ENTAGO

# Test-Driven Development im agilen Umfeld

*Bei moderner Softwareentwicklung wird Test-Driven Development und «agile» oft in einem Atemzug ausgesprochen. Doch was bedeutet Test-Driven Development genau und wie hängt es mit agiler Softwareentwicklung zusammen? Welche Vorgehensweise gibt es, um Software automatisiert zu testen? Der vorliegende Artikel geht auf diese Fragen ein und beschreibt die entsprechenden Techniken.*

Die Softwareentwicklung findet heute immer häufiger im agilen Umfeld statt. Schon fast vergessen sind die Zeiten, in welchen Monate lang Spezifikationen ausgearbeitet wurden, diese dann geschätzt und anschließend von Entwicklungsteams bis hin zu Jahren implementiert wurden.

Daraus ist die Motivation entstanden, kürzere Zyklen der Entwicklung durchzuführen und daraus resultierende Artefakte durch Fachbereiche und letztendlich durch Benutzer regelmässig abnehmen und verifizieren zu lassen.

Wenn beispielsweise solche Software-Artefakte innerhalb von 2 bis 4 wöchigen Zyklen produktiv bereitgestellt werden, ergibt sich daraus der grosse Vorteil, den Kurs der Entwicklung schneller und flexibler zu korrigieren. Des Weiteren lässt sich während der Entwicklung auch die Priorisierung ändern. So ist es möglich, rasch User-Feedback in die Priorisierung und Gestaltung von Anforderungen einfliessen zu lassen.

Daraus ergibt sich allerdings auch eine neue Herausforderung: Wenn alle 2 bis 4 Wochen ein neues Artefakt bereitgestellt wird, sind wochenlange Abnahmephasen durch Testing-Teams und Fachbereiche sehr teurer und logistisch nicht sinnvoll. Es soll also so viel wie möglich automatisiert getestet werden. Eine möglichst hohe Testabdeckung ist wünschenswert, das steigert dann ebenfalls die Sicherheit bei der Produktiv-Schaltung von neuen Software-Artefakten.

Eine Vorgehensweise namens «Test-Driven Development» hat sich daraus etabliert. Bevor die eigentliche Funktionalität implementiert wird, werden Testfälle für die noch zu erstellende Funktionalität geschrieben. Da die Funktion selbst noch nicht existiert, schlagen die Testfälle zunächst fehl. Die sukzessive Implementierung der Funktion sollte immer mehr Testfälle erfüllen, bis letztendlich alles «grün» ist.

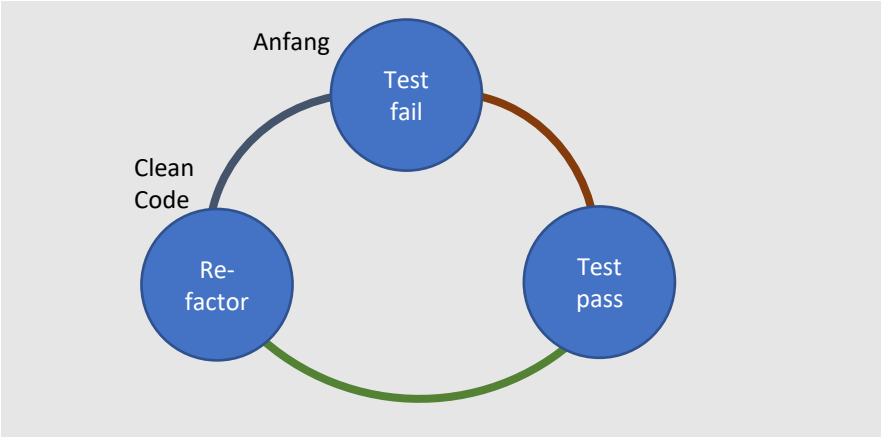


*Abbildung 1: Test-Driven Development*

ENTAGO

Der Einsatz von Test-Driven Development bietet einen weiteren grossen Vorteil: Fachbereich und Entwicklung – oder im agilen Jargon «Product Owner» und «Scrum Team» – sprechen die gleiche Sprache. So werden Anforderungen für gewöhnlich in der Vorgehensweise «Given-When-Then» formuliert.

- «Given» beschreibt den Zustand des Systems, der erfüllt sein muss, bevor eine bestimmte Operation durchgeführt wird oder ein Ereignis eintritt
- «When» beschreibt die Operation oder das Ereignis
- «Then» beschreibt den Zustand des Systems nach Eintreten des Ereignisses beziehungsweise nach Durchführen der Operation

Der Fachbereich benutzt diese «Sprache» bei der textuellen Erfassung der Anforderungen, das Entwicklungsteam kann daraus direkt Testfälle ableiten und implementieren.

In der Abbildung 2 ist ein Beispiel für solch ein Given/When/Then Kriterium gegeben.

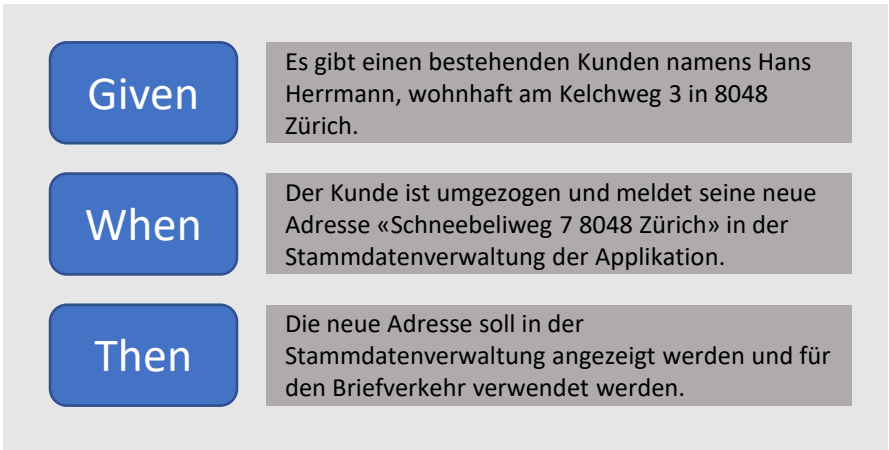


*Abbildung 2: Beispiel für ein Given/When/Then Kriterium*

## JUnit

JUnit ist die verbreiteste Bibliothek, um solche Testfälle in Java zu implementieren. Weiterhin wird es standardmässig von den grossen Frameworks wie beispielsweise Spring Boot verwendet.

## Unit-Test

Der Name JUnit leitet sich von Java und «Unit» ab. Ein «Unit-Test» ist ein Test, der eine bestimmte «Unit», also einen bestimmten Bestandteil der Applikation – weitläufig eine Klasse – testet. Beispielsweise kann eine Applikation eine Klasse implementieren, die Adressdaten ändert. Ein eingehendes Objekt für die Methode einer solchen Klasse wäre beispielsweise eine Benutzer-ID sowie ein Adressdatensatz, die Ausgabe der Methode könnte eine Information sein, ob die Adresse erfolgreich geändert wurde. Um den Gedanken fortzusetzen, könnte eine Adressänderung fehlschlagen, wenn ein Kunde angibt, dass er auf dem Mond lebt (Eingabe-Validierungsfehler). Um diese Fälle und den Fall aus Abbildung 2 automatisiert zu testen, bietet sich eine Test-Klasse mit JUnit an.

## Ende-zu-Ende Test

Ende-zu-Ende Tests bezeichnen Tests, die eine Funktion vom ersten Eingangspunkt bis zum letzten Ausgangspunkt testen. Bei Webapplikationen ist oft der Einstiegspunkt eine Web-Ressource wie beispielsweise ein Webservice, der Ausgangspunkt eine Veränderung in der Persistenz-Schicht wie einer Datenbank. Der vorher beschriebene Unit-Test würde beispielsweise jeden Bestandteil einzeln testen

- Funktioniert der Webservice? Ist dieser aufrufbar?
- Funktioniert die Business-Logik? Werden Adressänderungen validiert und gespeichert?
- Funktioniert die Persistenz? Wird ein neuer Adressdatensatz in der Datenbank persistiert?

Ein Ende-zu-Ende Test würde die Verkettung der einzelnen Bestandteile testen. Auch für solche Ende-zu-Ende Tests eignet sich JUnit hervorragend. Der gesamte Kontext der Applikation kann in einem JUnit-Test gestartet werden, dann kann durch JUnit innerhalb dieses Kontexts der Webservice aufgerufen werden. Abschliessend wird durch JUnit in der Datenbank der Datensatz geprüft.

## Integrations-Tests

Falls der Code der Applikation bereits flächendeckend durch Unit- und Ende-zu-Ende Tests abgedeckt wird, kann das Entwicklungsteam davon ausgehen, dass Funktionen richtig ausgeführt werden.

Wie kann nun aber die Integration in die Systemlandschaft automatisiert getestet werden? Durch die vorhergehenden Tests wissen wir zwar, dass das System prinzipiell Ende-zu-Ende funktioniert.

Allerdings funktioniert der Test bis dato nur Ende-zu-Ende im Kontext einer Testklasse.

Die Erstellung eines Integrationstests bietet die Möglichkeit, ebenfalls sicherzugehen, dass das zu testende System von anderen Systemen aufrufbar ist.

Erweitern wir den beschriebenen Anwendungsfall der Adressänderung. Neben dem Persistieren der Adresse informieren wir auch andere Systeme über diese Änderung per «Event». Das «Event» wird in einer Message-Queue veröffentlicht (Publishing). Andere Systeme, die an Adressänderungen interessiert sind, können diese Events lesen (Subscription).

Ein Beispiel für den Integrationstest des beschriebenen Falls ist in Abbildung 3 zu sehen. Der Integrationstest ist eine eigene Applikation, welche den Webservice zur Änderung der Adresse aufruft. Erwartet wird, dass die zu testende Applikation ein Event bezüglich der Adress-änderung veröffentlicht. Der Integrationstest startet ebenfalls eine Subscription auf der Message-Queue, sobald das erwartete Event gelesen und verarbeitet wird, ist der Integrationstest erfolgreich abgeschlossen.

Falls das erwartete Event nicht innerhalb einer bestimmten Zeit ankommt, kann man den Test fehlschlagen lassen.

Durch diese Art von Tests kann automatisiert sichergestellt werden, dass die Applikation integriert funktioniert. Solche Tests lassen sich auch automatisiert auf Vorproduktionssystemen, welche Produktionssysteme spiegeln, ausführen.

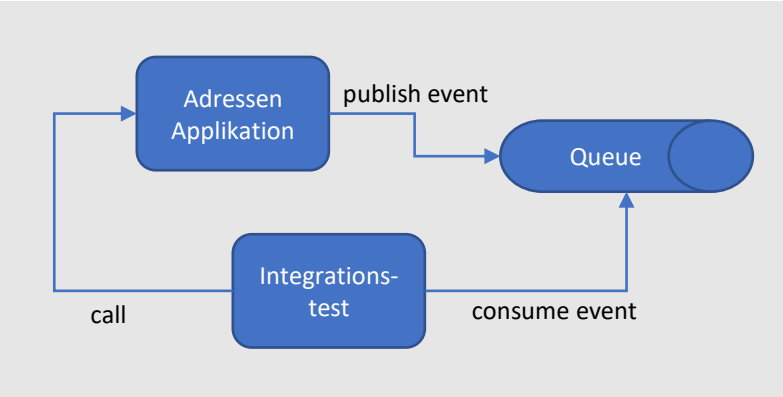


*Abbildung 3: Beispiel für einen Integrationstest*

## Die richtige Test-Strategie – Der Schlüssel zum Erfolg

Die Vorteile beim Einsatz von Test-Driven Development liegen auf der Hand:

- Inkremente können oft produktiv geschaltet werden. Automatisierte Tests geben dem Entwicklungsteam die Sicherheit, das «Richtige» aufzuschalten;
- Gerade beim agilen Vorgehen ist oft ein Refactoring notwendig. In vielen Entwicklungsteams gilt gemeinhin auch die Prämisse, dass ein agiles Vorgehen in der Software-Entwicklung ohne Refactoring unmöglich sei. Bei der Entwicklung werden Annahmen getroffen, die sich später ändern oder falsch waren. Dann müssen Bestandteile in der Implementierung geändert werden. Wenn aber die Testfälle immer noch «wahr» sind, können sie nah Durchführung von Änderungen im Progammcode erneut ausgeführt werden und stellen somit den korrekten Zustand der Applikation sicher;
- Bei der Verwendung von Continuous Integration und Continuous Development werden automatisierte Tests zwingend benötigt.

Eventuell regt der Artikel an, die aktuelle Systemlandschaft neu zu betrachten und zu eruieren, wo mehr «Test-Driven Development» sinnvoll sein kann. Vielleicht ist auch ein Coaching für die Einführung von agilen Vorgehensweisen hilfreich?

Gerne besprechen wir mit Ihnen zusammen das Vorgehen für die Erarbeitung von Test-Driven-Development in Ihrer Software-Entwicklung. Kontaktieren Sie uns unverbindlich.

Entago AG
Buckhauserstrasse 34
CH-8048 Zurich
info@entago.ch
www.entago.ch