JANUAR 2014

TECHNOLOGY RADAR

Zusammengestellt vom ThoughtWorks Technology Advisory Board

thoughtworks.com/radar

ThoughtWorks®

AKTUELLES

Folgende Trends werden in dieser Ausgabe vorgestellt:

- **Frühwarnsysteme und Wiederherstellung im Produktionsbetrieb** – Auf dem Markt sehen wir vermehrt neue Tools und Methoden zum Loggen, Monitoring, Archivieren und Abfragen von Daten aus dem Produktivbetrieb. In Kombination mit kurzen Wiederherstellungszeiten dank Virtualisierung und automatisierter Infrastruktur können Unternehmen den Testaufwand vor dem Deployment reduzieren. Unter Umständen können die Tests sogar in die Produktion verlegt werden.

- **Privatsphäre vs. Big Data** – Einerseits sehen wir die enormen Möglichkeiten, die umfangreiche Datensammlungen sowie neue Tools und Plattformen zur Datenarchivierung und -analyse für neue geschäftliche Erkenntnisse eröffnen, andererseits bereitet es uns Sorge, dass zahlreiche Unternehmen unnötigerweise große Mengen persönlicher Daten speichern. Deshalb setzen wir uns für das Prinzip der Datensparsamkeit ein und fordern Unternehmen auf, nur so viele Daten über ihre Kunden zu speichern wie unbedingt notwendig.

- **JavaScript weiter auf dem Vormarsch** – Das Ökosystem rund um JavaScript als ernstzunehmende Anwendungsplattform entwickelt sich erfolgreich weiter. In letzter Zeit kamen zahlreiche interessante Tools für Test, Build und Dependency Management für JavaScript Applikationen auf den Markt, sowohl auf Server- als auch auf Client-Seite.

- **Digital in der realen Welt** – Kostengünstige Geräte, offene Hardware-Plattformen und neue Kommunikationsprotokolle sorgen dafür, dass die Interaktion mit Computern nicht mehr nur an Bildschirmen stattfindet, sondern immer mehr in der realen Welt. Ein gutes Beispiel dafür ist die Verbreitung von tragbaren Geräten, die biometrische Daten des Trägers erfassen, und die Integration dieser Geräte mit Mobiltelefonen und Tablets.

Wir von ThoughtWorks lieben Technik. Wir entwickeln, erforschen und testen Technologien. Wir stellen sie als Open Source zur Verfügung, wir schreiben über sie und versuchen ständig, sie immer weiter zu verbessern – und zwar für jeden. Wir wollen als Vorreiter auf dem Gebiet der Softwareentwicklung vorangehen und neue Wege in der IT beschreiten. In unserem ThoughtWorks Technologie Radar wollen wir dieses Ziel fördern und mit anderen teilen. Erstellt wird der Radar vom ThoughtWorks Technical Advisory Board, das aus erfahrenen Technologieexperten besteht. In regelmäßigen Treffen sprechen sie über die globale Technologiestrategie von ThoughtWorks sowie die technischen Trends, die unsere Branche bewegen.

ThoughtWorks ist ein Technologie Unternehmen – Wir entwickeln, erforschen und testen Technologien. Wir stellen sie als Open Source Software zur Verfügung, wir schreiben über sie und versuchen ständig, Verbesserungen zu erreichen – und zwar für jeden. Wir wollen als Vorreiter auf dem Gebiet der Softwareentwicklung neue Wege in der IT beschreiten. Der Technologie Radar ist ein wesentlicher Bestandteil davon. Erstellt wird der Radar vom ThoughtWorks Technology Advisory Board, das aus erfahrenen Technologieexperten besteht. In regelmäßigen Treffen sprechen sie über die globale Technologiestrategie von ThoughtWorks sowie die technischen Trends, die unsere Branche bewegen.

In diesem Radar stellen wir zusammenfassend die Ergebnisse der Diskussionen des Technology Advisory Boards vor – in einem Format, das für unterschiedlichste Interessengruppen attraktiv ist, vom CIO bis hin zum Softwareentwickler. Wir möchten Sie mit dem Text dazu anregen, sich ausführlicher über die vorgestellten Technologien zu informieren. Der Radar ist sehr grafisch angelegt und ist unterteilt in Techniken, Werkzeuge, Plattformen, Sprachen & Frameworks. Themen, die für mehrere Quadranten relevant sind, werden in den zutreffendsten eingeordnet. Außerdem gruppieren wir die Einträge in vier Ringe, um unsere aktuelle Einschätzung darzustellen. Die Ringe sind:

- **Adopt:** (Einführen): Wir sind überzeugt, dass diese Technologien in der Branche eingeführt werden sollten. Wir nutzen sie, wenn sie für unsere Projekte relevant sind.

- **Trial:** (Ausprobieren): Es lohnt sich, den Einsatz dieser Technologien im Unternehmen auszuprobieren. Dies sollte nur bei Projekten geschehen, die ein gewisses Risiko vertragen.

- **Assess:** (Evaluieren): Es lohnt sich, die Möglichkeiten auszuloten. Wir halten es für wichtig zu verstehen, inwiefern sie Auswirkungen auf Ihr Unternehmen haben könnten.

- **Hold:** (Vorsicht): Gehen Sie achtsam vor.

AKTUELLES

Neue Einträge, die seit dem letzten Radar umfangreiche Änderungen erfahren haben, sind mit einem Dreieck gekennzeichnet. Einträge ohne Veränderung sind als Kreise dargestellt. Die detaillierten Abbildungen für die einzelnen Quadranten zeigen die entsprechenden Veränderungen. Generell interessiert uns natürlich viel mehr, als wir in einem Dokument dieser Größe erfassen können. Deshalb müssen wir leider viele Einträge aus dem letzten Radar ausblenden, um Platz für neue zu schaffen. Das heißt jedoch nicht, dass wir uns mit den ausgeblendeten Einträgen nicht mehr befassen.

Weitere Informationen über diesen Radar erhalten Sie unter **http://martinfowler.com/articles/radar-faq.html**

Ein Wort zur deutschen Übersetzung: Der Radar ist ein sehr technisches, schwer zu übersetzendes Dokument. Wir sehen diese Ausgabe als einen ersten Versuch an und freuen uns über Ihr Feedback!

AUTOREN - Zum ThoughtWorks Technology Advisory Board gehören:

Rebecca Parsons *(CTO)*
Martin Fowler *(Chief Scientist)*
Badri Janakiraman
Brain Leke
Claudia Melo
Darren Smith
Erik Dörnenburg
Evan Bottcher
Hao Xu
Ian Cartwright
James Lewis
Jeff Norris
Jonny LeRoy
Mike Mason
Neal Ford
Rachel Laycock
Sam Newman
Scott Shaw
Srihari Srinivasan
Thiyagu Palanisamy

(Die folgende Abbildung wird entsprechend der Radarbewegungen für die entsprechende Veröffentlichung aktualisiert)

THE RADAR

METHODEN

ADOPT
1 Erfassung von JavaScript-Fehlern auf Client-Seite
2 Continuous Delivery für Mobilgeräte
3 Mobiltesting für Mobilfunknetze
4 Separierte DOM plus Node für JS Testing
5 Windows-Infrastruktur-Automatisierung

TRIAL
6 Domain-Events explizit erfassen
7 Client- und Serverrendering mit demselben Code
8 HTML5-Speicher anstelle von Cookies
9 Instrument all the Things
10 Masterless Chef/Puppet
11 Microservices
12 Perimeterloses Unternehmen
13 Bereitstellung von Test-Tools
14 Strukturiertes Logging

ASSESS
15 Die Kluft zwischen physischer und digitaler Welt mit einfacher Hardware überbrücken
16 Gemeinschaftliche Analysen und Datenwissenschaft
17 Datensparsamkeit
18 Entwicklung von Umgebungen in der Cloud
19 Fokus auf Mean Time to Recovery
20 Maschinenbild als Artefakt
21 Tangible Interaction

HOLD
22 Cloud Lift & Shift
23 Ignorieren der OWASP Top 10
24 Silo Metrics
25 Schnelligkeit als Maß für Produktivität

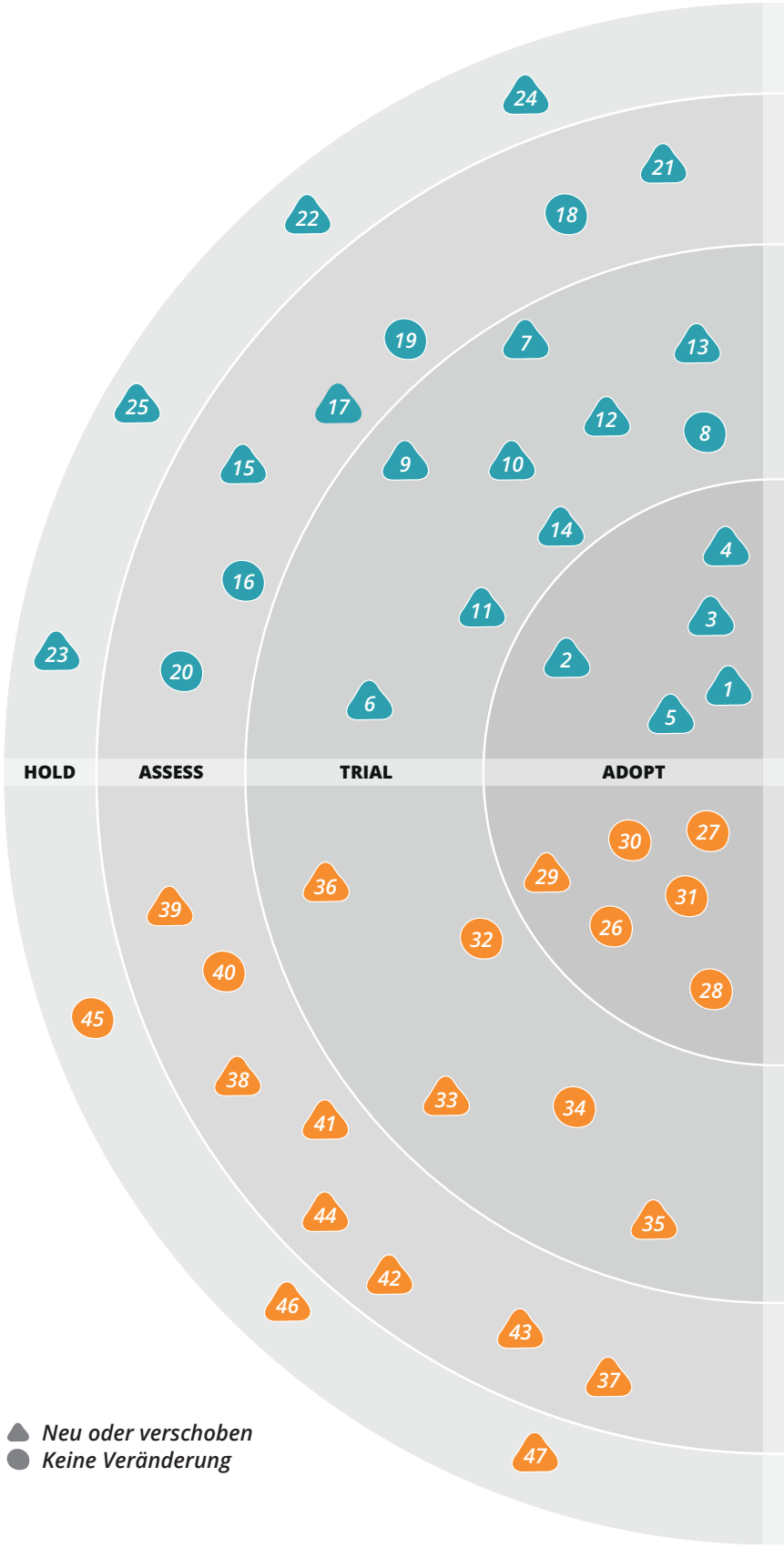
PLATTFORMEN

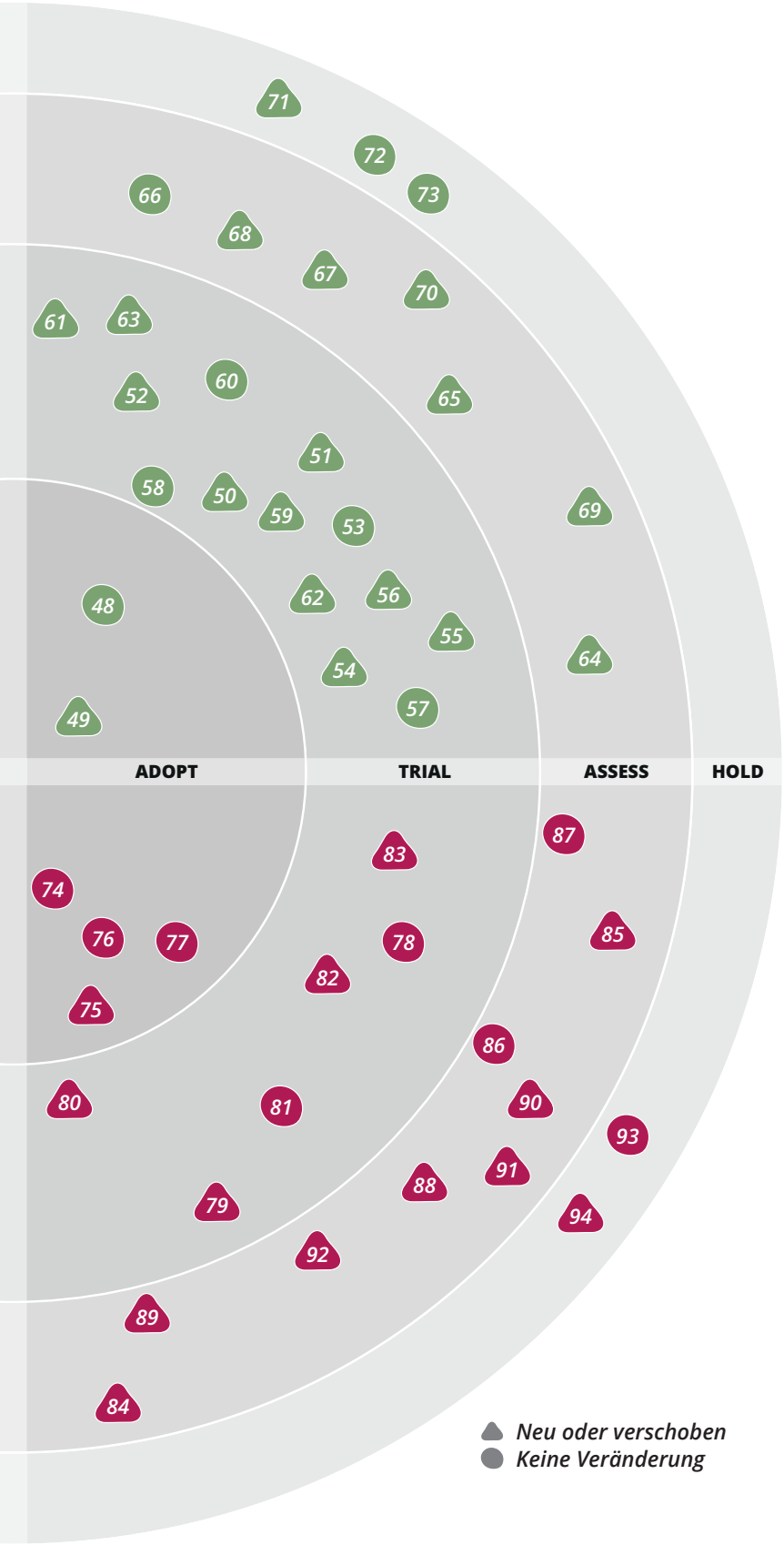
ADOPT
26 Elastic Search
27 MongoDB
28 Neo4J
29 Node.js
30 Redis
31 SMS and USSD as a UI

TRIAL
32 Hadoop 2.0
33 Hadoop as a service
34 OpenStack
35 PostgreSQL for NoSQL
36 Vumi

ASSESS
37 Akka
38 Backend as a service
39 Low-cost robotics
40 PhoneGap/Apache Cordova
41 Private Clouds
42 SPDY
43 Storm
44 Web Components standard

HOLD
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse





THE RADAR

TOOLS

ADOPT
48 D3
49 Dependency management for JavaScript

TRIAL
50 Ansible
51 Calabash
52 Chaos Monkey
53 Gatling
54 Grunt.js
55 Hystrix
56 Icon fonts
57 Librarian-puppet and Librarian-Chef
58 Logstash & Graylog2
59 Moco
60 PhantomJS
61 Prototype On Paper
62 SnapCI
63 Snowplow Analytics & Piwik

ASSESS
64 Cloud-init
65 Docker
66 Octopus
67 Sensu
68 Travis for OSX/iOS
69 Visual regression testing tools
70 Xamarin

HOLD
71 Ant
72 Heavyweight test tools
73 TFS

SPRACHEN & FRAMEWORKS

ADOPT
74 Clojure
75 Dropwizard
76 Scala, the good parts
77 Sinatra

TRIAL
78 CoffeeScript
79 Go language
80 Hive
81 Play Framework 2
82 Reactive Extensions across languages
83 Web API

ASSESS
84 Elixir
85 Julia
86 Nancy
87 OWIN
88 Pester
89 Pointer Events
90 Python 3
91 TypeScript
92 Yeoman

HOLD
93 Handwritten CSS
94 JSF

METHODEN

Durch die Erfassung von JavaScript-Fehlern auf Client-Seite konnten unsere Delivery-Teams Probleme identifizieren, die spezifisch für bestimmte Browser oder Plugin-Konfigurationen sind und die User Experience beeinträchtigen. Im Laufe des letzten Jahres haben mehrere Dienstleister versucht, diesen Bedarf zu decken. Diese Fehler werden dabei nicht mehr direkt durch die Anwendungen gespeichert, sondern an Webanalysetools oder bestehende Monitoring-Tools wie New Relic übertragen. Dadurch werden auch die Speicheranforderungen verringert.

Seit dem letzten Radar gab es einige Fortschritte, die **Continuous Delivery** für native Apps auf **Mobilgeräten** nun weniger problematisch machen. Das kürzlich als Open Source auf den Markt gekommene Xctool, ein „besseres Xcodebuild", verbessert Build Automatisierung und Unit Testing für iOS. Seit Einführung automatischer Updates mit iOS7 gibt es auch weniger Probleme bei regelmäßigen Releases. Travis-CI unterstützt jetzt auch OS X Agents und beseitigt damit eine weitere Hürde für nahtlos integrierte CD-Pipelines für mobile Plattformen. Wir empfehlen nach wie vor hybride Lösungen und möchten noch einmal die Wichtigkeit der Testautomatisierung für Mobile Apps unterstreichen.

Da JavaScript-Anwendungen auf Client-Seite immer anspruchsvoller werden, sehen wir eine steigende Notwendigkeit technischer Verfeinerung. Ein häufiger Fehler in der Architektur ist der ungehinderte Zugriff auf das DOM aus der gesamten Codebase, wodurch DOM-Manipulationen mit Applikationslogik und AJAX Calls vermischt werden. Dadurch werden das Verständnis des Codes und dessen Weiterentwicklung erschwert. Wir empfehlen eine strikte Trennung zwischen den verschiedenen Anliegen. Dazu gehört auch eine konsequente Konzentration des gesamten DOM-Zugriffs (was sich im Allgemeinen auch auf die gesamte jQuery Nutzung bezieht) in der Architektur. Ein positiver Nebeneffekt dieses Ansatzes ist, dass alles, was nicht in dieser **separierten DOM**-Ebene implementiert ist, außerhalb des Browsers mit einer schlanken JavaScript Engine wie zum Beispiel **node.js** getestet werden kann, was die Gesamtlaufzeit aller Tests reduziert.

Beim Einsatz von Methoden wie „Instrument all the Things" und Semantic Logging kann es sich als sehr nützlich erweisen, **Domain-Events explizit zu erfassen**. Indem man diese Veränderungen direkt modelliert, ist es nicht länger notwendig, Rückschlüsse auf Nutzerabsichten aus Statusveränderungen zu ziehen. Ein Weg dahin ist die Nutzung einer ereignisbasierten Architektur, in der die Applikations-Events als für das Unternehmen wichtige Ereignisse definiert sind.

HTML wird zunehmend nicht mehr nur auf dem Server, sondern zusätzlich auch auf dem Client, im Webbrowser, gerendert. In vielen Fällen wird dieses verteilte Rendering nach wie vor notwendig sein. Mit zunehmender Reife der JavaScript Templating Libraries wird dabei ein interessanter Ansatz möglich: **Client- und Serverrendering mit ein und demselben Code**.

Um auf wichtige geschäftliche Ereignisse zu reagieren, muss man sie protokollieren und beobachten. Das Prinzip **„Instrument all the Things"** regt bereits zu Beginn der Softwareentwicklung zum Nachdenken darüber an, wie dies erreicht werden kann. Dadurch können wesentliche Metriken erfasst und beobachtet, durch entsprechende Reports die operative Effizienz verbessert werden.

Chef und Puppet Server dienen als zentrale Stelle zur Speicherung von Receipts bzw. Manifesten, mit deren Hilfe Konfigurationsänderungen auf den verwalteten Rechnern

ADOPT
1 Erfassung von JavaScript-Fehlern auf Client-Seite
2 Continuous Delivery für Mobilgeräte
3 Mobiltesting für Mobilfunknetze
4 Separierte DOM plus Node für JS Testing
5 Windows-Infrastruktur Automatisierung

TRIAL
6 Domain-Events explizit erfassen
7 Client- und Serverrendering mit demselben Code
8 HTML5-Speicher anstelle von Cookies
9 Instrument all the Things
10 Masterless Chef/Puppet
11 Microservices
12 Perimeterloses Unternehmen
13 Bereitstellung von Test-Tools
14 Strukturiertes Logging

ASSESS
15 Die Kluft zwischen physischer und digitaler Welt mit einfacher Hardware überbrücken
16 Gemeinschaftliche Analysen und Datenwissenschaft
17 Datensparsamkeit
18 Entwicklung von Umgebungen in der Cloud
19 Fokus auf Mean Time to Recovery
20 Maschinenbild als Artefakt
21 Tangible Interaction

HOLD
22 Cloud Lift & Shift
23 Ignorieren der OWASP Top 10
24 Silo Metrics
25 Schnelligkeit als Maß für Produktivität

METHODEN

durchgeführt werden. Außerdem dienen sie als zentrale Datenbank für entsprechende Informationen und bieten die Zugriffskontrolle für Receipts und Manifeste. Ein Nachteil dieser Server ist, dass sie zum Engpass werden, vor allem auch, wenn sich viele Clients gleichzeitig mit ihnen verbinden. Sie stellen einen „Single Point of Failure" dar, und es erfordert einen gewissen Aufwand, sie stabil und zuverlässig laufen zu lassen. Deswegen empfehlen wir **Chef-Solo bzw. einzelne Puppet Server** zusammen mit einem Versionsverwaltungssystem, wenn der Server in erster Linie zur Speicherung von Receipts bzw. Manifesten eingesetzt wird. Teams können natürlich zu einem späteren Zeitpunkt nach Bedarf die entsprechenden Server einführen, anstatt in diesen Fällen neue Lösungen für Probleme zu finden, die die Server bereits gelöst haben.

Wir sind oftmals nicht mehr durch die Beschaffung und Provisionierung von Hardware beschränkt. Angesichts der stark erhöhten Flexibilität, die uns dadurch ermöglicht wird, zeigt sich jedoch, dass wir nun Beschränkungen durch die Größe und Komplexität der Software-Assets unterworfen sind, die wir zur Verwaltung unserer virtuellen Serverfarmen einsetzen. Durch die Nutzung von Technologien, die bekannter in der Welt der Softwareentwickler sind, wie zum Beispiel TDD, BDD und CI, entsteht ein Ansatz zum Umgang mit dieser Komplexität. Dieser gibt uns die Sicherheit, Veränderungen an unserer Infrastruktur wiederholbar und automatisiert vorzunehmen. **Provisionierungstest-Tools** wie rspec-puppet, Test Kitchen und serverspec sind für die meisten Plattformen verfügbar.

Durch die Behandlung von Logs als Daten erhalten wir bessere Einblicke in die operativen Aktivitäten der von uns gebauten Systeme. **Strukturiertes Logging** mit einem einheitlichen, klar definierten Format mit semantischen Informationen baut auf dieser Methode auf und ermöglicht es mit Tools wie Greylog2 und Splunk, bessere Einblicke zu gewinnen.

Hardware ist in letzter Zeit nicht nur kostengünstiger, sondern auch kleiner, einfacher und energiesparender geworden. Dies hat zu einer mengenmäßigen Explosion an Geräten geführt, die physikalische Domains für Software öffnen. Oftmals bestehen diese Geräte aus nicht viel mehr als einem Sensor und einer Kommunikationskomponente wie Bluetooth Low Energy oder WLAN. So müssen auch wir als Softwareingenieure unseren Horizont erweitern und uns immer vor Augen halten, dass **die Trennung zwischen der digitalen und realen Welt mit einfacher Hardware überbrückt werden kann**. Wir sehen dies in modernen Autos, Wohnungen und Häusern ebenso wie in Geräten am menschlichen Körper, in der Landwirtschaft und anderen Einsatzgebieten. Zum Glück wird immer weniger Zeit benötigt, um Prototypen solcher Geräte zu entwickeln. Somit können die kurzen, aus der Softwarewelt bekannten Durchlaufzeiten auch hier erreicht werden.

Es ist einfach in Versuchung zu geraten, so viele Daten wie möglich aufzuzeichnen. Schließlich wollen wir uns an schnell veränderliche Geschäftsmodelle anpassen, aus der Vergangenheit lernen und jedem Besucher ein optimales Erlebnis bieten. Auf der anderen Seite waren Hacker noch nie so angriffslustig wie heute; eine spektakuläre Sicherheitsverletzung jagt die andere. Zudem sind noch nie dagewesene Massenüberwachungen durch Regierungsbehörden bekannt geworden. Der Begriff **Datensparsamkeit** wurde von der deutschen Gesetzgebung zum Schutz der Privatsphäre geprägt. Dahinter steckt die Idee, nur so viele persönliche Informationen zu speichern, wie für das Geschäft unbedingt notwendig sind. Hierfür ein paar Beispiele: Anstatt in Zugriffsprotokollen die gesamte IP-Adresse eines Kunden zu speichern, genügt es, die ersten zwei oder drei Bytes zu erfassen. Bei der Aufzeichnung von Klickstrecken kann anstelle eines Nutzernamens auch ein anonymes Token verwendet werden. Für Informationen, die nicht gespeichert werden, muss man sich keine Sorgen machen, dass jemand diese stehlen könnte.

Die Grenzen zwischen Hard- und Software verschwimmen immer mehr. Das, was früher nur Computer leisten konnten, versteckt sich immer mehr auch in Alltagsgegenständen. Zwar sind die vernetzten Geräte allgegenwärtig – im Handel, im Auto, zu Hause und am Arbeitsplatz. Dennoch haben wir bisher nicht verstanden, wie wir all das zu einer sinnvollen Erfahrung bündeln können, die über den Computerbildschirm hinausgeht. **„Tangible Interaction" – Spürbare Interaktionen**, so nennt sich eine neue Disziplin, bei der Software und Hardware in Bezug auf Technologie, Architektur, User Experience und Design miteinander verschmelzen. Ziel ist die Erschaffung natürlicher Umgebungen, bestehend aus realen Objekten, in denen Menschen digitale Daten bearbeiten und verstehen können.

Mit wachsender Beliebtheit und Akzeptanz der Cloud geht jedoch auch der Trend einher, die Cloud wie einen Hosting-Provider zu behandeln. Dieser Trend, auch **„Cloud Lift & Shift"** genannt, wird leider von großen Anbietern noch verstärkt, die einfach bestehende Hosting-Angebote als „Cloud" neu vermarkten. Nur wenige dieser Angebote bieten echte Flexibilität oder werden wirklich nur entsprechend der Nutzung abgerechnet. Wer denkt, ohne Neuausrichtung der Architektur auf Cloud-Technologie umstellen zu können, liegt wahrscheinlich falsch.

METHODEN

Kaum eine Woche vergeht, in der die IT-Branche nicht von einem neuen Skandal erschüttert wird: massive Datenverluste, geknackte Passwörter oder das Eindringen in ein vermeintlich sicheres System. Es gibt zahlreiche zuverlässige Ressourcen, die dafür sorgen, dass die Sicherheitsaspekte bei der Softwareentwicklung entsprechend ernst genommen werden. Diese dürfen wir nicht weiter ignorieren. Die **OWASP Top 10** ist hierfür ein guter Anfang.

Seitdem sich immer mehr Unternehmen in Richtung online bewegen, beobachten wir eine Tendenz dazu, **Metriken** in Isolation zu verwalten. Mit Hilfe spezieller Tools werden bestimmte Daten erfasst und dargestellt: Ein Tool erfasst Seitenaufrufe und Browserverhalten, ein anderes betriebliche Daten und wieder ein anderes führt die Protokollangaben zusammen. So entstehen Datensilos und damit die Notwendigkeit, zwischen den einzelnen Tools hin- und herzuschalten, um die für das Geschäft notwendigen Daten zu bekommen. Durch diesen toolbedingten Bruch in der Analysedomain wird die Entscheidungsfähigkeit des Teams beeinträchtigt. Eine weitaus bessere Lösung ist die konsolidierte Betrachtung von Analysedaten in nahezu Echtzeit mit Hilfe integrierter Dashboards, die zeitnahe Domain- und teamrelevante Informationen anzeigen.

Von allen Ansätzen, die wir für problematisch halten, ist die Gleichsetzung von Velocity (im agilen Sinne) mit Produktivität so verbreitet, dass wir diese auf unsere „Hold“-Liste gesetzt haben. Bei richtiger Anwendung kann man mit Velocity die Daten aus vergangenen Iterationen in den Planungsprozess einbeziehen. Velocity ist nichts weiter als eine Kapazitätsschätzung eines bestimmten Teams zu einer bestimmten Zeit. Sie kann besser werden, je mehr das Team zusammenwächst oder je besser man Probleme wie technische Schulden oder einen unzuverlässigen Build-Server im Griff hat. Doch wie alle Informationen können auch diese Daten missbraucht werden. So bestehen einige übereifrige Manager zum Beispiel auf eine ständige Verbesserung der Velocity. Wenn man **die Velocity mit Produktivität** gleichsetzt, fördert man unproduktives Teamverhalten, das zwar die Zahlen schönt, dafür aber die tatsächlich fertiggestellte, funktionierende Software vernachlässigt.

TOOLS

Tools zur **Verwaltung von Abhängigkeiten für JavaScript** haben unsere Entwicklungsteams dabei unterstützt, große Mengen an JavaScript zu verwalten. Der Code wurde dabei strukturiert und die Abhängigkeiten zur Laufzeit geladen. Obwohl dadurch der Aufwand in den meisten Fällen verringert werden konnte, verkompliziert Lazy Loading die Unterstützung des Offline-Modus. Die einzelnen Tools zur Verwaltung von Abhängigkeiten haben jeweils unterschiedliche Stärken – treffen Sie Ihre Wahl also auf Grundlage Ihrer entsprechenden Aufgaben.

Im Bereich der DevOps Orchestrierungs-Engines stößt **Ansible** bei ThoughtWorks-Projekten fast ausnahmslos auf Zustimmung. Die Plattform verfügt über nützliche Tools und Abstraktionen bei einer gleichzeitig sinnvollen Granularität.

Bei Mobile-Projekten haben uns die Funktionalität und die ständig weiterentwickelten Funktionen sowie die Reife von **Calabash** beeindruckt. Hierbei handelt es sich um ein Tool für automatisierte Akzeptanzprüfungen für Android- und iOS-Anwendungen, das zudem übergreifende Tools wie Cucumber unterstützt. Calabash ist eine gute Wahl für heterogene Projekte.

Nach unserer Empfehlung im letzten Radar, den Fokus auf die Reduzierung der Mean Time to Recovery zu richten, möchten wir dieses Mal **Chaos Monkey** aus der Simian Army Suite von Netflix ins Rampenlicht rücken. Das Tool deaktiviert beim normalen Betrieb zufällig ausgewählte Instanzen in der Produktionsumgebung. Bei einem gleichzeitigen, umfassenden Monitoring und einem Team im Bereitschaftseinsatz können dadurch unerwartete Systemschwächen aufgedeckt werden. Dies wiederum ermöglicht es dem Entwicklerteam, frühzeitig automatisierte Recovery-Mechanismen zu entwickeln, anstatt auf überraschende Ausfälle reagieren zu müssen.

Mehrere Teams von ThoughtWorks, die sich mit der Entwicklung von Apps in Node.js beschäftigen, greifen auf **Grunt** zurück, um Entwicklungsaktivitäten wie Minifizierung, Kompilierung und Linting zu automatisieren. Viele häufige Tasks sind auch als Grunt-Plugins verfügbar. Falls erforderlich, können die Konfigurationen programmatisch erstellt werden.

Die Verwaltung der unzähligen Abhängigkeiten in einem verteilten System ist eine komplexe Angelegenheit, die vor dem Hintergrund immer feinerer Microservices zu einem zunehmenden Problem wird. **Hystrix** ist eine Bibliothek von Netflix für die JVM, die Muster für die Handhabung von nachgelagerten Fehlern implementiert, Echtzeit-Monitoring von Verbindungen bietet sowie über Caching- und Batching-Mechanismen verfügt, mit denen serviceübergreifende Abhängigkeiten effizienter gestaltet werden können. In Kombination mit Hystrix Dashboard und Turbine kann dieses Tool für die Entwicklung widerstandsfähigerer Systeme verwendet werden und liefert zudem Daten zu Durchsatz, Latenz und Fehlertoleranz nahezu in Echtzeit.

Die Prüfung von Microservices auf HTTP-Basis kann langwierig und kompliziert sein. Das gilt insbesondere für zwei Szenarien: das Konsumieren von Microservices durch Front-Ends sowie die Kommunikation zwischen Microservices. Zu diesem Zweck kann sich **Moco** als ziemlich praktisch erweisen. Moco ist ein leichtgewichtiges Stub-Framework zum Testen HTTP-basierter Endpunkte. Sie können einen eingebetteten Stub-Service bereits mit zwei Zeilen Java- oder Groovy-Codes zum Laufen bringen oder mit wenigen Zeilen JSON zum Beschreiben des gewünschten Verhaltens eine eigenständige Lösung auf die Beine stellen.

ADOPT
48 D3
49 Dependency management for JavaScript

TRIAL
50 Ansible
51 Calabash
52 Chaos Monkey
53 Gatling
54 Grunt.js
55 Hystrix
56 Icon fonts
57 Librarian-puppet and Librarian-Chef
58 Logstash & Graylog2
59 Moco
60 PhantomJS
61 Prototype On Paper
62 SnapCI
63 Snowplow Analytics & Piwik

ASSESS
64 Cloud-init
65 Docker
66 Octopus
67 Sensu
68 Travis for OSX/iOS
69 Visual regression testing tools
70 Xamarin

HOLD
71 Ant
72 Heavyweight test tools
73 TFS

TOOLS

Wir haben uns lange Zeit für die Verwendung handgezeichneter Low-Fidelity-Prototypen zur Veranschaulichung von Benutzerinteraktionen ausgesprochen, ohne uns in den vielen Details aus dem Grafikdesign zu verlieren. **Prototype On Paper** ist ein Tool, mit dem sich individuelle Modelle auf Papier zeichnen und via Kamera auf iOS oder Android erfassen und verknüpfen lassen, um die Benutzerinteraktion zu testen. Damit wird die Lücke zwischen den statischen LoFi-Prototypen und den HiFi-Prototyping-Verfahren elegant geschlossen.

Im letzten Radar haben wir über **SnapCI** von ThoughtWorks gesprochen – einem hosted Service, der Deployment-Pipelines zur Verfügung stellt. Seitdem haben viele Teams SnapCI erfolgreich bei ihren Projekten eingesetzt. Wenn Sie eine einfache Continuous-Delivery-Lösung für die Cloud benötigen, erhalten Sie diese bei SnapCi mit nur einem Klick.

Angesichts der zunehmenden Sensibilisierung für den Datenschutz bereitet es immer mehr Unternehmen Sorge, Web-Analytics-Daten mit Dritten zu teilen. **Snowplow Analytics und Piwik** sind Beispiele für Open-Source-Analytics-Plattformen, die selbst gehostet werden können und denen eine vielversprechende Zukunft prognostiziert wird.

Cloud-init ist eine simple, gleichzeitig aber leistungsstarke Technik, mit der sich beim Booten Aktionen auf einer Cloud-Instanz durchführen lassen. Sie eignet sich besonders in Verbindung mit Metadaten, um einer neu gestarteten Instanz das Abrufen von Konfigurationen, Abhängigkeiten und der zur Ausführung einer speziellen Funktion erforderlichen Software zu gestatten. In Verbindung mit dem Immutable- oder Phoenix-Server-Pattern lässt sich dadurch ein flexibler, leichtgewichtiger Mechanismus für das Deployment-Management in der Cloud einrichten.

Das **Docker** Open-Source-Projekt hat bei ThoughtWorks großes Interesse geweckt und gewinnt weiter an Dynamik und Reife. Über Docker lassen sich Anwendungen als leichte, portable Container packen und veröffentlichen, die auf einem Laptop oder Produktionscluster identisch laufen. Docker bietet Tools für die Erstellung und das Management von Anwendungscontainern sowie eine Laufzeitumgebung auf Basis von LXC (LinuX Containers).

Bei vielen Monitoring-Tools steht der Computer im Mittelpunkt. Wir überwachen, was der Rechner macht und welche Software auf ihm läuft. Bei Cloud-basierter Infrastruktur, insbesondere bei Pattern wie Phoenix- und Immutable-Servern, ist dieser Ansatz jedoch problematisch. Einzelne Server kommen und gehen. Wichtig ist aber, dass die Services immer funktionieren. **Sensu** erlaubt es einem Computer, sich selbst in einer speziellen Rolle zu registrieren, und es überwacht auf dieser Basis dann seine Aktivitäten. Wenn der Server nicht länger benötigt wird, kann er mit dem Tool einfach wieder abgemeldet werden.

Alle Entwicklungen für iOS müssen auf OS X durchgeführt werden. Aufgrund technischer sowie lizenzbezogener Einschränkungen ist der Betrieb von Serverfarmen mit OS X weder einfach noch häufig. Trotz dieser Schwierigkeiten bietet **Travis CI** nun – mit Unterstützung von Sauce Labs – Cloud-basierte Continuous-Integration-Services für iOS- und OS X-Projekte an.

Die zunehmende Komplexität von Web-Anwendungen hat das Bewusstsein geschärft, dass neben der Funktionalität auch das Erscheinungsbild getestet werden sollte. Dies hat verschiedene **visuelle Regressions-Testwerkzeuge** hervorgebracht, darunter CSS Critic, dpxdt, Huxley, PhantomCSS und Wraith. Die Techniken reichen von direkten Assertions von CSS-Werten bis hin zu Screenshot-Vergleichen. Obwohl in diesem Bereich noch immer aktiv weiterentwickelt wird, sind wir überzeugt, dass die Tests im Bereich der visuellen Regression zu Continuous-Delivery-Pipelines hinzugefügt werden sollten.

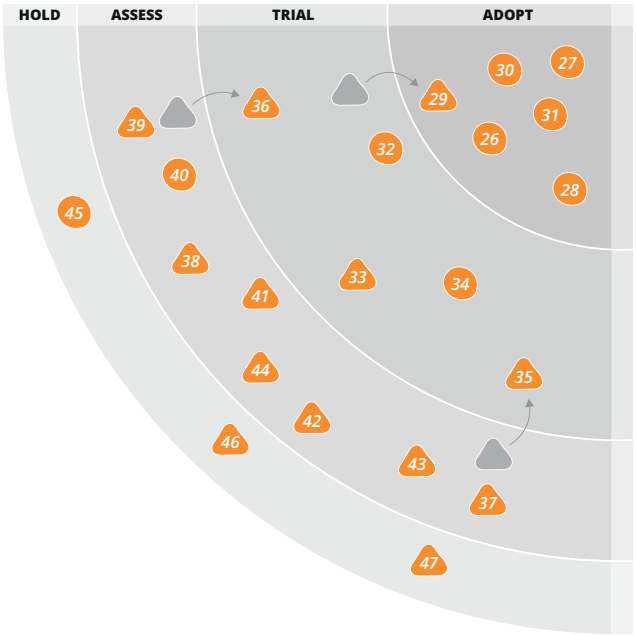
Von den vielen verschiedenen Möglichkeiten zur Entwicklung plattformübergreifender mobiler Apps bietet **Xamarin** ein ziemlich einzigartiges Toolset. Es unterstützt C# und F# als Primärsprachen mit Verbindungen zu plattformspezifischen SDKs und die Mono-Laufzeitumgebung, die unter iOS, Android und Windows läuft. Anwendungen werden zu systemeigenem Code kompiliert, anstatt den typischen plattformübergreifenden Ansatz zu wählen, der HTML-basierte UI in einem eingebetteten Browser rendert. Dadurch erhalten Apps ein natürlicheres Look & Feel. Bei Verwendung dieses Toolsets muss die plattformspezifische UI-Ebene unbedingt von den übrigen Ebenen getrennt werden, damit der Code über die verschiedenen Plattformen hinweg wieder verwendet werden kann. Die App-Binärdatei ist tendenziell etwas größer, was auf die integrierte Laufzeitumgebung zurückzuführen ist.

Wir stellen immer noch fest, dass sich einige Teams intensiv mit nicht-wartbaren **Ant**- und Nant-Buildskripts beschäftigen. Diese sind aufgrund ihrer mangelnden Ausdruckskraft und Modularität nur schwer verständlich und erweiterbar. Alternativen wie Gradle, Buildr und PSake haben ihre bessere Wartungsfreundlichkeit und Produktivität klar unter Beweis gestellt.

PLATTFORMEN

Wir sehen Unternehmen, die nach erfolgreicher Einführung von Hadoop damit begonnen haben, ihre Hadoop-Infrastrukturdienstleistungen in einer zentral verwalteten Plattform zusammenzufassen, bevor das System im gesamten Unternehmen eingeführt wurde. Typisch für diese Plattformen, die **Hadoop-as-a-Service** anbieten, ist eine Kontrollebene, die Schnittstellen zu verschiedenen Hadoop-Infrastrukturkomponenten besitzt und diese Komponenten koordiniert. Für das Unternehmen werden die Fähigkeiten der Plattform in der Regel über Abstraktionen auf einer höheren Ebene dargestellt. Eine solche Managed Platform gibt Unternehmen die Möglichkeit, Prozesse, Infrastruktur und Datenmengen im gesamten Unternehmen auf relativ einheitliche Weise anzuwenden. Diese Services sind in privaten Datenzentren und öffentlichen Cloud-Infrastrukturen verfügbar.

Akka ist ein Toolkit und eine Laufzeitumgebung für die Entwicklung hochgradig paralleler, verteilter und fehlertoleranter ereignisgesteuerter Applikationen auf der JVM. Es bietet leichtgewichtige ereignisgesteuerte Prozesse mit zirka 2,7 Millionen Aktoren per GB RAM und ein „Let-it-crash"-Modell der Fehlertoleranz, das speziell für verteilte Umgebungen geschaffen wurde. Akka kann als Bibliothek für Web-Apps oder als eigenständiger Systemkern zur Ausführung von Anwendungen genutzt werden.



Die aktuelle Explosion mobil ausgerichteter Produkte hat zusammen mit dem weit verbreiteten Ansatz der „Lean Start-ups", bei dem die Kürze der Zeit zur Produkteinführung von neuen Ideen an oberster Stelle steht, zur Entstehung eines Ökosystems geführt, das **Backend-as-a-Service** (BaaS) zum Inhalt hat. Dank dieser Angebote können sich die Entwickler auf die Client-Anwendung konzentrieren, ohne sich um das Backend Gedanken machen zu müssen. Betrachten Sie, ob es sich lohnen könnte, solche Dienstleistungen auch zu Ihrem Toolkit hinzuzufügen, wenn es darauf ankommt, schnell und kostengünstig den Wert einer neuen Produktidee zu beweisen.

Ansonsten gilt nach wie vor unsere Empfehlung bei allen Entwicklungs- und Kaufentscheidungen: Man muss sich im Klaren sein, welche Funktionsbereiche für das Geschäft von strategischer Bedeutung sind und bei welchen es sich um Standards handelt. Für alle Bereiche, die potenziell von strategischer Bedeutung sein könnten, sollten Sie einen Migrationspfad einplanen, bei dem Sie einen BaaS-Provider nutzen können, um schnell loslegen zu können, aber gleichzeitig Probleme vermeiden, wenn Ihre Architektur weiter wächst. Denn dann müssen Sie vielleicht dafür sorgen, dass Sie diese Funktion selbst kontrollieren und als Unterscheidungsmerkmal verwerten können.

Die Kosten für Industrieroboter sinken weiter, gleichzeitig werden diese immer sicherer und einfacher in der Anwendung. Damit eröffnet sich eine völlig neue Welt der kommerziellen Robotik mit zahlreichen nützlichen Anwendungen. Roboter wie Baxter* von Rethink Robotics oder der U5 von Universal Robotics geben auch kleinen und mittleren Unternehmen die Möglichkeit, eintönige Aufgaben zu automatisieren, die bisher von Menschenhand ausgeführt wurden. Und so steht auch die Unternehmenssoftware zunehmend vor der Aufgabe, **kostengünstige Robotik** als weiteren Bestandteil der Wertschöpfungskette zu integrieren. Hierbei kommt es darauf an, das gesamte Umfeld auch für die Mitarbeiter einfach und produktiv zu gestalten.

ADOPT
- 26 Elastic Search
- 27 MongoDB
- 28 Neo4J
- 29 Node.js
- 30 Redis
- 31 SMS and USSD as a UI

TRIAL
- 32 Hadoop 2.0
- 33 Hadoop as a service
- 34 OpenStack
- 35 PostgreSQL for NoSQL
- 36 Vumi

ASSESS
- 37 Akka
- 38 Backend as a service
- 39 Low-cost robotics
- 40 PhoneGap/Apache Cordova
- 41 Private Clouds
- 42 SPDY
- 43 Storm
- 44 Web Components standard

HOLD
- 45 Big enterprise solutions
- 46 CMS as a platform
- 47 Enterprise Data Warehouse

PLATTFORMEN

In den letzten Jahren ist der Bedarf an physischer Datenspeicherung in Ländern bzw. Unternehmen deutlich gestiegen. In Bezug auf die Speicherung sensibler Informationen in Clouds gibt es jedoch Bedenken. Für Organisationen sind Private Clouds eine Alternative, wenn Daten so gespeichert werden müssen, dass eine Kontrolle über Zugriff und Verteilung erfolgen kann. **Private Clouds** bieten eine Cloud-Infrastruktur zur exklusiven Nutzung durch eine Organisation mit folgenden Merkmalen: Selbstbedienung nach Bedarf, breitbandiger Netzwerkzugriff, Ressourcen-Pools, schnelle Elastizität und passender Service.

SPDY ist ein offenes Netzwerkprotokoll zum Transport von Webinhalten mit geringer Latenz, vorgeschlagen für HTTP2, dessen Support in modernen Browsern gestiegen ist. SPDY sorgt für kürzere Ladezeiten, indem die Übertragung von Unterressourcen priorisiert wird. So ist nur eine Verbindung pro Client erforderlich. Besonders in Umgebungen mit hoher Latenz ist dies von Vorteil.

Problematisch bei Big Data sind nicht nur die Heterogenitätund die überwältigend großen Datenmengen. In bestimmten Situationen kann die Verarbeitungsgeschwindigkeit genauso wichtig sein wie das Volumen. Storm ist ein verteiltes Echtzeit-Rechensystem. Die Skalierbarkeit ist mit Hadoop vergleichbar, der Datendurchlaufsatz kann bei einer Million Tupeln pro Sekunde liegen. Damit bietet das System für die Echtzeitverarbeitung dieselben Vorteile wie Hadoop für Batches.

Im vorherigen Radar warnten wir vor der Nutzung herkömmlicher komponenten-basierter Web-Frameworks, die ein Komponentenmodell auf Serverseite bieten. Der **Web Components Standard**, der bei Google entstanden ist, ist jedoch etwas völlig Anderes. Er bietet einfachere Möglichkeiten zur Entwicklung recycelbarer Widgets, indem er die Kapselung von HTML, CSS und JavaScript unterstützt, so dass diese nicht mit dem Rest der Seite ins Gehege kommen. Die Entwickler können so viele oder wenige Teile des Frameworks nutzen, wie sie möchten. Das Polymer Project bietet frühe Unterstützung für diesen Standard.

Während die zentralisierte Integration von Daten zu Analyse- und Reportingzwecken weiterhin eine gute Strategie ist, weisen traditionelle **Enterprise Data Warehouse** (EDW)-Initiativen eine Misserfolgsquote von mehr als 50 % auf. Umfangreiche Vorab-Datenmodellierung führt zu überdesignten Warehouses, deren Entwicklung viel Zeit in Anspruch nimmt und deren Wartung sehr kostspielig ist. In dieser Ausgabe des Radars mahnen wir bei diesen altmodischen EDWs und Verfahren daher zur Vorsicht. Vielmehr empfehlen wir die Entwicklung von EDW durch Testen und Lernen: mithilfe von kleinen wertsteigernden Schritten, die in regelmäßigen Abständen in die Produktion übergehen. Auch nicht-traditionelle Tools und Methoden können sich als hilfreich erweisen, beispielsweise die Verwendung eines Data-Vault-Schemas oder einer NoSQL-Dokumentablage wie HDFS.

Content-Management-Systeme (CMS) haben ihren Platz gefunden. In vielen Fällen ist es unvernünftig, Editing- und Workflow-Funktionalitäten von Grund auf neu zu schreiben. Jedoch haben wir ernste Probleme festgestellt, wenn **CMS-as-a-Platform** zu einer IT-Lösung wird, die über die Verwaltung einfacher Inhalte hinausgeht.

SPRACHEN & FRAMEWORKS

Scala ist eine umfangreiche Programmiersprache, die aufgrund ihrer Zugänglichkeit für neue Entwickler sehr beliebt ist. Doch die Fülle an Funktionen ist gleichzeitig auch problematisch: Viele Aspekte von Scala wie zum Beispiel implizite Konvertierung und Dynamics können leicht Schwierigkeiten verursachen. Für den erfolgreichen Einsatz von Scala müssen Sie sich intensiv mit der Sprache befassen und ein genaues Gespür dafür entwickeln, welche Aspekte Sie wirklich benötigen – Sie sollten also selbst festlegen, was für Sie bei Scala **wichtig** ist. Alle Aspekte von Scala, auf die Sie nicht zurückgreifen möchten, können Sie über so genannte Feature Flags deaktivieren.

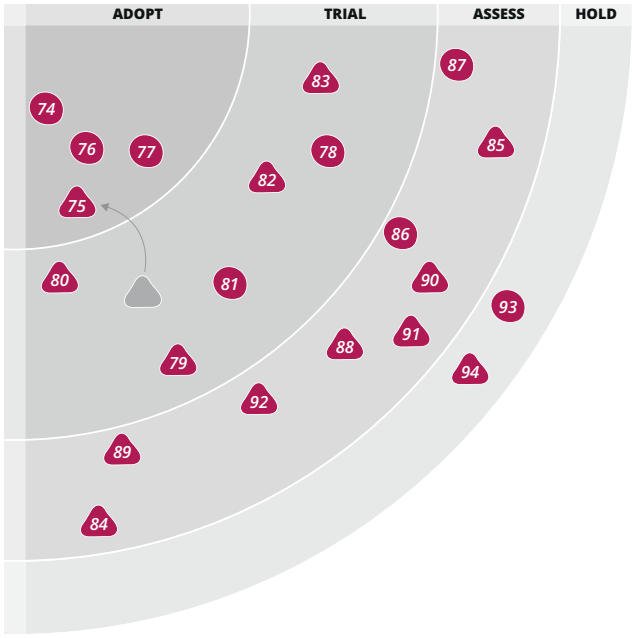
Die Programmiersprache **Go** wurde ursprünglich von Google zur Systemprogrammierung und als Ersatz für C und C++ entwickelt. Vier Jahre nach Einführung gewinnt Go in anderen Bereichen zunehmend an Bedeutung. Dank der Kombination aus sehr kleinen, statisch gelinkten Binärdateien und einer exzellenten HTTP-Bibliothek ist Go vor allem bei Unternehmen beliebt, die auf Microservice-Architekturen setzen.

Hive ist eine Data-Warehouse-Erweiterung für Hadoop, die eine SQL-ähnliche Abfrage- und Datendefinitionssprache ermöglicht. Damit können Anfragen in Map-Reduce-Jobs konvertiert werden, die innerhalb des gesamten Hadoop-Clusters ausgeführt werden können. Wie alle nützlichen Abstraktionen ignoriert Hive die zugrundeliegenden Mechanismen von Hadoop nicht und unterstützt individuelle Map-Reduce-Operationen als mächtige Erweiterungen. Trotz der oberflächlichen Ähnlichkeiten zu SQL versteht sich Hive nicht als Ersatz für Echtzeit-Abfragemodule mit niedriger Latenz, wie man sie bei relationalen Datenbanksystemen findet. Wir raten dringend davon ab, Hive für Online-Adhoc-Abfragen zu verwenden.

Play Framework 2 hat intern zu zahlreichen Diskussionen geführt und die unterschiedlichsten Ansichten hervorgebracht. Diese Unterschiede beziehen sich in erster Linie auf die konkreten Anwendungen, für die es verwendet werden soll, und welche Erwartungen damit verknüpft sind. Obwohl keines dieser Probleme auf Play beschränkt ist, hat Play in der Standard-Debatte zwischen Bibliothek und Framework weitaus mehr Kontroversen als üblich hervorgebracht. Wir möchten noch einmal auf die Vorsichtsmaßnahmen aus dem vorherigen Radar hinweisen und werden weiter im Auge behalten, wie Play in Zukunft weiter reifen und seine Position ausbauen kann.

Reaktives Programmieren beschäftigt sich mit Streams oder Werten, die sich im Laufe der Zeit ändern. Durch Datenfluss-Elemente, implizite Nebenläufigkeit sowie transparente Ereignispropagation ermöglichen diese Techniken ein effizientes Verarbeiten von Ereignissen im großen Maßstab mit einem hohen Maß an Effizienz und niedriger Latenz. Im letzten Radar sind wir auf Reactive Extensions in .NET eingegangen – insbesondere aufgrund der umfangreichen Bemühungen von Microsoft, Rx zu einem wesentlichen Bestandteil des .NET Framework werden zu lassen. Seitdem wurde mit der Einführung der Reactive-Cocoa-Bibliothek für Objective C, dem Java-Port von Reactive Extensions, der React-JavaScript-Bibliothek, der Elm-Sprache auf Basis von Haskell sowie der Flapjax-JavaScript-Bibliothek die Grundlage für unsere **sprachübergreifende** Empfehlung von **Reactive Extensions** geschaffen.

Bis vor Kurzem war das **Web API** von Microsoft die am wenigsten problematische Option für RESTful Services mit ASP.NET. Web API 2 glättet dank besserem Support für flexibles Routing, Sub-Resources, Medientypen und verbesserten Testmöglichkeiten einige kleinere Mängel. Web API bleibt unsere bevorzugte Bibliothek für .NET REST APIs.



ADOPT
- 74 Clojure
- 75 Dropwizard
- 76 Scala, the good parts
- 77 Sinatra

TRIAL
- 78 CoffeeScript
- 79 Go language
- 80 Hive
- 81 Play Framework 2
- 82 Reactive Extensions across languages
- 83 Web API

ASSESS
- 84 Elixir
- 85 Julia
- 86 Nancy
- 87 OWIN
- 88 Pester
- 89 Pointer Events
- 90 Python 3
- 91 TypeScript
- 92 Yeoman

HOLD
- 93 Handwritten CSS
- 94 JSF

SPRACHEN & FRAMEWORKS

Elixir ist eine dynamische, funktionale, homoikonische Programmiersprache, die auf der virtuellen Maschine von Erlang aufbaut – mit einem leistungsstarken Makrosystem, das sie ideal für den Aufbau domain-spezifischer Sprachen macht. Elixir verfügt über einzigartige Features wie den Pipe-Operator, mit dem Entwickler analog zur UNIX-Shell die Ausgabe eines Befehls direkt auf einen anderen Befehl weiterleiten lassen können. Über den gemeinsamen Bytecode kann Elixir mit Erlang zusammenarbeiten und bestehende Bibliotheken nutzen. Gleichzeitig werden Tools wie das Mix-Buildtool, die interaktive Shell Iex oder das Unit-Testing-Framework ExUnit unterstützt. Elixir ist eine praktische Alternative zu Erlang beim Aufbau von domain-spezifischen Sprachen.

Julia ist eine dynamische, prozedurale und homoikonische Programmiersprache, die vor allem für die hohen Anforderungen von Berechnungen im wissenschaftlichen Umfeld entwickelt wurde. Im Zentrum der Implementierung der Sprache steht das Konzept der generischen Funktionen und des dynamischen Methoden-Dispatches. Mit Julia werden zum Großteil Funktionen geschrieben, die eine Vielzahl von Definitionen für verschiedene Argumenttypen-Kombinationen enthalten können. Dank der Kombination dieser Sprach-Features mit dem Just-in-Time-Compiler auf LLVM-Basis kann Julia eine sehr hohe Performance erreichen. Darüber hinaus unterstützt Julia eine Multiprocessing-Umgebung auf Basis von Message Passing, sodass Programme auf mehreren Prozessen laufen können. Programmierer sind dadurch in der Lage, verteilte Programme auf Basis eines beliebigen Modells zur parallelen Programmierung zu erstellen.

PowerShell ist und bleibt eine weit verbreitete Option zur Low-Level-Automation auf Windows-Rechnern. **Pester** ist eine Testing-Library, mit der PowerShell-Befehle ausgeführt und validiert werden können. Pester vereinfacht die Überprüfung von Skripts bei der Entwicklung durch ein leistungsstarkes Modellierungssystem, mit dem Stubs und Dopplungen in Tests eingerichtet werden können. Pester-Tests können zudem in ein Continuous-Integration-System integriert werden, um Regressionsdefekte zu verhindern.

Python 3 war eine größere Änderung der Vorgängerversion Python 2.x, bei der rückwärts inkompatible Änderungen eingeführt wurden. Es ist bekannt dafür, dass Sprach-Features entfernt wurden, um es anwenderfreundlicher und einheitlicher zu machen, ohne dass dabei Mächtigkeit verloren geht. Das führte zu gewissen Problemen bei der Annahme durch Entwickler, da einige der gern genutzten Bibliotheken nicht portiert wurden und die Entwickler von Python oftmals neue Wege dafür einschlagen mussten. Dennoch ist die Entwicklung hin zur Vereinfachung von Sprachen sicherlich der richtige Weg. Und wenn Sie selbst als Python-Entwickler aktiv sind, gönnen Sie Python 3 einen zweiten Blick.

Mit einiger Verzögerung – vor allem infolge von Patentansprüchen von Apple – hat das W3C nun die Spezifikation für Touch Events verabschiedet. Mit **Pointer Events** gewinnt mittlerweile jedoch ein neuerer, weiter gefasster und reichhaltigerer Standard an Fahrt. Wir empfehlen, Pointer Events bei HTML-Schnittstellen zu berücksichtigen, die über verschiedene Eingabemethoden hinweg funktionieren müssen.

Hinter **TypeScript** verbirgt sich ein interessanter Ansatz, um eine neue Programmiersprache in den Browser zu bringen. Mit TypeScript lassen sich die neuen Sprach-Features auf normales JavaScript herunterbrechen. Als Superset von JavaScript fühlt es sich dennoch nicht wie eine komplett neue Sprache an. Weder handelt sich um eine Entweder-oder Entscheidung noch wird JavaScript zu einer intermediären Ausführungsplattform degradiert. Viele der Sprach-Features basieren auf geplanten zukünftigen Erweiterungen von JavaScript.

Yeoman will Entwickler von Web-Anwendungen produktiver machen, indem Aktivitäten wie Scaffolding, Building und Package-Management vereinfacht werden. Es handelt sich hierbei um eine Sammlung der Tools Yo, Grunt und Bower, die wunderbar im Set funktionieren.

Noch immer stellen wir fest, dass einige Teams bei der Verwendung von **JSF** – JavaServer Faces – Schwierigkeiten haben und empfehlen daher, auf diese Technologie zu verzichten. Viele Teams entscheiden sich scheinbar für JSF, da es sich hier um einen Java EE-Standard handelt. Dabei wird aber nur selten beurteilt, ob das Programmiermodell wirklich sinnvoll ist. Unserer Meinung nach verfolgt JSF einen falschen Ansatz, da es versucht, HTML, CSS und HTTP wegzuabstrahieren – das genaue Gegenteil von dem, was moderne Frameworks machen. Genauso wie ASP.NET WebForms versucht JSF über das zustandlose Protokoll HTTP einen Zustand herzustellen, was zu verschiedensten Problemen mit serverseitiger Zustandsverwaltung führt. Wir sind uns der Verbesserungen bei JSF 2.x durchaus bewusst. Dennoch ist das Modell unserer Meinung nach grundlegend falsch. Wir empfehlen daher den Einsatz einfacher Frameworks zusammen mit Web-Technologien wie HTTP, HTML und CSS.

REFERENZEN

Tangible Interaction
http://www.interaction-design.org/encyclopedia/tangible_interaction.html
http://www.computer.org/csdl/mags/co/2013/08/mco2013080070-abs.html
http://www.theverge.com/2012/9/21/3369616/co-working-robots-baxter-home
http://robohub.org/rethink-robotics-baxter-and-universal-robots-ur5-and-ur10-succeeding/

Web Components standard
http://www.polymer-project.org

Hystrix
https://github.com/Netflix/Hystrix/wiki
https://github.com/Netflix/Hystrix/tree/master/hystrix-dashboard
https://github.com/Netflix/Turbine/wiki

Reactive Extensions across Languages
https://github.com/blog/1107-reactivecocoa-for-a-better-world
http://facebook.github.io/react/
http://techblog.netflix.com/2013/02/rxjava-netflix-api.html
http://elm-lang.org/
http://www.flapjax-lang.org/

Pointer Events
http://www.w3.org/TR/pointerevents/
http://www.w3.org/TR/touch-events/
http://www.w3.org/2012/te-pag/pagreport.html
http://msopentech.com/blog/2013/06/17/w3c-pointer-events-gains-further-web-momentum-with-patch-for-mozilla-firefox

ThoughtWorks®

ThoughtWorks ist ein Software-Unternehmen und eine Gruppe engagierter Menschen, die es sich zur Aufgabe gemacht haben, das Design, die Erstellung sowie die Verbreitung von Software grundlegend zu verbessern und gleichzeitig mittels Software-Lösungen positive soziale Veränderungen herbeizuführen. Das Besondere dabei ist, dass diese Lösungen gemeinsam mit dem Kunden erarbeitet werden. Die drei Säulen des Geschäftsmodells und grundlegend für die Arbeit von ThoughtWorks sind nachhaltige Ausrichtung des Unternehmens, Förderung und Entwicklung von herausragender Software sowie Engagement für soziale und wirtschaftliche Gerechtigkeit.

Kunden von ThoughtWorks sind Personen und Unternehmen mit ehrgeizigen Zielen, denen neuartige Denkweisen und Technologien geliefert werden. Die Produktabteilung ThoughtWorks Studios erstellt wegweisende Tools wie Mingle®, Go™ und Twist® die Unternehmen dabei helfen, mittels hochwertiger Lösungen Aussergewöhnliches zu leisten.

ThoughtWorks beschäftigt derzeit mehr als 2.800 Mitarbeiter, die von Australien, Brasilien, Ecuador, Kanada, China, Deutschland, Indien, Singapur, Südafrika, Uganda, Großbritannien und den USA aus Kunden betreuen.