

Integration eines Feature-orientierten Testsystems in den Entwicklungszyklus technischer Systeme

Bachelorarbeit
zum Erlangen des akademischen Grades

Bachelor of Science in Engineering (BSc)

Fachhochschule Vorarlberg
Informatik - Software and Information Engineering

Betreut von
Dipl.-Ing. Dr. techn. Ralph Hoch

Vorgelegt von
Marco Prescher

Dornbirn, am 1. Juli 2023

Kurzreferat

Integration eines Feature-orientierten Testsystems in den Entwicklungszyklus technischer Systeme

Die Entwicklung technischer Systeme ist ein komplexer und kostspieliger Prozess. Daher ist es wichtig, dass die Produkte vor der Auslieferung an den Kunden gezielt und sorgfältig getestet werden. Dies wird durch Zeitdruck und Deadlines oftmals vernachlässigt, oder nur unzureichend durchgeführt. Mit einem gut strukturierten Testplan kann dieses Risiko allerdings minimiert werden, und die Durchführung der Tests mit dem Produktentwicklungszyklus integriert werden. Dadurch kann stabile Hardware sowie effiziente und gut strukturierte Software ohne große Verzögerung an den Kunden ausgeliefert werden.

Durch Methodiken wie zum Beispiel Feature-orientierte Entwicklung ist es möglich, dass bestimmte Features vor dem Release der Produkte abgenommen und getestet werden müssen. Das wiederum ermöglicht, dass neu implementierte Features gründlich getestet werden und somit zu einer hohen Qualität beitragen.

Dies kann erreicht werden, indem Features strukturiert in einer Datenbank eingepflegt werden. Ein auf diesen Featurebeschreibungen basierendes System kann die automatische Testdurchführung unterstützen, gezielt Tests für einzelne Features durchführen und deren Abnahme beschleunigen. Dadurch kann der Testaufwand verringert und den Entwicklern ein fokussiertes Feedback vermittelt werden.

Abstract

Integrating a feature-oriented testing system into the development cycle of technical systems

The development of technical systems is a complex and costly process. Therefore, it is important that products are thoroughly tested before delivery to the customer. However, this is often neglected or done insufficiently due to time pressure and deadlines. A well-structured test plan can minimize this risk and integrate the testing process into the product development cycle. This allows stable hardware and efficient, well-structured software to be delivered to the customer without significant delays.

Using techniques such as feature-oriented development, certain features must be accepted and tested before the product release. This in turn allows newly implemented features to be thoroughly tested and contribute to high quality.

This can be achieved by structuring features in a database. A system based on these feature descriptions can support automatic testing, conduct targeted tests for individual features, and accelerate their acceptance. This reduces testing effort and provides focused feedback to the developers.

Inhaltsverzeichnis

Abbildungsverzeichnis	7
Abkürzungsverzeichnis	8
1 Einleitung	10
1.1 Motivation	10
1.2 Problemstellung	10
1.3 Zielsetzung	11
2 Stand des Wissens	12
2.1 Testarten	12
2.1.1 Unit-Tests	13
2.1.2 Integrationstests	13
2.1.3 Funktionstests	13
2.1.4 Leistungstests	13
2.2 Black-Box und White-Box Testing	14
2.3 Test Driven Development	14
2.4 Test Case Management System	15
2.4.1 Allgemeine Vorteile von Test Case Management Systemen . . .	16
2.4.2 Überblick von bestehenden Test Case Management Systeme . .	17
2.5 Verwendete Technologien	18
2.5.1 MariaDB	18
2.5.2 Docker	19
2.5.3 Microsoft .NET Core	20
2.5.4 Microsoft ASP.NET Core	20
2.5.5 Entity Framework Core	20
2.5.6 OpenAPI	21

3	Anforderungen	22
3.1	Test	22
4	Konzept	23
4.1	Architektur	23
4.1.1	Backend	23
4.1.2	Domain Driven Design	23
4.2	Vorentwicklungsphase	23
4.2.1	Projektstruktur	23
4.2.2	Domain Model	23
4.2.3	ER Model	23
4.2.4	Featurestruktur	23
4.2.5	Featurebedingungen	24
4.2.6	Featureverwaltung	24
4.3	Entwicklungsphase	24
4.3.1	Verwaltung und Struktur von Features	24
4.3.2	Featurestruktur zur Datenbank	24
4.3.3	Visualisierung von Featurestrukturen	24
5	Technische Umsetzung	25
5.1	Scrum	25
5.2	User Stories	25
5.2.1	Story 1	25
5.2.2	Story 2	25
5.2.3	Story n	25
6	Ergebnisse	26
7	Fazit und Ausblick	27
	Literaturverzeichnis	28

Abbildungsverzeichnis

2.1	Black-Box/White-Box Testing (Quelle: Khandelwal (2019))	14
2.2	Test Driven Development cycle (Quelle: <i>Test-driven development - IBM Garage Practices</i> (2023))	15
2.3	Liste von aktuell führenden Test Case Management Systemen/Tools (Quelle: <i>7 Best Test Management Tools</i> (2023))	18
2.4	Funktionsweise von Docker (Quelle: Böllhoff (2022))	19
2.5	Update von einem Parameter einer JavaScript Object Notation Spalte	21
7.1	Ein String wird in einem Scope erzeugt und der Variable hello zugewiesen. Am Ende des Scopes wird der Speicher freigegeben.	27
7.2	Nach 100 Simulationsschritten, einem minimalen Gruppenanteil $B_{min} = 0.4$ und einer Nachbarschaftsgröße von einem Feld sind die Gruppen bereits sichtlich separiert.	27

Abkürzungsverzeichnis

API Application Programming Interface

JSON JavaScript Object Notation

TCMS Test Case Management System

TDD Test Driven Development

EF Entity Framework

ORM Object Relational Mapping

OAS OpenAPI Spezifikation

HTTP Hypertext Transfer Protocol

Danksagung

Ich möchte mich aufrichtig bei Ralph Hoch, der Firma Gantner Instruments und allen Mitarbeitenden bedanken, die mir bei der Vollendung dieser Arbeit geholfen haben. Ihre Unterstützung und Expertise waren von unschätzbarem Wert und haben zum erfolgreichen Abschluss dieses Projekts beigetragen. Vielen Dank für Ihre harte Arbeit und Ihr Engagement. Ich schätze Ihre Zusammenarbeit sehr.

1 Einleitung

Proper Text

In Zusammenarbeit der Firma *Gantner Instruments....* Diese Arbeit fokussiert sich daher auf die Implementierung von einem *Test Case Management System* (TCMS) backends und vergleicht verschiedene vorhandene Lösungen.

1.1 Motivation

Die Entwicklung technischer Systeme ist ein komplexer Prozess, der eine hohe Qualität erfordert, um den Anforderungen der Kunden gerecht zu werden. Damit diese Qualität auch gewährleistet wird, müssen Fehler sowie Mängel identifiziert und ausgebessert werden. Ein TCMS bietet eine Lösung, um diesen Prozess zu vereinfachen und effektiver zu gestalten. Durch die Verwendung eines TCMS können Angestellte aus verschiedenen Abteilungen, wie z.B. der Software-, Hardware-, Support- oder Marketingabteilung, die Qualität des Produkts gemeinsam verbessern.

1.2 Problemstellung

Durch die von Abschnitt 1.1 angesprochene Komplexität technischer Systeme ist bekannt, dass Fehler, die erst spät im Entwicklungsprozess entdeckt werden, viel kostspieliger zu beheben sind als Fehler, die frühzeitig identifiziert und behoben werden.

Zitat

Infolgedessen suchen Unternehmen nach Lösungen, um den Testprozess effektiver zu gestalten und Fehler früher im Entwicklungszyklus zu identifizieren. TCMS bietet eine solche Lösung, indem es Entwicklern und Testern ermöglicht, Testfälle effizient zu planen und zu verwalten sowie Testergebnisse zu erfassen, darzustellen und somit auch zu verfolgen. Obwohl TCMS in der Industrie weit verbreitet sind und es einige fertige

Lösungen gibt, gibt es jedoch nicht immer die perfekte Lösung um ein bestehendes TCMS für das eigene Projekt anzuwenden. Insbesondere gibt es Bedenken hinsichtlich der Anwendbarkeit und Integration von einem schon bestehendem TCMS, da dieses System auch mit internen Tools kommunizieren können muss. Diese Probleme stellen Hindernisse dar, die die Einführung von TCMS in einem Unternehmen erschweren. Diese Arbeit beschäftigt sich mit diesen Problemen und untersucht, wie ein TCMS effektiv eingesetzt werden kann.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, die Verwendung von TCMSs zu untersuchen und zu bewerten. Zudem ein auf unser eigenes Produkt angepasstes TCMS backend zu entwickeln. Hiermit ergeben sich drei relevante fragen:

- Wie kann man Features, Testfälle und Testimplementierungen beschreiben?
- Wie in Datenbank schreiben und lesen?
- Wie mit Testläufen verknüpfen?

Insbesondere möchten wir die folgenden Ziele erreichen:

- Die Vor- und Nachteile der Verwendung von einem TCMS zu identifizieren und zu analysieren.
- Die Entwicklung eines TCMS backends.
- Die Konnektivität von TCMS mit anderen Tools und Systemen, die in der Softwareentwicklung verwendet werden, zu untersuchen und zu bewerten.
- Empfehlungen für die erfolgreiche Implementierung von TCMS in der Softwareentwicklung zu geben, einschließlich der Identifizierung bewährter Praktiken.

Durch die Erfüllung dieser Ziele wird diese Arbeit dazu beitragen, das Verständnis für die Verwendung von einem TCMS in der Produktentwicklung zu verbessern und Unternehmen dabei zu unterstützen, den Testprozess zu optimieren und die Qualität ihrer Produkte zu verbessern.

2 Stand des Wissens

Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik von Testarten, TCMS sowie deren Verwendung und Integration mit bestehenden Systemen. Weiters werden Technologien, auf denen das in dieser Arbeit entwickelte TCMS aufbaut, beschrieben.

2.1 Testarten

Während der Entwicklung eines Produktes kommen unterschiedliche Testarten zum Einsatz. Atlassian (2023) beschreibt sieben unterschiedliche Testverfahren die je nachdem andere Aspekte eines Produkts testen. In *Software Testing/Qualitätssicherung - Alle Methoden und Tools* (2023) wird beschrieben, dass dabei zwischen *Funktionalen* und *Nicht-Funktionalen* Tests unterschieden werden. Zu der funktionalen Testfamilie zählen beispielsweise Unit-Tests und Integrationstests. Nicht funktionale Testarten sind beispielsweise Leistung, Last und Stresstests sowie Usability-Tests zu den nicht funktionalen Testfamilie gehören.

Ein paar der am häufigsten vorkommenden Testarten sind:

- Unit-Tests
- Integrationstests
- Funktionstests
- Leistungstests

Um eine Testabdeckung sicherzustellen und den allgemeinen Testprozess zu optimieren werden TCMS eingesetzt. Diese Systeme bieten eine zentrale Plattform zur Verwaltung von Testfällen, wo Tests, die für ein Release oder Produkt erforderlich sind, erstellt und dokumentiert werden können (Siehe Abschnitt 2.4).

Ammann und Offutt (2016)

2.1.1 Unit-Tests

Im Allgemeinen sind Unit-Tests Tests die beispielsweise Methoden einer Klasse mit unterschiedlichen Parametern testet. Sie sind automatisierbar und können von einer Continuous-Integration-Pipeline durchgeführt werden. Diese Tests ermöglichen eine kontinuierliche Überprüfung der Funktionsfähigkeit des Produkts im Entwicklungsprozess und unterstützen das frühzeitige Finden von Fehlern.

2.1.2 Integrationstests

Integrationstests sind Tests, die sicherzustellen, dass verschiedene Module oder Services, problemlos miteinander interagieren können. Durch diese Tests kann die Funktionalität einzelner Teile der Anwendung überprüft werden, wie beispielsweise die Interaktion mit einer Datenbank oder der Zusammenarbeit von Microservices.

2.1.3 Funktionstests

Funktionstests werden integriert, um ausschließlich Ergebnisse einer gegebenen Funktion zu überprüfen. Dabei werden Spezifikation vor der Testausführung festgelegt und diese dann mit den Ergebnissen verglichen.

Während Integrationstests beispielsweise nur prüfen, ob Datenbankabfragen generell möglich sind, wird bei einem Funktionstest ein bestimmter Wert aus der Datenbank abgerufen und dieser mit den angegebenen Spezifikationen geprüft.

2.1.4 Leistungstests

Leistungstests prüfen das Verhalten eines Systems unter verschiedenen Lastprofilen zu überprüfen. Häufig wird Zuverlässigkeit, Geschwindigkeit, Skalierbarkeit und Reaktionsfähigkeit einer Anwendung getestet. Außerdem können Leistungstests mögliche Engpässe in einer Anwendung identifizieren.

2.2 Black-Box und White-Box Testing

Sowohl *Black-Box* als auch *White-Box* Tests werden in der Software Entwicklung häufig verwendet, um Fehler als auch die Qualität des Produktes zu evaluieren. Dabei gibt es zwischen den beiden wichtige Unterschiede.

Verbesserung
der Erklärung
von white box
testing

Bei White-Box Testing handelt es sich grundsätzlich um Unit-Tests die den Code und die Struktur des zu testeten Produkts überprüfen. Wobei der Inhalt des Codes für den Tester einsehbar ist. White-Box Testing bezieht sich dabei nur auf die interne Funktion der Software.

Bei Black-Box Testing wird die interne Struktur, das Design und die Implementierung nicht berücksichtigt. Hier werden nur die Ausgaben oder Reaktionen von dem System geprüft. Black-Box Testing bezieht sich hiermit nur auf die externe Funktion der Software.

Nidhra und Dondeti (2012, Seite 12)

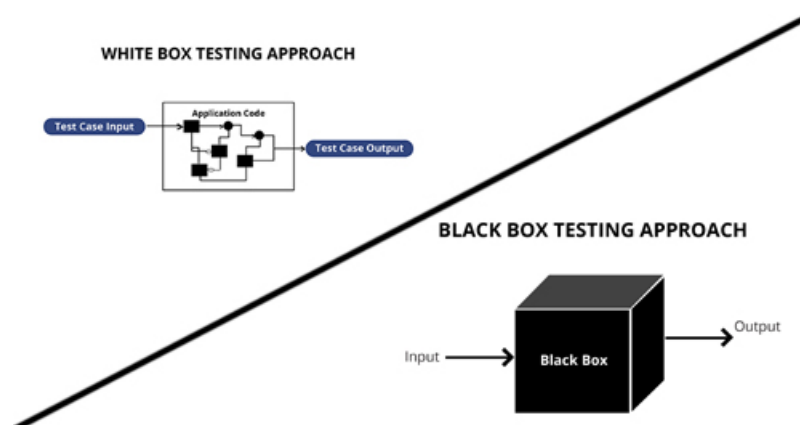


Abbildung 2.1: Black-Box/White-Box Testing (Quelle: Khandelwal (2019))

2.3 Test Driven Development

Test Driven Development (TDD) ist ein Konzept, bei dem Tests zuerst geschrieben werden und erst anschließend eine passende Implementierung erstellt wird die genau soviel beinhaltet, dass der Test erfolgreich durchgeführt werden kann. TDD bietet daher mehrere Vorteile:

- Geschriebener Code kann überarbeitet oder verschoben werden, ohne dass die Gefahr besteht, Funktionalität zu beschädigen.
- Die Tests selbst werden durch die Implementierung getestet.
- Die Anforderungen können mit geringerem Aufwand umgesetzt werden, da nur die benötigte Funktion geschrieben wird.

Ammann und Offutt (2016)

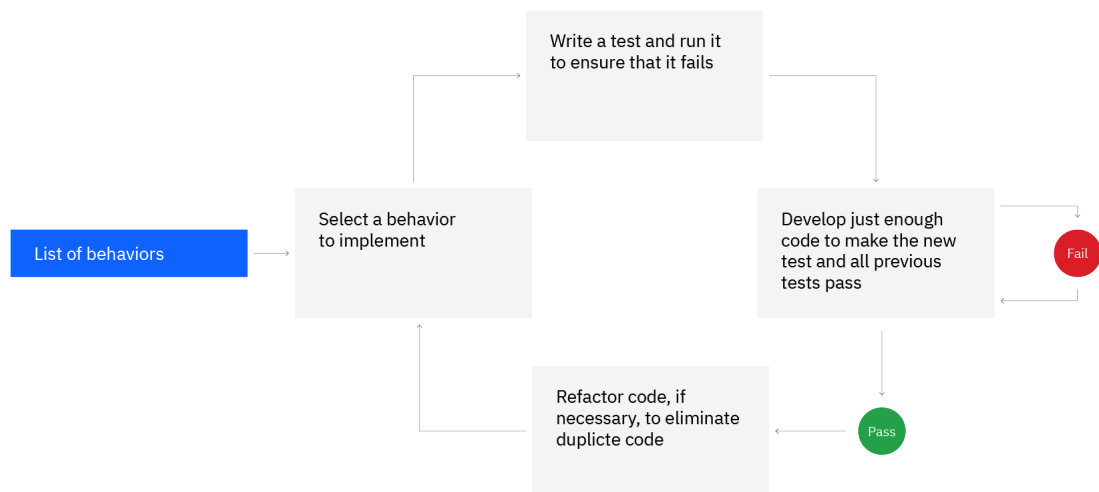


Abbildung 2.2: Test Driven Development cycle (Quelle: *Test-driven development - IBM Garage Practices* (2023))

2.4 Test Case Management System

Ein Test Case Management System (TCMS) ist eine Software, mit dem Test-Teams für ein bestimmtes Projekt oder eine Anwendung Testfälle verwalten, organisieren und analysieren können. Es hilft bei der Planung, Überwachung und Dokumentation von Tests und ermöglicht es, Testfälle sicher und effizient zu verwalten.

Ein TCMS ist ein wichtiges Werkzeug, um einen strukturierten und effektiven Testprozess zu gewährleisten und den Qualitätsstandard einer Anwendung zu verbessern. Es verfügt über Basisfunktionen wie Testfall-Erstellung, Testfall-Verwaltung, Testfall-Ausführung und Ergebnisberichterstattung. Weiters kann ein solches System auch eine

integrierte Umgebung für die Zusammenarbeit von verschiedene Abteilungen in einer Firma bereitstellen.

Zusätzlich zu den Basisfunktionen sind weitere wichtige Merkmale:

- Attraktive Benutzeroberfläche und benutzerfreundliches Design
- Nachvollziehbarkeit
- verbesserte Zeitplanung und Organisation für Releases durch Reports
- Überwachung und Metriken
- Flexibilität

Lead (2023)

2.4.1 Allgemeine Vorteile von Test Case Management Systemen

Einer der Hauptvorteile eines TCMS besteht darin, dass sie den Testprozess verbessern. Sie unterstützen auch die Kontrolle der Gesamtkosten, indem sie die Testautomatisierung nutzen, um einen reibungslosen Ablauf zu gewährleisten.

Einige Vorteile von TCMS bezüglich der Testausführungsläufe sind:

- Sie geben einen besseren Überblick über das zu testende System, halten den gesamten Prozess auf Kurs und koordinieren die Testaktivitäten.
- Sie helfen bei der Feinabstimmung des Testprozesses, indem sie die Zusammenarbeit, Kommunikation und Auswertung unterstützen.
- Sie dokumentieren Aufgaben, Fehler und Testergebnisse und vereinfachen den Prozess, indem sie alles in einer einzigen Anwendung erledigen.
- Sie sind skalierbar und können eingesetzt werden, wenn die Testaktivitäten umfangreicher und komplexer werden.

Lead (2023)

2.4.2 Überblick von bestehenden Test Case Management Systeme

Auf dem Markt gibt es eine Vielzahl fertiger und direkt verwendbaren TCMS, die die Verwaltung von Tests vereinfacht.

Einige bekannte TCMS sind:

- *TestRail – Orchestrate Testing. Elevate Quality.* (2023)

Webbasiertes TCMS, ist zentralisiert und hat folgende Features:

- Test Planung, Verwaltung und Ausführung
- Echtzeit Berichterstattung und Analysen
- Rückverfolgbarkeits- und Abdeckungsberichte für Anforderungen, Tests und Fehler

- *Tricentis Test Management for Jira* (2023)

Webbasiertes TCMS und hat folgende Features:

- Test Planung, Verwaltung und Ausführung
- Echtzeit Test Status Update
- Berichterstattung

- *Zephyr Test Management Products / SmartBear* (2023)

Webbasiertes TCMS, kann in JIRA integriert werden und hat folgende Features:

- Test Planung, Verwaltung und Ausführung
- Test Automatisierung
- Echtzeit Visualisierung von Projektstatus

- *Tricentis qTest for Unified Test Management* (2023)

Webbasiertes TCMS und hat folgende Features:

- Test Planung, Verwaltung und Ausführung
- Migrationsmöglichkeit von alten Test Management Lösungen
- Integrationsmöglichkeit mit Jenkins, Azure Pipelines, Bamboo oder jedem anderen CI/CD-Tool

- Anpassbare Dashboards, um über alle Releases, Projekte oder Programme im gesamten Unternehmen zu berichten
- Berichte per E-Mail oder URL teilen

Welches Tool am besten geeignet ist, hängt natürlich von der jeweiligen Anwendung ab.



Abbildung 2.3: Liste von aktuell führenden Test Case Management Systemen/Tools
(Quelle: *7 Best Test Management Tools* (2023))

2.5 Verwendete Technologien

Dieser Abschnitt gibt eine Übersicht über die verwendeten Technologien, die für die Implementierung des in dieser Arbeit entwickelten TCMS backends eingesetzt worden sind.

2.5.1 MariaDB

MariaDB ist eine weit verbreitete relationalen Open-Source-Datenbanken. Sie wurde auf einem Fork von MySQL basierend von den ursprünglichen Entwicklern von MySQL

entwickelt. Der Fokus von MariaDB liegt auf Leistung, Stabilität und Offenheit. Zu den jüngsten Erweiterungen gehören Clustering mit Galera Cluster 4, Kompatibilitätsfunktionen mit der Oracle-Datenbank und temporäre Datentabellen, mit denen Daten zu jedem beliebigen Zeitpunkt in der Vergangenheit abgefragt werden können.

MariaDB documentation (2023)

2.5.2 Docker

Docker ist eine Plattform mit der Entwickler einfach und schnell Container erstellen, bereitstellen, ausführen, aktualisieren und verwalten können. Container sind eigenständige, ausführbare Einheiten die unabhängig vom OS deployed werden können. Container vereinfachen die Entwicklung und Bereitstellung von verteilten Anwendungen. Entwickler können Container auch ohne Docker erstellen doch Docker macht die Containerisierung schneller, einfacher und sicherer. Telepresence ist die neuste Erweiterung und bietet einen einfachen Weg mit Kubernetes zu entwickeln.

Ghosh (2020)

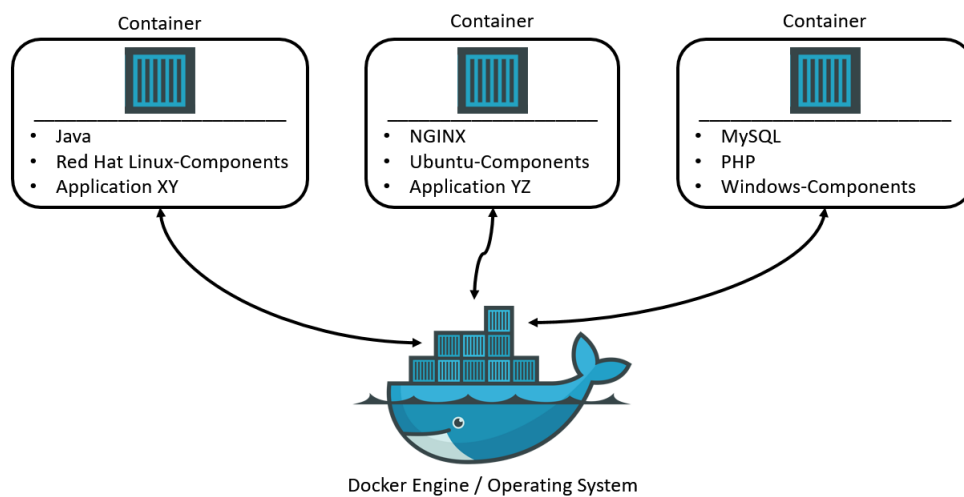


Abbildung 2.4: Funktionsweise von Docker (Quelle: Böllhoff (2022))

2.5.3 Microsoft .NET Core

Microsoft .NET Core ist ein Open-Source und plattformübergreifendes Framework um Applikationen auf Android, Apple, Linux und Windows Betriebssystemen zu entwickeln. Microsoft .NET Core unterstützt mehrere Programmiersprachen wie C#, F# und Visual Basic. Zudem bietet .NET einen Paketmanager um einfach und effizient Bibliotheken von Drittanbietern zu verwenden. Die neue Version .NET 8 bietet einige Neuerungen wie beispielsweise Performance-focused Typen, die die Leistung einer Anwendung erheblich verbessern soll. .NET 8 wird offiziell im November 2023 veröffentlicht.

BillWagner (2023)

2.5.4 Microsoft ASP.NET Core

Microsoft ASP.NET Core ist im Gegensatz zu .NET Core ein Framework um Webanwendungen zu entwickeln. ASP.NET verfügt über Vorteile wie beispielsweise Blazor mit dem man einfach und effizient eine Interaktive Webbenutzeroberfläche mit C# entwickeln kann. Dieses Framework unterstützt die Entwicklung und Implementierung von *Application Programming Interface* (API) und Mikroservices. Zudem ist ASP.NET laut *TechEmpower Web Framework Performance Comparison* (2023) schneller als jedes andere beliebte Web-Framework.

BillWagner (2023)

2.5.5 Entity Framework Core

Entity Framework (EF) Core ist eine *Object Relational Mapping* (ORM) Technik für Microsofts .NET Core Technologie (Unterabschnitt 2.5.3). Die Technologie ist Open-Source, erweiterbar und plattformübergreifend. EF Core unterstützt LINQ-Abfragen, Änderungsverfolgung, Aktualisierung sowie Schemamigration. Weiteres unterstützt EF Core mehrere Datenbanken wie beispielsweise MySQL, PostgreSQL, Azure CosmosDB etc. und auch MariaDB. Die aktuelle Version von EF Core ist 7.0 und verfügt über neue Funktionen wie z.B. das Mapping und Abfragen auf *JavaScript Object Notation* (JSON) Spalten. Dabei ist es möglich einzelne Parameter von einem JSON Objekt abzufragen und zu ändern.

```

var jeremy = await context
    .Authors
    .SingleAsync(author => author.Name.StartsWith("Jeremy"));

jeremy.Contact = new() {
    Address = new("2 Riverside", "Trimbridge", "TB1 5ZS", "UK"),
    Phone = "01632 88346"
};

await context.SaveChangesAsync();

```

Abbildung 2.5: Update von einem Parameter einer JavaScript Object Notation Spalte

BillWagner (2023)

2.5.6 OpenAPI

Die OpenAPI-Initiative hat die *OpenAPI Spezifikation* (OAS) entwickelt, die eine API Beschreibung standardisiert. Die OAS ist eine Sprache für *Hypertext Transfer Protocol* (HTTP) APIs, durch diese APIs standardisiert beschrieben werden können. Mithilfe von einem OpenAPI-Code-Generator (*OpenAPI Generator* (2023)) kann direkt Client-Code für verschiedene Technologien wie z.B. Typescript generiert werden.

OpenAPI (2023)

3 Anforderungen

Proper Anforderungen

3.1 Test

Test

4 Konzept

Proper Lösungs-
ansatz

4.1 Architektur

Proper Architek-
tur Description

Richards und Ford (2020)

4.1.1 Backend

Proper Backend
Description

4.1.2 Domain Driven Design

Proper Domain
Driven Design
Description

Vernon (2013)

4.2 Vorentwicklungsphase

Proper Vorent-
wicklungsphase
Description

4.2.1 Projektstruktur

Proper Projekt-
struktur Descrip-
tion

4.2.2 Domain Model

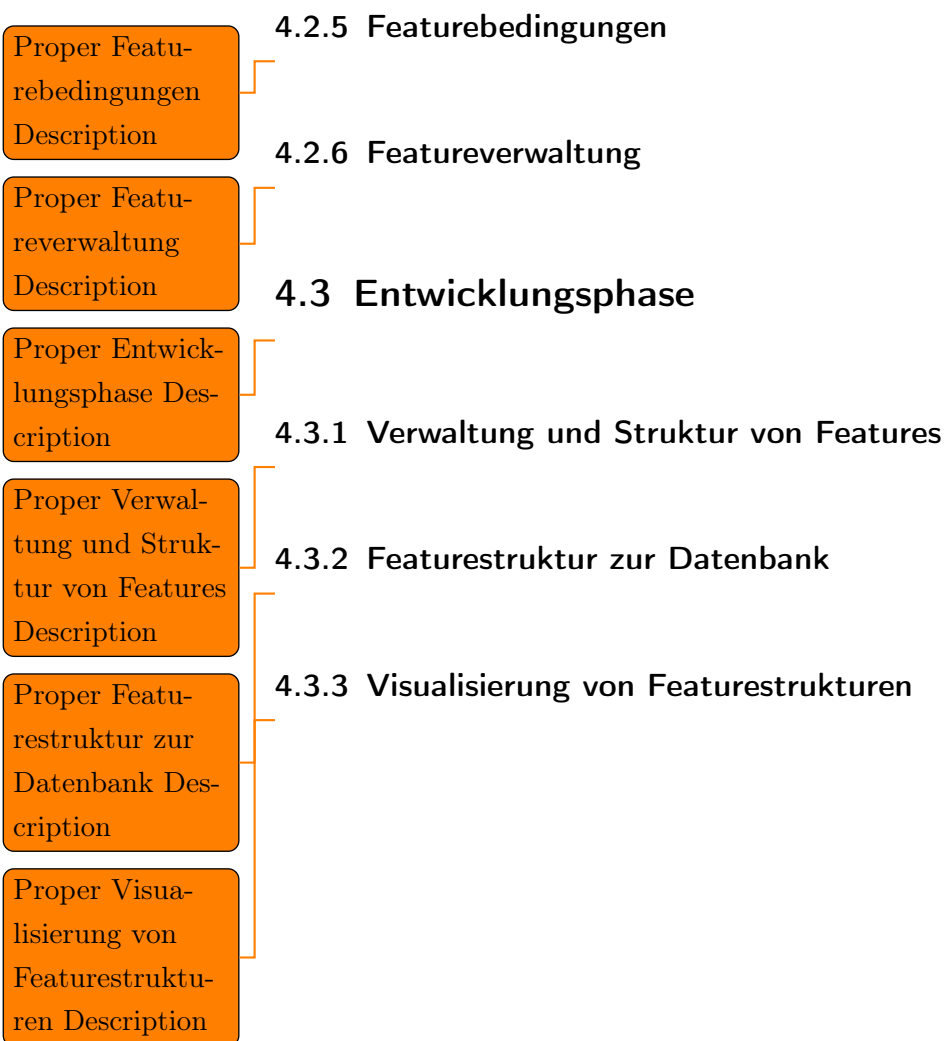
Proper Domain
Model Descripti-
on

4.2.3 ER Model

Proper ER Mo-
del Description

4.2.4 Featurestruktur

Proper Feature-
struktur Descrip-
tion



5 Technische Umsetzung

Proper Implementierung

5.1 Scrum

Proper Scrum Description

Rubin (2012)

5.2 User Stories

Proper User Stories Description

5.2.1 Story 1

Proper Story 1 Description

5.2.2 Story 2

Proper Story 2 Description

5.2.3 Story n

Proper Story n Description

6 Ergebnisse

Proper Ergebnisse
Description

7 Fazit und Ausblick

Proper Fazit
und Ausblick

Literatur

- 7 Best Test Management Tools (2023). URL: <https://www.practitest.com/test-management-tools/> (besucht am 18.04.2023).
- Ammann, Paul und Jeff Offutt (13. Dez. 2016). *Introduction to Software Testing*. Google-Books-ID: 58LeDQAAQBAJ. Cambridge University Press. 367 S. ISBN: 978-1-316-77312-3.
- Atlassian (2023). *Die unterschiedlichen Arten von Softwaretests*. Atlassian. URL: <https://www.atlassian.com/de/continuous-delivery/software-testing/types-of-software-testing> (besucht am 15.04.2023).
- BillWagner (2023). *.NET documentation*. URL: <https://learn.microsoft.com/en-us/dotnet/> (besucht am 19.04.2023).
- Böllhoff, Patrick (6. Jan. 2022). *Kubernetes vs Docker: eine Kooperation statt Konkurrenz*. Section: DevOps. URL: <https://kruschecompany.com/de/kubernetes-vs-docker/> (besucht am 05.05.2023).
- Ghosh, Saibal (3. Okt. 2020). *Docker Demystified: Learn How to Develop and Deploy Applications Using Docker*. Google-Books-ID: 650AEAAAQBAJ. BPB Publications. 243 S. ISBN: 978-93-89845-87-7.
- Khandelwal, Abhik (12. Okt. 2019). *Difference between Black Box and White Box Testing / Testing Types*. TestingGenez. URL: <https://testinggenez.com/black-box-and-white-box-testing/> (besucht am 01.05.2023).
- Lead, The QA (2023). *Articles Archives*. The QA Lead. URL: <https://theqalead.com/topics/> (besucht am 19.04.2023).
- MariaDB documentation (2023). MariaDB.org. URL: <https://mariadb.org/documentation/> (besucht am 19.04.2023).
- Nidhra, Srinivas und Jagruthi Dondeti (30. Juni 2012). „BLACK BOX AND WHITE BOX TESTING TECHNIQUES -A LITERATURE REVIEW“. In: *International journal of embedded systems and applications*. DOI: 10.5121/ijesa.2012.2204. URL: <https://www.scinapse.io/papers/2334860424> (besucht am 01.05.2023).
- OpenAPI (2023). OpenAPI Initiative. URL: <https://www.openapis.org/> (besucht am 19.04.2023).

- OpenAPI Generator* (2023). URL: <https://openapi-generator.tech/> (besucht am 19.04.2023).
- Richards, Mark und Neal Ford (28. Jan. 2020). *Fundamentals of Software Architecture: An Engineering Approach*. Google-Books-ID: xa7MDwAAQBAJ. Ö'Reilly Media, Inc." 422 S. ISBN: 978-1-4920-4342-3.
- Rubin, Kenneth S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Google-Books-ID: HkXX65VCZU4C. Addison-Wesley Professional. 501 S. ISBN: 978-0-13-704329-3.
- Software Testing/Qualitätssicherung - Alle Methoden und Tools* (2023). Software Testing/Qualitätssicherung - Alle Methoden und Tools. URL: <https://q-centric.com/> (besucht am 15.04.2023).
- TechEmpower Web Framework Performance Comparison* (2023). www.techempower.com. URL: <https://www.techempower.com/benchmarks/#section=data-r21&hw=ph&test=plaintext> (besucht am 01.05.2023).
- Test-driven development - IBM Garage Practices* (2023). URL: https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/ (besucht am 01.05.2023).
- TestRail – Orchestrate Testing. Elevate Quality*. (25. Jan. 2023). URL: <https://www.testrail.com/> (besucht am 17.04.2023).
- Tricentis qTest for Unified Test Management* (2023). Tricentis. URL: <https://www.tricentis.com/products/unified-test-management-qtest/> (besucht am 18.04.2023).
- Tricentis Test Management for Jira* (2023). Atlassian Marketplace. URL: <https://marketplace.atlassian.com/apps/1228672/tricentis-test-management-for-jira> (besucht am 17.04.2023).
- Vernon, Vaughn (6. Feb. 2013). *Implementing Domain-Driven Design*. Google-Books-ID: X7DpD5g3VP8C. Addison-Wesley. 656 S. ISBN: 978-0-13-303988-7.
- Zephyr Test Management Products | SmartBear* (2023). URL: <https://smartbear.com/test-management/zephyr/> (besucht am 17.04.2023).

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 1. Juli 2023

Marco Prescher