Jan Jürjens, Kurt Schneider (Hrsg.): Software Engineering 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 121

# Agiles Testen: Auch Anwender können Unit Tests

Ursula Oesing [1] Alexander Georgiev Jahn Langenbrink Stefan Jonker [2]

**Abstract:** In der Softwareentwicklung werden insbesondere in agilen Projekten diverse Unit Tests im Qualitätssicherungsprozess intensiv eingesetzt. Ebenso ist die Notwendigkeit erkannt worden, Kunden, Anwender oder Product Owner, in diesem Beitrag als Wissensträger bezeichnet, so früh wie möglich in den Prozess des Softwaretestens, am besten bei der Durchführung von Unit Tests, einzubinden, um Anforderungen wie kurze Projektentwicklungszyklen und Flexibilität gegenüber Kundenwünschen erfüllen zu können. In der Praxis steigen aber Kunden und Anwender nach wie vor häufig wegen fehlender Programmierkenntnisse frühestens bei der Durchführung von Akzeptanztests in den Prozess des Softwaretestens ein, zu spät, um auf die entdeckten Fehler ohne größeren Zeitverlust reagieren zu können. Product Owner haben zusätzlich das Interesse, dass die Transparenz in der Qualitätssicherung erhöht wird. Um die Situation entscheidend zu verbessern, wird in diesem Beitrag ein Prozess für die Durchführung von Unit Tests vorgestellt, welcher beschreibt, wie Entwickler, Kunden und Anwender gemeinsam Unit Tests erstellen können und jeweils ihre Expertise, Programmierkenntnisse auf der einen Seite und fachliches Know How auf der anderen Seite, einfließen lassen können. Für einen Product Owner werden Grundlagen für einen Überblick über den aktuellen Stand der Unit Tests bereitgestellt. Zur Unterstützung dieses Prozesses wurde die Java-Anwendung KBUnit (Knowledge Based Unit Testing Application) entwickelt, welche ebenfalls in diesem Beitrag vorgestellt wird. Die Integration von KBUnit in die täglichen Arbeitsabläufe in der Praxis wird berücksichtigt.

**Keywords:** Unit Testfälle ohne Programmierung, Agiles Testen, Testautomatisierung.

## 1 Ausgangssituation und Zielstellung

Die Funktionalität einer Anwendung soll in einer möglichst frühen Projektphase von Wissensträgern getestet werden können. Beispielsweise möchte ein Finanzdienstleister eine Anwendung entwickeln, welche Berechnungen zu seinen Finanzprodukten erlaubt. Eine Teilfunktion sei die Berechnung der Gesamtschuld, wobei der Anwender unter anderem die Höhe des Darlehens, den Zinssatz und die Laufzeit vorgibt. Entwickler können Unit Tests hierzu implementieren, aber sie kennen die Fachlichkeit nicht genügend, hier beispielsweise den maximal zugelassenen Zinssatz. Ausreichende Fachkenntnisse hat der Wissensträger, der aber wegen fehlender Programmierkenntnisse und fehlender Kenntnisse zur Entwicklungsumgebung nicht die Unit Tests implementieren kann. Ziel ist es, einen Prozess zur Durchführung von Unit Tests zu ermöglichen, in welchem Entwickler und Wissensträger gemeinsam Testfälle entwickeln, und in welchem die Transparenz bezüglich des aktuellen Stands der Tests erhöht wird.

[1] Hochschule Bochum, Fachbereich Elektrotechnik und Informatik, Lennershofstraße 140, 44801 Bochum, ursula.oesing@hs-bochum.de
[2] Hochschule Bochum, Fachbereich Elektrotechnik und Informatik, Lennershofstraße 140, 44801 Bochum, stefan.jonker@hs-bochum.de

## 2 Lösungsansatz

Der Prozess zur Durchführung von Unit Tests wird in Abbildung 1 dargestellt.

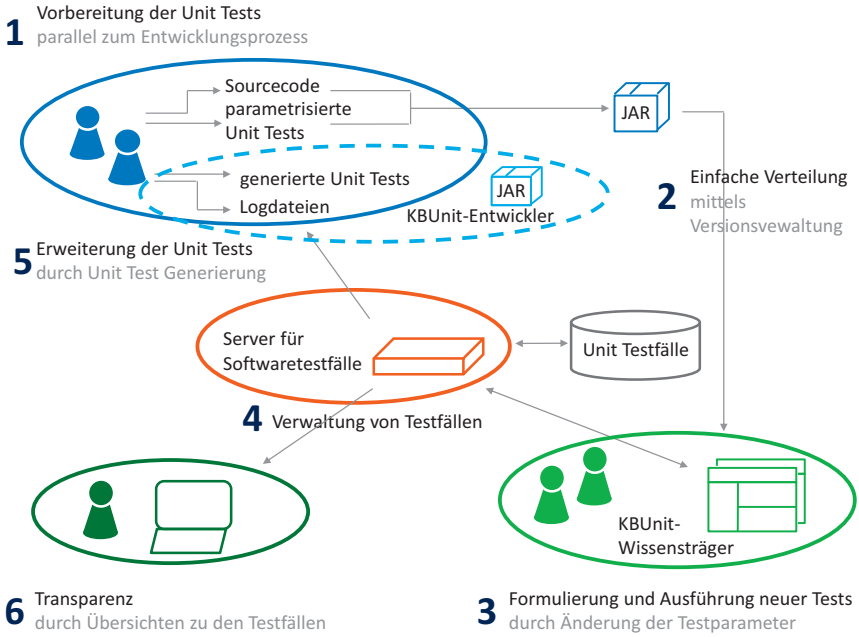


Abb. 1: Prozess zur Durchführung von Unit Tests

1. **Vorbereitung der Unit Tests:** Die Entwickler implementieren eine Funktionalität und zu dieser einen Unit Test, welcher einen Testfall überprüft. Dieser Unit Test enthält Parameter, die von dem Wissensträger variiert werden sollen. Die Entwickler brauchen zu jeder Funktion beziehungsweise Methode nur einen Unit Test zu schreiben.
2. **Einfache Verteilung der Unit Tests:** Der Sourcecode wird zusammen mit den Unit Tests den Wissensträgern zugänglich gemacht.
3. **Formulierung und Ausführung neuer Tests:** Die Wissensträger greifen auf die von den Entwicklern entworfenen Testfälle zu und erstellen weitere Testfälle, die aus ihrer Sicht geprüft werden müssen, indem sie die Werte der Parameter variieren. Sie können sämtliche Testfälle auch abspielen.
4. **Verwaltung von Testfällen:** Die Wissensträger speichern die neuen Testfälle inklusive deren Ergebnis in einer Datenbank und können diese auch verwalten.
5. **Erweiterung der Unit Tests:** Die Entwickler haben die Möglichkeit, auf die neuen Testfälle zuzugreifen. Sie können diese in ihrer Entwicklungsumgebung abspielen und entsprechende Logdateien erstellen lassen. Sie können ebenfalls sich zu den neuen Testfällen Unit Tests generieren lassen.

6. **Transparenz:** Der Product Owner hat zusätzlich die Möglichkeit, sich den Stand zu den Softwaretests anzeigen zu lassen.

## 3 Funktionsweise von KBUnit

Die in Java geschriebene Anwendung KBUnit, von welcher eine frühe Version auf der Software Engineering 2013 vorgestellt wurde, unterstützte die Schritte 1 bis 3 der Abbildung 1 für JUnit Tests. Insbesondere wurde bereits vorgestellt, wie KBUnit im Schritt 3 die Wissensträger durch die Generierung von Testfällen unterstützt und dass auch bereits die Möglichkeit existierte, Exceptions zu testen, also auch Testfälle, die nicht funktionieren dürfen.

KBUnit ist weiterentwickelt worden. Das Tool unterstützt jetzt auch die Schritte 4 und 5 des im Abschnitt 1 beschriebenen Prozesses. Die Entwickler können jetzt auf die Testfälle, welche die Wissensträger erstellen, aus ihrer Entwicklungsumgebung heraus zugreifen. Sie können die neuen Testfälle abspielen (Logger). Als Ergebnis erhalten sie Logdateien zu den Testfällen inklusive der Testergebnisse. Weiterhin können Sie sich neue JUnit Tests zu den Testfällen generieren lassen (Runner) und diese dann in ihrer Entwicklungsumgebung abspielen.

Die folgende Erläuterung erfolgt am Beispiel der im Abschnitt 1 genannten Anwendung zu Finanzprodukten. In diesem Beispiel wurde zu einer Methode zur Berechnung der Gesamtschuld einer Klasse Tilungsdarlehen ein JUnit Test testBerechneGesamtschuld in der Klasse TilgungsdarlehenTest implementiert und an den Wissensträger als jar-Datei weitergeleitet. Der Wissensträger hat einen Testfall mit einem negativen Darlehen erstellt, so dass er eine Exception erwartet. Dieser Testfall wurde erfolgreich ausgeführt, die gewünschte Exception wurde geworfen. Der Testfall wurde in der Datenbank gespeichert, auf welche der Entwickler zugreifen kann. Letzterer hat den Testfall abgespielt und eine Logdatei TilgungsdarlehenTest_160920_094600.xml erhalten, siehe Abbildung 2.

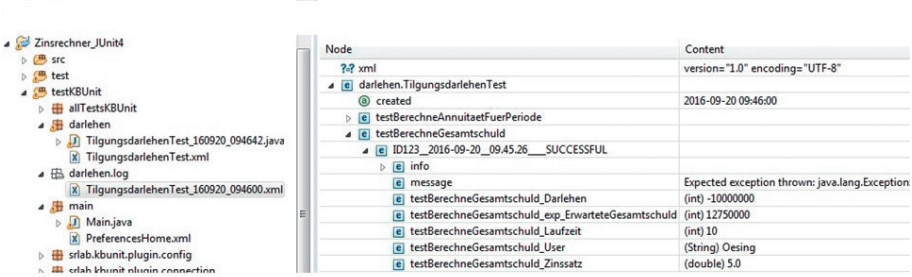


Abb. 2: Ergebnis des Loggers: eine xml-Datei mit Informationen zu den Testfällen

Weiterhin hat der Entwickler sich einen neuen JUnit Test generieren lassen, siehe Abbildung 2. Die JUnit Testklasse erhält als Namen den der ursprünglichen JUnit Testklasse, hier TilgungsdarlehenTest.java, zusätzlich mit dem Datum und der Uhrzeit der Generierung versehen, hier TilgungsdarlehenTest_160920_094642.java, siehe Abbildung 2. Dieser

JUnit Test enthält die Testparameterwerte, die der Wissensträger vorgegeben hat. Attribute der Testklasse, hier beispielsweise testBerechneGesamtschuld_Darlehen, werden vor der Ausführung des JUnit Tests mit diesen Werten belegt. Die ursprüngliche Testmethode wird dann mit den veränderten Testparameterwerten durchgeführt, siehe Listing 1.

```
@Test (expected = Exception.class)
public void testBerechneGesamtschuld_123()
    throws Exception {
    testBerechneGesamtschuld_Darlehen = -10000000;
    testBerechneGesamtschuld_exp_ErwarteteGesamtschuld
        = 12750000;
    testBerechneGesamtschuld_Laufzeit = 10;
    testBerechneGesamtschuld_User = "Oesing";
    testBerechneGesamtschuld_Zinssatz = 5.0;
    testBerechneGesamtschuld();
}
```

List. 1: Ergebnis des Runners: ein generierter JUnit Test zu einem Testfall

Die Weiterentwicklung von KBUnit erfüllt die Voraussetzungen, um in die alltägliche Projektarbeit integriert zu werden. Das Tool kann über das Build-Management-Tool Maven eingebunden werden, so dass das Abspielen des Loggers und des Runners automatisiert erfolgen. Die kontinuierliche Integration wird mittels Jenkins und Tools zur Versionsverwaltung unterstützt, die Verteilung des Sourcecodes und der JUnit Tests als jar-Datei an die Wissensträger kann beispielsweise über Jenkins erfolgen. Die Client-Server-Schnittstellen wurden auf Basis von RESTful Web Services umgesetzt.

## 4 Bewertung und Ausblick

Der vorgestellte Prozess und das vorgestellte Tool, welches diesen unterstützt, ermöglichen Entwicklern und Wissensträgern die gemeinsame Durchführung von Unit Tests, so dass hier eine bisher bestehende Lücke beim Testen von Software geschlossen werden kann. Das Tool unterstützt im hohen Maße die Testautomatisierung.

Nicht berücksichtigt wurde bisher die nähere Beschreibung der Auswertungen der Tests für den Product Owner. Die Daten zu den Testfällen stehen zur Verfügung, es müssen aber noch konkrete Übersichten entwickelt werden. Auf welche Art und Weise soll der Stand zu Unit Tests aufbereitet werden? Welche Informationen sind für den Product Owner von Interesse? Welche Auswertungen sollen vorgenommen werden? Hier könnte beispielsweise die Testabdeckung überprüft und abgebildet werden.