeldung und schnelle Reaktionsfähigkeit sowie, wie der Name bereits andeutet, das Auflösen der Teamgrenzen [5]. Immerhin sollen die Entwicklung und der Betrieb als eine Einheit agieren und Teams anhand von Komponente oder fachlichen Zuständigkeiten aufgeteilt werden, wodurch auch die vollständige Verantwortung für ein Modul bei dem entsprechenden Team liegt. Damit geht eine Änderung der Organisationsform einher, um eine Kultur und ein Umfeld zu schaffen, indem das Erstellen, Testen und Freigeben von Software schnell, häufig und zuverlässig erfolgen kann. Ein Großteil von DevOps befasst sich demnach mit der Prozessverbesserung durch eine radikale Automatisierung manueller Prozesse [7].

Viele Unternehmen sind bereits dabei, DevOps erfolgreich umzusetzen. Allerdings bleiben nach wie vor kritische Barrieren bei der Entwicklung, wie sie im folgenden Kapitel näher beschrieben werden. Um diese Philosophie anzuwenden gibt es verschiedene Möglichkeiten. Dabei ist vermutlich die größte Hürde die Angst vor Veränderungen (siehe vorherige Abneigung gegenüber Clouds) [3].

Es muss einem klar sein, was DevOps für das Unternehmen bedeutet. Zwei der größten Hindernisse, diese Disziplin erfolgreich umzusetzen, sind die Unternehmensarchitektur sowie Unternehmenskultur, die davon beeinflusst werden.

4.3 Architektur

Für die Architektur bedeutet DevOps eine Herausforderung. Da eine zentrale Instanz, die die Architektur in eine bestimmte Richtung zwingt einen Widerspruch zu den selbständigen Teams darstellt, müssen an dieser Stelle Abstimmungen erfolgen. Die Teams entwickeln so die Architektur des Systems untereinander. Die Erfahrung hat gezeigt, dass entsprechend moderierte Meetings stattfinden, in welchen die Teams ihre Abstimmung vornehmen können [1].

Continuious Delivery und DevOps sind Ansätze, mit denen Software effizienter und effektiver betrieben und in Produktion gebracht werden kann. Sie haben vor allem Auswirkungen auf Deployment- und Betriebsprozesse. Allerdings beeinflusst Continuous Delivery auch die Architektur der Lösungen. Wenn bei einem System an einer Stelle etwas geändert wird, muss die ganze Pipeline durchlaufen werden. Es werden dabei nicht nur die geänderten Komponenten getestet, sondern alle. Das führt zu einer Menge unnötiger Arbeit und verzögert das Feedback. Bis zu einem gewissen Maße ist dies sinnvoll, da beim Durchlaufen der Pipeline alle Bestandteile getestet werden sollen. Immerhin kann sich ein Fehler in einer Komponente erst dadurch manifestieren, wenn eine abhängige Komponente ein falsches Verhalten aufweist [1].

Durch Continuous Delivery ergibt sich eine andere technische Umsetzung der Komponenten, denn jede Komponente kann als eine eigene Deployment-Einheit implementiert werden, welche dann jeweils eine eigene Pipeline durchlaufen. Es wird demnach auch nicht mehr das gesamte System deployt, sondern nur ein Teil. Das schnelle Feedback und die in angestrebte Risikominimierung wird hierdurch erreicht. Es werden nun also zwei Merkmalen der Unternehmensarchitektur eine hohe Priorisierung zuteil: Testbarkeit und Einsatzfähigkeit. Das Design für die Implementierung und die entsprechenden Tests beginnt zudem mit der Sicherstellung, dass die Services aus lose gekoppelten, gut gekapselten Komponenten oder Modulen bestehen (im Rahmen der objektorientierten Programmierung folgen solche Systeme den Prinzipien des SOLID-Designs). Durch verschiedene Anpassungen der Architektur und den bisher erwähnten Anforderungen an die Organisation ergibt sich ein umfangreiches Zusammenwirken von Continuous Delivery und dem Softwarearchitekturansatz von Microservices. Microservices entsprechen sehr gut den Anforderungen an einer Architektur für ein Continuous Delivery System und setzen auch andere Aspekte (Technologiefreiheit, unabhängige Einheiten etc.) um [1]. Die Microservices-Bewegung

hat diese architektonischen Eigenschaften sogar explizit priorisiert. Müssen beispielsweise viele Microservices gleichzeitig freigegeben werden („Big Bang-Event"), ist es erforderlich, dass viele Teams in einer sorgfältigen Art und Weise zusammenarbeiten. Solche Bereitstellungen können dabei typischerweise viele Stunden oder sogar Tage in Anspruch nehmen und erfordern eine erhebliche Planung. Auch hier ist die Verbindung zu DevOps wieder erkennbar. Ein Unternehmen muss kontinuierlich und vor allem agil planen. Das setzt voraus, dass es in der Lage ist, sich schnell an die sich ändernden Marktbedingungen anzupassen [3]. Bei der Einführung von Continuous Delivery muss also auch die Architektur der Software angepasst werden. Lediglich die Prozesse anzupassen bzw. zu ändern, reicht nicht aus [1].

4.4 Kultur

Auch die Unternehmenskultur ist von der Transformation betroffen. Obwohl diese nicht greifbar und nur schwer zu ändern ist, ist es wichtig, eine Kultur zu schaffen, in welcher alle Menschen der Organisation bei der Verfolgung gemeinsamer Ziele zusammenarbeiten. Dies ging bereits durch den Soziologen Ron Westrum hervor (...), der die Wichtigkeit einer solchen Kultur betonte. Bei DevOps wird immer die primäre Bedeutung der Kultur hervorgehoben, mit besonderem Fokus auf einer effektiven Zusammenarbeit zwischen Entwicklungsteams und IT-Betriebsteams. Tatsächlich warten die leistungsstärksten Unternehmen nicht darauf, dass schlimme Dinge passieren, um daraus zu lernen und sich zu verbessern. Sie schaffen regelmäßig (kontrollierte) Unfälle, um schneller Optimierungen abzuleiten und schneller als die Konkurrenz zu lernen [3]. Leistungsstarke Organisationen versuchen also stets besser zu werden. Durch die Eigenverantwortung und Selbstorganisation, um die es bei DevOps im Grunde geht, übernehmen die Teams die vollständige Verantwortung für ihre Komponenten. Dies Betrifft sowohl den Betrieb als auch die Entwicklung. Somit können viele Entscheidungen direkt in den Teams gefällt werden. Eine dieser Entscheidungen betrifft den zu verwendenden Technologie-Stack des Teams.

4.5 Technolgie

Die Technologiefreiheit bedeutet, dass die vollständige Zuständigkeit der Auswahl von Implementierungstechnologien wie Frameworks, Programmiersprachen oder Application Server bei den Teams liegt. Treten mit den ausgewählten Technologien Probleme auf, muss sich auch nur dieses Team mit der Behebung möglicher Fehler auseinandersetzen. Andere Teams bleiben davon unbeeinflusst. So können sehr viele unterschiedliche Technologien in einer Organisation genutzt werden. Eine klassische Organisation versucht, die Anzahl von Technologien möglichst einzuschränken und zu kontrollieren, um so Risiken zu minimieren und ein Zusammenspiel zu realisieren. Wenn alle Projekte dieselben Programmiersprachen und Infrastrukturelemente nutzen, kennt jeder Entwickler die notwendigen Technologien und kann jedes Projekt bis zu einem gewissen Maße unterstützen. Der Betrieb kann sich somit auch auf diese Technologien fokussieren und ist mit Problemen dieses Stacks besonders gut vertraut. Bei DevOps stehen Synergien nicht so sehr im Vordergrund. Der Fokus liegt auf der Freiheit der Teams, eigene Entscheidungen zu treffen. Dadurch kann jedes Team den Technologie-Stack wählen, der für die jeweiligen Herausforderungen am besten geeignet ist. Auf diese Weise kann jedes Team mit optimaler Produktivität arbeiten. Einen Standard-Technologie-Stack kann etabliert werden, indem auf Erfahrungen anderer Teams zurückgegriffen wird. Somit könnte es bereits einen Werkzeugkasten geben, welcher auch für andere Teams attraktiv ist [1].

5 Fehler bei der Transformation

In Kapitel 4 wurde gezeigt wie komplex die Etablierung von Continuous Delivery für Unternehmen ist. Bei diesen Prozess können Probleme auftreten, die in diesem Abschnitt diskutiert werden. Es tritt häufig der Fehler auf, dass Unternehmen dazu neigen, Continuous Delivery als einen Endzustand anzusehen und diese erwähnte Mentalität nicht weiter entwickeln. Allgemein lassen sich die Menge der Probleme in sechs Kategorien klassifizieren: Builddesign, Systemmodularisierung, Integration, Testphase, Release, Mensch und Organisationsstruktur werden. In der Arbeit von Laukkanen et al treten hier 40 Probleme auf (siehe Abbildung 2) [4].

5.1 Builddesign

Bereits die Entscheidung, wie das Builddesign aufgebaut und konfiguriert wird, kann eine zukünftige Problemursache darstellen. Ein Grund hierfür kann die Erstellung von komplexen und unflexiblen Buildskripten sein. Das Builddesign ist dadurch stark an das Zielsystem gekoppelt, sodass minimale Skritpänderungen zu Buildfehlern führen. Dies erfordert intensive Pflege- und Wartungsarbeiten, welche die Einführung einer DevOps-Kultur drosseln. Aber auch stellt die Modellierung des Systems eine Abhängigkeit für das Builddesign dar, da die Auflösung von Abhängigkeiten kritisch sein könnte, wie z. B. zyklische Abhängigkeiten.

5.2 Systemmodularisierung

Wie in Abschnitt 5.1 aufgezeigt wurde stellt die Systemmodularisierung eine weitere Problemursache dar. Eine geeignete Systemmodellierung führt zu einer autonomen und unabhängigen Entwicklung. Von einer ungeeigneten Architektur ist genau dann die Rede, wenn sie monolithisch gekoppelt sei. Eine ungeeignete Systemmodularisierung führt zu einem überhöhten Entwicklungsaufwand, Testbarkeit und Wartbarkeit. Dadurch erhöht sich sowohl die Fehleranfälligkeit des Systems als auch die Code-Inkonsistenz. Draus folgt, dass sich die Software in einem nicht aus-lieferbaren Zustand befindet. Jedoch kann auch bei einer geeigneten Architektur Probleme auftreten, indem zu starke Systemmodellierung betrieben wird. Das führt zu einer Erhöhung der Komplexität, sodass neue Teammitglieder die Software schwer weiterentwickeln und warten können [4].

5.3 Integration

Die Integration umfasst Probleme, die bei der Zusammenführung von Softwarekomponenten entstehen [6]. Sobald diese nicht wie vorgesehen zusammengeführt werden,

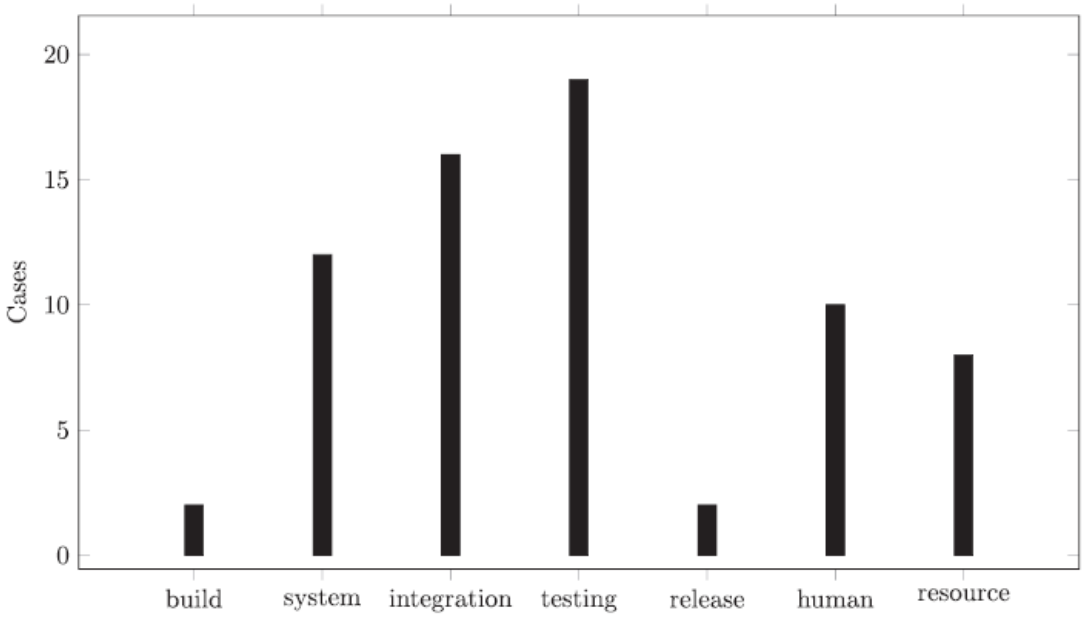

Abbildung 2. Visualisierung der Probleme der einzelnen Bereiche [4].

fangen die Probleme an. Nicht selten kommt es vor, dass Entwickler einmal täglich Quellcodeänderungen hochladen. Das hat die Folge, dass eine hohe Anzahl von Änderungen mit sich bringt und diese im Konflikt mit anderen Änderungen stehen. Im Fall eines Fehlers im Buildprozess kann der Fehler nicht sofort identifiziert werden und es entsteht eine Arbeitsblockade, siehe Abbildung 3. Des Weiteren hat diese Vorgehensweise den Nachteil, dass überhöhter Netzwerklatenz erzeugt wird. Eine oft genutzte Methode ist hier das Erstellen von mehreren Abzweigungen des Hauptentwicklungsstrangs. Dies ist allerdings keine optimale Lösung. Während die Entwicklung am Hauptentwicklungsstrang voranschreitet, divergieren Verzweigungen immer weiter vom ursprünglichen Strang, sodass die Vereinigung dieser beiden Stränge ein Problem wird. Die Lösung dieser Vereinigungsproblemen resultiert in mehr Aufwand, was zur einer reduzierten Produktivität führt [4].

5.4 Testphase

Tests sollten eindeutige Validierung vornehmen. Bei bestimmten Tests, wie beispielsweise ”flaky tests”, entstehen hier langwierige Probleme. Bei diesen Tests werden zufällige Resultate erzeugt, die nicht determinierbar sind und erschweren somit die Testphase, sodass Entwickler verwirrt werden, anstatt Auskunft zu erhaltenk, welches aus dem folgenden Zitat hervorgeht:

> „...several of the automated activities do not yield a clear “pass or fail”result. Instead, they generate logs, which are then inspected in order to determine whether there were any problems—something only a small minority of project members actually do, or are even capable of doing. “ [4, S.65]

5.5 Mensch und Organisationsstruktur

Eine letzte Problemursache ist der Mensch, welches aktuell die größte Herausforderung darstellt. Die Adaption von Continuous Delivery erwartet wie erwähnt Agilität und Flexibilität. Das setzt voraus, dass eine Akzeptanz für solche Vorgehensweisen vorhanden ist, welches oft die erste Hürde darstellt wie im folgenden Zitat zu sehen ist. Sowohl auf Management- als auch Mitarbeiterebene wird Motivation und Disziplin erfordert. Des Weiteren muss erwähnt werden, dass in Softwareteams mehr Druck herrscht, da die Software sich in einem ständig aus-lieferbaren Zustand befinden muss. Das kann negative Auswirkungen auf die Motivation der Teammitglieder haben und führt gegebenfalls zu einer angespannten Arbeitsatmosphäre. Aus technischer Sicht verlangt Continuous Integration/Delivery tiefes technisches Wissen im Bereich Build-Management und sowohl die Programmierung und Erstellung von unterschiedlichen Skripten [4].

> „...But it was hard to convince them that we needed to go through our implementation “hump of pain”to get the pieces in place that would allow us to have continuous integration. I worked on a small team and we didn’t seem to have any “extra”time for me to work on the infrastructure we needed. “ [7, S.373]

6 Auswirkungen auf das Unternehmen

Dadurch, dass eine DevOps-Kultur anders ist, ergibt dies eine andere Mentalität. DevOps ermöglicht es aussagekräftige ROI-Metriken zu liefern. Ob die Veränderung Auswirkungen hat, sagen einem die Zahlen. Wichtig ist es somit, entsprechend zu messen. Unternehmen müssen wissen, wie effizient oder ineffizient bestehende Softwareentwicklungsprozesse sind, bevor sie sich zu verändern beginnen. Verstehen Sie auch, welche Ergebnisse Sie erwarten und planen Sie entsprechend. Basierend auf unserer Erfahrung sollte die Bereitstellung acht- bis zehnmal schneller sein, mit einer Qualitätssteigerung von 50 Prozent (Messung der Reduzierung von Vorfällen und Vorfalldauer), während Software-Builds drei- bis viermal schneller sein sollten. Wenn Sie

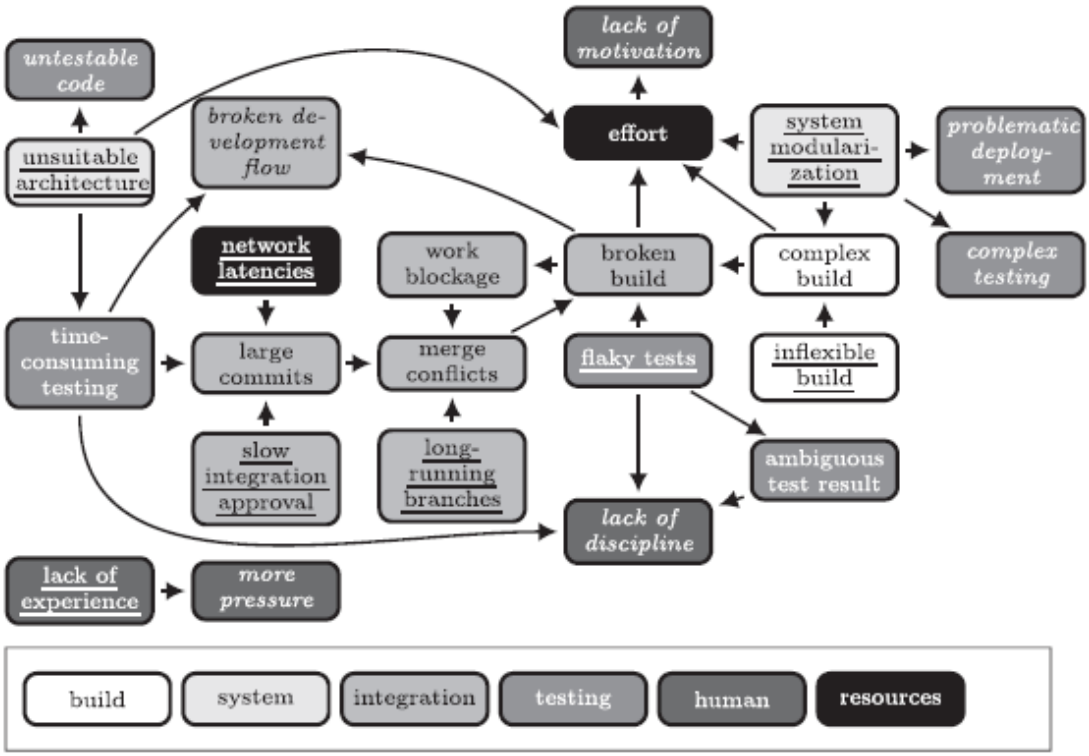

Abbildung 3. All [4].

die perfekte DevOps-Kultur richtig einrichten, können die Ergebnisse erstaunlich sein. Wir haben festgestellt, dass ein Unternehmen eine DevOps-Struktur einführt und die Veröffentlichungsfenster durch Wiederholbarkeit und Skalierbarkeit von zwei Tagen auf zwei Stunden reduziert. Das Ziel: Barrieren abbauen Obwohl DevOps zunehmend an Bedeutung gewinnen, bleiben Fragen von denen, die über ihre Effektivität skeptisch sind. Es erinnert mich an eine Zeile, die ich oft auf Konferenzen und Shows höre, die immer wieder zum Lachen kommt: Ïrren ist menschlich. In 10 Minuten auf 10.000 Servern zu irren ist DevOps. "Die Wahrheit ist jedoch, dass DevOps ein sehr einfaches Konzept für Zusammenarbeit und intelligenteres Arbeiten ist. Es ist die Realisierung eines effizienteren, produktiveren Organisationsmodells. Es ist an der Zeit für alle Unternehmen, Barrieren abzubauen, eine moderne Kultur zu schaffen und bessere Software in kürzeren Zyklen zu liefern [8].

7 Diskussion

Continuous Delivery ist ohne Frage ein lohnenswerter Schritt, den ein Unternehmen gehen kann, um seine Softwarequalität weiter zu verbessern und das Risiko beim Release drastisch zu minimieren. Wie bereits erwähnt, kann Continuous Delivery bzw. ein Teil davon auch ohne DevOps umgesetzt werden, allerdings gehen wir in diesem Artikel davon aus, dass Continuous Delivery vollständig eingeführt werden soll um alle Vorteile dieser Disziplin zu erhalten. Dabei reicht es nicht einfach nur .... Es gilt diverse Hürden zu überwinden, wie aus den obigen Kapiteln bereits hervorgegangen ist. Um eine geeignete DevOps-Kultur in das Unternehmen einzuführen bedeutet zunächst auch einen enormen initialen Aufwand für das Unternehmen. Auch im laufenden System fallen nun regelmäßig Weiterentwicklungen und womöglich Wartungen an.

Die Erstellung von Tests muss zudem ein integraler Bestandteil der Entwicklung werden. Dabei muss entsprechendes Know-how zum Schreiben der Tests in den Teams verteilt und Testframeworks evaluiert werden. Die Tests müssen dabei unabhängig voneinander sein. Anschließend dreht sich sehr viel um die Automatisierung. Diese umfasst die Ausführung genannter Tests und die Bereitstellung resultierender Daten. Damit diese Tests zuverlässige Ergebnisse liefern, muss die Entwicklung und der Betrieb das Deployment und die Konfiguration der Anwendung gemeinsam planen, anpassen und warten [2].

Dabei sollte dieser Ansatz möglichst zu Beginn eines Projekts verfolgt werden, um die Pipeline schrittweise aufzubauen. An eine bestehende Codebasis anzusetzen bedeutet oftmals eine fundamentale Änderung der bisherigen Techniken und Abläufe, denn wie bereits aus den obigen Kapiteln hervorgegangen ist, bedeutet das Einrichten einer DevOps-Kultur unter anderem eine Änderung an der gesamten Architektur des Systems. Dies bedeutet oftmals einen gewissen Leistungsdruck. Die Implementierung der Automatisierung benötigt ebenfalls Zeit. Der hier erbrachte Aufwand ist in dieser Zeit nicht für eine Weiterentwicklung des Systems bzw. der Features verfügbar.

Eine enge Zusammenarbeit der Teams rückt in den Vordergrund. Dies lässt sich allerdings oft auch recht einfach gestalten. Dabei reicht es, die Arbeitsplätze der Mitarbeiter aus dem Betrieb direkt mit den Arbeitsplätzen der Entwicklung in einen Raum zusammenzulegen. So können beide Abteilungen viel einfacher miteinander kommunizieren, was die Zusammenarbeit wesentlich einfacher gestaltet. Dazu ist keine Änderung in der Organisation notwendig. DevOps kann so erreicht werden, indem man lediglich ein anderes Verständnis von Entwicklung und Betrieb bekommt, wodurch eine Änderung nicht unbedingt notwendig ist, allerdings sehr förderlich.

DevOps beeinflusst also weit über Continuous Delivery hinaus die Prozesse, Teams und das Vorgehen. Gleichzeitig werden mit DevOps noch viele weitere Vorteile realisiert. Für eine vollständige Continuous-Delivery-Pipeline sind sowohl Fähigkeiten aus dem Betriebs- wie aus dem Entwicklungsumfeld notwendig. Ohne eine gewisse Kooperation kann eine Continuous-Delivery-Pipeline also nicht vollständig umgesetzt werden.

LITERATUR

[1] Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through build, test and deployment automation*. Boston, MA 02116: 2011 Pearson Education, 2011.

[2] E. Wolff, *Continuous Delivery: Der pragmatische Einstieg*, 2nd ed., 2016.

[3] A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikainen, *Product-Focused Software Process Improvement*. Cham: Springer International Publishing, 2014, vol. 8892.

[4] E. Laukkanen, J. Itkonen, and C. Lassenius, "Problems, causes and solutions when adopting continuous delivery—a systematic literature review," *Information and Software Technology*, vol. 82, pp. 55–79, 2017.

[5] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[6] L. Chen and P. Power, "Continuous delivery: Huge benefits, but challenges too," *IEEE Access*, pp. 50–54, 2015.

[7] S. Stolberg, "Enabling agile testing through continuous integration," in *Agile '09, Agile Conference, 2009*, Y. Dubinsky, Ed. Piscataway, NJ: IEEE, 2009, pp. 369–374.

[8] DevOps, "Devops and continuous delivery: Not the same," 2016. [Online]. Available: https://devops.com/devops-and-continuous-delivery-not-same/