ASERVO Software

KNOW-HOW

Continuous Testing - vom Trend zum Pattern

Wieso automatisierten Tests die Zukunft gehört

Wieso automatisierten Tests die Zukunft gehört

Was ist denn das nun schon wieder? Brauche ich das wirklich? Das kostet mich doch nur wieder viel Zeit und Geld?! Kann nicht einfach alles so bleiben wie es ist? Solche Fragen stellen sich viele immer wieder, wenn es einen neuen Trend gibt und Unternehmen überlegen, ob man diesem Trend folgen möchte oder auch nicht. Was Continuous Testing (CT) ist, wie wir das Konzept dahinter zum Vorteil nutzen können und warum die Automation von Tests mit CT weit mehr als ein Trend ist, erläutern wir im Folgenden.

Was ist eigentlich Continuous Testing und welche Vorteile bringt es?
//

Ganz einfach formuliert, ist Continuous Testing die Integration von automatisierten Tests in den kontinuierlichen Software Delivery Prozess. Sogar die mehrfache Integration von automatisierten Tests innerhalb eines Prozesses, wie nach bestimmten Schritten eines Software Delivery Prozesses, ist nicht unüblich.

Prinzipiell eignen sich alle Arten von Tests für den Einsatz. Sowohl funktionale Tests, wie Unittests, Integrationstests, API Tests, End2End-Tests oder auch Layouttests, als auch nicht-funktionale Tests, wie Last- und Performancetests, Sicherheitstests oder Usability-Tests. Es macht meist aber keinen Sinn alle vorhandenen Tests nach jedem Schritt im Delivery Prozess durchzuführen. Soll heißen, es gibt kein Rezept oder eine klar formulierte Schritt-für-Schritt Anleitung, wie Continuous Testing ein- oder umzusetzen ist. **Continuous Testing ist** eher eine Art **Pattern fürs kontinuierliche Testen mit kurzen Feedbackschleifen**. Basis ist dabei immer, dass man sich für sein spezielles Umfeld, seine Software, seine Prozesse, seine Infrastruktur etc. überlegen muss, wie sich automatisierte Tests gemessen am Verhältnis Aufwand/Nutzen und entsprechend ihrem Aussagegehalt je Testart am effektivsten einsetzen lassen. Hierbei ist es enorm wichtig, dass sich alle Beteiligte im Vorfeld überlegen und definieren müssen, wo, wann welche Tests Sinn machen. Dies können nur alle zusammen entscheiden, da unterschiedliche Personen unterschiedlicher Rollen auch unterschiedliche Ansichten und Prioritäten haben.

Konsistenz gewährleistet Sicherheit und Qualität
//

Automatisierte Tests haben den Vorteil, dass sie nur einmalig implementiert werden müssen, meist schneller laufen als manuelle Testdurchführungen und immer exakt den gleichen Ablauf gewährleisten. Diese Vorteile bietet insbesondere das Continuous Testing. Einmalig implementierte automatisierte Tests werden nach jedem Change immer wieder durchgeführt in den vorhandenen Software Delivery Prozessen. Auf den ersten Blick wirkt es so, als ob sich die Prozesse nur unnötig verlängern würden, wenn Tests immer und immer wieder laufen und sich somit die Produktivsetzung der Software verzögert. Aus rein zeitlicher Sicht mag das stimmen, natürlich kosten Testdurchführungen Zeit. Aber es lohnt sich diesen Zeitaufwand in Kauf zu nehmen, denn:

- CT ermöglicht ein schnelles Feedback über den korrekten Ablauf meiner Software Delivery Prozesse
  - durch Rückmeldungen zu jedem einzelnen Schritt, zu dem automatisierte Tests nachgelagert integriert sind.
  - durch die Vermeidung von Wartezeit bis der komplette Prozess abgeschlossen wurde.
- CT spart manuellen Aufwand, speziell im Bereich Testing und Debugging.
- CT mindert das Risiko, fehlerhafte Software und Konfigurationen produktiv zu nehmen.
- CT ermöglicht eine bessere Ursachenforschung bei Problemen (Defect Analysis), da der Software Delivery Prozess sofort abbricht, wenn Tests fehlschlagen.
- CT ist die Grundlage für durchgehende Delivery Prozesse von Dev bis Ops.

Kurz gefasst, bietet Continuous Testing dem ganzen Team mehr Sicherheit und ein gutes Gefühl für erfolgreiche Deliveries.

Wie kam es zum Ansatz Continuous Testing?
//

Vom Grundgedanken her ist Continuous Testing ein wirklich sehr einfacher Ansatz. Die Komplexität liegt dabei, wie so oft, im Detail und der Umsetzung. Um zu erfahren, wie es zu diesem Ansatz kam, ist es sinnvoll sich die Entwicklung der Software Delivery Prozesse zu verdeutlichen.

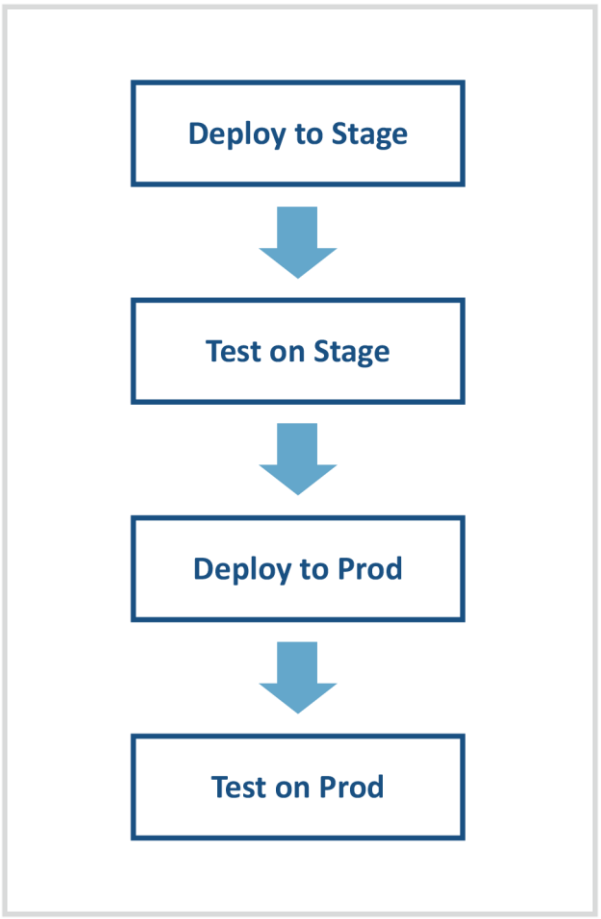

Abbildung 1: *Software Delivery Prozess (damals)*

Der Software Delivery Prozess damals ...
//

„Früher" (s. Abbildung 1) wurde Software meist nach dem Wasserfallmodell entwickelt. Dies bedeutet, dass die Software zuerst geplant und die Anforderungen in **Schriftform** dokumentiert wurden. Anschließend fand dann die **Entwicklung** statt und ganz am Ende stand für alles zusammen das **Testing** an. Es gab meist nur wenige Releases im Jahr. Die Deployments auf die Testumgebung (Stage) und die Produktivumgebungen hingen oft noch mit viel manuellem Aufwand zusammen und die Tests, die durchgeführt wurden, waren zum großen Teil manueller Art und häufig explorativer Art.

Typisch waren auch Checklisten für Tests, die von Punkt 1 bis Punkt n nach jedem Release immer und immer wieder abgearbeitet wurden. Zu der Zeit war das Thema der Unittests schon aktuell, aber noch nicht ausgereift; Integrationstests oder API Tests bildeten eher die Ausnahme. Allerdings war zu der Zeit das Thema der End2End-Tests (User-Acceptance-Tests) bereits aktuell und erfuhr im Bereich Web durch die Einführung von Selenium weiteren Aufschwung. Diese Tests zeigten sich aber auch damals schon sehr fehleranfällig und vergleichsweise langsam und träge. Teilweise gab es auch bereits Last- oder Performancetests, die nachgelagert durchgeführt wurden. Nicht selten hieß das Motto „Der Nutzer ist unser bester Tester und wird uns schon Feedback geben."

Durch die wachsende Popularität der agilen Ansätze und Methoden, hat sich auch das Testing und die Denkweise rund um den Software Delivery Prozess angepasst und weiterentwickelt .

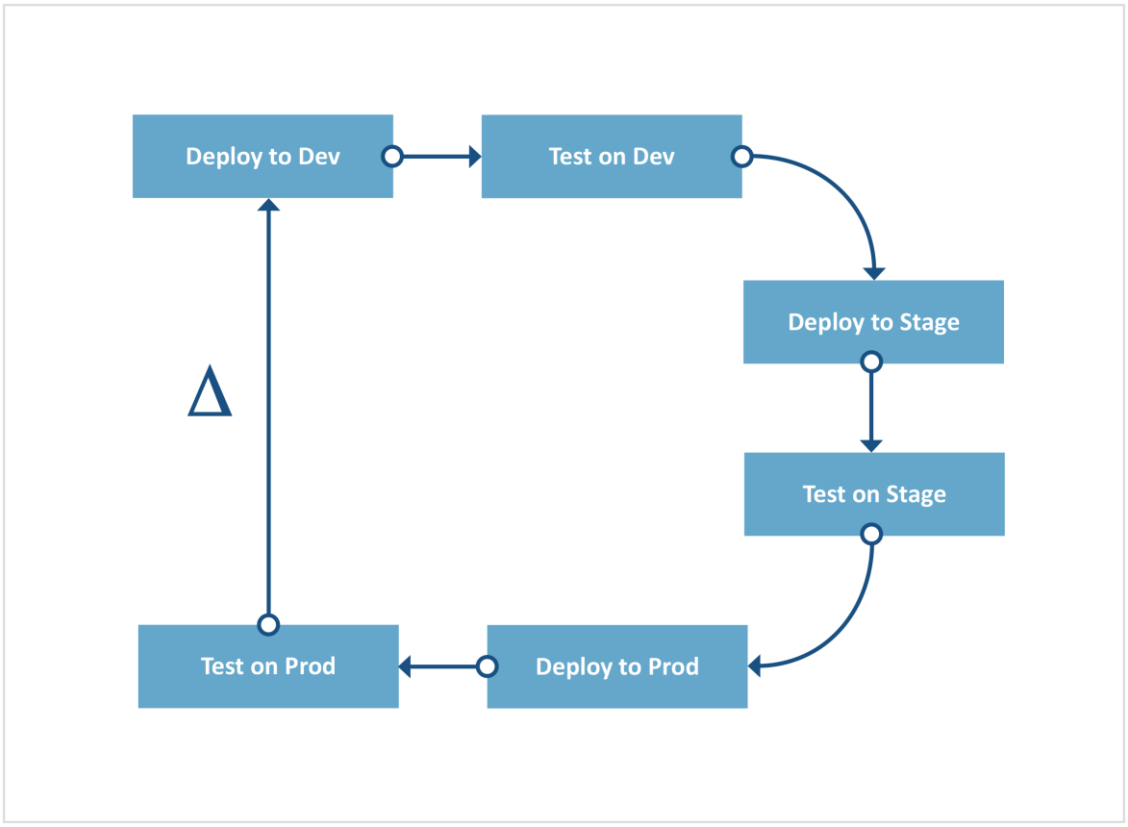

Abbildung 2: Software Delivery Prozess (heute)

... heute
//

Die Leitsprüche „time to market" und „potentially shippable" führten dazu, dass Deployments zum Großteil in Form von **CI/CD Pipelines** automatisiert wurden. Der zeitliche Zyklus der Releases verkürzte sich so deutlich. Oft ist es aber noch so, dass es große Deployments von großen Softwarekomponenten sind, die ausgerollt werden.

Auch im Bereich Test hat es viel Optimierung gegeben. Die Bedeutung von Unittests ist weiterhin gestiegen. Integrationstests und API Tests haben an Wert gewonnen und werden häufiger automatisiert. Es gibt weiterhin oft viele End2End Tests. All diese **Tests sind nun bereits schon integriert in die automatischen Deploymentprozesse**. Teilweise laufen End2End Tests noch als nachgelagerte Prozesse, da sie weiterhin fehleranfälliger und langsamer sind als Unit- oder Integrationstests. Sogar nichtfunktionale Tests, wie Last- oder Performancetests, werden teilweise bereits integriert. (Abbildung 2)

… und in Zukunft
//

Der erste Schritt in Richtung Continuous Testing ist also bereits gemacht. Und wie wird nun aus dem aktuellen Modell ein Continuous Testing Modell? Eigentlich ist die Antwort ganz einfach: Teile deine Software in **Services / Microservices** auf und gewährleiste, dass jeder Service unabhängig von allem anderen über einen Deployment-Prozess bereitgestellt werden kann. Dabei wird jeder Deployment-Schritt eines jeden Services, sofern notwendig und sinnvoll, durch zu definierende, geeignete automatisierte Tests auf Korrektheit überprüft.

Das klingt in der Theorie wirklich sehr simpel, es bedarf im Detail aber wirklich einiges an Aufwand für die Umsetzung. Jeder Service sollte hierbei seine eigene CI/CD Pipeline haben. Um Redundanzen bei der Erstellung und Wartung der Pipeline zu vermeiden und um ein einheitliches Pipelinekonstrukt zu haben, empfiehlt es sich ein Pipeline Template zu definieren, welches dann jeder Service nutzt und für sich implementiert und gegebenenfalls ausprägt (zur grafischen Verdeutlichung siehe Abbildung 3).

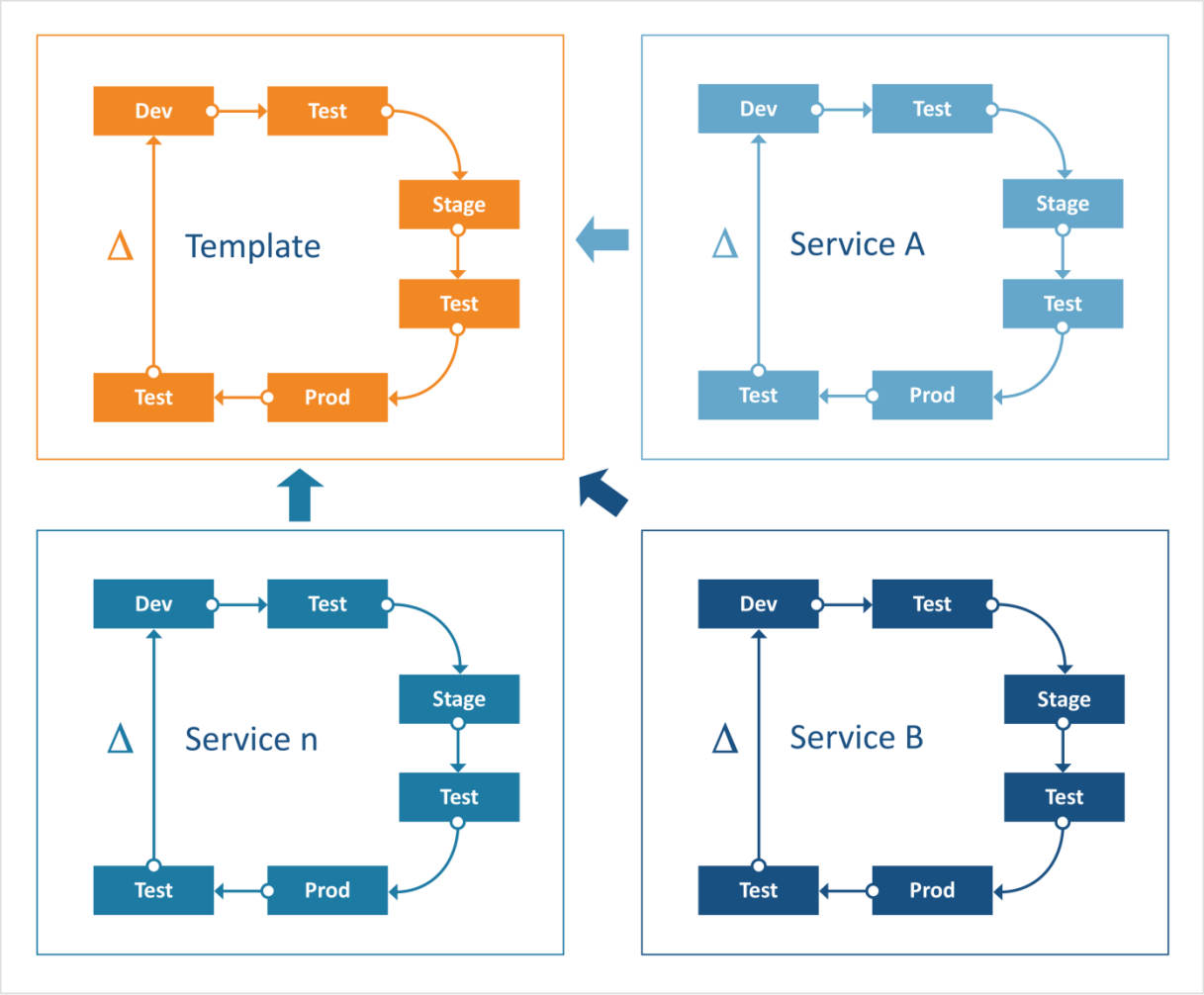

Abbildung 3: Software Delivery Prozess mit Continuous Testing

So lässt sich erreichen, dass automatisierte Tests, wie Unittests, Integrations- und API-Tests, End2End-Tests, Last- und Performancetests oder welche Art von Tests auch immer für jedes Deployment eines jeden definierten Service durchgeführt werden. Welche Tests dabei wann notwendig bzw. sinnvoll sind und einen Mehrwert schaffen, ist hierbei immer im Einzelfall zu definieren.

Speziell in Hinblick auf die fehleranfälligen und recht langsamen **End2End-Tests** sollten folgende Fragen geklärt sein, um ein **richtiges Maß** an Tests **für die Deployment-Pipelines** zu schaffen:

- Ist eine Smoketestsuite für die End2End-Tests sinnvoll und möchte ich diese anstelle aller End2End-Tests durchführen?
- Ist es notwendig die End2End-Tests in jedem Service, teilweise eventuell mehrfach während eines Pipeline-Ablaufs, durchzuführen?
- Welche End2End-Tests sind wirklich wichtig zur Integration in meine Service-Deployment-Pipeline?
- Genügt die Durchführung der End2End-Tests als eine Art Integrationstest mehrerer Services?
- Ist eine separate Durchführung der End2End-Tests über einen eigenen Prozess eventuell sinnvoll oder hilfreich, um ein regelmäßiges Feedback zu bekommen?

Und wie fange ich nun an mit einer Umsetzung?
//

Die Thematik wirkt zunächst sehr theoretisch. Um das Thema aber auch praktisch näher zu verdeutlichen, soll ein **Beispiel** die **Einführung von Continuous Testing** erläutern. Die Abbildung 4 veranschaulicht das mögliche Vorgehen bei der Umsetzung dazu grafisch.

Nehmen wir an, ein Unternehmen hat sich für den Einsatz von Continuous Testing entschieden. Im eigenen Haus kann es nicht auf die dafür notwendigen Ressourcen oder Kompetenzen zurückgreifen und holt sich das entsprechende Know-how bei Consultants, die das eigene Team unterstützen.

Die **Ausgangslage** kennzeichnet eine webbasierte Software, die aus einem Plattform Frontend und einem Plattform Backend besteht. Sowohl das Backend als auch das Frontend basiert auf einem Service, der wiederum auf REST basiert. Weitere funktionale Webservices / Microservices werden bereitgestellt, um vom Frontend oder über eine API gesteuert zu werden (ebenfalls über REST). Einerseits gibt es manuelle Tests, die über die Web GUI der Anwendung laufen. Andererseits werden in zeitlichen Abständen die automatisierten End2End-Tests lokal auf einem Entwicklerrechner ausgeführt. Schnittstellentests existieren nicht und auch die Durchführung von existierenden automatisierten Tests in Jenkins sind weder umgesetzt und noch geplant.

Die vorgeschlagene **Lösung** beinhaltet die folgenden Konstrukte:

- Frontend, Backend und die Services sind separat bereitzustellen über eigene Deployment-Pipelines.
- Hierbei wird von Pipeline zu Pipeline definiert, welche Tests integriert und somit bei jedem Durchlauf ausgeführt werden.
- Zusätzlich existiert ein separater Prozess zur Durchführung der End2End-Tests. Diese werden täglich in unterschiedlichen Zielsystemkonfigurationen aus Betriebssystem und Browsern mithilfe einer Continuous Testing Cloud durchgeführt.

Als **Mehrwert** resultiert so ein durchgehendes integriertes, automatisiertes Testing in alle Pipelines wobei die **Qualität** jeder Pipeline und jedes Deploymentgegenstands **kontinuierlich geprüft** wird. Ein entsprechendes Reporting und gegebenenfalls Abhängigkeiten zwischen Deployment-Pipelines verhindern somit das Ausrollen von bekannten und aufgedeckten Fehlern. Darüber hinaus wird das Eingreifen durch manuelle Testaktivitäten und der damit verbundene Aufwand vermieden und ein Deployment im theoretischen Rhythmus von 24/7 ermöglicht.

Die konkrete Aufteilung in unabhängige (Micro-)Services inkl. der Erstellung der Deployment-Pipelines und der Integration der Tests in diesem Prozess ist durchaus mit zeitlichem Aufwand verbunden und lässt sich nicht von heute auf morgen realisieren. Continuous Testing ist nur ein Bruchteil des gesamten Continuous Lifecycles. Das Resultat der erfolgreichen Umsetzung aber ist lohnenswert.

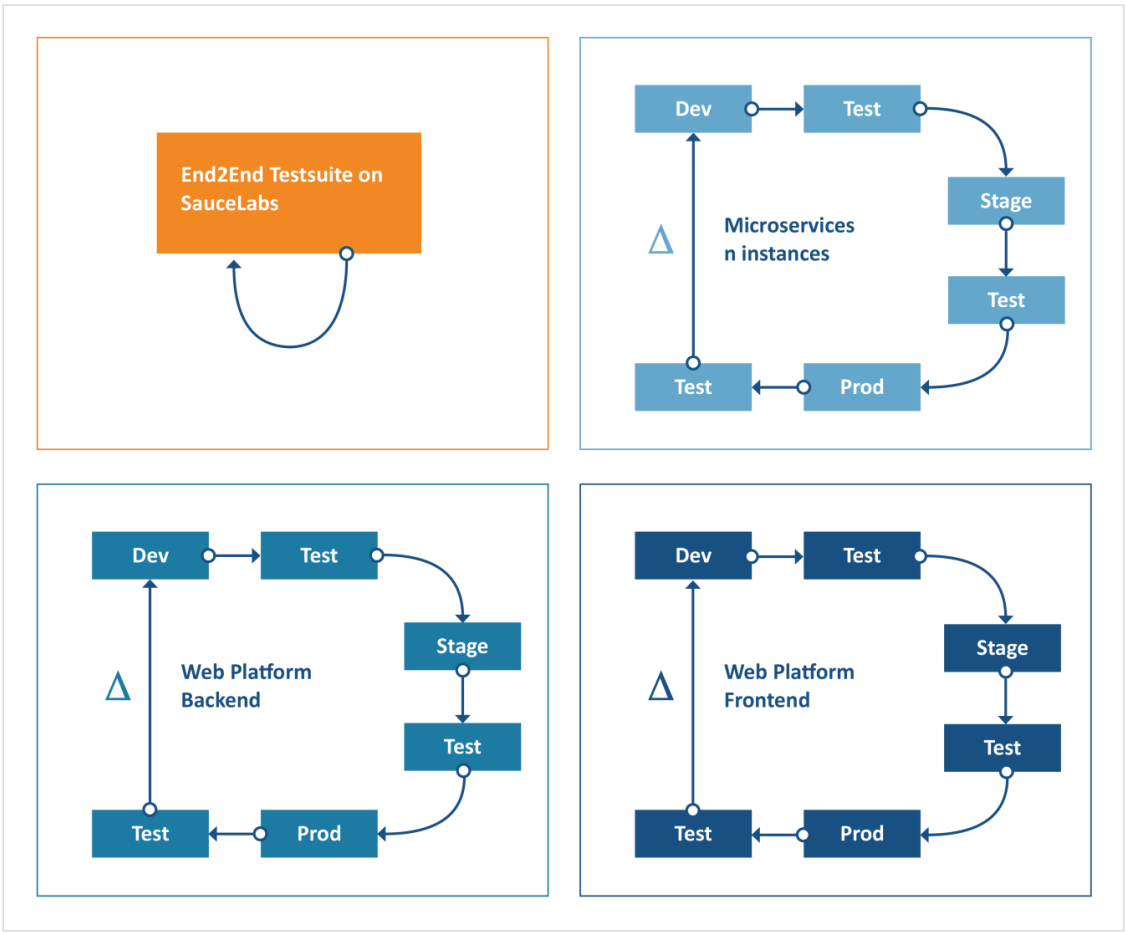

Abbildung 4: Beispiel praktische Umsetzungsplanung CT

Was bleibt? ...ein Fazit
//

Continuous Testing ist ein sehr hilfreiches Konzept ist, um die Qualität der Software, sowie der Delivery Prozesse zu sichern und sogar zu steigern. Insofern ist Continuous Testing auch sicherlich mehr als nur ein Trend, der sich rasch als Konstante oder Pattern im Umfeld Continuous Integration etablieren kann.

Continuous Testing im Rahmen von Continuous Deployment / Delivery ist und bleibt ein spannendes Thema. Unternehmen werden in Zukunft wohl kaum noch auf diese Thematik verzichten können, **um eine hohe Qualität beim Bereitstellen einer Software in einem quantitativen Zeitzyklus sicherzustellen**. In den Zeiten, in denen möglichst alles möglichst schnell geliefert werden muss, soll und darf die Qualität nicht darunter leiden. Und genau hierfür ist Continuous Testing der korrekte Lösungsansatz.

8 Tipps zur Umsetzung von Continuous Testing

- Entscheide von Service zu Service welche Tests sinnvoll und hinreichend sind,um sie in den Deployment-Prozess zu integrieren.
- Führe automatische Tests nach jedem möglichen Schritt (in jeder Deployment Stage) durch.
- Führe nicht alle existierenden Tests in jedem Deployment-Prozess eines jeden Services durch.
- Versuche End2End-Tests zu reduzieren bzw. sie in nebenläufige Prozesse zu exkludieren.
- Motiviere und lehre andere automatisierte Tests zu entwickeln und einzubinden, um das Verständnis und das Wissen zu verteilen (Think like a DevOps).
- Erweitere deine End2End-Tests nur, wenn notwendig und pflege diese Tests mit dem Ziel, dass sie stabil, zuverlässig und robust sind.
- Konzentriere dich auf das Wesentliche:
  - Das saubere Deployment der Services ist das oberste Ziel.
  - Integrierte Tests sollen dabei unterstützen, aber nicht den Mittelpunkt darstellen.
- Teile und diskutiere deine Erfahrungen, Probleme und Lösungen zu Continuous Testing mit anderen.

ASERVO Software GmbH
Konrad-Zuse-Platz 8
81829 München

Tel.: +49 (0) 89 7 16 71 82-40
Fax: +49 (0) 89 7 16 71 82-55
Email: kontakt@aservo.com