Entwicklung einer webbasierten Anwendung unter Verwendung des Spring Frameworks mit Test-Driven Development (TDD)

Studienarbeit T3_3101

des Studienganges Angewandte Informatik an der
Dualen Hochschule Baden-Württemberg Mosbach

von
Isabel Lind

Matrikelnummer, Kurs	8471449, INF21B
Gutachter der Dualen Hochschule	Philipp Abele

18. März 2024

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema:
„Entwicklung einer webbasierten Anwendung unter Verwendung des Spring Frameworks mit Test-Driven Development (TDD)“
selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

TDD Test-Driven Development

1 Einleitung

2 Grundlagen

2.1 Test-Driven Development

Bei dem Test-Driven Development (TDD), zu Deutsch auch „test-getriebene Entwicklung“, handelt es sich um ein Entwicklungs- und Designverfahren für Software, bei dem Testfälle bereits vor oder spätestens parallel zur Implementierung spezifiziert werden. Die zu erstellende Software wird quasi über Tests entworfen. [1, S. 151]

TDD sollte bereits bei der Anforderungsanalyse angewendet werden. Die Anforderungen sind so zu definieren, dass sichergestellt werden kann, dass die Erfüllung dieser Anforderungen mithilfe von Tests validiert werden können. Dies kann durch Testdefinitionen in Textform gewährleistet werden, in die genau die Vorbedingungen, die Ausführung und die zu erwartenden Ergebnisse beschrieben werden. [2, S.188]

Die Abbildung 2.1 zeigt exemplarisch einen schematischen Ablauf des Test-Driven Development-Ansatzes. Am Anfang wird der Testfall definiert (Test Definition, TD), darauf folgt die Umsetzung und Programmierung (Programming, P). Im Anschluss werden die Testfälle ausgeführt (Test Execution, TE). [1, S. 151]

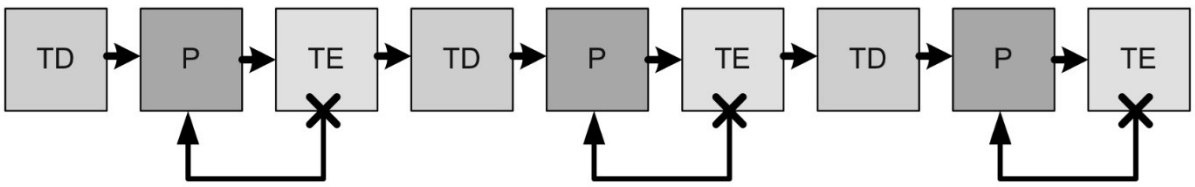

Abbildung 2.1: schematischer Ablauf des Test-Driven Development [1, S. 151]

Im Gegensatz dazu wird beim traditionellen Testen erst parallel zur Entwicklung oder nach Abschluss der Implementierung geeignete Testfälle definiert und ausgeführt, wie ein schematischer Ablauf in Abbildung 2.2 zeigt. [1, S. 150]

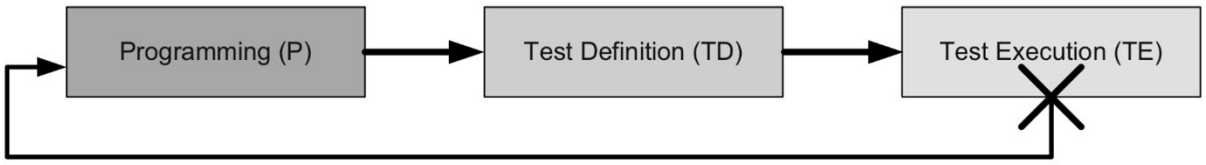

Abbildung 2.2: schematischer Ablauf des traditionellen Testen [1, S.151]

Durch das frühe Definieren und Ausführen von Testfällen im TDD-Ansatz erhalten die Entwickler ein unmittelbares Feedback zur erstellten Lösung und Probleme wie Seiteneffekte können frühzeitig erkannt werden [1, S. 151]. Da der Entwickler sich für die Testerstellung intensiv mit den Anforderungen auseinander setzen muss, erhält er ein verbessertes Verständnis für das zu entwickelnden Produkt und die eigentliche Entwicklungszeit kann deutlich verkürzt werden. Die Anzahl verschleppter und dann aufwändig zu reparierender Fehler kann zudem, durch die von der Entwicklung unabhängigen Erstellung der Testfälle, reduziert werden. Denn, wenn erst nach der Implementierung die Testfälle definiert werden, besteht eine große Wahrscheinlichkeit, dass Denkfehler bei der Implementierung bei der Testerstellung wiederholt werden und somit falsche Tests, die falsche zu testende Software, als korrekt überprüfen [2, S.188].

Die frühzeitig erstellten Tests eignen sich ferner auch zur Automatisierung und des Weiteren kann durch die Testfälle und deren Testergebnisse die Kommunikation verbessert werden. TDD wird meistens im Rahmen der agilen Software-Entwicklung verwendet und ist fester Bestandteil verschiedener Vorgehensmodelle, wie eXtreme Programming und SCRUM [1, S. 151].

2.1.1 Ablauf von Test-Driven Development

Der entscheidende Bestandteil des Test-Driven Development ist, dass die Testfälle vor oder spätestens parallel zur Implementierung definiert werden. Im Anschluss werden die Softwarekomponenten implementiert bzw. angepasst, bis die definierten Tests erfolgreich durchlaufen. Konkret lässt sich TDD in vier grundlegende Schritte unterteilen, wie sie in Abbildung 2.3 skizziert werden. [1, S. 153]

In dem ersten Schritt „think“ wird die Anforderung ausgewählt, die im nächsten Schritt umgesetzt werden soll. Anhand der ausgewählten Anforderung werden die geeigneten Tests definiert. Hierbei ist sicherzustellen, dass die ausgewählte Anforderung auch tat-

sächlich durch die Tests abgedeckt wird. Unit-Tests können beispielsweise auf der Implementierungsebene z.B. für Komponenten verwendet werden. [1, S. 153]

In dem zweiten Schritt „red“ werden diese Tests ausgeführt. Da es noch keine Implementierung dazu gibt, schlagen die Tests fehl und sind im Status „red“. [1, S. 153]

Dann erfolgt die Implementierung, bis die erstellten Tests erfolgreich durchlaufen und den Status „green“ erreichen (Schritt drei). Dabei wird die Anforderung schrittweise implementiert und getestet. Wenn die Testfälle nicht erfolgreich durchlaufen, werden Fehler korrigiert, falls die Anforderung bereits umgesetzt wurde oder die Funktionalität implementiert, falls die Anforderung noch nicht umgesetzt wurde. [1, S. 153]

Im letzten Schritt erfolgt die Optimierung und Anpassung des geschriebenen Softwarecodes (vierter Schritt Refactoring). Da durch das Refactoring der Code verändert wird, müssen alle Testfälle im Anschluss jeweils durchgeführt werden, um sicherzustellen, dass alle Tests weiterhin erfolgreich durchlaufen und der Status „green“ nicht mehr verlassen wird. [1, S. 153]

Wenn das Refactoring erfolgreich durchgeführt wurde, ist die Implementierung für die ausgewählte Anforderung abgeschlossen und die nächste Anforderung kann durch die gleiche Schrittfolge umgesetzt werden. [1, S. 153 f.]

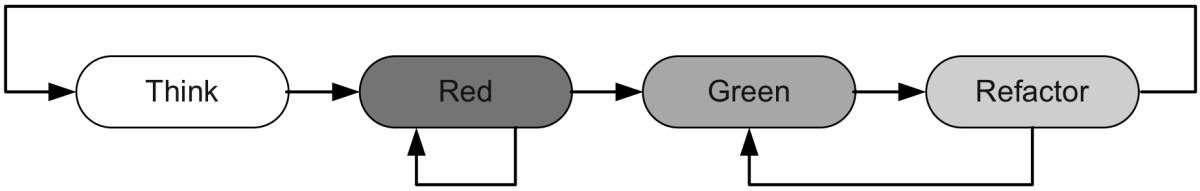

Abbildung 2.3: Phasen Ablauf des Test-Driven Development [1, S. 153]

Die Abbildung 2.4 zeigt einen beispielhaften schematischen Ablaufes von TDD in der Praxis. Dabei wird veranschaulicht, dass für die ausgewählten Anforderungen geeignete Testfälle erstellt werden, um die Erfüllung der Anforderung validieren zu können. In dem Anwendungsbeispiel wird für die Anforderung A drei Testfälle benötigt. Die Testläufe, welche auf der x-Achse dargestellt werden, geben den Status der Testfälle an und wechseln von „red“ in „green“, entsprechend den jeweiligen TDD-Schritten. Dabei wird ersichtlich, dass die Anforderungen schrittweise implementiert und getestet werden. Dieses Beispiel zeigt ebenfalls auf, wie durch die schnelle Rückmeldung, durch die Tests, Seiteneffekte

frühzeitig entdeckt werden können. Eine Implementierung, welche für den Testfall C2 bestimmt war, bewirkte, dass nicht nur der Test C2 weiterhin fehlschlägt, sondern auch der Testfall B2. Dieser Testfall wäre wohlmöglich bei einem traditionellen Testanfall erst sehr spät entdeckt worden und müsste aufwendig korrigiert werden. [1, S. 154]

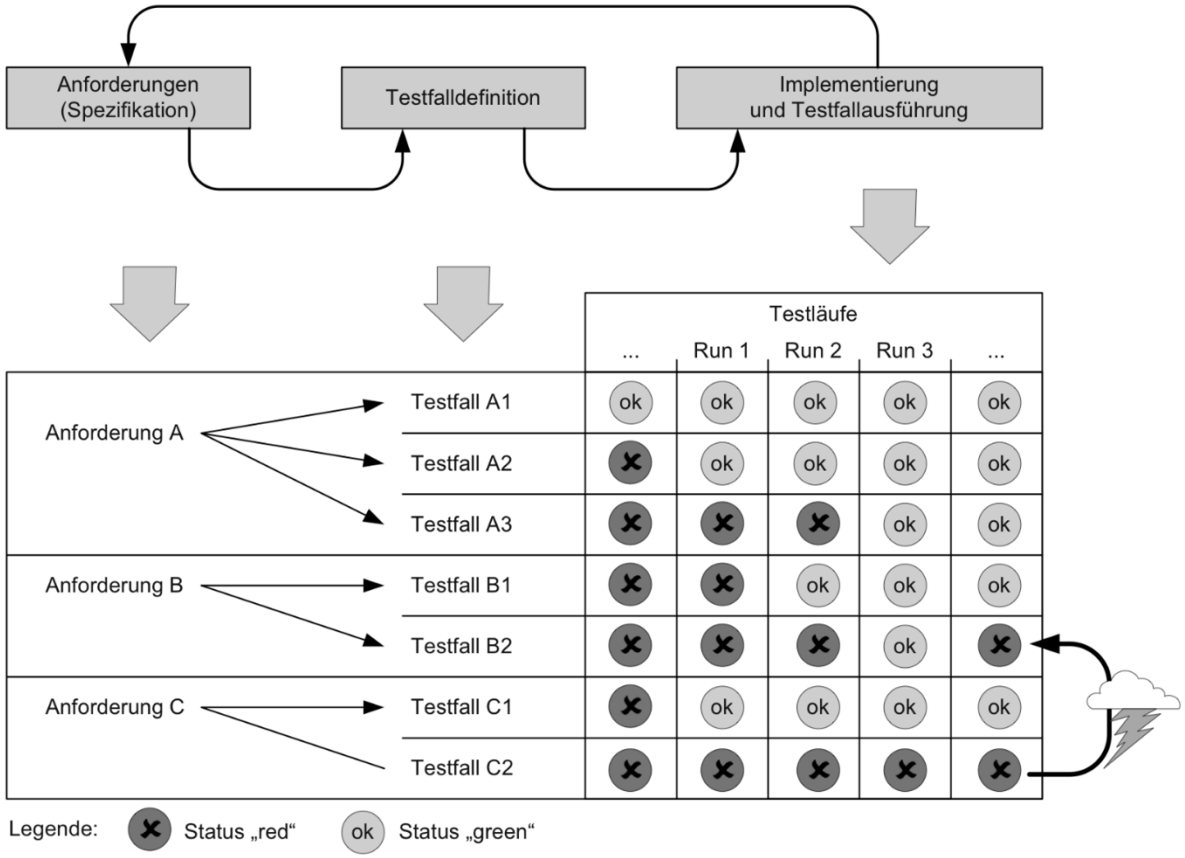

Abbildung 2.4: schematischer Ablauf von Test-Driven Development in der Praxis [1, S. 154]

2.1.2 Arten des Test-Driven Development

Literatur

[1] Alexander Schatten u.a. *Best Practice Software-Engineering*. de. Heidelberg: Spektrum Akademischer Verlag, 2010. ISBN: 978-3-8274-2486-0 978-3-8274-2487-7. DOI: 10.1007/978-3-8274-2487-7. URL: http://link.springer.com/10.1007/978-3-8274-2487-7 (besucht am 18. 03. 2024).

[2] Stephan Kleuker. *Qualitätssicherung durch Softwaretests: Vorgehensweisen und Werkzeuge zum Testen von Java-Programmen*. de. Wiesbaden: Springer Fachmedien, 2019. ISBN: 978-3-658-24885-7 978-3-658-24886-4. DOI: 10.1007/978-3-658-24886-4. URL: http://link.springer.com/10.1007/978-3-658-24886-4 (besucht am 18. 03. 2024).