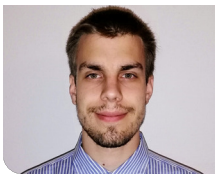
Von DDD zu BDD: Methoden, Werkzeuge, Fallstudien

Diplomanden

Saskia Stillhart



Timothée Moos

Ausgangslage: Domain Driven Design (DDD) ist eine weit verbreitete Praktik in der Softwaremodellierung. Komplementär dazu wird Behavior-Driven Development für "Specification by Example" und die systematische Spezifikation von Testcases angewendet. Wie die beiden Techniken miteinander kombiniert werden können, ist noch nicht ausreichend untersucht und dokumentiert. In dieser Arbeit soll anhand von Praxisbeispielen der Stand der Techniken erhoben und eine integrierte Arbeitsmethode konzipiert werden. Diese Arbeitsmethode soll von einem neuen oder bereits existierenden Werkzeug unterstützt werden und als Einstiegshilfe für BDD dienen.

Vorgehen / Technologien: Als erstes analysierten wir die vorhanden DDD- und BDD-Tools. Im Bereich DDD waren dies Context Mapper und Sculptor Generator, zwei Modellierungs-Frameworks. Im BDD-Umfeld untersuchten wir zwei Werkzeug-Unterstützungen für Java, Cucumber und JBehave. Ebenfalls recherchierten wir zu weiteren Praktiken, die im BDD-Umfeld genutzt werden, wie beispielsweise Example Mapping. Die Recherche nach einem geeigneten Werkzeug, welches direkt für die kombinierte Arbeitsmethode verwendet werden könnte, verlief erfolglos. Deshalb entschieden wir uns, einen eigenen Prototypen, DDD-2-BDD, zu entwickeln. Dieser nutzt Context Mapper-Modelle als Input und generiert als Output Testszenarien in Gherkin-Syntax. Gherkin wird von vielen BDD-Frameworks unterstützt; BDD-Tools arbeiten mit Textdokumenten, die durch Schlüsselwörter zu ausführbaren Spezifikationen werden. Für den Context Mapper haben wir uns entschieden, weil das Projektteam bereits Erfahrungen damit gesammelt hatte, eine bereits erprobte Java Library zur Verarbeitung des Models zur Verfügung steht und umfangreiche Modelle erstellt werden können. Der Prototyp ist als Webapplikation umgesetzt mit Spring Boot und Thymeleaf. Abbildung 1 zeigt mit einem Screenshot der Applikation auf, wie die Selektion der zu generierenden Tests funktioniert und in welcher Form die generierten Tests zur Verfügung gestellt werden. Anhand der Systemübersicht in Abbildung 2 ist das EVA-Prinzip mit Eingabe(blau), Verarbeitung(orange) und Ausgabe(grün) ersichtlich.

Ergebnis: Zusätzlich zum DDD-2-BDD Toolprototypen entstand eine Dokumentation der Testgenerierung. Es werden in drei verschiedenen Bereichen Testgenerierungen angeboten: Validierungen von Attributen, Testen von Beziehungen und Überprüfung von DDD-Patterns. Der Ansatz, aus dem Domänenmodel Gherkin-Features zu generieren, bietet den Vorteil, dass bereits modellierte Informationen wiederverwendet werden und so eine Zeitersparnis erreicht wird. Ein Nachteil dieses Ansatzes ist es jedoch, dass ein hoher Detailgrad im Modell-Input benötigt wird, um sinnvolle Tests generieren zu können. Ausserdem bietet das Tool zwei Tutorials an. Diese Tutorials erklären mithilfe von Beispielen, wie der Output generiert wird und wie dieser mit Cucumber verwendet werden kann, um Tests zu automatisieren. Es ist ebenfalls beschrieben, wie ein Projekt mit Maven oder Gradle aufgesetzt wird, wie die Gherkin Files integriert werden und wie die sogenannten Step Definitions in Cucumber definiert werden können. Zusätzlich werden die Beispiele als Maven- und Gradle-Projekte zur Verfügung gestellt. In Abbildung 3 ist das Zusammenspiel zwischen Context Mapper, DDD-2-BDD, Gherkin Files, Cucumber Step Definitions und Java Code illustriert.

Abbildung 1: Screenshot Toolprototyp
Eigene Darstellung

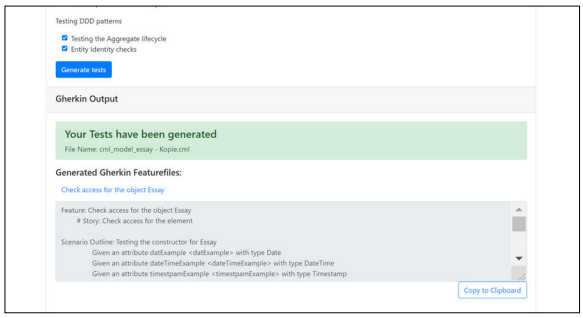


Abbildung 2: Systemübersicht
Eigene Darstellung

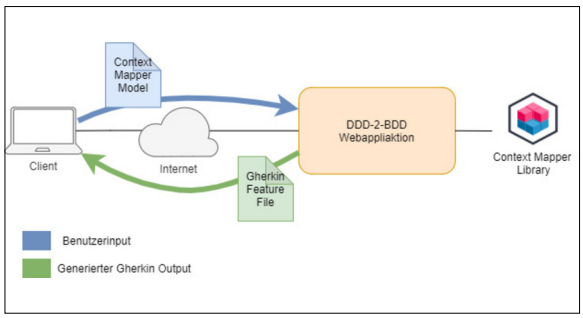
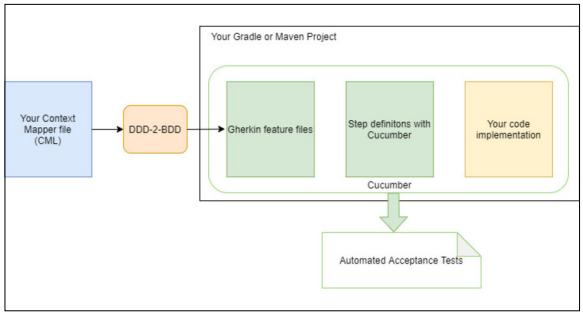


Abbildung 3: Übersicht Arbeitsmethode
Eigene Darstellung



Examinator
Prof. Dr. Olaf Zimmermann

Experte
Dr. Daniel Lübke

Themengebiet
Software