präsentiert von: heise Developer
www.heise-developer.de

Mehr als drei Buzzwords:
Continuous Delivery, DevOps und Container

iX DEVELOPER

2/2016

Auf der Heft-DVD

Massig Material für Entwickler

Software:
Ubuntu, Eclipse, Docker, Kubernetes, Ansible, Chef, Puppet, Jenkins, Vagrant, Mesos, VirtualBox etc.

Multimedia:
4 Continuous-Lifecycle-Videos zu Continuous Delivery, DevOps und Branch-Modellen; mehrere Episoden des SoftwareArchitek-TOUR-Podcasts

Literatur:
Microservices (Broschüre), Continuous Delivery (Buchauszüge), *iX* Developer „Bessere Software" (Sonderheft)

Effektiver entwickeln

Container:

Docker & Co. revolutionieren die IT-Landschaft

Grundlagen, Ökosystem und Orchestrierung

DevOps:

IT-Kulturwandel eint Entwickler und Administratoren

Compliance, Sicherheit, APM und Change Management

Continuous Delivery:

Basiswissen, Einführung, Tools und Management

Chef, Puppet und Ansible im Dienst der Continuous Integration

Microservices:

Architekturparadigma stützt Continuous Delivery und DevOps

Mannheim, Congress Center Rosengarten,
14.-16. November 2016

continuous lifecycle 2016

Prozesse – Tools – Erfahrungen

CONTINUOUS WHAT? CONTINUOUS EVERYTHING!

Die drei Auflagen der Continuous Lifecycle haben es deutlich gezeigt: Continuous Delivery, DevOps und Containertechniken sind in der Praxis angekommen – nicht nur in Web-2.0- und Cloud-Firmen, sondern auch in Unternehmen, die klassische Unternehmens- oder Embedded-Anwendungen herstellen.

Wie aber setzt man diese Konzepte und die damit verbundenen Techniken sinnvoll im eigenen Projekt oder Unternehmen ein? Wo steckt das größte Potenzial, wo liegen die Fallstricke? Antworten auf diese Fragen gibt Ihnen die Continuous Lifecycle – die wichtigste deutsche Konferenz zu Continuous Delivery, DevOps und Containerisierung. Ganzheitlich widmet sie sich den Konzepten, Prozessen und Werkzeugen hinter Continuous Delivery, DevOps und Co. und bietet Erfahrungen, die Ihnen praktisch weiterhelfen.

Call for Papers bis 30. Mai 2016

THEMEN (Auszug)

- Der richtige Umgang mit Continuous Delivery
- Praktische Umsetzung von DevOps-Methoden
- Werkzeuge für agiles Application Lifecycle Management (Versionskontrolle, Continuous Integration, Ticketing und Bugtracking)
- Containerisierung mit Docker und den Werkzeugen aus dem Docker-Ökosystem
- Build Management
- Code-Reviews
- Testen und Qualitätssicherung
- Betrieb und Monitoring
- Fallstricke und Best Practices verteilt arbeitender Software-Teams

ZIELGRUPPE

- Softwareentwickler
- Softwarearchitekten
- Administratoren
- Projektleiter
- IT-Strategen

Gold-Sponsor:

codecentric XebiaLabs Deliver Faster

Silber-Sponsoren:

adesso business. people. technology. CloudBees The Enterprise Jenkins Company OPITZ CONSULTING

Veranstalter:

iX heise Developer dpunkt.verlag

www.continuouslifecycle.de

Der ewige Kreis

Die Softwareentwicklung zeigt derzeit Mut zur Größe – zur kleinen Größe. Der Trend geht weg von monolithischen Systemen hin zu kleinen Bausteinen wie Microservices. Auch die Zeiträume zwischen Releases schrumpfen: Statt alle ein bis drei Jahre ein großes Major Release herauszubringen und durch Service Packs zu ergänzen, veröffentlichen Unternehmen mithilfe von Continuous Delivery eine neue Softwareversion, sobald sich eine Komponente ändert. Große Internetfirmen wie Google, Amazon oder Facebook aktualisieren ihre Plattformen inzwischen im Minutentakt. Dass die Nutzer davon nur bei Änderungen der Oberfläche etwas mitbekommen, zeigt, wie reibungslos die kontinuierlichen Prozesse funktionieren können. Ein Wartungsfenster am Wochenende existiert im Internet nicht. Die Softwareentwicklung wird zum ewigen Kreis.

Container bieten Entwicklern eine in sich abgeschlossene und damit überschaubare Umgebung zum Testen und Ausrollen der Anwendungen. Dank der einfachen Reproduzierbarkeit können die Teams auf große Testumgebungen verzichten. Ändern sich die Anforderungen oder Umgebungen, passen sie einfach die Schablone an und rollen neue Container aus.

Containerisierung, Komponenten oder kontinuierliche Prozesse sind keine Erfindung der 2010er-Jahre, sondern haben in der ein oder anderen Ausprägung immer wieder den Weg in die Schlagzeilen geschafft. Oft scheiterten sie in der Realität jedoch an Komplexität und Inkompatibilitäten. Es gibt jedoch entscheidende Unterschiede, die den aktuellen Techniken und Tools zum Erfolg verhelfen sollten.

Microservices punkten durch die Unabhängigkeit der einzelnen Dienste voneinander: Jeder hat seine eigene Ablaufumgebung und kommuniziert über Standardnetzwerkprotokolle mit der Außenwelt. Frühere Modularisierungskonzepte bedienten oftmals eine spezielle Sprache oder verwendeten einen Applikationsserver und scheiterten an der Zusammenarbeit über die Grenzen hinweg. Docker-Container haben einen klaren Fokus auf Entwickler und sind leichter zu verwenden als vorherige Ansätze. Jenkins ist besonders wegen seiner Flexibilität und des offenen Plug-in-Konzepts zur festen Größe für Continuous Delivery geworden. Bei den Konfigurations-Management-Systemen buhlen vor allem Chef, Puppet und Ansible mit unterschiedlichen Stärken um die Gunst der Anwender.

Die technische Basis und die weitgehend quelloffenen Werkzeuge sind jedoch nur die halbe Miete. DevOps, also das Zusammenspiel von Softwareentwicklern (Dev) und den Administratoren (Ops), erfordert ein Umdenken der Beteiligten: Die Fraktionen müssen zusammenarbeiten und bei Problemen gemeinsam nach Lösungen suchen. Wenn Entwickler im Prozess der kontinuierlichen Verteilung eine Komponente ändern und damit einen neuen Build anstoßen, der die Änderung sofort ins Livesystem übernimmt, kommt ihnen und ihrem Werk zudem eine höhere Bedeutung zu als bisher. Der Satz „You build it, you run it“ von Amazons CTO Werner Vogels bringt es auf den Punkt: Jeder Entwickler ist für seinen Code verantwortlich und bekommt dadurch mehr Bezug zum Gesamtprodukt.

Der Kulturwandel erfordert Umdenken auf allen Ebenen. In einigen Projekten erweist es sich als sinnvoll, dass sich das Management aus dem Tagesgeschäft heraushält und somit den einzelnen Teams mehr Verantwortung zuteilt. Auch traditionelle Unternehmen wie Maschinenbauer können durchaus von moderner Softwareentwicklung profitieren und mit den passenden Ansätzen den Produktionsbetrieb in der Fabrik optimieren, ohne dort den reibungslosen Ablauf zu gefährden. Damit schließt er sich wieder, der ewige Kreis.

Mit diesem Heft möchten wir Ihnen helfen, sich bei den Themen Continuous Delivery, DevOps, Containerisierung und Microservices zurechtzufinden. *iX* und *heise Developer* wünschen Ihnen viel Spaß beim Lesen!

RAINALD MENGE-SONNENTAG & ALEXANDER NEUMANN

Continuous Delivery

... verspricht Produktivsetzungen der entwickelten Software auf Knopfdruck bei besserer Qualität. Was sind die zentralen Ideen dahinter, welche Voraussetzungen muss man schaffen, um damit erfolgreich zu sein, und was sind die wichtigsten Tools?

ab Seite 36

DevOps

... lebt nicht nur von geeigneten Werkzeugen, sondern vor allem von gegenseitigem Vertrauen und geteilter Verantwortung. Erst dann lässt sich die Idee hinter DevOps umsetzen, wie Erfahrungen mit entsprechenden Tools aus der Praxis zeigen.

ab Seite 66

Einführung

Continuous Delivery

DevOps

Microservices

... lassen sich unabhängig skalieren, und der Ausfall eines Service beeinflusst die anderen nicht. Je kleiner er ist, desto größer der Vorteil. Aber wo liegt die Grenze für die Größe eines Microservice? Und wann ergeben sie überhaupt Sinn?

ab Seite 100

Docker

... vereinfacht den Test- und Produktivbetrieb. Die Docker-eigenen Werkzeuge decken die Grundbedürfnisse ab, darüber hinaus ist inzwischen ein reiches Ökosystem zur Orchestrierung der Container in nahezu beliebig komplexen Szenarien gewachsen. Best Practices sorgen für einen sicheren Betrieb.

ab Seite 126

Microservices

Container

Sonstiges

Alle Links: www.ix.de/ix1614004

Artikel mit Verweisen ins Web enthalten am Ende einen Hinweis darauf, dass diese Webadressen auf dem Server der *iX* abrufbar sind. Dazu gibt man den *iX*-Link in der URL-Zeile des Browsers ein. Dann kann man auch die längsten Links bequem mit einem Klick ansteuern. Alternativ steht oben rechts auf der *iX*-Homepage ein Eingabefeld zur Verfügung.

Auf der Heft-DVD

Multimedia

Videos (Continuous Lifecycle 2015)

The Rationale for Continuous Delivery

Dave Farleys Keynote-Vortrag von der Continuous Lifecycle 2015 ist ein zeitgemäßer Überblick zu den mit Continuous Delivery und DevOps verbundenen Entwicklungsprozessen.

Reinventing your Workplace – Struktur, Technik und Kultur bei 1&1 MyWebsite

Der Vortrag von Matthias Kainer und Marcel Devantier zeigt, wie Struktur, Technik und Kultur Hand in Hand gehen, welche Werkzeuge man dabei verwenden kann und wie schwierig es ist, alles zu ändern.

Microservices, DevOps, Continuous Delivery – mehr als drei Buzzwords

Eberhard Wolffs Vortrag zeigt, wie Microservices, Continuous Delivery und DevOps zusammenpassen und dass Microservices mehr als ein Hype sind.

Branch-Modelle mit Git

Verschiedene Entwicklungsszenarien erfordern passende Branches. Der Vortrag von Bjørn Stachmann und René Preißel zeigt Möglichkeiten, diese und weitere Release-Prozesse mit Git umzusetzen.

Audio (SoftwareArchitekTOUR)

- Modularisierte Architektur für große Systeme
- Know-how für Architekten
- Architekturanalyse und -bewertung
- Soziale Kompetenz für Architekten

Software

Ubuntu 14.04.4

Long-Term-Support-Release der populären Linux-Distribution, die für viele Artikel des Hefts herangezogen wurde.

Eclipse 4.5.2

Auf der Heft-DVD finden Leser das zweite Update der Entwicklungsumgebung in der Mars-Distribution für Java-Entwickler.

DevOps- und Produktivitätswerkzeuge

Continuous Delivery und DevOps sind ohne diese auf DVD zu findenden Tools nur schwer vorstellbar: Ansible, Chef, Puppet, Docker, Kubernetes, Marathon, Aurora, Mesos, Vagrant, Flyway, Liquibase, Jenkins, VirtualBox.

Listings und Lizenzen

Die Listings zu den Heftartikeln und die Lizenzen zu den Softwarepaketen auf der Heft-DVD.

Literatur

Microservices (Broschüre zum Buch)

Diese von Eberhard Wolff konfektionierte Broschüre gibt eine kurze Einführung in das Thema Microservices und stellt dar, was Microservices überhaupt sind und welche Vorteile sie haben. Das soll den Einstieg in das Thema erleichtern und helfen, die Einsetzbarkeit und den Nutzen von Microservices in einem bestimmten Kontext abzuschätzen. So können Microservices vom Hype zum sinnvollen Bestandteil im Werkzeugkasten eines Architekten werden.

Continuous Delivery – Der pragmatische Einstieg (Buchauszüge)

Continuous Delivery ermöglicht es, Software viel schneller und mit wesentlich höherer Zuverlässigkeit in Produktion zu bringen. Eberhard Wolffs Buch hilft, sich mit dem Aufbau einer Continuous-Delivery-Pipeline vertraut zu machen, und erklärt, welche Technologien Entwickler dazu einsetzen können.

***iX* Developer „Bessere Software"**

Das von *heise Developer* 2013 zusammengestellte Heft zur Qualitätssicherung in der Softwareentwicklung setzt auf 180 Seiten den Fokus auf die Schlagwörter „Agile ALM", „Continuous Delivery" und „DevOps".

Hinweis für Käufer

- PDF-, Android- und iPad-Version: In der *iX*-App finden Sie einen Button zum Download des DVD-Images.
- PDF-E-Book: Folgen Sie im Browser der unter „Alle Links" angegebenen URL.

Alle Links: www.ix.de/ix1614006

Testqualität, Build-Security, Systemautomatisierung

Die hier im Heft behandelten Themen Continuous Delivery, DevOps und Microservices sind mehr als nur ein Hype, auf den sich alle stürzen. Grundlagen hierfür wie agile Testverfahren sowie Build- und Systemautomatisierung sind seit Jahren bewährt. Das nährt die Hoffnung auf eine goldene Zukunft.

Wie DevOps, Continuous Delivery und Containerisierung zum idealen Gespann werden

Triumvirat

Alexander Neumann, Rainald Menge-Sonnentag

Ohne Frage ist DevOps mehr als nur ein Hype, auf den Tool-Hersteller, Dienstleister und Konferenzanbieter aufspringen. Mehrere Indikatoren sprechen für die Durchdringung der mit dieser Bewegung einhergehenden Prinzipien in den IT-Abteilungen.

Die Zahl der Untersuchungen und Konferenzen zum auch in diesem Sonderheft zentralen Thema DevOps nimmt kontinuierlich zu, genauso die Zahl der Unternehmen, die mit Werkzeugen, Services und Dienstleistungen, aber auch Fortbildungsangeboten an dem großen Kuchen rund um DevOps partizipieren wollen.

Der Begriff selbst geht auf den Belgier Patrick Debois zurück, der Ende Oktober 2009 die ersten DevOpsDays mitorganisierte [a]. Die Kombination aus „Dev“ für Anwendungsentwicklung (Development) und „Ops“ für IT-Betrieb (Operations) steht für das Zusammenrücken der beiden Bereiche mit der Absicht, dass eine Organisation Software schneller und fehlerfreier erstellen und verfügbar machen kann. Ziel ist also eine Verbesserung ihrer Prozesse durch die hoffentlich gelebten DevOps-Prinzipien. Sie beruhen vorrangig auf denen der agilen Softwareentwicklung, genau genommen auf den Forderungen des dieses Jahr den 15-jährigen Geburtstag feiernden agilen Manifests [b].

Immer wieder ist auch davon die Rede, dass mit DevOps nun die agilen Entwicklungspraktiken Einzug in den IT-Betrieb gehalten haben. De facto sind die Zeiten vorbei, in denen Software sprichwörtlich „über die Mauer geworfen“ wird. Der Kommunikation zwischen den Abteilungen kommt so eine viel größere Bedeutung zu, wie schon das agile Manifest propagiert.

Das Ziel von DevOps-Praktiken wie Continuous Delivery – übrigens ebenfalls ein Schlagwort im erwähnten Manifest –, Versionskontrolle und automatisierte Tests ist eine häufigere Inbetriebnahme von Software bei gleichzeitiger Reduzierung der Ausfälle. Das impliziert ein reibungsloses Zusammenspiel zwischen den Abteilungen und mündet in Zufriedenheit mit dem Job, da Softwareentwickler ihr Produkt schnell und transparent ausliefern – ein Erfolgserlebnis ist sichtbar – und Administratoren einen zuverlässigeren Betrieb gewährleisten können. Hinzu kommt, dass alle Beteiligten am Entwicklungs- und Auslieferungsprozess oder zumindest möglichst früh in die Designentscheidungen eines Systems eingebunden sind. Das fördert einerseits das Gemeinschaftsgefühl, andererseits werden Fehler womöglich früher erkannt.

Dass die Einführung der DevOps-Maßnahmen in der Praxis tatsächlich fruchtet, ist aus jüngsten Erhebungen [c] zum Thema zu beobachten: Mehr als drei Viertel der deutschen Entwickler sind demnach der Meinung, dass seit der Anwendung von DevOps weniger Probleme beim Live-Gang von Neuentwicklungen auftreten. Zugleich kommt es wohl zur Verkürzung der Release-

Köln, KOMED Veranstaltungszentrum,
10. – 12. Mai 2016

building IoT

Die neue Softwareentwicklerkonferenz zu Internet of Things und Industrie 4.0

EINTAUCHEN INS INTERNET DER DINGE

Das Internet der Dinge verspricht viel: eine Fülle neuer Produktideen und Geschäftsmodelle sowie weitreichende Veränderungen im privaten und beruflichen Leben. Während es an kreativen Ideen nicht mangelt, ist die technische Umsetzung nicht trivial, und viele Softwareentwickler stehen vor großen Herausforderungen wie einem schwer überschaubaren Ökosystem aus Hardware, Protokollen und Softwarekomponenten. Auf der anderen Seite müssen sich Entwickler mit Hardware-Erfahrung für sie neue Themen erarbeiten, unter anderem Vernetzung / Connectivity, Sicherheit und Datenschutz.

Jetzt registrieren!

Die building IoT 2016 hilft Ihnen, sich im Dschungel der neuen Aufgaben zurechtzufinden. Treffen Sie die Gestalter des Internet of Things, das sich langsam zum Internet of Everything mausert. Zwar ist das IoT noch am Anfang, allerdings helfen das in Vorträgen vermittelte Grundlagenwissen und die Möglichkeit zum Austausch über aktuelle Forschungsvorhaben dabei, eigene Projekte bereits jetzt erfolgreich anzugehen und umzusetzen.

AUS DEM PROGRAMM

- Das Internet der Dinge aus Architekturperspektive
- Boards für das IoT-Prototyping
- IoT-Protokolle: Grundlagen und Vergleich
- IoT in der Cloud – ein Vergleich verschiedener Plattformen
- Attacken auf IoT-Geräte – Trends und Prognosen für die Zukunft
- Testen von IoT-Produkten
- Software-Updates im Internet der Dinge
- Wem gehören die IoT-Daten?
- BeeData – Wie das Internet Imkern und Bienen hilft

KEYNOTES

- **Andy Stanford-Clark** (IBM):
  Innovation Begins at Home
- **Joshua Corman** (I am The Cavalry / Sonatype):
  CyberSafety for the Internet of Everything:
  Bits & Bytes meet Flesh & Blood

Gold-Sponsoren:

adesso business. people. technology.

SAP

Silber-Sponsoren:

applied informatics

codecentric

OPITZ CONSULTING

ThoughtWorks

Bronze-Sponsoren:

heise COMPUTER

intel Software Elite Reseller

intel Software

Veranstalter:

iX

heise Developer

dpunkt.verlag

www.buildingiot.de

Zyklen und zur Steigerung der Produktivität in der IT insgesamt. Allerdings scheint bislang noch nicht mal die Hälfte aller deutschen Entwicklungsabteilungen die DevOps-Konzepte oder -Praktiken eingeführt zu haben. Jedoch planen anscheinend viele, auch auf den DevOps-Zug aufzuspringen. Die damit einhergehenden Prinzipien zu etablieren, stellt jedoch kein leichtes Unterfangen dar. Denn DevOps bedeutet an erster Stelle einen Kulturwandel, auf den sich mindestens die IT-Abteilung eines Unternehmens einigen muss.

Die Mauern überwinden

Doch sind in vielen Unternehmen Softwareentwicklung und IT-Betrieb immer noch zwei getrennte Einheiten, die traditionell gegenläufige Ziele verfolgen und zumindest bis vor kurzem unterschiedliche Werkzeuge einsetzten. Während die Entwicklung neue Features in möglichst hoher Frequenz veröffentlichen möchte, ist der Betrieb für eine hohe und schnelle Verfügbarkeit verantwortlich. Dieser Gegensatz zwischen Veränderung und Stabilität hat häufig Konflikte zur Folge – insbesondere dann, wenn in Unternehmen keine übergreifende Sicht auf dieses Thema existiert, was die Spannungen noch verstärkt. Man spricht hier von einer „Wall of Confusion“ zwischen Entwicklern und Admins, die nicht selten sogenannte „Blame Games“ zur Folge haben („Die Artefakte und Config-Dateien waren fehlerhaft.“; „Bei uns hat alles funktioniert, ihr habt was falsch gemacht.“ etc.). Währenddessen kann niemand mit der Software arbeiten – irgendwie blöd, denn Kunden ist es egal, ob der Fehler bei Development oder Operations liegt. Die Konsequenz muss also sein, dass die beteiligten Parteien näher zusammenrücken.

Die womöglich über Jahrzehnte gewachsenen Strukturen aufzubrechen, ist jedoch nicht einfach. Deswegen verwundert es nicht, dass viele an der DevOps-Einführung Beteiligte von einem anfänglichen Chaos sprechen. Fast drei Viertel der mit DevOps experimentierenden Unternehmen greifen deswegen auf externe Dienstleister zurück, um sich helfen zu lassen [d]. Die zum Einsatz kommenden Werkzeuge können dabei eine zunehmend gewichtigere Rolle spielen. Glücklicherweise haben viele der auch in diesem Heft zur Sprache kommenden Tools mittlerweile den Status eines De-facto-Standards: Beispielsweise kommt der Continuous-Integration-Server Jenkins unter den CI-Tools schon auf eine Verbreitung von 75 Prozent. Quelloffene Konfigurations-Management-Systeme wie Ansible, Chef und Puppet genießen große und immer noch zunehmende Verbreitung.

Mit Docker gibt es nun das Werkzeug schlechthin, mit dem sich (nicht nur) die Zusammenarbeit von Devs und Ops sinnvoll praktizieren lässt. Es bietet neue Möglichkeiten, wie Software gebaut, verteilt und betrieben wird. Denn es definiert Schnittstellen zwischen den Teams mit der Folge, dass Admins Monitoring, Logging, Networking und Persistenz für Anwendungs-Container bereitstellen können, ohne dass das Auswirkungen auf die Software im Container hat. Das bedeutet in der Konsequenz, dass Anwendungen, die in der Testumgebung fehlerfrei arbeiten, in der Produktivumgebung ebenso funktionieren. Sollten denn doch Fehler auftauchen, lassen sie sich immer genau zuweisen.

Die Nutzung von Docker bedeutet jedoch auch, dass sich die Arbeitsweise, wie bislang Software entwickelt und administriert wurde, grundlegend verändert – Softwareentwickler sind mehr denn je als Kommunikatoren gefragt. Nicht wenige sprechen bei Docker gar fast schon von einer Revolution für die Softwareentwicklung; zumindest lässt sich feststellen, dass es bisher kaum eine Software gegeben hat, die von Entwicklern und Administratoren gleichermaßen begrüßt wurde. Das beiseite gelassen, ist die Popularität von Docker konkret mit Zahlen festzumachen. So hat sich laut der erwähnten RightScale-Studie [d] der Einsatz des Werkzeugs innerhalb eines Jahres offenbar von 13 auf 27 Prozent mehr als verdoppelt. Weitere 35 Prozent planen die zukünftige Verwendung. Bei den über 1000 befragten Entwicklern arbeiteten mehr als 40 Prozent für Unternehmen mit mehr als 1000 Angestellten.

DevOps ist womöglich erst der Anfang

In einem weiteren und hier letzten Schritt mag es sogar sinnvoll sein, das Thema DevOps noch größer zu denken und weitere Abteilungen (Schlagwort „DevOpsBiz“), ja sogar das ganze Unternehmen in die agilen Prozesse einzubinden. Einige wenige praktizieren das schon, andere informieren sich in die Richtung, für die meisten Firmen ist das jedoch ferne Zukunftsmusik – wenn überhaupt. Die Zeiten, in denen IT-Abteilungen innerhalb von Unternehmen eine Randrolle spielten, sind jedoch vorbei. „Software Is Eating the World“ – das Bonmot von Marc Andreessen, dem Erfinder des Netscape-Browsers und heutigen Investor, trifft im Zuge einer zunehmend digitalisierteren Gesellschaft heute noch mehr zu als im Jahr 2011, aus dem der Satz stammt [e].

Derzeit befinden sich Softwareentwicklung und -betrieb allemal auf einem guten Weg. Was mit agiler Softwareentwicklung und Continuous Integration vor über 15 Jahren seinen Auftakt nahm, dann seine Fortführung in Konzepten wie Continuous Delivery gefunden hat, erreicht bei DevOps ein erstes Zwischenhoch, das in andere Unternehmensbereiche hineinwirkt. Tools wie Docker tragen ihren Teil dazu bei, möglichst konfliktfrei und gleichzeitig hochqualitativ Software über den ganzen Anwendungslebenszyklus hinweg zu entwickeln und auszuliefern. (ane)

Onlinequellen

[a] Erste DevOpsDays in Ghent
www.devopsdays.org/events/2009-ghent/

[b] Agiles Manifest
agilemanifesto.org/iso/de/

[c] Developer Week; Umfrage zu DevOps und Cloud vs. Server, Teil 1: DevOps
www.developer-week.de/content/download/20642/225705/file/DWX2016-Studie_zu_DevOps.pdf

[d] Cloud Computing Trends: 2016 State of the Cloud Survey (Studie von RightScale)
www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey

[e] Marc Andreessen; Why Software Is Eating The World
www.wsj.com/articles/SB10001424053111903480904576512250915629460


Alexander Neumann und Rainald Menge-Sonnentag
sind Redakteure von heise Developer und haben dieses Sonderheft konzipiert.

Alle Links: www.ix.de/ix1614008

Zuverlässige Tests in der agilen Softwareentwicklung

Mit Argusaugen

Markus Gärtner

Gute Tests sind ein Muss beim Erstellen zuverlässiger Software. Die kurzen Releasezyklen im Prozess der Continuous Delivery stellen neue Anforderungen an die Tester. Sie müssen die richtigen Methoden für die jeweiligen Stufen auswählen und reproduzierbar wiederholen.

Mit dem Aufkommen agiler Softwareentwicklung stellen sich viele Teams die Frage, wie sie das Testen der Software effektiv gestalten. Kurze Zyklen von wenigen Wochen sind die Norm. Am Ende jedes Zyklus soll eine potenziell auslieferbare Version der Software entstehen. Das einzige, was der Auslieferung im Wege steht, soll die Entscheidung eines Verantwortlichen wie des Product Owner in Scrum sein. Dafür ist der ordentliche Test der Software unerlässlich, um Risiken vor der Auslieferung zu vermeiden und dem Endanwender keine „Bananensoftware“ zu liefern, die beim Kunden reift. Das Testen von Software erfüllt damit einen essenziellen Mehrwert in der Bewältigung von Geschäftsrisiken.

Klassisch unterscheidet die Welt des Testens verschiedene Testarten, -stufen und unterschiedliche Herangehensweisen. Unit-Tests, User-Acceptance-Tests, Integrationstests mit unterschiedlichen Komponenten, Last- und Performancetests, manuelles Testen und Testautomatisierung sind weitverbreitete Methoden, um die Qualität einer Software sicherzustellen. Für den Erfolg kommt es vor allem auf die ausgewogene Mischung im jeweiligen Projektkontext an. In einem agilen Projekt liegt ein großer Teil der Verantwortung beim Entwicklungsteam.

Bei der Antwort auf die Frage, wie ein agiles Team zu einer sinnvollen Auswahl unterschiedlicher Vorgehensweisen kommt, hilft eine grundlegende Kenntnis der jeweiligen Stärken und Einsatzgebiete. Das Modell von Nassim Nicholas Taleb [1] unterscheidet zwischen bewusstem und unbewusstem Wissen und Unwissen (s. Abb. 1). Bei unbewusstem Wissen handelt es sich um eingespielten Abläufe, die im Englischen „muscle memory“ genannt werden. Bewusstes Wissen umfasst all das, was Menschen mit etwas Anstrengung ihrer kognitiven Fähigkeiten abrufen können. Bewusstes Unwissen bezieht sich auf jene Dinge, über die sie (noch) nichts wissen. Das unbewusste Unwissen ist alles, von dem sie noch nicht einmal wissen, dass sie es nicht wissen. Unterschiedliche Testarten können Teams bei ihrem Wissen und Nichtwissen rund um ihre Software unterstützen.

Bewusstes Wissen

Übertragen auf die Softwareentwicklung fällt unter bewusstes Wissen beispielsweise die Kenntnis der Applikationsanforderun-

gen oder das Domänenwissen zum Einsatzgebiet der Software. Bei vorhandenen Systemen zählt auch das Know-how der Architektur und deren Design, also welche Elemente in der Software welchem Zweck dienen, die Aufgabe jeder kleineren Klasse sowie das Zusammenspiel von mehreren Klassen, um das darunter liegende Businessproblem zu lösen.

Das wirft die Frage auf, in welcher Form Teams das gesammelte Wissen festhalten können. Klassische Ansätze orientieren sich zumeist an Anforderungsdokumenten sowie Architektur- und Designspezifikationen. Über die Zeit können die Dokumente jedoch veralten und nicht mehr viel mit dem Anwendungsverhalten zu tun haben. Die agile Softwareentwicklung setzt einen stärkeren Fokus auf funktionierende Software als auf allzu ausführliche Dokumentation, weil Letztere nur schwerlich alles notwendige Wissen vermitteln kann. Die Einarbeitung neuer Projektmitarbeiter durch die Dokumentation ist nicht unmittelbar möglich. Zudem prüfen Kunden ausführliche Anforderungsdokumente nicht unbedingt vollständig auf ihre Korrektheit.

Automatisierte Tests können an die Stelle klassischer Dokumentation treten. Die Vorteile davon liegen auf der Hand: Wenn die Tests regelmäßig, beispielsweise in einem Continuous-Integration-System, mit jeder Änderung ausgeführt werden, ist sichergestellt, dass sie das Wissen widerspiegeln, das zur Entwicklung eines Teils der Software vorhanden war. Automatisierte Unit-Tests auf der kleinsten Ebene wie individuelle Klassen und Funktionen dokumentieren dabei das Softwaredesign. Tests, die mehrere Klassen heranziehen, dokumentieren die größeren Design- und Architekturentscheidungen. Automatisierte Akzeptanztests halten zum einen die Anforderungen an die Software – also das Domänenwissen – fest, zum anderen steckt auch die Kenntnis der Architektur im Automatisierungscode.

Im Gegensatz zu klassischer Dokumentation erfüllen die automatisierten Tests eine weitere Bedingung: Durch regelmäßiges Ausführen ist sichergestellt, dass sie auf dem aktuellen Stand hinsichtlich der Codebasis sind. Damit kann eine gute automatisierte Testbasis Wissen zur Anwendung auch für Neulinge im Code dokumentieren und effektiver transportieren als geschriebene Texte.

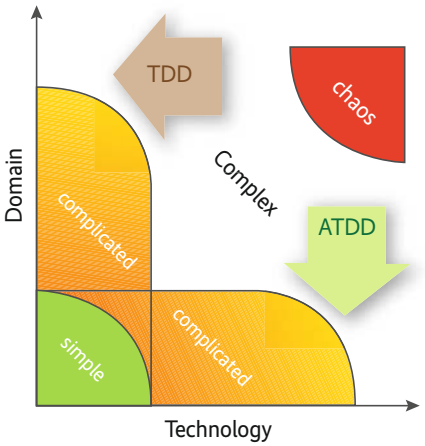
Unbewusstes Wissen

Unbewusstes Wissen umfasst die Dinge, die der Mensch ohne gezielt eingesetzte Überlegungen tut. Wer morgens immer den gleichen Weg zur Arbeit nimmt, kennt den Effekt. Man steigt morgens in das Beförderungsmittel seiner Wahl und weiß am Ende der Reise gar nicht mehr, wie man ins Büro gekommen ist. In dem Fall hat der Mensch unbewusstes Wissen für das Autofahren oder das Umsteigen bei öffentlichen Verkehrsmitteln verwendet.

Notfalleinsatzkräfte trainieren in regelmäßigen Abständen Abläufe, die sie in einer brenzligen Situation unbewusst abrufen können müssen. Dazu gehört die obligatorische Auffrischung im Abstand von zwei Jahren.

bewusstes Wissen	bewusstes Unwissen
unbewusstes Wissen	unbewusstes Unwissen

Wissen und Unwissen können bewusst und unbewusst sein (Abb. 1).



Test-driven Development (TDD) und Acceptance Test-driven Development (ATDD) adressieren unterschiedliches Wissen. Während es bei TDD um die eingesetzte Technologie geht, hilft ATDD beim Verstehen der Anwendungsdomäne (Abb. 2).

Ähnlich verhält es sich mit diversen Praktiken rund ums Entwickeln und Testen: Wer in einem stressigen Projekt ist und keine Zeit hat, an seinen Fertigkeiten zu arbeiten, sollte im Eifer des Gefechts Grundwissen abrufbereit haben. In der Software Craftsmanship Community ist daher das stetige Durchführen von Code-Katas verbreitet. Das sind kleine, abgeschlossene Übungen, bei denen es um die effektive Anwendung von testgetriebener Entwicklung und emergentem Design geht. Softwarehandwerker wissen, dass sie regelmäßig einstudierte Praktiken in stressigen Situationen unbewusst abrufen können. Dafür gibt es den englischen Fachausdruck „Deliberate Practice“, also wohldurchdachtes Praktizieren.

Vergleichbares kann man auf mehreren Ebenen für Softwaretests anwenden. In einem stressigen Projekt oder Sprint, in dem ein Mitstreiter ausgefallen ist, sollen Teams ein Grundmaß an Wissen schnell abrufen können, ohne lange darüber nachzudenken. Zum Einstudieren können sie beispielsweise regelmäßig an Testing Dojos teilnehmen. Dabei schärfen die Teilnehmer anhand kleiner Applikationen ihre Fähigkeiten und lernen, nützliches Wissen außerhalb der Arbeitsumgebung in einem sicheren Umfeld anzuwenden. Vor allem entsteht während der Trainingsphase kein Schaden, wenn doch ein Fehler übersehen werden sollte.

Weitere Möglichkeiten zur Schärfung des unbewussten Wissens bestehen darin, frei im Internet verfügbare Applikationen zu testen und dabei beispielsweise Wissen aus einem frisch gelesenen Blog-Eintrag anzuwenden. Vor einigen Jahren hat der Autor bei einem Kunden einen Testing-Dojo-Workshop organisiert, bei dem es darum ging, das Wissen rund um den Billing-Teil der Applikation zu vertiefen. In den Teams vor Ort kam es bei Änderungen an der Abrechnungskomponente immer wieder zu Problemen, aber kein Teilnehmer wusste, wie sie die Modifikationen testen konnten. Geholfen hat schließlich, dass mehrere testaffine Teilnehmer jeweils in Paaren unter Zuhilfenahme der REST-API-Dokumentation gearbeitet haben. Am Ende des Workshops hat die Gruppe zusätzliche Schritte zur Verbesserung der Testbarkeit der betroffenen Komponente auf den Weg gebracht, sodass sich die unterschiedlichen Personen jetzt sicherer im Umgang mit ihr fühlen.

Bewusstes Unwissen

Das bewusste Unwissen umfasst die Dinge, von denen der Mensch weiß, dass er sie nicht weiß. Darunter fällt etwa das Do-

mänenwissen beim Entwickeln einer neuen Applikation. Es kann auch das Know-how der eingesetzten Techniken sein: Wenn Entwickler ein neues Framework im Code ausprobieren wollen, treffen sie auf eine unbekannte Funktionsweise. Teile des bewussten Unwissens kann man ignorieren. Wenn jedoch beispielsweise das Spezialwissen eines Teammitglieds gefragt ist, das in Urlaub ist oder ausfällt, hindert die Unkenntnis das Team daran, weiteren Wert für den Kunden zu schaffen, bis der Kollege zurück ist.

Weg zur Erkenntnis

Bei bewusstem Unwissen können sich die Betroffenen mit der neuen Technologie oder dem Unbekannten auseinandersetzen. Sie erlangen neues Wissen, wenn sie mehr über das Unbekannte lernen. Kent Beck setzt etwa testgetriebene Entwicklung (engl.: Test-driven Development, TDD) auch dazu ein, um Entwickler mit einer neuen Umgebung vertraut zu machen. Als Seiteneffekt entstehen bei diesem Vorgehen Unit-Tests, die das Gelernte in bewusstes Wissen überführen.

Diese Technik lässt sich analog auf die Anwendungsdomäne anwenden (s. Abb. 2). Wenn Teams bei neuen Funktionen Akzeptanztests festhalten und sie automatisieren, schaffen sie den Übergang von bewusstem Unwissen zu bewusstem Wissen. Die automatisierten Akzeptanztests reflektieren nach der Implementierung der Funktion das, was die Mitarbeiter bei der Umsetzung gelernt haben. Das ermöglicht eine gezielte Transformation von Unkenntnis zu einer manifestierten Wissensbasis.

Damit das gelingt, müssen entsprechende Unit- oder Akzeptanztests aussagekräftig geschrieben sein. Wenn sich jemand Neues mit der Funktion in der Zukunft auseinandersetzt, sollten die Entwickler die aktuelle Lernkurve nachvollziehen können. Unit-Tests müssen daher fokussiert und überschaubar sein. Die gleichen Prinzipien gelten für automatisierte Akzeptanztests. Wenn sie geskriptet sind, dokumentieren sie nur auf schwer überschaubare Art und Weise das darunter liegende Businessproblem und dessen Lösung. Deswegen sind für akzeptanztestgetriebene Entwicklung (engl.: Acceptance Test-driven Development, ATDD) ebenfalls aussagekräftige Tests ein wichtiger Bestandteil. Teams sollen sich nicht lange erarbeiten, was sie sich bei einer Umsetzungsentscheidung gedacht haben, sondern das im Idealfall in drei prägnanten Zeilen bereits ablesen können.

TDD sowie ATDD können wichtige Entscheidungen gezielt von bewusstem Unwissen zu bewusstem Wissen überführen. Bei TDD halten die Mitarbeiter Designentscheidungen für die Zukunft fest. Bei ATDD erhalten sie langfristig eine Wissensbasis, die sie beispielsweise bei neuen Entwicklungen heranziehen können, um vorab Seiteneffekte erkennen zu können.

Unbewusstes Unwissen

Der Umgang mit unbewusstem Unwissen ist schwieriger. Es umfasst die Dinge, von denen Menschen noch nicht mal wissen, dass sie über etwas keine Kenntnis haben. Dazu gehören beispielsweise überraschende Effekte im Reiseverkehr, etwa wenn ein Vulkan Asche in die Atmosphäre pustet und deshalb keine Flugzeuge mehr starten oder landen dürfen.

Solche Effekte existieren ebenso in der Softwareentwicklung. Der Autor war beispielsweise vor einigen Jahren für den Support eines Systems eingeteilt, das Mobilfunktelefonate in Echtzeit abrechnete. Als er abends einen Anruf bekam, schaute er sich die Log-Informationen des Systems an und fand mehrere Fehler. Es dauerte über einen Tag, bis er die Probleme des Systems erkannt hatte: Es handelte es sich um eine Neuinstallation eines Systems, auf dem 100 Mobilfunkteilnehmer testweise aktiv geschaltet waren. Natürlich hatte die Installation einen User Acceptance Test (UAT) im Vorfeld hinter sich. Trotzdem war es faktisch für keinen der Teilnehmer möglich, ein Telefonat zu tätigen. Innerhalb von 24 Stunden hatte der Autor etwa 20 kritische Fehler aus den Log-Dateien identifiziert. Auf keinem der Testsysteme traten die Fehler zuvor auf. Viele davon ließen sich darauf zurückführen, dass unterschiedliche Subkomponenten in Produktion in Grenzfällen nicht miteinander sprechen konnten. Selbstverständlich gab es im Vorfeld umfangreiche Testmaßnahmen, von denen aber keine auf die Probleme hinwies. Erst im Produktionssystem war das Team in der Lage, das unerwünschte Verhalten zu beobachten.

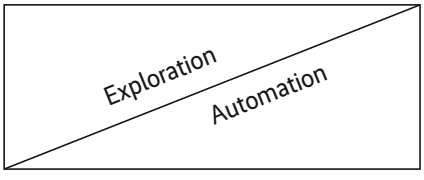
Risiken einschätzen

Unbewusstes Unwissen steht oft im Fokus des traditionellen Risikomanagements: Man versucht, Projektrisiken vorab einzuschätzen, um sie bekämpfen zu können. Das soll Unwissen bewusst machen. In Bezug auf das Testen von Software kann ein risikobasierter Ansatz dabei helfen, unbekanntes Unwissen zu ergründen. Ein agiles Team mit seinen kurzen Iterationen will möglichst früh Gefahren erkennen. Dabei kann ein Ansatz über risikobasierte Tests helfen, unbekanntes Unwissen aufzudecken.

Exploratives Testen kann in beiden Fällen dafür genutzt werden, sich mit der Applikation vertraut zu machen und so vorhandenes Unwissen zu entdecken. Dabei arbeiten Teams in kurzen Zeitabschnitten von beispielsweise 60 Minuten unter einem Thema mit der Applikation. Das auch Testcharter genannte Thema [2] hilft dabei, die Aktivitäten in der kurzen Zeit zu fokussieren. Wenn etwas Unvorhergesehenes eintritt, können Tester entscheiden, ob es zur Testcharter passt und sie das Problem weiterverfolgen oder ob sie sich eine kurze Notiz machen und sich in einer zukünftigen Session damit auseinandersetzen. Nach Ablauf der 60 Minuten geht der Tester seine Notizen durch und identifiziert neue Themen, auf die er noch ein Auge werfen möchte.

Das große Zusammenspiel

Bewusstes Unwissen lässt sich mithilfe testgetriebener Entwicklung in bewusstes Wissen überführen. Auf neues Unwissen stoßen Teams dabei, indem sie in regelmäßigen Abständen exploratives Testen verwenden, um all die Dinge, die sie bislang noch nicht antizipiert haben, entdecken zu können. Bei jedem neuen Problem, das sie mit explorativem Testen entdecken, können sie sich überlegen, ob sie den Aufwand in die Automatisierung eines Tests stecken, damit sie das Know-how zukünftig präsent haben werden. Damit schaffen sie den



Zeit für Automatisierung kann nicht mehr in das Erforschen von Risiken gesteckt werden; Zeit für manuelles Testen kann nicht mehr für die Automatisierung genutzt werden (Abb. 3).

Weg von unbewusstem Unwissen über bewusstes Unwissen hin zu bewusstem Wissen. Die Feedbackschleife von explorativem Testen zu automatisierten Tests führt dazu, dass sie neue Risiken in regelmäßigen Abständen entdecken und mitigieren können.

Dabei muss das Team allerdings auch den Aufwand für die jeweiligen Aktivitäten im Auge behalten. Die Zeit, die es in die Automatisierung eines Tests steckt, fehlt eventuell für das Adressieren von Risiken durch explorative Tests. Andersherum gilt: Jeder manuelle Test nimmt dem Team die Zeit, einen Test zu automatisieren (s. Abb. 3).

Fazit

Die Frage nach dem richtiges Maß an Automatisierung und Exploration lässt sich nicht kontextfrei beantworten: Hat ein Team viele Risiken in Form von Altlasten im Code beziehungsweise in Form technischer Schulden aufgebaut und kann deshalb nur schwer automatisiert testen, wird es vermutlich mehr manuellen Testaufwand einplanen. Wenn die Applikation mit einer hohen Codeabdeckung testgetrieben auf Unit- und Akzeptanztestebene entwickelt wurde, wird das Team mehr Aufwand in die Pflege der vorhandenen Testbasis stecken und dadurch weniger Zeit für manuelle Tätigkeiten haben. Das gesunde Maß hängt auch davon ab, wie gut das Team die Anwendungsdomäne und die eingesetzten Techniken kennt. Daher bestimmt der jeweilige Projektkontext die richtige Mischung aus automatisierten und rein manuellen Tests. In der Agilität setzt man dazu bewusst auf regelmäßige Retrospektiven, in denen eine ungesunde Balance zwischen den beiden transparent werden kann. Das Team kann dadurch gezielt sein Vorgehen auf den Projektkontext anpassen.

Darüber hinaus sollten sich alle Mitarbeiter kontinuierlich weiterentwickeln. Coding- und Testing-Dojos helfen dabei, Fähigkeiten rund um testgetriebene Entwicklung, emergentes Design und Architektur sowie exploratives Testen zu verbessern, damit alle Beteiligten sie in ihren Projekten effizient anwenden können. (rme)

Literatur

[1] Nassim Nicholas Taleb; The Black Swan – The Impact of the Highly Improbable; Random House 2010

[2] Elisabeth Hendrickson; Explore It! – Reduce Risk and Increase Confidence with Exploratory Testing; O'Reilly 2013

Markus Gärtner

arbeitet als Organizational Design Consultant, Certified Scrum Trainer (CST) und Agile Coach für die it-agile GmbH in Hamburg. Er schrieb unter anderem ATDD by Example – A Practical Guide to Acceptance Test-driven Development und erhielt den Most Influential Agile Testing Professional Person Award in 2013.

Alle Links: www.ix.de/ix1614012

Risiken beim Bau von Anwendungen und Lösungsansätze für die Sicherheit

Achtung: Baustelle in Gefahr

Andreas Krüger

Unternehmen härten Anwendungen, um zu verhindern, dass ein Angreifer Schadfunktionalität einschmuggeln kann. Sie schützen ihre Produktionsumgebungen und schotten sie ab. Aber Angreifer können versuchen, „über Bande zu spielen" und in eine Anwendung einzubrechen, indem sie die Build-Umgebung manipulieren.

In einem Projekt läuft ein Build auf einen Fehler. Die Details der Fehlermeldung zeigen auf eine bestimmte Datei, eine JavaScript-Bibliothek. Nur, dass diese Datei keinen JavaScript-Code enthält, sondern stattdessen HTML, mit der Botschaft: „Diese Domain können Sie kaufen." Der Autor der Bibliothek hatte augenscheinlich seine Domainregistrierung nicht verlängert, sodass sie an einen Grabber gefallen war, der nicht mehr das ursprüngliche JavaScript, sondern stattdessen seine Werbung als HTML auslieferte.

An dieser Anekdote ist leider nichts erfunden; sie ist dem Autor genau so passiert. Es handelte sich um ein großes Enterprise-Projekt, in dem die Anwendungssicherheit kontinuierlich thematisiert wurde und verschiedene Reviews und Prozesse vorgeschrieben waren, um sie zu stützen. Die Build-Umgebung hatte man dabei jedoch weitgehend außen vor gelassen.

Zu Unrecht: Denn bei entsprechende krimineller Energie hätte auch Schlimmeres in der veränderten Datei stecken können als nur Werbung. So gesehen ist das Projekt haarscharf am Albtraum eines erfolgreichen Angriffs vorbei geschrammt.

Die Spitze des Eisbergs

Die Projektbeteiligten hatten sich dabei keineswegs außergewöhnlich leichtsinnig verhalten, sondern nur gehandelt, wie es heute üblich ist: Sicherheitsanalysen konzentrieren sich auf den selbstentwickelten Anwendungscode und die Infrastruktur der Produktionsumgebung. Mit dem eigenen Anwendungscode überprüft man aber nur einen kleinen Bruchteil dessen, was schlussendlich in der Produktionsumgebung läuft. Eine beispielhafte Zählung der Codezeilen einer gerade verfügbaren produktiven Ruby-on-Rails-Anwendung ergab die in Abbildung 1 dargestellten Ergebnisse.

Das passt ins allgemeine Bild: Softwareentwickler verdanken die eigenen Erfolge auch der immer leistungsfähigeren vorge-