



Fakultät für Informatik und Mathematik 07

Bacheloarbeit

über das Thema

⁵ **Generative Testerstellung für Microservice-Architekturen**

Autor: Fabian Wilms
holtkoet@hm.edu

Prüfer: Prof. Dr. Ulrike Hammerschall

Abgabedatum: xx.xx.2017

I Kurzfassung

Viele Softwareprojekte scheitern an fehlerhafter Kalkulation. Die Kosten werden meist viel zu gering angesetzt und sorgen für unzufriedene Kunden, geringe Erträge und fehlende Folgeaufträge. Doch warum ist das so?

- 5 Der größte Kostenfaktor sind Fehler im laufenden System. Diese Fehler zu beseitigen ist häufig eine große Herausforderung die viel Zeit, Personal und somit Geld kostet. Es gilt also, vor Auslieferung der Software eine entsprechende Qualität gewährleisten zu können.

Aus diesem Grund ist eine umfassende Testabdeckung ein sehr wichtiger Bestandteil in der Definition of Done der Landeshauptstadt München. Unit- und Integrations Tests sind
10 fest in viele Entwicklungsprozesse integriert und finden besonders auch in agilen Entwicklerteams großen Zuspruch. Allerdings sind auch Tests finanzielle, wie auch zeitliche Kostenfaktoren. Dennoch wird für Tests, trotz des Bewusstseins für die damit entstehenden Probleme, häufig viel zu wenig Zeit eingeplant.

Um diesen Problemen entgegen zu wirken wurde bei der LHM im vergangenen Jahr das
15 Softwareprojekt Barrakuda entwickelt, welches anhand eines nach Domain Driven Design (DDD) aufgebauten Domänen-Modells Microservice Architekturen generieren kann. Die Idee: Software anhand von simplen Modellen generieren, um möglichst viel Zeit beim Entwickeln von "Boiler PlateCode einzusparen, und die Software-Qualität nicht nur durch ersparte Entwicklungszeit, sondern auch durch homogene Projektstrukturen in allen ge-
20 nerierten Systemen, und damit erhöhte Wartbarkeit, zu verbessern.

Im Rahmen dieser Bachelor-Arbeit soll die These geprüft werden, dass sich aus einem solchen Modell genügend Informationen zum Testen eines Microservice-Frameworks extrahieren lassen, um eine ausreichende Testabdeckung für die generierten Services zu bieten.

- 25 Hierzu ist ein tieferes Verständnis der zu Generierung eingesetzten Sprache Barrakuda, welche von Martin Kurz in seiner Masterarbeit „Verbesserung des Softwareentwicklungsprozesses der Landeshauptstadt München durch modellgetriebene Softwareentwicklung“ entwickelt wurde, nötig. Diese wird in einem weiteren Abschnitt erklärt und anhand von Beispielen demonstriert. Um die Verbindung von Barrakuda zum DDD hervorzuheben,
30 wird dies im weiteren Vorgehen näher beleuchtet.

Im zweiten Teilabschnitt werden dann die gängigsten Methoden und Praktiken, sowie Frameworks, die beim Testen von Microservice-Architekturen eingesetzt werden, identifiziert und weitere Anforderungsanalysen für diese durchgeführt, um die benötigte Bandbreite

der generierten Tests abzugrenzen. Vorstellbare Testarten sind Integrationstests, Unit-Tests und Smoke Tests.

Der letzte Schritt der Vorbereitung soll dann aus den zusammengetragenen Informationen feststellen, für welches Framework ein generativer Ansatz zum Erstellen von
5 Testfällen am sinnvollsten ist. Außerdem sollen eventuell benötigte Erweiterungen und/o-
der Änderungen von Barrakuda festgelegt werden, die für den generativen Ansatz von
Nöten sind.

Im praktischen Teil der Arbeit werden dann zunächst einige Referenzimplementierungen
umgesetzt, anhand derer im nächsten Teilschritt die Generierung der Tests, unter Zuhilfe-
10 nahme von Barrakuda und den im vorherigen Teil festgelegten Änderungen, implementiert
werden.

Abstract

Das ganze auf Englisch.

II Inhaltsverzeichnis

I	Kurzfassung	I
II	Inhaltsverzeichnis	III
III	Abbildungsverzeichnis	IV
5 IV	Tabellenverzeichnis	IV
V	Listing-Verzeichnis	IV
VI	Abkürzungsverzeichnis	IV
1	Einführung und Motivation	1
2	Software Testen	3
10 2.1	Bekannte Testmethoden	3
2.2	Testen von Microservices	3
2.2.1	Test-Strategien und deren Nutzen für Barrakuda	3
2.2.2	Frameworks zum Umsetzen von Test-Strategien für Barrakuda	3
3	Architektur	3
15 3.1	Generelle Architektur für Microservices	3
3.2	Vorgegebene Architektur der LHM	3
3.2.1	Authentifizierungsservice	3
3.2.2	Discoveryservice	3
3.2.3	Configurationsservice	3
20 3.2.4	Service	3
3.2.5	Webservice	3
3.3	Unterschiede zwischen Microservices und Monolith-Architekturen	3
3.3.1	Auswirkungen	3
4	Anforderungen an generierte Tests	3
25 4.1	Benötigte Daten	3
4.2	Notwendige Änderungen/Erweiterungen von Barrakuda	3
5	Implementierung in Barrakuda	3
5.1	Referenz-System	3
5.1.1	Komponenten und Aufbau	3
30 5.1.2	Implementierung des Systems	3
5.1.3	Implementierung der Tests	3
5.2	Übernahme der Referenz-Implementierung in Barrakuda-Templates	3
6	Quellenverzeichnis	3
Anhang		I
35 A	Code-Fragmente	I

III Abbildungsverzeichnis

Abb. 1	SQS Report Costs of Defect Correction	2
--------	---	---

IV Tabellenverzeichnis

V Listing-Verzeichnis

5 VI Abkürzungsverzeichnis

1 Einführung und Motivation

IT nimmt sowohl im privaten als auch geschäftlichen Alltag eine immer größere Rolle ein. Die Übernahme von Bereichen, die ehemals als nicht durch Computer austauschbar erachtet wurden, schreitet immer weiter fort. Doch dadurch steigen nicht nur bestehende
5 Anforderungen an Software, sondern es entstehen auch neue Kriterien. Ganz abgesehen davon steigt die Komplexität von modernen Software-Systemen immens an.

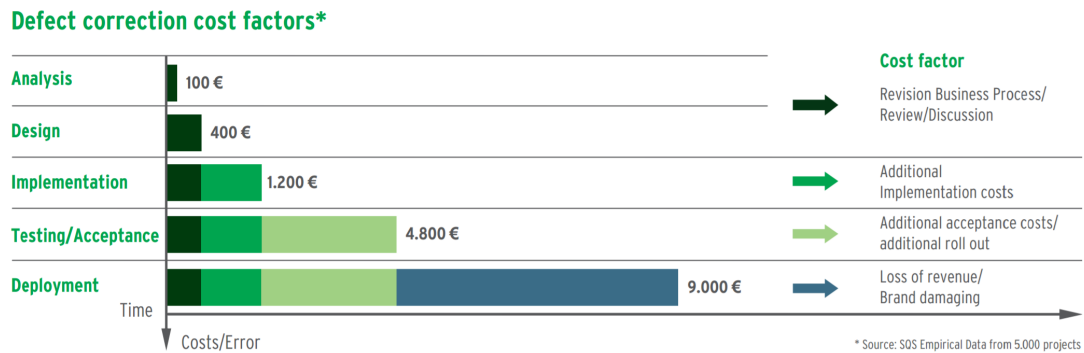
Mit steigender Komplexität und höherer Nachfrage am Markt, sowie engen Zeitplänen für Projekte wird leider häufig aus Zeit- und Kostengründen auf Qualität nur geringfügig
10 Rücksicht genommen. Zunächst verursacht eine gute Software-Qualität nämlich nur Mehrkosten. Personelle wie zeitliche. Die langfristige Sinnhaftigkeit bleibt dabei außen vor, aus der Vergangenheit wird nur selten gelernt.

Für die größte Kostenexplosion sorgt nämlich das Bug-Fixing in Produktivsystemen, also das Beheben sogenannter *field defects*. Wurde in den ersten Phasen eines Projekts nicht viel, oder kein Wert auf eine ausreichende Test-Abdeckung gelegt schaffen es viele Fehler in
15 die Produktivsysteme der Hersteller. Doch werden diese Fehler erst im laufenden Betrieb beim Kunden festgestellt, ist es bereits zu spät. Robert N. Charette kritisiert eben dies in seinem Artikel *Why Software Fails*.

*If the software coders don't catch their omission until final system testing—or worse, until after the system has been rolled out—the costs incurred to correct
20 the error will likely be many times greater than if they'd caught the mistake while they were still working on the initial [...] process.¹*

Ein Bericht der Kölner Beratungsfirma SQS zeigt dies anhand von gesammelten Zahlen aus Beratungsaufträgen. Hier wird besonders deutlich wie viel es für ein Projekt bedeutet, frühzeitige Qualitätssicherung durchzusetzen. Und dazu zählt auch das Testen von
25 Software.

¹Charette, Robert N.: Why Software Fails. (URL: <http://spectrum.ieee.org/computing/software/why-software-fails>).

Abbildung 1: SQS Report Costs of Defect Correction ²

Diese Zahlen machen es auf einfachem Wege sehr deutlich. Umso früher Fehler entdeckt und bemerkt werden, umso weniger kostet es auch diese zu beheben. Wenn bereits vor dem Start der Implementierungsphase auf eine hohe Testabdeckung, beispielsweise durch den Einsatz von test-driven development, Wert gelegt wird, können je auftretenden Fehler um die 7800€ eingespart werden. Mit diesen Zahlen sind die Mehrkosten, die für ein solches vorgehen entstehen um ein vielfaches leichter zu verdauen.

Es scheint also ganz einfach: Viel Zeit und Geld in gute Softwarequalität investieren und schon sollte das Projekt keine unvorhergesehenen Kosten verursachen. Doch so einfach ist es leider nicht. Wie bereits am Anfang der Einleitung erwähnt stehen Projektleiter und ihre Mitarbeiter unter hohem zeitlichen Druck. Und Tests kosten nun mal Zeit. Es entsteht in dieser Zeit aber kein messbarer Fortschritt am Produkt. Dies ist auch der Grund, weshalb das Testen bei Entwicklern nicht an oberster Stelle der liebsten Aufgaben steht. Es kostet Zeit ohne erkennbaren Fortschritt zu erreichen.

Es muss also die Zeit, die für das Entwickeln von Tests benötigt wird, reduziert werden, sowie mehr Zeit geschaffen werden, um für eine gute Softwarequalität zu sorgen. Diesen Ansatz verfolgt die Eigenentwicklung von it@M, Barrakuda, und deren Weiterentwicklung im Rahmen dieser Arbeit.

²SQS: Detect errors early on, reduce costs and increase quality. (URL: <https://www.sqs.com/en/academy/download/fact-sheet-EED-en.pdf>)

2 Software Testen

2.1 Bekannte Testmethoden

2.2 Testen von Microservices

2.2.1 Test-Strategien und deren Nutzen für Barrakuda

5 2.2.2 Frameworks zum Umsetzen von Test-Strategien für Barrakuda

3 Architektur

3.1 Generelle Architektur für Microservices

3.2 Vorgegebene Architektur der LHM

3.2.1 Authentifizierungsservice

10 3.2.2 Discoveryservice

3.2.3 Configurationsservice

3.2.4 Service

3.2.5 Webservice

3.3 Unterschiede zwischen Microservices und

15 Monolith-Architekturen

3.3.1 Auswirkungen

4 Anforderungen an generierte Tests

4.1 Benötigte Daten

4.2 Notwendige Änderungen/Erweiterungen von Barrakuda

20 5 Implementierung in Barrakuda

5.1 Referenz-System

5.1.1 Komponenten und Aufbau

5.1.2 Implementierung des Systems

5.1.3 Implementierung der Tests

25 5.2 Übernahme der Referenz-Implementierung in Barrakuda-Templates

6 Quellenverzeichnis

Charette, Robert N.: Why Software Fails. (URL: <http://spectrum.ieee.org/computing/software/why-software-fails>)

30 SQS: Detect errors early on, reduce costs and increase quality. (URL: <https://www.sqs.com/en/academy/download/fact-sheet-EED-en.pdf>)

Anhang

A Code-Fragmente

Viel Beispiel-Code