entwickler.press
shortcuts

Dependency Injection in Java

Testing mit CDI-Unit und DI-Frameworks

Christian Laboranowitsch, Philipp Bayer,
Tanja Schmidt und Sven Ruppert

Christian Laboranowitsch, Philipp Bayer, Tanja Schmidt und Sven Ruppert

Dependency Injection in Java

Testing mit CDI-Unit und DI-Frameworks

ISBN: 978-3-86802-544-6
© 2015 entwickler.press
Ein Imprint der Software & Support Media GmbH

1 Integrationstests von CDI-Komponenten

Bei der Entwicklung von Java-EE-Anwendungen vermisst der Spring-verwöhnte Entwickler ein integriertes Testframeworks schmerzlich. Denn während mit Spring das Spring-Test-Modul mitgeliefert wird, ist man bei Java-EE-Applikationen auf Third Party Libraries und einen gewissen Erfindergeist der Entwickler und Architekten im Projekt angewiesen. Neben den allseits bekannten Größen wie Arquillian [1] steht mit CDI-Unit [2] ein vielversprechendes Framework in den Startlöchern, wenn es um das Testen von komplexen Java-EE-6-/CDI-Applikationen geht. Im Folgenden werden wir die Vor- und Nachteile der einzelnen Ansätze darstellen und exemplarische Anwendungsfälle aufzeigen.

Ein wichtiger Erfolgsfaktor für Softwareentwicklungsprojekte ist das frühzeitige Testen durch die Entwickler. Aber wie kann man eine zufriedenstellende Testabdeckung mit angemessenem Aufwand erreichen? Dafür müssen die gewählten Frameworks und Werkzeuge die Erstellung und Ausführung von Tests bestmöglich unterstützen und gleichzeitig zum gewählten Technologiestack passen. Werkzeuge, die Hand in Hand greifen, vereinfachen die Erstellung und Ausführung der Tests und steigern nicht zuletzt die Motivation des Entwicklers, was ein nicht zu unterschätzender Aspekt ist.

Doch Testen ist in vielen Projekten leider immer noch eine Aufgabe mit C-Priorität. Es wird erledigt, wenn es die Zeitplanung oder die Budgetsituation zulässt. Häufig bleibt es auch noch heute auf der Strecke (eine Auswirkung ist beispielsweise, dass fehlgeschlagene Tests mit *@Ignore* annotiert werden, um sie dann „später“ mal in Ordnung zu bringen). Ein Grund dafür mag sein, dass die Bedeutung von Entwicklertests für Nichtentwickler oft schwer zu greifen ist.

Denn immerhin gibt es in nahezu allen Projekten Testphasen, in denen von Testern getestet wird. Dabei ist schon lange bekannt, dass ein Bug umso weniger Kosten verursacht, je früher er gefunden wird [3]. Man kann also argumentieren, dass Entwicklertests einen Return-on-Investment bieten, der umso höher ist, je früher die notwendigen Tests zur Anwendung kommen. Wenn also Entwicklertests nur nebenbei und irgendwann gemacht werden, leidet die Qualität der Software. Dies führt häufig zu Budgetbelastungen und Zeitdruck, gerade in der Endphase eines Projekts. Nicht zu vernachlässigen sind die hohen Belastungen der einzelnen Projektmitglieder, wenn es in der Auslieferungsphase zu Qualitätsproblemen mit langen Buglisten kommt. Auch wenn dies eine bereits bekannte Argumentationskette darstellt, gilt es immer noch bei dem einen oder anderen Stakeholder Überzeugungsarbeit zu leisten. Daneben bieten Entwicklertests noch eine Reihe anderer Vorteile:

- Fehler, die durch spätere Refactorings entstehen, werden entdeckt. Ohne Entwicklertests ist es meist schwer zu sagen, ob das Refactoring erfolgreich durchgeführt werden konnte, wenn nicht sogar unmöglich (Regressionstests).
- Spätere Änderungswünsche können ungewollte Auswirkungen auf Bereiche des Systems haben, die so nicht absehbar waren und nur durch das Fehlschlagen der Entwicklertests frühzeitig identifiziert werden.
- Das Schreiben von Entwicklertests bewirkt häufig eine intensivere Auseinandersetzung mit den fachlichen Anforderungen und der technischen Umsetzung.
- Entwicklertests dokumentieren das gewollte und ungewollte Verhalten von Code. So ist es häufig eine gute Idee, zur Einarbeitung in unbekannten Code bei den Tests zu starten.

Welche Art von Test darf es sein?

In der Softwareentwicklung gibt es für den richtigen Einsatzzweck unterschiedliche Arten von Tests (**Abb. 1.1**). Bei unserer Betrachtung wollen wir uns auf Entwicklertests beschränken.

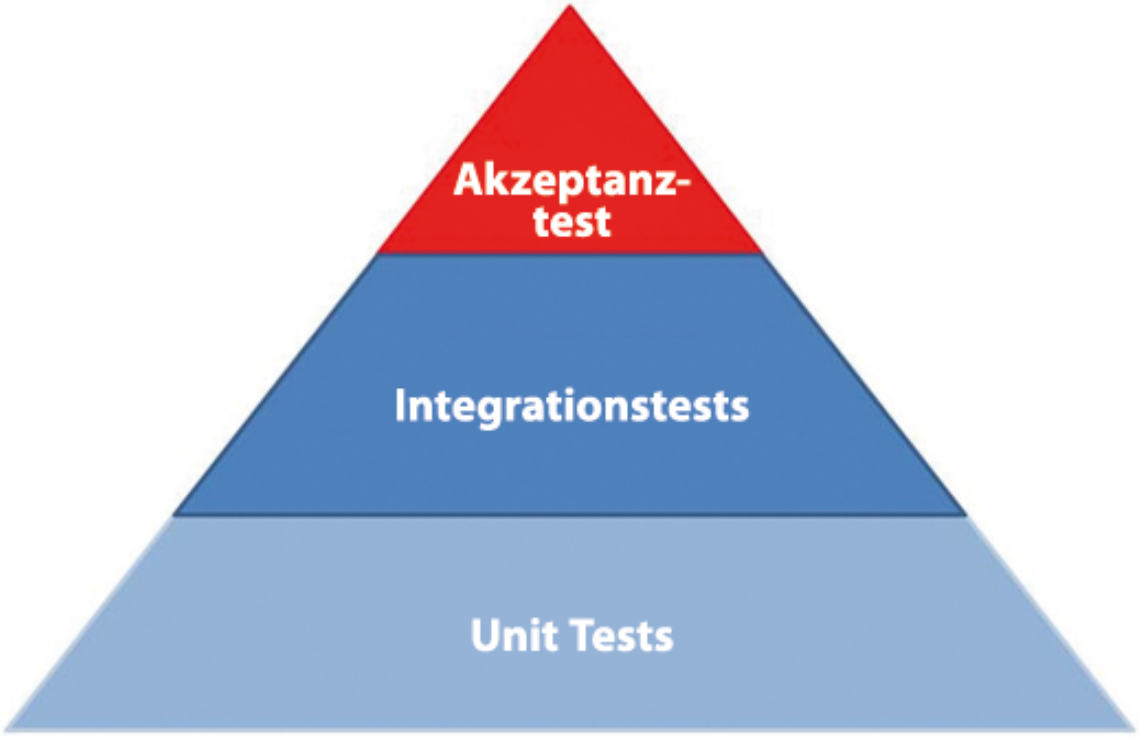

Abbildung 1.1: Testpyramide

Unter Entwicklertests verstehen wir Unit und Integrationstests. Da diese beiden Begriffe nicht immer eindeutig verwendet werden, möchten wir kurz auf diese Testarten eingehen und die Unterschiede und Einsatzzwecke aufzeigen.

Unit Test

Unit Tests dienen dazu, die Module einer Software auf korrekte Funktionalität zu testen. Synonyme Bezeichnungen sind auch die Begriffe Modultest und Komponententest. Merkmal eines Unit Tests ist, dass er ein Modul isoliert von anderen Modulen testet. Unter einem Modul beziehungsweise einer Komponente wird dabei der kleinstmögliche Teil funktionalen Codes verstanden; in der Regel also eine Methode oder Klasse. Die