ADAP CW#04

Projektdaten

Projekt name:	3D-Printing
Projekt repository:	https://github.com/**ModischFabrications**/wahlzeit
This week's tag:	adap-hw03 on master
Homework diff:	https://github.com/ModischFabrications/wahlzeit/compare/adap-hw02...adap-hw03
CI:	https://travis-ci.org/**ModischFabrications**/wahlzeit
Docker Hub Repo:	https://hub.docker.com/r/**modischfabrications**/wahlzeit

Hausaufgaben

Implementation

Neue Klassen (falls vorhanden)

EmailTestSuite, AllTests

Veränderte Klassen (falls vorhanden)

EmailAddressTest, EmailServiceTest

[build.gradle]

Erklärung

Die erste Änderung ist die Anpassung der alten Methoden mit den Erkenntnissen der Übung, um den Vergleich von Double-Werten mit Unschärfe zu realisieren.

Dann habe ich mich in die Klasse „EmailAddress.java“ eingelesen, um ein Gefühl für die Funktionen zu bekommen. Dann habe ich diese Funktionen mit den bisherigen Tests verglichen und mich für einen Test der Gleichheit entschieden, da diese Funktion essenziell für fast jede Anwendung ist.

Beim Test der Adressklasse erstelle ich zwei (nach String) identische Email-Adressen und prüfe beide auf Gleichheit. Ich habe die Methode wie in der Übung besprochen mit “@Test” markiert und mit “test” prefixed, bin allerdings auch überrascht, dass das nicht bei den anderen Methoden der Fall ist (Legacy?). Diesen Test habe ich vor dem Commit ausgeführt, um dessen Lauffähigkeit und den Match zur aktuellen Implementierung zu überprüfen.

Für den Service habe ich einen Test für leere Mailadressen implementiert, formal war das wieder die gleiche Bearbeitung wie beim letzten. Hier habe ich zusätzlich eine “message” eingefügt, um den Entwickler auf den Soll-Zustand hinzuweisen. In diesem Rahmen habe ich einen zusätzlichen, nicht geforderten Test auf ein leeres “Subject” eingefügt, es sind dennoch viele Tests nicht implementiert, die für eine formal vollständige Test-Suite implementiert sein sollten. Idealerweise sollten die existierenden Tests auch weiter zerlegt werden, um die Präzision der Tests zu erhöhen.

Für die Test-Suites habe ich ein eigenes package angelegt, um eine transparente, übersichtliche Struktur zu schaffen. Hier habe ich zuerst einen Testcase für die Email-Dienste angelegt, um Featureorientiert alles zu Emails zu testen. Diese Suite und alle anderen Tests habe ich dann in die allgemeine Test-Suite "AllTests " eingefügt. Hier sollten weitere Suites als Zwischenebene eingeführt werden, um einen modulareren Aufbau zu erhalten. Alle Provider-Klassen, die im Test-package existieren, werden nicht ausgeführt, da diese keine ausführbaren Tests enthalten.

Dann wurde diese Klasse in build.gradle als Einstiegspunkt definiert und der Test-Log einer beispielhaften Ausführung als Referenz hinterlegt.

Zuletzt habe ich alle Änderungen gepusht und den letzten Commit mit dem oben genannten Tag für die Abgabe markiert.

Fragen

Which external systems could you mock in Wahlzeit to isolate the application?

Ein sinnvolles externes System wäre der Datenbankzugriff, um (im Falle real verwendeter Entwicklung) Fehlaufrufe ins Produktivsystem zu vermeiden und die Testzeit zu reduzieren. Genauso ist es sinnvoll die Persistenzebene zu mocken, um die Isolation zu verstärken und setups/teardowns zu minimieren. Auch Oberflächeninteraktionen sollten (wo sinnvoll) gemockt werden, da diese aufwendig zu automatisieren und latenzbehaftet sind (siehe Test Automation Pyramid).

How did you decide that you didn't need to write more tests?

Für mich persönlich, in diesem konkreten Fall? Ein neuer Test pro Klasse reicht, weil die Anforderung „a new test", also **einen** neuen Test, fordert. Mehr Aufwand entspricht nicht mehr den Anforderungen.

Allgemein:

Tests sind in der Realität niemals vollständig, Limit ist der Schnittpunkt zwischen Sicherheitsanforderungen und Arbeitsaufwand. Es lässt sich kein generischer Messwert angeben, der Umfang ist Kontextabhängig. Sicherheitskritische Domänen wie zB. die Bahntechnik erfordern sehr umfassende Tests, Programmierhausaufgaben für Kurse brauchen oft gar keine. Maßgeblich ist häufig die Lebensdauer, „one-offs" brauchen im Allgemeinen weniger Tests als große mehrjährig entwickelte Software. Sinnvoll ist auch eine Bestimmung von Äquivalenzklassen, um den getesteten Umfang in Vergleich zur Testmenge zu maximieren.

Which benefits does introducing multiple test suites have?

Teilbereiche der Anwendung können zielgerichtet getestet werden, hierdurch kann die Iterationszeit deutlich reduziert werden, insbesondere bei zeitaufwendigen Tests in unveränderten Bereichen der Software. Achtung: Es sollten dennoch allumfassende Tests vor einer Auslieferung gemacht werden, um die Systemintegrität zu gewährleisten!