Feature Toggles/Branches in der agilen Softwareentwicklung

Timo Hahn
virtual 7 GmbH
Karlsruhe

Schlüsselworte
Feature Toggles, Feature Branch, Continuous Integration, Continuous Delivery, Agile Softwareentwicklung, Test, Unit Test

Einleitung

In der agilen Softwareentwicklung werden mittels Continuous Integration und Continuous Delivery kurze Release-Zyklen realisiert. Somit kann permanent Software bis hin zu den produktiven Systemen bereitgestellt werden, um Neuerungen dem Kunden schnell verfügbar zu machen.
Für Funktionalitäten (Features) die zur Entwicklung länger als einen Release-Zyklus benötigen entsteht daraus das Problem, wie diese Funktionalitäten während der Entwicklungszeit zu behandeln bzw. von anderen Auslieferungen abzugrenzen sind. Weiter zu betrachten sind die verschiedenen Systeme (Development, Test, Integration und Produktion) auf denen die Artefakte bereitzustellen sind.
Grundsätzlich stehen zur Lösung dieses Problems mehrere Vorgehensweisen zur Verfügung. Die gängigsten sind Feature Branches und Feature Toggles.
Im Folgenden gehen wir auf die einzelnen Bestandteile des Softwareentwicklungsprozess in einem Großprojekt ein, welches agil durchgeführt wird. Anschließend betrachten wir die Problemstellung und beleuchten die zur Umsetzung gemachten Entscheidungen und deren Zustandekommen.

Definitionen, Begriffe

Beginnen wir mit einer kurzen Einführung der Begriffe

Agile Softwareentwicklung, Agiler Prozess

Ziel der Vorgehensweise ist es, den Softwareentwicklungsprozess durch Abbau der Bürokratie und durch die stärkere Berücksichtigung der menschlichen Aspekte effizienter zu gestalten.
Den meisten agilen Prozessen liegt zu Grunde, dass sie versuchen, die reine Entwurfsphase auf ein Mindestmaß zu reduzieren und im Entwicklungsprozess so früh wie möglich zu ausführbarer Software zu gelangen, die dann in regelmäßigen, kurzen Abständen dem Kunden zur gemeinsamen Abstimmung vorgelegt werden kann. Auf diese Weise soll es jederzeit möglich sein, flexibel auf Kundenwünsche einzugehen, um so die Kundenzufriedenheit insgesamt zu erhöhen. Sie stellen damit einen Gegensatz zu den klassischen Vorgehensmodellen wie dem V-Modell oder dem Rational Unified Process (RUP) dar.
Inzwischen gibt es eine Vielzahl von agilen Prozessen. Zu den bekanntesten zählen unter anderem:

- Adaptive Software Development (ASD)
- Crystal
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Kanban (Software-Kanban)
- Scrum
- Agiles Testen

- Behavior Driven Development (BDD)

Scrum (stark vereinfacht):

Scrum wird als Vorgehensrahmen oder -gerüst (Framework) und bewusst nicht als Vorgehensmodell bezeichnet, da es nur aus wenigen Regeln besteht. Diese Regeln definieren fünf Aktivitäten, drei Artefakte und drei Rollen, die den Kern von Scrum ausmachen. Die Regeln sind im ‚Agile Atlas' oder ‚Scrum Guide' definiert. Das Scrum-Framework muss durch Techniken für die Umsetzung der Aktivitäten, Artefakte und Rollen konkretisiert werden, um Scrum tatsächlich umsetzen zu können.

Das zu entwickelnde System wird über Produkteigenschaften (Features) definiert, die im Product Backlog in einer priorisierten Reihenfolge festgelegt sind. Das Product Backlog ist anfangs nicht vollständig und wird mit der Zeit um weitere Eigenschaften erweitert bzw. um Eigenschaften reduziert. Die eigentliche Entwicklung des Systems ist in sogenannte Sprints unterteilt, die zwischen einer Woche und einem Monat dauern. In einem Sprint werden alle Eigenschaften umgesetzt, die zuvor aus dem Product Backlog ausgewählt und in das Sprint Backlog eingetragen worden sind. Das Ergebnis eines Sprints ist ein lauffähiges (Teil-)System. An einen Sprint schließt sich die Retrospektive an, in der der soeben abgeschlossene Projektabschnitt bewertet wird, um Erfahrungen zu verarbeiten und in Verbesserungen einfließen zu lassen.

Continuous Integration

Bezeichnet die permanente Integration, das Bauen und Testen von durch die Entwicklung geschriebenen Code in die Mainline der Entwicklungsumgebung.

Continuous Delivery

Bezeichnet eine Sammlung von Techniken, Prozessen und Werkzeugen, die den Softwarelieferprozess verbessern. Techniken wie Testautomatisierung, kontinuierliche Integration (Continuous Integration) und kontinuierliche Installation erlauben die Entwicklung qualitativ hochwertiger Software, die durch automatisierte Release-Erstellung auf Entwicklungs-, Test-, Integrations- und Produktivumgebung eingespielt werden kann.

Feature Toggle

Feature Toggles beschreiben eine Programmiertechnik in der modernen Softwareentwicklung, bei der ein in der Entwicklung befindliches Feature oder eine Funktionalität zur Laufzeit der Software an- oder ausgeschaltet werden kann. Das Entwicklungsteam oder der Entwickler schaltet in der eigenen Umgebung das Feature zur Laufzeit ein, um es erweitern und testen zu können. Beim Einchecken des Sourcecodes in die Integration bleibt das Feature Toggle standardmäßig ausgeschaltet, bis es einen akzeptablen Reifegrad erreicht hat, so dass andere Teams, Testteams oder auch Benutzer damit arbeiten können.

Feature Branch

Feature Branches beschreiben eine Vorgehensweise in der Softwareentwicklung bei der für ein umzusetzendes Feature ein eigener Branch von der Basislinie im VCS angelegt wird, auf dem das Feature dann entwickelt wird. Dies erlaubt die Entwicklung des Features isoliert von anderen Features und Einflüssen. Nach Fertigstellung des Features werden die Änderungen in die Basislinie migriert.

Problemstellung und Vorgaben

Die diesem Vortrag zugrundeliegende Problemstellung und die Vorgaben, die zur Lösung einzuhalten sind, beeinflussen maßgeblich die Entscheidungsfindung. Es soll kurz das Umfeld und die Vorgaben aufgezeigt werden.

Alle Entwicklungen des Kunden werden agil mittels Scrum durchgeführt. Insofern werden die oben in der Definition des Agilen Prozess aufgeführten anderen Modelle nicht weiter betrachtet.

Im Bild unten sehen wir die schematische Systemarchitektur mit ihren Komponenten.

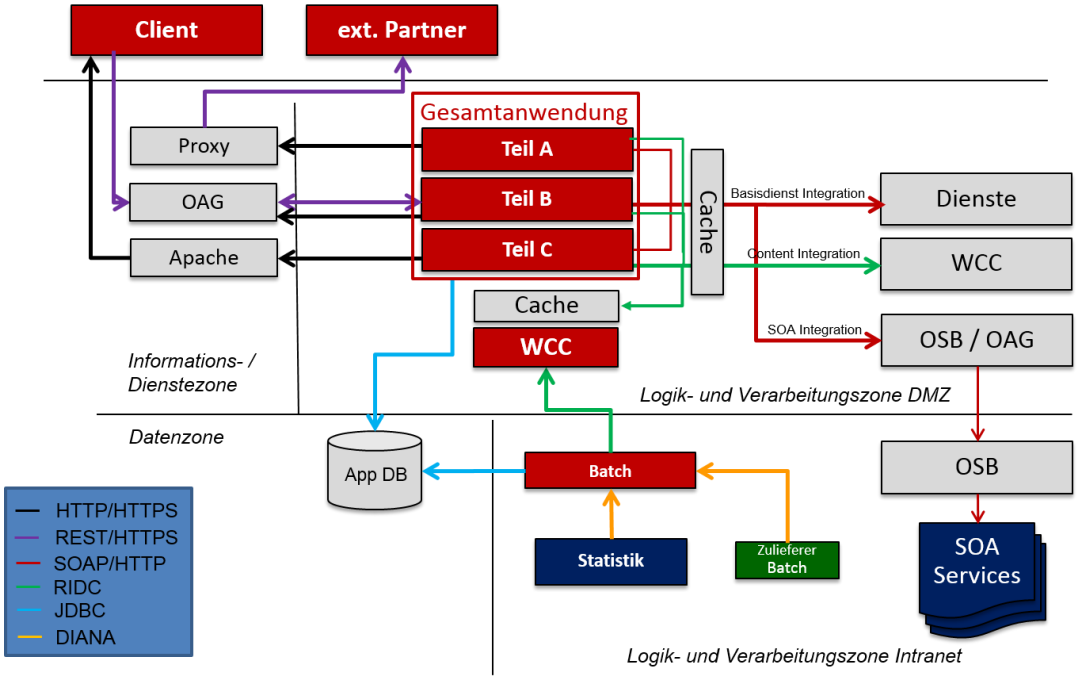

Abb. 1: Schematische Systemarchitektur

Die Anwendungen müssen auf unterschiedlichen Systemen wie lokaler Entwicklungsrechner, Test, Integration, Lasttest und Produktion ausführbar sein. Die verschiedenen Systeme sind in ihrer Größenordnung (Anzahl Server, Speicher,…) abgestuft. Ein System besteht dabei aus ein bis mehreren WebLogic Servern (Clustern), Datenbank, verschiedenen WebServices die Daten liefern, einem Content Management System (WCC). Das Projekt wird mittels Continuous Integration durchgeführt und soll eine Continuous Delivery Strecke aufbauen.

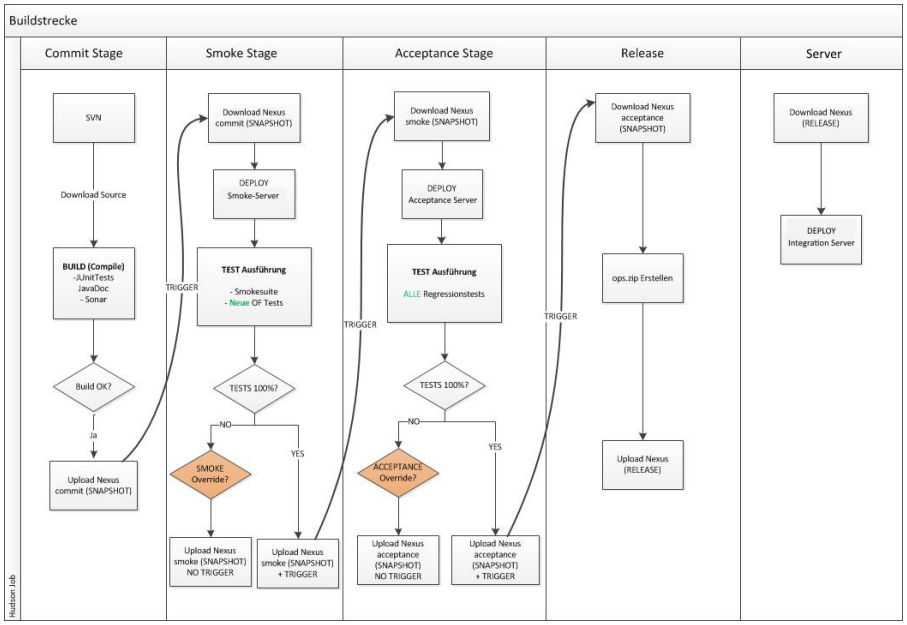

Abb. 2: Typische Continuous Delivery Strecke

Innerhalb des Projektes kommt es immer wieder zu Situationen, in denen Anforderungen systemübergreifend zu implementieren sind, aber nicht während eines Scrum Zyklus (Sprint) abgeschlossen werden können. Mit systemübergreifend ist hier gemeint, dass ein Feature nicht nur in Java für die Web Anwendung zu entwickeln ist, sondern auch Teile in abhängigen Systemen die Datenbank, Batch und/oder Content Server zu realisieren ist.
Durch die Beteiligung der verschiedenen Systeme kann es zu erhöhter zeitlicher Dauer der Realisierung kommen.
Änderungen an nur einem System, die auch noch innerhalb eines Sprints fertiggestellt werden können stellen im Allgemeinen kein Problem dar. Sind aber mehrere Systeme und Dienste (SOA, Rest,…) an einer solchen Änderung beteiligt, die auch noch in unterschiedlichen Abteilungen angesiedelt sind, wird es kompliziert. Meist kann eine solche Änderung oder Erweiterungen nicht innerhalb des Zeitraums eines Sprints durchgeführt werden. Im Projekt trat eine solche Situation mehrfach auf.
Es stellt sich somit die Frage, wann die Änderungen an den verschiedenen Codeteilen auf der Zeitschiene in den verschiedenen Systemen entwickelt werden und anschließend tatsächlich gemeinsam fertiggestellt werden. Nicht zu vergessen sind die ebenfalls notwendigen Tests.

Beispiel eines systemübergreifenden Features: Ein erweiterte Funktion bedingt eine Änderung im Zugriff auf den Content Server und gleichzeitig Änderungen an den vom Content Server ausgelieferten Dokumenten, die dynamisch mittels XSLT aufbereitet werden.

Lösung mittels Feature Branches
Die Lösung des Problems mittels Feature Branches hört sich anfangs einfach an. Von der aktuellen Mainline (dem Master Code Strang) wird ein Branch gezogen, auf dem das Feature bis zur Produktionsreife entwickelt wird. Anschließend werden die gemachten Änderungen vom Branch in die Mainline zurückgemerged.
Leider stellt es sich in der Praxis als erheblich schwieriger heraus. Je länger der Zeitraum zur Fertigstellung dauert, desto schwieriger gestaltet die die notwendige Zusammenführung der Branches. Dies ist in der Literatur als ‚Merge-Hölle' bekannt. Es existieren Projekte, bei denen ein Feature nicht in die Produktion übernommen wurde, da sich die verschiedenen Stränge in der Zwischenzeit zu weit auseinander bewegt haben.
Natürlich gibt es Lösungsstrategien, die dieses Phänomen abmildern (siehe FeatureBranch). Es allerdings kann nicht verschweigen werden, dass die Arbeitsweise mit Feature Branches eigentlich der Definition von Continuous Integration wiederspricht, die ja gerade davon ausgeht, das aller Code immer in der Mainline lebt.
Es gibt auch durchaus große Projekte, die mittels Feature Branches erfolgreich laufen. Ein Beispiel dazu ist das Linux Kernel.

Lösung mittels Feature Toggels
Wird ein Feature mit Toggles entwickelt, werden die Änderungen permanent in die Mainline des Code eingebaut, was die Merge-Hölle komplett umgeht. Man handelt sich aber leider auch ein paar Probleme ein, die den Einsatz dieser Technik nicht ganz einfach gestalten. Prinzipiell wird nicht fertiget Code in die Mainleine eingecheckt. Dies stellt einige Anforderungen an diesen Code. Um nur ein Paar zu nennen:

- Kompilierbarkeit muss gewährleistet sein -> Sonar
- Bestehende Funktionalität darf nicht gebrochen werden -> Test
- Die Funktionalität darf für den Benutzer nicht sichtbar werden, solange sie nicht als fertig gilt. -> Toggles

Auch die Testbarkeit der Anwendung muss im Auge behalten werden. Dabei sollte es reichen, die Anwendung mit allen Feature Toggles zu testen, die im Sprint fertiggestellt werden und alle Toggles angeschaltet. Werden beide Test erfolgreich bestanden, kann von der Korrektheit der Anwendung ausgegangen werden, ohne dass alle Kombinationen von Toggles gegeneinander getestet werden müssen (siehe FeatureToggle 2).
Es darf auch nicht vergessen werden, dass die Toggles auch wieder ausgebaut werden sollten, wenn ein Feature final fertiggestellt wurde. Sollen Features z.B. pro Kunde an/abgeschaltet werden, bleiben die Schalter im Code.

Zusammenfassung
Keine der Vorgehensweisen geht als klarer Sieger hervor. Es ist am Ende immer eine Entscheidung des PO und des Entwicklungsteam wie vorgegangen wird. Im beschriebenen Projekt wurden Feature Branches schnell verworfen, das die Schwierigkeiten mit auseinanderdriftendem Code zu groß wurden.
Feature Toggles erlauben Funktionalitäten parallel zur normalen Entwicklung einfach durchzuführen. Die Nachteile lassen sich beherrschen. Zusätzlich existieren Tools zur Unterstützung (Togglz, FT4J um nur einige zu nennen).

Referenzen:
Continuous Integration & Feature Branches: Tonis Pool, University of Tartu Faculty of Mathematics and Computer Sience, 2013
Feature Toggle, Wikipedia, https://de.wikipedia.org/wiki/Feature_Toggle
Feature Branch, Martin Fowler, http://martinfowler.com/bliki/FeatureBranch.html
Feature Toggle 2, Martin Fowler, http://martinfowler.com/bliki/FeatureToggle.html
Make Large Scale Changes Incrementally with Branch By Abstraction, jez, http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/
Experience Report: Branch by Feature,Sarah Taraporewalla's Technical Ramblings, http://sarahtaraporewalla.com/design/experience-report-branch-by-feature/
Experience Report: Feature Toggling, Sarah Taraporewalla's Technical Ramblings, http://sarahtaraporewalla.com/design/experience-report-feature-toggling/
Feature Flags Made Easy | Outbrain Techblog, Eran Harel, http://techblog.outbrain.com/2011/07/feature-flags-made-easy/

Kontaktadresse:
Timo Hahn
virtual 7 GmbH
Zeppelinstrasse 2
D-76185 Karlsruhe

Telefon: +49 (0) 721-61901759
Fax: +49 (0) 721-61901729
E-Mail timo.hahn@virtual7.de
Internet: www.virtual7.de