Drei Methoden, ein Ziel: Testautomatisierung mit BDD, MBT und KDT im Vergleich

Christian Brandes[1], Benedikt Eberhardinger[2], David Faragó[3], Mario Friske[4], Baris Güldali[5], Andrej Pietschker[6]
[1] imbus AG, [2] Universität Augsburg, [3] QPR-Technologies, [4] T-Systems International GmbH,
[5] Universität Paderborn, [6] Giesecke & Devrient GmbH

1 Einleitung

Behavior-Driven Development (BDD) hat sich – nicht zuletzt durch die Ausbreitung agiler Entwicklungsprozesse – in IT-Projekten als Methode zur Spezifikation automatisierter System- und Akzeptanztests etabliert. Andere Ansätze wie Model-Based Testing (MBT) und Keyword-Driven Testing (KDT), die vergleichbare Ziele verfolgen, sind darüber in ihrer Wahrnehmung und Verbreitung scheinbar ins Hintertreffen geraten. Woran liegt das? Worin unterscheiden sich die drei Ansätze? Wo haben sie ihre Stärken? In welchen Situationen sind sie wie gut geeignet? Und lassen sie sich vielleicht sogar nutzbringend kombinieren?

Alle diese Fragen haben wir im Arbeitskeis „Model-Based Testing (MBT)“ der GI-Fachgruppe „Testen, Analyse und Verifikation (TAV)“ seit Anfang 2015 intensiv diskutiert. Als erstes Ergebnis stellen wir in diesem Beitrag die drei genannten Ansätze zunächst kurz vor, wenden sie dann zur Veranschaulichung auf ein gemeinsames Beispiel an und nehmen abschließend eine vergleichende Bewertung vor, die sich vorrangig auf die Projekterfahrungen der Autoren stützt.

2 Vorstellung der drei Methoden

Behavior-Driven Development (BDD) wurde 2006 von Dan North eingeführt, um Test-Driven-Development (TDD) zu vereinfachen [6]. BDD kombiniert Test-Driven Development (TDD), Object-Oriented Analysis (OOA), Object-Oriented Design (OOD) und Domain-Driven Design (DDD), um eine einheitliche Sprache und Vorgehensweise von der Anforderungsanalyse bis zur Implementierung zu bieten. Die einheitliche Sprache kann von Qualitätsmanagern, Domänenexperten und Softwareentwicklern gemeinsam verstanden und benutzt werden.

BDD-Anforderungen beschreiben deklarativ das Verhalten häufig in der textuellen Beschreibungssprache Gherkin:

> **GEGEBEN** initialer Kontext/Zustand (Vorbedingung)
> **WENN** Ereignis/Aktion auftritt (zu testendes Verhalten)
> **DANN** zu beobachtende Auswirkung (Nachbedingung)

Solche Beschreibungssprachen sind semi-formal, mit den fettgedruckten Wörtern als vordefinierte Schlüsselwörter. Durch die einfache Grammatik und die natürlichsprachlichen Schlüsselwörter können die BDD-Anforderungen von fachlichen Testern verstanden und manuell ausgeführt werden. Aber auch eine Testautomatisierung wird möglich: BDD-Tools, wie z.B. Cucumber [1], können aus jedem Szenario automatisiert Testfälle erzeugen, bei denen jede nicht fettgedruckte Phrase auf einen parametrisierten Funktionsaufruf abgebildet wird. Dazu werden die Schlüsselwörter schrittweise durchlaufen und abgearbeitet. In Cucumber werden hierfür Ruby-Skripts eingesetzt, welche die Schritte (Schlüsselwörter) durch automatisierte Programmaufrufe definieren und somit die BDD Beschreibung automatisiert ausführbar machen.

Model-Based Testing (MBT) bezeichnet die Nutzung von hauptsächlich grafischen Modellen, z.B. UML-Diagrammen, für die Generierung von Testartefakten im Testprozess, insbesondere die Generierung von Testfällen. Auf diese Weise werden Aktivitäten des Testdesigns (teil-)automatisiert. Es existieren verschiedene MBT-Varianten (siehe [7]), die sich hinsichtlich der eingesetzten Modelle (vorhandene Systemmodelle und/oder eigenständige Testmodelle), dem Umfang der Generierung (abstrakte oder konkrete Testfälle, Testdaten oder Testorakel) und dem Automatisierungsgrad unterscheiden. MBT muss also jeweils individuell in den vorhandenen Entwicklungs- und Testprozess eingebettet werden.

Keyword-Driven Testing (KDT) bezeichnet die Spezifikation von Testfällen mit Hilfe sogenannter wiederverwendbarer Schlüsselwörter (engl. Keywords), die jeweils für einen atomaren oder zusammengesetzten Testschritt stehen, wie z.B. Eingaben ins System under Test (SUT) oder Soll-Ist-Vergleiche. Ergebnis ist eine formale Notation für Testfälle, die als Domain Specific Language (DSL) für Fachtester fungiert und zugleich den Testautomatisierern eine wartungsarme Automatisierungsarchitektur bereitstellt. Somit werden Testschritte nicht in freier Prosa formuliert, sondern mit Hilfe eines exakt definierten Vokabulars, den Keywords. Erster Schritt beim KDT ist das Keyword-Design: Die Keywords werden so entwickelt, dass sie von Entwicklern und Domänenexperten gleichermaßen verstanden und benutzt werden können. Laut ISO/IEC/IEEE 29119-5 [5] sind verschiedene Informationsquellen hierfür denkbar. Fachliche Keywords (ohne Bezug zur Implementierung) sollten im Imperativ, d.h. als „Anweisung“ für den Testdurchführer,

formuliert sein und so gewählt werden, dass sie in möglichst vielen Testfällen verwendet werden können. Auf Basis der definierten Keywords können einerseits Testfälle komponiert werden und auf der anderen Seite automatische Testprozeduren erstellt werden, welche dann zur Ausführung der Testfälle eingesetzt werden.

3 Die Methoden im Einsatz an einem Beispiel

Als Beispiel für die Anwendung der Ansätze haben wir einen Geldautomaten (Fallstudie) gewählt, die auf [4] basiert. Abbildung 1 zeigt einige Anwendungsfälle und Akteure eines typischen Bankautomaten. Nach einer exemplarischen Beschreibung einer Bankautomatenfunktion zeigen wir in diesem Abschnitt, wie man bei den drei Techniken vorgehen würde, um Testdesign bzw. Testrealisierung umzusetzen.

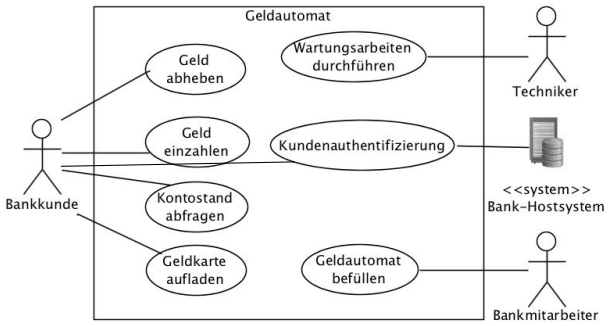

Abbildung 1: UML Anwendungsfalldiagramm für einen Geldautomaten.

Zum Anwendungsfall „Geld abheben“ existiert folgende Beschreibung [4]:

> Nachdem der Kunde die Karte eingeschoben und die PIN und den Betrag eingegeben hat, wird der Betrag vom Konto abgebucht und die Karte und das Geld ausgegeben. Falls die PIN das 1. oder 2. Mal falsch eingegeben wurde, wird der Kunde benachrichtigt und die PIN wird erneut eingegeben. Falls die PIN ein 3. Mal falsch eingegeben wurde, protokolliert das System den Versuch, zieht die Karte ein und bricht die Kommunikation ab. Falls die Karte nicht gültig ist, wird dies dem Kunden mitgeteilt und die Karte wieder ausgegeben.

3.1 BDD am Beispiel

Eine BDD-User Story und ein dazu gehöriges Szenario bzw. Testfall zum Anwendungsfall „Geld abheben“ könnten wie folgt aussehen:

> **User Story:** Bankkunde hebt Geld ab
> **Als** Bankkunde,
> **möchte ich** vom Geldautomaten Geld abheben
> **damit** ich unabhängig von den Filial-Öffnungszeiten auf mein Guthaben in Form von Bargeld zugreifen kann.
>
> **Szenario:** Gewünschter Geldbetrag ist auf dem Konto vorhanden
> **Gegeben** das Konto verfügt über den gewünschten Betrag
> **Und** die Bankkarte ist gültig
> **Und** der Bankkunde ist durch PIN authentifiziert
> **Und** der Bankautomat enthält das Geld für den erwünschten Betrag
> **Wenn** der Kunde den gewünschten Betrag abhebt
> **Dann** wird der Betrag ausgezahlt
> **Und** das Konto mit dem Betrag belastet
> **Und** die Bankkarte zurückgegeben

Das oben beschriebene Szenario ist von der Abstraktion her so formuliert, dass es von fachlichen Testern erstellt und verstanden werden kann. Alternativ kann aber auch detaillierter spezifiziert werden, so dass auch Personen die Tests durchführen können, die über weniger domänenspezifisches Wissen verfügen. Als Beispiel könnte man die Aktion unter „Wenn“ auch wie folgt detaillierter spezifizieren:

> ...
> **Wenn** der Kunde die Karte einführt
> **Und** der Kunde die PIN eingibt
> **Und** die Transkation „Geld abheben“ wählt
> **Und** den Betrag eingibt
> ...

3.2 MBT am Beispiel

Das Ziel des modellbasierten Testens des Geldautomaten ist, aus den natürlichsprachlichen Anforderungen ein formales generisches Modell zu erstellen und aus diesem Modell systematisch Testfälle abzuleiten. Die Systematik besteht darin, die Modellelemente algorithmisch abzudecken und diese in Testfallelemente zu übersetzen. Das Modellabdeckungsmaß dient dabei als Testendekriterium.

Basierend auf der Vorgehensweise der objektorientierten Analyse (OOAD) [2], haben wir aus der funktionalen Anforderung für „Geld abheben“ ein Verhaltensmodell mittels UML-Aktivitätsdiagramm erstellt (siehe Abbildung 2). Dabei werden Standard- und Alternativabläufe während einer Interaktion zwischen einem Bankkunden und einem Geldautomaten für den Anwendungsfall „Geld abheben“ modelliert.

Der Algorithmus für die Testfallableitung soll das Testkriterium „Kantenabdeckung“ erfüllen, d.h., alle Übergänge zwischen den Aktivitäten müssen mindestens einmal in einem Testfall berücksichtigt werden.

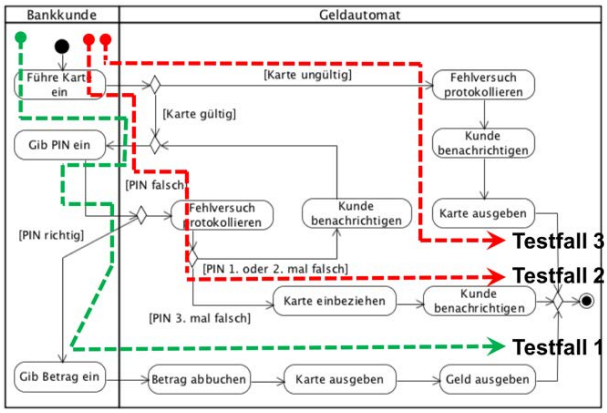

Abbildung 2: Verhaltensmodell als UML Aktivitätsdiagramm für den Anwendungsfall „Geld abheben“ mit drei exemplarischen Testfällen.

Die Abbildung zeigt drei Testfälle (der positive Testfall 1 und die negativen Testfälle 2 und 3), die von 21 Übergängen (Kanten) 19 abdecken.

Das in diesem Detailgrad formulierte Szenario könnte durch die Verwendung von KDT-Keywords automatisiert werden, wie es im folgenden Abschnitt erläutert wird.

3.3 KDT am Beispiel

Im vorliegenden Beispiel konnten aus der funktionalen Beschreibung folgende fachliche Keywords für KDT abgeleitet werden:

Führe Karte ein (mit Parameter „gültig/ungültig“)
Entnehme Karte
Gib PIN ein (mit Parameter <PIN> oder den beiden Äquivalenzklassen „gültige/ungültige PIN“)
Wähle Transaktion (mögliche Werte: „Abheben“, „Kontostand“, ...)
Gib Betrag ein (<Betrag>)
Entnehme Geld (<Betrag>)
Breche Vorgang ab

Zu beachten dabei ist, dass wir die Keywords – mit Blick auf Testobjekt und Testziel – aus der Perspektive des Testers definieren. Ein „interner“ Arbeitsschritt wie „Validiere PIN“ taucht hier nicht auf, könnte aber ein Keyword an anderer Stelle im Testvorgehen sein. Anders gesagt: Keywords in diesem Beispiel ergeben sich aus der Sicht des Endnutzers, dem Bankkunden.

Zweiter Schritt nach dem Keyword-Design ist die Spezifikation von Testfällen unter Verwendung der definierten Keywords. Ein Testfall, der dem Standardfall wie im BDD-Szenario oder dem Testfall 1 im Verhaltensmodell entspricht, wäre:

Führe Karte ein („gültig“)
Gib PIN ein („0815“)
Wähle Transaktion („Abheben“)
Gib Betrag ein („500“)
Entnehme Karte
Entnehme Geld („500“).

Man sieht, dass KDT zu Testfällen führt, die nicht nur von Fachtestern, sondern auch von Testern mit wenig fachlichem Hintergrund, ähnlich wie beim BDD-Beispiel, völlig problemlos verstanden und manuell durchgeführt werden können. Der große Nutzen liegt aber – neben der Wohldefiniertheit der Spezifikation – in der Automatisierbarkeit. Abhängig von der gewählten Implementierung und dem Automatisierungswerkzeug entstehen hierzu Automatisierungsskripte für jedes Keyword.

Eine weitere Erkenntnis ist, dass KDT genutzt werden kann, um die Testautomatisierung bei BDD zu implementieren, wenn die BDD-Szenarien mit den gleichen Keywords wie bei KDT spezifiziert sind, wie beim zweiten BDD-Beispiel und dem KDT-Beispiel.

4 Vergleich und praktische Erfahrungen

Wichtige Unterschiede zwischen den drei Methoden lassen sich entlang zweier Dimensionen gut darstellen: der *Formalitätsgrad* einerseits und die adressierte *Phase des Testprozesses* andererseits:

MBT ist im Gegensatz zu KDT & BDD eine Testdesignmethode. Werden Tests eigens in Modellform spezifiziert (sog. Testmodelle), bekommt MBT zusätzlich den Charakter einer Realisierungsmethode. KDT und BDD hingegen sind reine „Notationen“ für Tests, unterstützen aber nicht bei deren inhaltlichem Design.

Durch den Einsatz von Modellen hat MBT per se einen meist hohen Formalisierungsgrad; dies kann in der Beschriftung von Modellelementen variieren (z.B. können die Knoten in Ablaufdiagrammen Prosatext beinhalten oder definierte Keywords). KDT hingegen ist immer hochformal: Nur die definierten Keywords sind erlaubt. BDD bietet wie MBT Abstufungen: Jenseits von GEGEBEN-WENN-DANN kann wahlweise Freitext stehen, aber auch Keywords [3]. Bei all dem gilt: Je formaler (d.h. je weniger Freitext), desto größer das Generierungs- und Automatisierungs-Potenzial. KDT unterstützt als Automatisierungs-„Enabler“ also sowohl MBT als auch BDD, wenn diese jeweils formal ausgeprägt sein sollen.

Hinsichtlich Teststufen und Testzielen zeichnen sich ebenfalls Unterschiede ab: BDD fokussiert ausschließlich auf die fachliche Benutzersicht und unterstützt dadurch System- und Abnahmetests sehr gut. KDT und MBT hingegen können sowohl fachliche als auch technische Sichten auf das SUT abbilden und dessen unterschiedliche Schnittstellen ansprechen. Dadurch

wird eine Unterstützung von mehreren Teststufen von Komponententest bis zum Abnahmetest möglich.

Unterschiede gibt es beim Thema Skills und Tools: Hier stellt MBT die höchsten Anforderungen, weil Modellierungsskills und spezielle Skills für Werkzeuge zur Bearbeitung von Modellen und zur Generierung von Testartefakten benötigt werden. BDD hingegen verlangt bei beidem scheinbar weniger: Tests können in Freitext beschrieben werden, geeignete Werkzeuge sind in der „eh-da"-Kategorie zu finden (etwa ein Wiki oder die üblichen Office-Verdächtigen). KDT wird in der Praxis nur breite Akzeptanz finden, wenn Werkzeuge bei der korrekten Eingabe und Verwendung definierter Keywords unterstützen. Außerdem verlangt der Einsatz die Rolle eines Keyword-Designers (was auch für BDD gilt, wenn Keywords genutzt werden sollen).

Spannend wird es bei Aufwand und Kosten. Beginnen wir mit dem initialen Aufwand zur Einführung der Methoden: Verglichen mit MBT und auch KDT ist die Anfangsinvestition bei informellem BDD gering. KDT verlangt die Erstellung eines Keyword-Katalogs und Toolunterstützung bei der Eingabe, Wiederverwendung und Parametrisierung von Keywords. MBT muss sich, um den gewünschten Nutzen zu erzielen, mit Fragen des modellbasierten Testprozesses, Metamodellen, Editoren, Transformatoren, Generatoren und Schnittstellen beschäftigen, was folglich zu den höchsten Kosten in dieser Kategorie führt.

Das Bild ändert sich nach unserer Erfahrung aber bei den laufenden Kosten: Ist MBT etabliert, lauten die Hauptaufgaben Modellanpassung und Neugenerierung. Ähnliches gilt für KDT, was durch extrem hohe Modularisierung die laufenden Kosten ebenfalls kontrollierbar hält. BDD hingegen kann zum Pulverfass werden, wenn es um die Verwaltung mehrerer hundert oder tausend Szenarien, um deren Tracing zu den User Stories und Keywords und um Impact Analysis geht.

Ähnlich gestaltet sich der Vergleich hinsichtlich Skalierbarkeit und Wiederverwendung: Sowohl MBT als auch KDT bauen auf einer sauber strukturierte Modularisierung auf, welche sich im Projektverlauf auszahlt. Bei BDD wird dies in der Freitext-Ausprägung schwierig, lässt sich aber ebenfalls adressieren, wenn man Keywords ergänzend heranzieht.

5 Fazit & Ausblick

Bei komplexen Testaufgaben (d.h. mit vielen Tests, vielen Ablauf- und/oder Datenvarianten, komplexen Spezifikationen) kann MBT sowohl hinsichtlich Testqualität als auch Wartbarkeit seine Stärken ausspielen. Für einfachere Aufgabenstellungen mag der Ansatz hingegen oft zu schwergewichtig sein. Wie eine „gewöhnliche" Testautomatisierung auch bringt MBT eine Aufwandskurve mit sich, die anfangs hoch ausfällt und sich dann in der Pflege (komplexe Änderungen) auszahlt. Die nötigen Modellierungskenntnisse sind eine Haupt-Hürde bei der Definition und Einführung eines modellbasierten Testprozesses.

BDD sieht leichtgewichtig aus und entwickelt deshalb schnell Charme, gerade auch für fachliche Stakeholder. Zu beachten ist die längerfristige Perspektive: Wie regelt man das Tracing der Tests zu den getesteten Anforderungen? Wie achtet man auf hohe Wiederverwendbarkeit? Wie verwaltet man eine große Zahl solcherart spezifizierter Tests? Die Gefahr besteht (vielleicht vergleichbar mit Capture-Replay-Automatisierungen), dass man sich für BDD entscheidet, weil schnelle Automatisierungs-Ergebnisse möglich sind und sowohl Tool- als auch Lernkurve sehr gering ausfallen, man sich aber zu wenige Gedanken über Pflege und Skalierung des Ansatzes macht.

KDT ergänzt sich ausgezeichnet mit beiden Ansätzen und sorgt für die Minimierung von Pflege- und Automatisierungs-Kosten. Akzeptanz muss durch definierte Ownership der „erlaubten" Keywords und durch geeignete Editor-Unterstützung geschaffen werden. Und das mehrschichtige Design von Keywords ist eine Aufgabe, die nicht unterschätzt werden darf.

Des Weiteren sei darauf hingewiesen, dass – unabhängig vom Ansatz – die Arbeit mit der Erstellung automatisierter Testfälle nicht getan ist. Die Automatisierungs-Praxis zeigt, dass mehrere weitere Themen zu bearbeiten sind, insb. die Kontrollierbarkeit der späteren Testumgebung und Testdaten für die Durchführung, sowie die Testbarkeit und Automatisierbarkeit des Testobjekts. Dies sind typische „Stolpersteine" für jede Testautomatisierung.

Literatur

[1] *Cucumber BDD-Werkzeug.* `http://cucumber.io/`. Version: 2015

[2] Booch, G. ; Maksimchuk, R. ; Engle, M. ; Young, B. ; Conallen, J. ; Houston, K. : *Object-oriented analysis and design with applications.* Addison-Wesley Professional, 2007

[3] Buwalda, H. : Testing with Action Words. In: Fewster, M. (Hrsg.) ; Graham, D. (Hrsg.): *Software Test Automation.* Addison-Wesley, 1999, Kapitel 22

[4] Hahn, R. : *Folien zur Vorlesung "Objektorientierte Analyse und Design", HS Darmstadt.* `https://www.fbi.h-da.de/organisation/personen/hahn-ralf.html`. Version: 2014

[5] ISO/IEC/IEEE: *International Standard 29119–5 Software and systems engineering - Software testing - Part 5: Keyword-driven testing (Draft).* 2015

[6] North, D. : Introducing BDD / Better Software Magazine. 2006. – Forschungsbericht

[7] Rossner, T. ; Brandes, C. ; Götz, H. ; Winter, M. : *Basiswissen modellbasierter Test.* dpunkt.verlag, 2010