Fakultät für Informatik und Mathematik 07

Bacheloararbeit

über das Thema

Generative Testerstellung für Microservice-Architekturenit

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?

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 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 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 einzsuparen, und die Software-Qualität nicht nur durch ersparte Entwicklungszeit, sondern auch durch homogene Projektstrukturen in allen generierten Systemen, und damit erhöhte Wartbarkeit, zu verbessern.

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

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, 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 isnd Integrationstests, Unit-Tests und Smoke Tests.

Der letzte Schritt der Vorbereitung soll dann aus den zusammengetragenen Informationen festgestellen, für welches Framework ein generativer Ansatz zum Erstellen von Testfällen am sinnvollsten ist. Außerdem sollen eventuell benötigte Erweiterungen und/oder 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 Zuhilfenahme von Barrakuda und den im vorherigen Teil festgelegten Änderungen, implementiert werden.

Abstract

Das ganze auf Englisch.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Listing-Verzeichnis

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 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 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.

Der Chaos Report von 2014 zeigt erschreckend wie viele IT-Projekte in der Versenkung verschwanden oder aber erst durch enorme Mehrkosten fertiggestellt werden konnten.

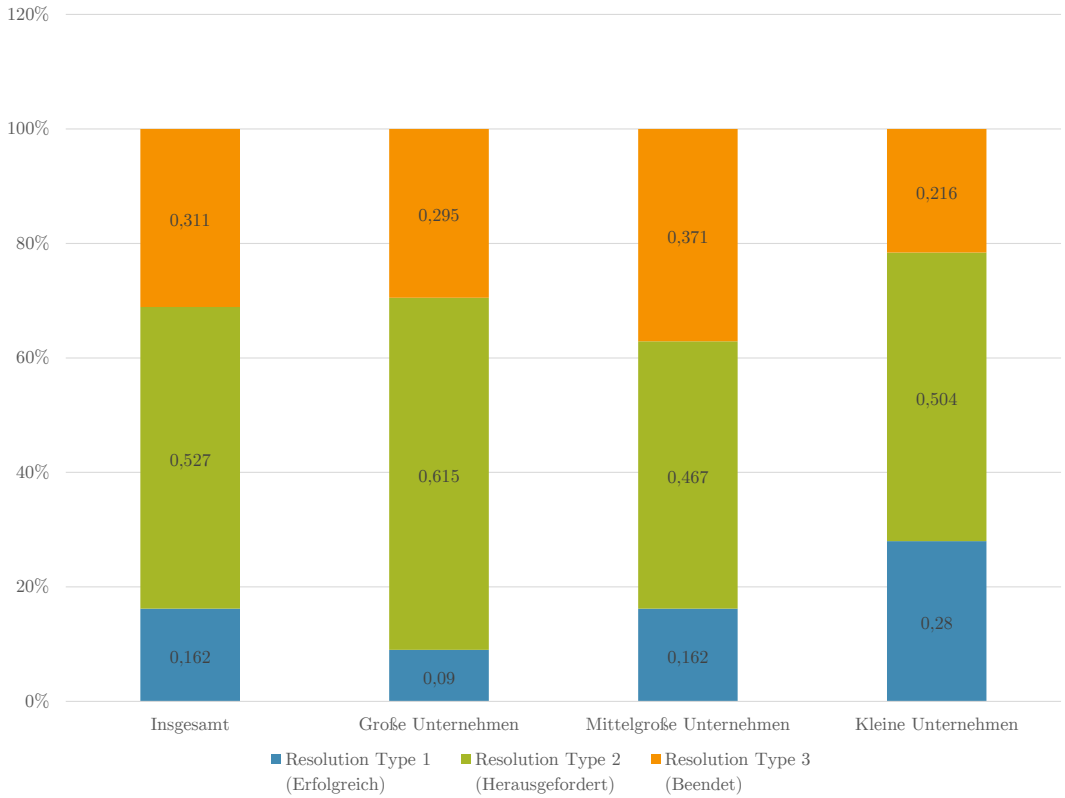

Abbildung 1: Chaos Report 2014 Failure Statistics[a]

[a] 1995, The Standish Group: The Standish Group Report - CHAOS Report. ⟨URL: `https://www.projectsmart.co.uk/white-papers/chaos-report.pdf`⟩.

Die Projekte wurden drei verschiedenen Klassen zugeteilt. RT1 entspricht einem erfolg-

reich abgeschlossenen Projekt, welches zeitlich und finanziell fertiggestellt wurde, sowie alle Features und Funktionalitäten die gefordert wurden enthält. RT2-Projekte wurden zu spät, zu teuer, und nicht mit allen Features ausgeliefert. RT3 sind abgebrochene Projekte.

Doch wie kommt es dazu, dass in derart vielen Projekten die festgelegten Ziele nicht ganz, oder sogar gar nicht erreicht werden? Es gibt unzählige Faktoren die dazu führen. Doch sind es einige die immer wieder in diesem Zusammenhang genannt werden. Dazu zählen ... und ... wie ...

Für die Größte Kostenexplosion sorgt jedoch das Bug-Fixing in Produktivsystemen. Wurde in den ersten Phasen eines Projekts nicht viel, oder kein Wert auf eine ausreichende Test-Abdeckung gelegt schaffen es viele Fehler in 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 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 Software.

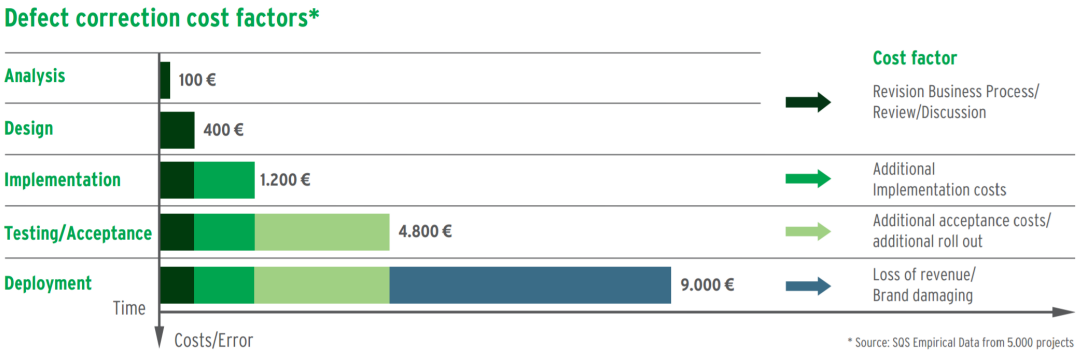

Abbildung 2: SQS Report Costs of Defect Correction [2]

Es ist also eigentlich ganz einfach: Viel Zeit und Geld in gute Softwarequalität investieren

[1] Charette, Robert N.: Why Software Fails. ⟨URL: `http://spectrum.ieee.org/computing/software/why-software-fails`⟩.

[2] SQS: Detect errors early on, reduce costs and increase quality. ⟨URL: `https://www.sqs.com/en/academy/download/fact-sheet-EED-en.pdf`⟩

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 seine 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 an den 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.

2 Architektur-Generierung für Software-Projekte

2.1 Barrakuda

Im Rahmen seiner Master-Arbeit hat der Angestellte des Eigenbetriebs it@M der Landeshauptstadt München mithilfe von X-Text und X-Tend eine Sprache entwickelt, die es erlaubt, anhand eines Domänen-Ähnlichen Modells eine Microservice-Architektur zu generieren.

2.1.1 Idee

2.1.2 Einführung

2.1.3 Verbindung zum Domain-driven Design

Domain-driven Design in Microservices

Umsetzung

1995, The Standish Group: The Standish Group Report - CHAOS Report. ⟨URL: `https://www.projectsmart.co.uk/white-papers/chaos-report.pdf`⟩

Charette, Robert N.: Why Software Fails. ⟨URL: `http://spectrum.ieee.org/computing/software/why-software-fails`⟩

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