Unit Testing

Lukas Panni
TINF18B5

DHBW Karlsruhe
Vorlesung Advanced Software Engineering Semester 5/6

Inhaltsverzeichnis

1 Analyse und Begründung für Umfang der Tests

Um die Funktionalität der einzelnen Komponenten gewährleisten zu können werden Unit-Tests eingesetzt. Dabei werden für die einzelnen Tests nur die, für diesen Test, relevanten Teile des Systems verwendet. Da Abhängigkeiten zu anderen Komponenten die Tests nicht beeinflussen sollen werden alle anderen Komponenten durch Fake-/Mock-Objekte ersetzt. Das Zusammenspiel mit den anderen Komponenten kann in Integrationstests getestet werden. Außerdem tragen Unit-Tests auch zur Dokumentation bei, indem das gewünschte Verhalten der Komponente für Regel- und Ausnahmefälle in den Testfällen dokumentiert ist.
Für dieses Projekt wird *JUnit* als Framework für die Erstellung und Ausführung von Java-Unit-Tests verwendet. Bei der Implementierung der Tests wurde darauf geachtet die ATRIP-Regeln (*Automatic, Thorough, Repeatable, Independent, Professional*) möglichst zu beachten.
Teile des Codes in diesem Projekt werden für Android-Spezifische UI-Aufgaben benötigt und können deshalb nur schlecht getestet werden. Das führt auch dazu, dass die Code-Coverage über das gesamte Projekt vergleichsweise klein sein kann und in diesem Fall keine sinnvolle Aussage über die Genauigkeit der Tests ermöglicht. Stattdessen sollte hier zur Beurteilung der Testabdeckung nur die Code-Coverage der nicht-UI-Klassen betrachtet werden.
Allgemein wird darauf verzichtet triviale Funktionen, wie zum Beispiel Getter zu testen. Tests dieser Funktionen würden bei großem Aufwand nur einen minimalen Mehrwert bringen, da keine echte Funktionalität getestet wird. Stattdessen sollen sich die Tests auf relevante Funktionalität des Systems fokussieren. Das bedeutet auch, dass vor allem die Klassen getestet werden sollen, die häufig verwendet werden.

Zum Beispiel werden in diesem System oft Konvertierungen von API-Datentypen zu Datentypen der Anwendung durchgeführt. Implementiert sind diese Konvertierungen in den Anwendungsdatentypen, die im *data*-Package zusammengefasst sind. Diese werden häufig verwendet und werden deshalb ausführlich getestet, wobei auch hier auf Tests trivialer Funktionalität verzichtet wird.
Im *data*-Package befindet sich zusätzlich ein *Cache*-Package, das Klassen enthält, die für das Caching von Anwendungsdaten notwendig sind. Die Caching-Funktionalität wird eingesetzt, um die über eine Netzwerkverbindung übertragene Datenmenge zu reduzieren, da sehr viele Daten mehrfach benötigt werden. Da die Cache-Komponente häufig benutzt wird und damit von hoher Relevanz für die Funktionalität der gesamten Anwendung ist, ist auch hier eine hohe Testabdeckung notwendig.
Deshalb ist das Ziel, im gesamten *data*-Package die Code-Coverage hoch zu halten. Trotz-

dem sollen auch hier keine trivialen Funktionen getestet werden.

2 Analyse und Begründung für Einsatz von Fake-/Mock-Objekten

Fake- und Mock-Objekte werden benötigt, um Abhängigkeiten einer Komponente zu anderen Komponenten in Unit-Tests zu reduzieren. Sie implementieren dafür zum Beispiel das benötigte Interface, aber davon nur die aktuell benötigte Funktionalität.