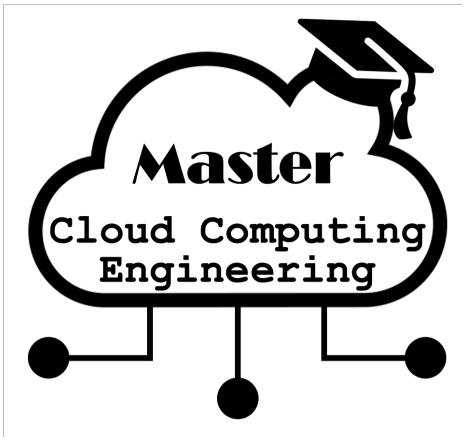


Aufgabenstellung 1 Taucher

Harald Beier* Susanne Peer† Patrick Prugger‡
Philipp Palatin§

27. April 2025

*2410781028@hochschule-burgenland.at
†2410781002@hochschule-burgenland.at
‡2410781029@hochschule-burgenland.at
§2310781027@hochschule-burgenland.at

Inhaltsverzeichnis

1 Aufgabenstellung 1 Taucher **3**
1.1 Teilbereich Build and Code 3
1.1.1 Vorgehensmodelle 3
1.1.2 Agile Methoden und Praktiken 8
1.1.3 Extreme Programming 10
1.1.4 Zusammenfassung von Artikel Spotify Scaling 12
1.1.5 Git Features 14
1.1.6 Qualitätssteigernde Maßnahmen 14
1.2 Teilbereich DevOps 16
1.2.1 Aufgabenstellung 16
1.2.2 Mögliche Probleme in der aktuellen Organisation und Arbeitsaufteilung 16
1.2.3 Notwendige Schritte um in dieser Organisation DevOps einzuführen 17
1.2.4 Reihenfolge der Schritte 17
1.2.5 Benötigte Tools 18
1.2.6 Wesentliche Stakeholder und Argumente 19

1 Aufgabenstellung 1 Taucher

1.1 Teilbereich Build and Code

Die Agile Softwareentwicklung in der Art und Weise wie Sie heute praktiziert und rezitiert wird, basiert auf dem Manifest für Agile Softwareentwicklung, dieses ist die Basis für agiles Projektmanagement und wurde 2001 als Manifesto for Agile Software Development veröffentlicht. Dieses Manifest besteht aus vier Prinzipien:

- Menschen und ihre Interaktion miteinander statt Prozesse und Werkzeuge.
- Funktionierende Produkte oder Dienstleistungen statt umfassender Dokumentation.
- Zusammenarbeit mit Kunden statt Vertragsverhandlungen.
- Reagieren auf Veränderungen statt Befolgen eines Plans. [Beck et al., 2001]

Es lässt sich also sagen, dass Agile Software Entwicklung sich aus, an Agilität ausgerichteten Methoden und agilen Prozessen zusammensetzt.

1.1.1 Vorgehensmodelle

Fragestellung: *Welche bereits vorgestellten bzw. zusätzlich recherchierte Vorgehensmodelle ermöglichen ein schnelles Iterieren und somit die Möglichkeit Feedback zeitnah durch die jeweiligen Stakeholder einzubringen? Zusätzlich soll auch darauf eingegangen werden, weshalb die gewählten Vorgehensmodelle dies im Vergleich zu anderen Modellen ermöglichen.*

Aufbauend auf das agile Manifest haben sich eine Vielzahl von Frameworks, für das Management der agilen Prozesse entwickelt. Diese sind mit Fokus auf die Software Entwicklung entstanden, haben aber ihren Weg auch schon in andere Bereiche angetreten. Einige ausgewählte Frameworks sollen nachfolgenden kurz präsentiert werden.

Tabelle 1: Agile Prozessmanagement Frameworks der Software Entwicklung

Framework	**Abkürzung**
Extreme Programming	XP
Feature Driven Development	FDD
Kanban in der IT	IT-Kanban
Adaptive Software Development	ASD
Crystal Family	CF
Agile unified process	AUP
Lean software development	LSD
Large Scale Agile Frameworks	LSAF

Die Leitwerte des **XP** sind Kommunikation, Einfachheit, Feedback und Mut und sollen durch die Praktiken: Vor-Ort-Kunde, Planspiel, Metapher, einfaches Design, kleines Release, Pair Programming, Testen, Refactoring, kontinuierliche Integration, 40-Stunden-Woche, Coding Standard und kollektive Verantwortung für die Entstehung eines optimalen Produktes sorgen. Einige dieser Praktiken bzw. Methoden werden auch in anderen agilen Frameworks genutzt, im XP wird aber stark auf die Synergieeffekte zwischen den Leitwerten und Praktiken wert gelegt. [Fojtik, 2011]
Mohammad Alshayeb und Wie Li untersuchten die spezifischen Praktiken und stellten im XP folgende Tätigkeiten als Hauptaktivitäten der Entwickler fest: Refactoring, neues Design, Beheben von Fehlern und Implementieren von Unit- sowie Funktionstests, um die Funktionstüchtigkeit des Produktes sicherzustellen. [Alshayeb and Li, 2006]

Das **FDD** hat sich aus der Coad Methode von Peter Coad entwickelt und besteht lediglich aus zwei Hauptphasen. Der Entdeckungs- und der Implementierungsphase. Hauptaugenmerk wird dabei auf die Entdeckungsphase gelegt, da hier sowohl die Liste der Features erstellt wird, als auch die UML-Diagramme der spezifischen Kundendomäne. Die Mitwirkung des Kunden ist besonders wichtig, damit die Wartbarkeit und Erweiterbarkeit des Codes im weiteren Projektverlauf gewährleistet werden kann. Die verwendete Sprache sollte von sowohl von Entwickler:innen, als auch von Kundenseite verstanden werden. [Chowdhury and Huda, 2011]

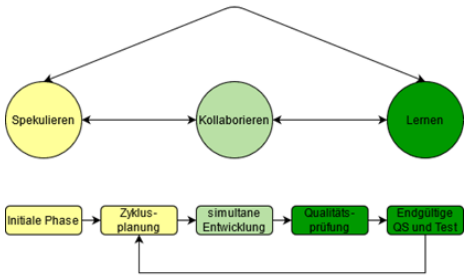
Kanban ist eine Methodik aus der Produktionsindustrie und wurde von Mary und Tom Poppenieck, und später von David Anderson für die Software Entwicklung adaptiert. Im Gegensatz zu den meisten anderen agilen Frameworks gibt es keine spezifischen Arbeitsabläufe, Rollen oder zeitliche Begrenzungen des Arbeitsprozesses in Iterationen. Hauptaufgabe ist es, den Arbeitsfluss zu koordinieren und in Teilaufgaben zu unterteilen. Anderson beschreibt dazu fünf Kernprinzipien für IT-Kanban:

- Arbeitsablauf visualisieren
- Arbeit, die noch nicht abgeschlossen ist, begrenzen des Work in Progress
- Arbeitsfluss messen und verwalten
- Richtlinien für den Arbeitsprozess definieren und explizit sichtbar machen
- Nutzung von Modellen aus Praxis oder Theorie, um Verbesserungsmöglichkeiten im Arbeitsablauf zu erkennen.

Die Visualisierung erfolgt durch das sogenannte Kanban-Board, das sowohl die Begrenzung der Arbeitslast bzw. des Arbeitspakets, als auch die Priorisierung der Aufgabe und das Eingreifen bei Engpässen im Arbeitsfluss ermöglicht.[Ahmad et al., 2018, Granulo and Tanovic, 2019]

In **ASD** wird der Fokus auf ein zyklisches Weiterentwicklungsmodell gelegt, das die unterschiedlichen Phasen im Lebenszyklus der Software widerspiegelt. [Abdelaziz et al., 2015]

Abbildung 1: ASD-Diagramm



- **Spekulieren:** ersetzt planen, um hier mehr Raum für Innovation und Ungewissheit bei komplexen Problemen zu schaffen.
- **Kollaborieren/Zusammenarbeiten:** Bei komplexen Software-Anwendungen und sich ändernden Anforderungen ist der Informationsfluss nur durch die Zusammenarbeit im Team schaffbar.
- **Lernen:** In dieser Phase können die technischen Parameter und Kundenwünsche regelmäßig nach den abgeschlossenen einzelnen Iterationen überprüft werden

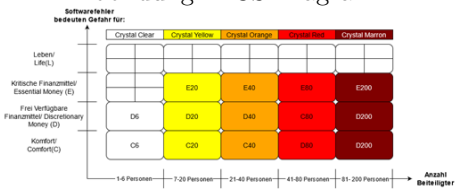
[Alnoukari et al., 2008, Abdelaziz et al., 2015]

Die **CF** wurde von Alistair Cockburn erstellt, um im Rahmen der Entwicklung von Software, ähnlich wie bei einem Kristall oder Edelstein, je nach Größe des Projektes unterschiedliche Methoden, Techniken und Richtlinien nutzen zu können. Die Methoden fokussieren sich dabei auf folgende Parameter [Ibrahim et al., 2020]:

- **Menschen**
- **Interaktionen**
- **Gemeinschaft**
- **Fertigkeiten**
- **Talente und Kommunikation**

Die Methoden von Crystal werden nach Farben benannt und nach der Größe des Projektes und der Anzahl der Projektmitarbeiter*Innen eingeteilt. Auf der Y-Achse werden die vier Level des Gefahrenlevels definiert und auf der X-Achse die Anzahl der Beteiligten. Hier angezeigt werden die Crystal Methoden, die

Abbildung 2: CS-Diagramm



noch nicht das Risikolevel Leben abdecken können. Crystal Clear sollen Projekte mit fixiertem Preis und klaren Parameter sein und sobald Projekte Gefahr für Leib und Leben beinhalten, handelt es sich dabei um Crystal Diamond und Crystal Sapphire.[Cockburn, 2004, Ibrahim et al., 2020]

AUP ist ein Framework bzw. Modellierungsansatz, der von Scott Ambler aus dem Rational Unified Process (RUP) mit agilen Methoden kombiniert entwickelt wurde. Ambler definiert dafür folgende Kernprinzipen [Christou et al., 2010]:

- Die meisten Menschen werden keine detaillierte Dokumentation lesen. Sie werden jedoch hin und wieder Anleitung und Schulung benötigen
- Das Projekt sollte einfach mit wenigen Seiten beschrieben werden.
- Die AUP entspricht den von der Agile Alliance beschriebenen Werten und Prinzipien.
- Das Projekt muss sich darauf konzentrieren, einen wesentlichen Wert zu liefern und nicht unnötige Funktionen.
- Die Entwickler müssen die Freiheit haben, Werkzeuge zu verwenden, die für die jewei-lige Aufgabe am besten geeignet sind, und nicht, um eine Vorschrift zu erfüllen.
- AUP lässt sich über gängige HTML-Editierwerkzeuge leicht anpassen.

Es werden dann vier seriell ablaufende Phasen definiert [Li and Wang, 2010, ShuiYuan et al., 2009]:

- **Inception/Beginn**: Ziel ist es, den anfänglichen Umfang des Projekts und eine potenzielle Architektur für Ihr System festzulegen, sowie die anfängliche Projektfinanzierung und die Akzeptanz der Stakeholder zu erhalten.
- **Elaboration/Ausarbeitung:** Die Architektur des Systems wird überprüft.
- **Construction/Konstruktion:** Es soll regelmäßig und schrittweise funktionierende Software erstellt werden, welche die Anforderungen der Projektbeteiligten mit höchster Priorität erfüllt.

- **Transition/Übergabe:** Das Ziel ist die Validierung und der Einsatz ihres Systems in ihrer Produktionsumgebung

LSD hat als Vorbild das Toyota Produktionssystem und definiert sieben Prinzipien und 22 Praktiken für die Umsetzung im Rahmen der agilen Softwareentwicklung. Die sieben Prinzipien sind [Janes, 2015, Jonsson et al., 2013]:

- Verschwendung vermeiden
- Lernen unterstützen
- So spät entscheiden wie möglich
- Verantwortung an das Team geben
- Integrität einbauen
- Das Ganze sehen

Unterschiedliche **LSAF** sind entwickelt worden, um agile Praktiken auf große Projekte und Softwareentwicklung mit weltweit verteilten Teams anzuwenden. Zu den bekanntesten Modellen zählen[Beecham et al., 2021, Ebert and Paasivaara, 2017]:

- **Scrum of Scrums (SoS)**
- **Scaled Agile Framework (SAFe)**
- **Large-Scale Scrum (LeSS)**
- **Disciplined Agile Delivery (DAD)**
- **Lean Scalable Agility for Engineering (LeanSAFE)**

Kieran Conboy und Noel Carroll definieren ausgehend von einer 15-jährigen Retrospektive folgende Herausforderungen bei der Implementation von Large Scale Agile Frameworks [Conboy and Carroll, 2019, Kasauli et al., 2021]:

- Definieren von Konzepten
- Vergleich und Gegenüberstellung von Frameworks
- Bereitschaft und Appetit auf Veränderung
- Abgleich von Organisationsstruktur und Rahmenbedingungen
- Top-down- versus Bottom-up-Implementierung
- Überbetonung der 100 %igen Einhaltung der Regeln des Frameworks gegenüber dem Mehrwert für die Firma
- Fehlender evidenzbasierter Einsatz
- Erhaltung der Autonomie der Entwickler und Entwicklerteams
- Fehlende Abstimmung zwischen Kundenprozessen und Frameworks

1.1.2 Agile Methoden und Praktiken

Es existiert eine große Vielfalt an unterschiedlichen Methoden. Zur Vereinfachung werden diese kurz in Englisch angeführt:

Tabelle 2: Tabellarische Auflistung agiler Methoden

Methoden	**Abkürzung**
Acceptance test-driven development	ATDD
Agile modeling	AM
Agile testing	AT
Backlogs	BL
Behavior-driven development	BDD
Cross-Functional Team	DST
Daily Stand-up	CFT
Domain-driven design	DDD
Iterative and incremental development	IID
Planning poker	PlP
Refactoring	RF
Retrospective	RS
Specification by example	SBE
Story-driven modeling	SDM
Timeboxing	TB
Velocity tracking	VT

ATTD: Hierbei handelt es sich um eine Methode, welche großen Wert auf die Kommunikation der Kund*Innen, Entwickler*Innen und Tester*Innen legt. Wichtig sind hierbei die Implementierung von Akzeptanztests bevor die Entwicklungsteams mit dem coden beginnen. Diese Tests sollen in der Sprache der Geschäftsdomäne geschrieben werden. Eine Anforderung bzw. User Story für welche kein Test vor Anforderungsumsetzung geschrieben wurde, ist schlecht implementiert.[Downs, 2011]

AM: Dies ist eine Methode für Modellierung und Dokumentation von Softwaresystemen, die auf unterschiedlichen Kernprinzipien beruht.[Ambler, 2010]

AT: Hiermit werden die Parameter für das Testen und Testszenarien im agilen Softwareentwicklungsumfeld beschrieben. Der Fokus liegt auf der Unterstützung der Entwicklungsteams. Ein Beispiel für die Implementierung ist die Definition of Test (DoT), die die Parameter für die Testdurchführung und Strategie definiert. [Crispin and Gregory, 2009]

BL sind eine Liste von Arbeitspaketen, die geordnet sind nach Priorität und Reifegrad der einzelnen Artefakte. Die bekanntesten Varianten der Arbeitspaketformate sind User Stories oder Use Cases. Es werden Features, Pro-

blemlösungen und andere nicht technische Anforderungen dokumentiert, um ein Softwareprodukt liefern zu können.[Svensson et al., 2019]

BDD ist eine agile Testtechnik, deren Ziel es ist, Software-Anforderungen als Beispielinteraktionen mit dem System zu spezifizieren, wobei eine strukturierte natürliche Sprache verwendet wird. Während die Beispiele (in der Theorie) von nicht-technischen Stakeholdern gelesen werden können, können sie auch gegen die Codebasis ausgeführt werden, um Verhaltensweisen zu identifizieren, die noch nicht korrekt implementiert sind. [Binamungu et al., 2020]

CFT: Hierbei handelt es sich um Teams aus unterschiedlichen Fachbereichen einer Firma. Ziel ist es, die Wissenssilos innerhalb der eigenen Organisation aufzubrechen und damit die Kundenbe-dürfnisse an die erste Stelle zu bringen. Innerhalb der Softwareentwicklung bedeutet dies oft, dass Frontend, Backend, Devops, Tester*Innen, Interface Designer*Innen und Personen, die für die Geschäftslogik zuständig sind, zusammenarbeiten.[McDonough, 2000]

DST: Dies ist eine spezifische Meeting-Art, die täglich stattfinden soll, um sich gegenseitig über den aktuellen Stand der Arbeit zu informieren. Diese Meetings sollten time-boxed (siehe weiter unten) sowie im Stehen stattfinden, um das Meeting fokussiert durchzuführen. Je nach Umsetzung gibt es dafür einen kurzen Fragenkatalog.[Stray et al., 2016]

DDD ist eine Vorgehensweise, in der versucht wird, das Domänenwissen für den Softwareentwurf korrekt abzufragen, zu erfassen und modellbasiert darzustellen. Ziel ist, die Beziehungen unterschiedlicher Domänenkonzepte zu identifizieren. DDD findet oft Anwendung beim Entwurf von Microservice-Architekturen, da es eine Schnittmenge zwischen den einzelnen funktionalen Microservices und den verschiedenen Geschäftsfeldern gibt.[Rademacher et al., 2018]

IID ist die Sammelbezeichnung für unterschiedliche Konzepte und Modelle für iterative und inkrementelle Vorgehensmodelle in der Softwareentwicklung. Diese verschiedenen Ansätze sind aber alle der Versuch eines Gegenmodelles zum traditionellen Ansatz einer gleichzeitigen In-tegration aller Teilkomponenten zum Abschluss des Projektes. Durch kleine, häufige Schritte und Auslieferung von Teilsystemen, soll eine sequenzielle, dokumentengesteuerte Auslieferung in einem Durchgang vermieden werden. [Larman and Basili, 2003]

PlP ist Teil der Planungsmeetings für die nächste Iteration und dient der Schätzung der Zeitres-sourcen, die notwendig sind, um ein neues Feature zu implementieren. Die Kalkulation erfolgt zumeist in sogenannten Story Points, die in der Regel einem idealen Arbeitstag entsprechen. Zumeist werden für die Schätzung vordefinierte Werte wie die Fibonacci-Folge verwendet. [Mahnič and Hovelja, 2012]

RF ist der Prozess, in dem der Quellcode so umstrukturiert wird, dass die interne Codequalität und Struktur der Software verbessert wird, dabei aber

das externe Verhalten, das für die Benutzer ersichtlich ist, gleichbleibt. Ziel ist es, die Wartungskosten zu verringern und die Lebensdauer der Software zu verlängern.[Kaur and Singh, 2019]

RS sind Meetings bzw. Aktivitäten zur Erfahrungssicherung der Teammitglieder. Oft auch als Post-Mortems bezeichnet, in der die Probleme sowie Ursachen eines abgeschlossenen Projektes oder einer Softwareiteration analysiert werden. Es sollen durch diese Untersuchung sowohl die Erfolgsfaktoren als auch die Auslöser von Problemen erkannt werden.[Lehtinen et al., 2014]

SBE ist ein Leitfaden oder eine kollaborative Methode, deren oberstes Ziel die Entwicklung von Software ist, welche die Kundenanforderungen erfüllt. Es wird hier mit Workshops gearbeitet, in denen die verschiedenen Rollen und Standpunkte vertreten sind und die gemeinsam spezifischen Szenarien der Nutzung anhand von Beispielen entwerfen. Mit Hilfe dieser Szenarien sollen folgend die automatisierten und funktionalen Tests entwickelt werden. Ziel ist es, eine Dokumentation zu erstellen, die in der Sprache der jeweiligen Fachdomäne geschrieben wurde und auch von Personen aus den Fachabteilungen, die keine Programmierer sind, gelesen werden kann.[Blasquez and Leblanc, 2017, Bache and Bache, 2014]

Im Gegensatz zu anderen Arten der objektorientierten Modellierung wird im **SDM** die Struktur nicht durch statische Klassendiagramme und der Beziehung dieser Klassen zueinander Wert gelegt, sondern auf die Erstellung von spezifischen Beispielen, die Szenarien in der Nutzung abbilden. Zudem sollen die daraus entstehenden Objektdiagramme sich im Laufe des Szenarios und der Ausführung der Software weiterentwickeln.[Wautelet et al., 2017]

TB ist die Definition eines Zeitrahmens oder einer Zeitphase, die innerhalb der Projektplanung für gewisse Vorgänge genutzt werden darf. Es wird hier ein Zeitbudget für die jeweiligen Aufgaben erstellt. Insbesondere im Scrum und der agilen Softwareentwicklung wird Timeboxing für einzelne Vorgänge wie Meetings und Iterationszyklen verwendet. [Miranda, 2011]

VT ist die Messung und Darstellung der Team Performanz. Zwei für die Darstellung verwendete Methoden sind Burn-Down und Burn-Up Charts.[Al-Sabbagh and Gren, 2018]

Bei diesen verschiedenen Praktiken ist ersichtlich, dass einige bereits die Produktivität (z.B. IID, VT, PlP, BL), Ergebnisorientierung (z.B.: RF, TB) sowie die Weiterentwicklung der Teams (z.B.: RF, PP, AT, AM) innerhalb der agilen Softwareentwicklung mitgedacht bzw. angedacht haben.

1.1.3 Extreme Programming

Fragestellung: *Beschreiben Sie mindestens 3 Praktiken aus Extreme Programming im Detail und den Einfluss die diese Praktik auf die Softwareentwicklung*

und dem Ergebnis haben.

Pair Programming Beim Pair Programming arbeiten zwei Entwickler gemeinsam an einem Computer. Eine Person (der Driver) schreibt den Code, während die andere Person (der Navigator) den Code überprüft, Probleme identifiziert und strategische Entscheidungen trifft. Die Rollen werden regelmäßig gewechselt.

Einfluss auf die Softwareentwicklung:

- **Verbesserte Codequalität:** Durch kontinuierliches Review werden Fehler früher erkannt und behoben.
- **Wissenstransfer:** Entwickler lernen voneinander, was zu einer breiteren Verteilung von Wissen im Team führt.
- **Bessere Lösungsansätze:** Durch die Kombination verschiedener Perspektiven entstehen oft kreativere und effizientere Lösungen.
- **Reduzierte technische Schulden:** Die kontinuierliche Überprüfung verhindert Abkürzungen und schlechte Praktiken.

Continuous Integration (CI) Bei der Continuous Integration werden Codeänderungen mehrmals täglich in ein gemeinsames Repository integriert. Nach jeder Integration werden automatisierte Tests durchgeführt, um sicherzustellen, dass die Änderungen keine Fehler verursachen.

Einfluss auf die Softwareentwicklung:

- **Frühe Fehlererkennung:** Probleme werden unmittelbar nach ihrer Entstehung identifiziert, was die Behebungskosten drastisch reduziert.
- **Reduzierte Integrationszeit:** Durch häufige kleine Integrationen werden große, problematische Merges vermieden.
- **Höhere Softwarestabilität:** Die Software bleibt kontinuierlich in einem funktionsfähigen Zustand.
- **Schnelleres Feedback:** Entwickler erhalten unmittelbare Rückmeldung zu ihren Änderungen.

Test-Driven Development (TDD) Bei TDD werden Tests geschrieben, bevor der eigentliche Code implementiert wird. Der Entwicklungsprozess folgt einem Red-Green-Refactor-Zyklus: Zuerst wird ein fehlschlagender Test geschrieben (Red), dann wird gerade genug Code implementiert, um den Test zu bestehen (Green), und schließlich wird der Code verbessert, ohne seine Funktionalität zu ändern (Refactor).

Einfluss auf die Softwareentwicklung:

- **Klarere Anforderungen:** Das Schreiben von Tests zwingt Entwickler, die Anforderungen genau zu verstehen.

- **Besseres Design:** TDD fördert modularen, entkoppelten Code, der leichter zu testen ist.
- **Umfassende Testabdeckung:** Jede Funktionalität wird durch Tests abgedeckt, was zu robusterer Software führt.
- **Dokumentation durch Tests:** Tests dienen als lebende Dokumentation, die zeigt, wie der Code verwendet werden soll.
- **Sicherheit bei Refactoring:** Entwickler können Code mit Vertrauen umgestalten, da Tests Regressionen aufdecken.

1.1.4 Zusammenfassung von Artikel Spotify Scaling

Aufgabenstellung: *Lesen Sie den Artikel https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf. Diese Informationen fassen Sie bitte in 1 bis maximal 2 Seiten zusammen*

Das Spotify-Modell zur Skalierung agiler Organisationen: Das Spotify-Modell stellt eines der meistzitierten Beispiele für die Skalierung agiler Methoden in der zeitgenössischen Softwareentwicklung dar. Es wurde 2012 von Kniberg und Ivarsson öffentlich vorgestellt. In einer Phase rapiden Wachstums mit Niederlassungen in mehreren Städten entwickelte Spotify eine Organisationsstruktur, die darauf abzielte, Agilität durch kleine, autonome Teams zu bewahren und gleichzeitig Kohärenz, Wissensaustausch und architektonische Integrität sicherzustellen.

1. Squads - Autonome Feature-Teams: Ein Squad bei Spotify ist ein funktionsübergreifendes, selbstorganisierendes Team mit einer langfristigen Mission (z.B. Android-Client, Backend-Skalierung, Zahlungssysteme). Die Squads wählen ihre eigenen Arbeitsprozesse (Scrum, Kanban oder hybride Ansätze) und integrieren Lean-Startup-Prinzipien - sie veröffentlichen MVPs (Minimum Viable Products) und nutzen Daten aus A/B-Tests, um dem Motto *Denken, Bauen, Ausliefern, Optimieren* zu folgen. Um Innovation zu fördern, widmet jedes Squad etwa 10% seiner Zeit sogenannten Hack Days. In diesen Phasen verfolgen Teammitglieder Nebenprojekte oder experimentieren mit neuen Werkzeugen, was häufig zu Verbesserungen bei Produkten oder Entwicklungswerkzeugen führt. Product Owner verantworten die Priorisierung des Backlogs, während Agile Coaches (in die Squads integriert) Retrospektiven moderieren und die Weiterentwicklung der Arbeitsprozesse unterstützen.

2. Tribes - Kohärente Domänen: Thematisch zusammenhängende Squads bilden einen Tribe, dessen Größe typischerweise auf etwa 100 Personen begrenzt ist - in Anlehnung an die Dunbar-Zahl, um informellen sozialen Zusammenhalt zu gewährleisten und Bürokratie zu minimieren. Tribes veranstalten regelmäßige Demonstrationen, bei denen Squads ihre Fortschritte, Werkzeuge und Ergebnisse ihrer Hack-Days präsentieren.

Abhängigkeitsmanagement Spotify führt vierteljährliche Umfragen unter den Squads durch, um Abhängigkeiten zu erfassen und blockierende Probleme zu identifizieren. Diese werden als Kreise (aktueller Zustand) und Pfeile (Trend) visualisiert. Die erhobenen Daten führen zu Priorisierungsanpassungen, Reorganisation oder architektonischen Veränderungen, um Engpässe zu beseitigen. Anstelle eines permanenten Scrum of Scrums findet die Koordination zwischen Squads bedarfsorientiert statt - beispielsweise als temporäre tägliche Synchronisation mit einem gemeinsamen Board für Notizen im Rahmen eines Squad-übergreifenden Projekts.

3. Chapters und Guilds - Horizontale Gemeinschaften: Um Silobildung zu vermeiden, führte Spotify Chapters (fachbereichsbezogene Gruppen innerhalb eines Tribes, geleitet von einem Chapter Lead, der gleichzeitig als Linienmanager fungiert) und Guilds (organisationsweite, freiwillige Interessengemeinschaften) ein. Chapters gewährleisten technische Konsistenz, während Guilds (z.B. Web Guild Unconference) den Wissensaustausch und die Werkzeugentwicklung über Tribe-Grenzen hinweg ermöglichen, vergleichbar mit Communities of Practice.

4. Architektur und Systemeigentümerschaft: Das Backend von Spotify besteht aus über 100 lose gekoppelten Diensten. Squads können diese unabhängig bereitstellen, riskieren dabei jedoch *Architekturerosion*. Um diesem Risiko entgegenzuwirken, verfügt jedes System über einen System Owner – häufig ein Squad-Mitglied oder ein DevOps-Paar, das für Stabilität, Dokumentation, technische Schulden und periodische *System-Owner-Tage* zur Systempflege verantwortlich ist. Ein Chief Architect bietet übergeordnete Leitlinien in beratender Funktion und stellt sicher, dass neue Dienste mit der architektonischen Gesamtvision übereinstimmen, ohne die Autonomie der Squads zu untergraben.

5. Operations-Team als Ermöglicher: Anstelle eines Übergabemodells baut und pflegt ein Operations-Team bei Spotify die Release-Infrastruktur (Skripte, Routinen, Umgebungen), damit Squads ihre Releases selbstständig durchführen können. Diese *Straße zur Produktion* reduziert Reibung zwischen Entwicklung und Betrieb und verkörpert DevOps-Ideale geteilter Verantwortung.

6. Implikationen für DevOps und skalierte Agilität: Das Spotify-Modell veranschaulicht exemplarisch, wie dezentralisierte Autonomie und leichtgewichtige Abstimmungsstrukturen koexistieren können. Es unterstreicht:

- **Kontinuierliche Experimentation** durch MVPs, A/B-Tests und Hack Days
- **Dynamische Koordination** durch bedarfsgesteuerte Synchronisation und Abhängigkeitsanalysen
- **Gemeinsame Verantwortung** durch System Owners und ein unterstützendes Operations-Team

- **Wissensnetzwerke** ermöglicht durch Chapters und Guilds

Als Fallstudie illustriert es den iterativen, menschenzentrierten Ansatz, der erforderlich ist, um Agilität in großen, schnell wachsenden Softwareorganisationen aufrechtzuerhalten.

1.1.5 Git Features

Fragestellung: Beschreiben Sie Git im Detail sowie mindestens 2 Features im Detail.

Was ist Git? Git ist ein verteiltes Versionskontrollsystem, das 2005 von Linus Torvalds entwickelt wurde. Es ermöglicht die Verwaltung von Änderungen an Dateien und Projekten, wobei jeder Entwickler eine vollständige Kopie des Projekts und seiner Historie auf seinem lokalen System hat. Git ist besonders für seine Geschwindigkeit, Datenintegrität und Unterstützung für nicht-lineare, verteilte Workflows bekannt. [GitHub, 2024]

Feature 1: Branching und Merging Git bietet ein leistungsfähiges Branching-System, das es ermöglicht, parallele Entwicklungslinien zu erstellen und zu verwalten. Branches sind leichtgewichtige Zeiger auf einen bestimmten Commit. Entwickler können schnell neue Branches erstellen, zwischen ihnen wechseln und sie zusammenführen. Dies ermöglicht:

- Isolierte Entwicklung neuer Features
- Experimentieren ohne Risiko für den Hauptcode
- Parallele Arbeit an verschiedenen Aspekten des Projekts
- Einfaches Zusammenführen von Änderungen durch Merging

Feature 2: Distributed Version Control Git ist ein vollständig verteiltes System, was bedeutet, dass jeder Entwickler eine vollständige Kopie des Repositories besitzt. Dies bietet mehrere Vorteile:

- Offline-Arbeit ist möglich
- Schnelle Operationen durch lokale Ausführung
- Redundanz und Backup durch multiple Kopien
- Flexible Workflow-Möglichkeiten
- Keine zentrale Schwachstelle

1.1.6 Qualitätssteigernde Maßnahmen

Fragestellung: *Welche qualitätssteigenden Maßnahmen kennen Sie? Beschreiben Sie 2 beliebige im Detail.*

Statische Code-Analyse: Bei der statischen Code-Analyse wird der Quellcode automatisiert auf potenzielle Fehler, Sicherheitslücken und Code-Smells untersucht, ohne dass er tatsächlich ausgeführt wird. Tools wie SonarQube bieten hierfür umfassende Reports zu Duplikaten, Komplexität und Schwachstellen und lassen sich direkt in IDEs und CI-Pipelines integrieren.[SonarSource, 2023, Ayewah and Pugh, 2008] Bereits beim Build-Prozess aufgesetzte Quality Gates sorgen dafür, dass Builds bei Verstößen gegen definierte Schwellenwerte (z.B. erhöhte Zyklomatische Komplexität oder Sicherheitswarnungen) automatisch blockiert werden. [Beller et al., 2016]

Durch diesen maschinellen Vorfilter verringert sich die Menge an einfachen, aber zeitaufwändigen Review-Aufgaben erheblich - Entwickler werden gezielt auf die kritischsten Schwachstellen hingewiesen und können sie schon in einer frühen Phase beheben.[Ayewah and Pugh, 2008] Gleichzeitig schafft die durchgängige Analyse historischer Trends in Metriken wie Code-Duplikation oder Testabdeckung eine transparente Basis, um technische Schulden systematisch abzubauen. Langfristig steigert sich so nicht nur die Wartbarkeit, sondern auch die Sicherheit und Zuverlässigkeit der Software. [SonarSource, 2023]

Code Reviews Code Reviews sind strukturierte Peer-Inspektionen, bei denen Entwickler:innen Änderungen im Quellcode ihrer Kolleg:innen über Pull- oder Merge-Requests prüfen. Dieses Vorgehen geht zurück auf klassische Inspektionsmethoden nach Fagan (1976), bei denen ein formalisiertes Review-Meeting systematisch alle Aspekte des Codes beleuchtet.[Fagan, 1976] Moderne Plattformen wie GitHub, GitLab oder Bitbucket unterstützen diesen Prozess durch integrierte Kommentar-Threads und automatisierte Review-Checks. [Atlassian, 2024, Cohen, 2010]

Der unmittelbare Nutzen zeigt sich in frühzeitigem Feedback und einem deutlich geringeren Aufkommen von Produktionsfehlern: Studien belegen, dass Teams mit etablierten Review-Prozessen bis zu 65 % weniger kritische Bugs in der Produktivumgebung haben.[Bacchelli and Bird, 2013] Darüber hinaus fördert Code Review den Wissensaustausch: Teammitglieder lernen neue Patterns und Architekturstile kennen, was die kollektive Codekompetenz stärkt und Einarbeitungszeiten verkürzt [Rigby et al., 2013].

Best-Practices empfehlen kurze, regelmäßige Reviews anstelle großer Inspektionsschlachten, um den Kontext frisch zu halten und schnelle Korrekturschleifen zu ermöglichen. Die Verantwortlichkeiten sollten rotieren, damit nicht immer dieselben Personen reviewen, und Review-Metriken (z.B. durchschnittliche Review-Dauer, Anzahl der Kommentare) in Dashboards sichtbar gemacht werden. So wird das Review-Feedback unmittelbar ins Entwicklerteam zurückgeführt und hohe Codequalität nachhaltig verankert. [Atlassian, 2024, Rigby et al., 2013].

1.2 Teilbereich DevOps

1.2.1 Aufgabenstellung

Das Unternehmen ABC Ad Tech stellt eine SaaS-Lösung für Kunden bereit mit denen ihre Werbekampagnen verwaltet werden können. Aktuell gibt es ein Entwicklungsteam namens Ad-Dev mit 5 Personen die gemeinsam die Lösung entwickeln. Der Source Code ist hierzu in git abgelegt. Der Output des Build-Prozesses (=Artefakt) wird auf dem Firmen-PC von einem speziellen Mitarbeiter erstellt und dann händisch auf eine Netzwerkdateifreigabe kopiert. Für die Kunden wird die Lösung aktuell durch das Team namens Ad-Ops betrieben. Die beiden Teams arbeiten unabhängig voneinander. Das Entwicklungsteam Ad-Dev liefert alle 3 Monate eine neue Produktivversion und alle zwei Woche eine neue Testversion. Beide werden von Team Ad-Ops bei Verfügbarkeit eingespielt, d.h. es wird das Artefakt von der Netzwerkdateifreigabe heruterkopiert und dann händisch eingespielt. Die Testversion kommt immer auf eine Staging-Umgebung, welche vom Team Ad-QS getestet wird und anschließend wird das Feedback mittels einer Besprechung an die Entwicklung rückgemeldet. Auch die Produktivversion wird zuerst auf der Staging-Umgebung getestet und sobald die Freigabe seitens Ad-QS gegeben wird erfolgt die Einspielung auf das Produktivsystem durch das Team Ad-Ops. Bei Problemen und sonstigen Auffälligkeiten wird vom Team Ad-Ops aus Kontakt mit dem Entwicklungsteam Ad-Dev aufgenommen.

1.2.2 Mögliche Probleme in der aktuellen Organisation und Arbeitsaufteilung

Fragestellung: *Beschreibe kurz mögliche Probleme in der aktuellen Organisation und Arbeitsaufteilung*

DevOps-Einführung bei ABC Ad Tech: Von manuellen Handoffs zur Continuous Delivery

Kurzüberblick - Warum sich der aktuelle Handoff bei ABC Ad Tech wie stille Post anfühlt Sie kennen das Szenario: Fünf *Ad-Dev-Ingenieure* kämpfen mit Git, ein einzelner Build-Spezialist kopiert Artefakte manuell, und ein *Ad-Ops-Team* klickt sich alle zwei Wochen - oder vierteljährlich - durch den Deployment-Prozess. Es funktioniert irgendwie - aber es ist fehleranfällig, intransparent und hemmt Innovationen. Lassen wir uns einen reibungsloseren Weg skizzieren, der zu Continuous Delivery (CD) führt, ohne das Organigramm über Nacht umzuschreiben.

Was bremst uns aus? - Mainpainpoints im heutigen Setup

- **Siloartige Teams und Übergabereibung:** Ad-Dev erstellt Artefakte auf einem PC und legt sie in einer Freigabe ab; Ad-Ops holt sie dann ab. Das ist fehleranfällig und schafft blinde Flecken, wenn Fehler auftreten.
- **Rhythmus-Diskrepanz:** Alle zwei Wochen eine Testversion, alle drei Monate ein Produktiv-Rollout - doch Feedback-Schleifen hängen von manuellen Staging-Umgebungen und Meetings ab. Diese Verzögerung killt die Dynamik.
- **Mangel an Automatisierung** Keine CI, keine automatisierten Tests zur Qualitätssicherung und keine Pipeline-Transparenz. Builds, Tests und Deployments sind vollständig manuell.
- **Verborgenes Wissen und Engpässe** Ein Spezialist besitzt das Wissen über den Build-Prozess. Ist diese Person nicht verfügbar, kommt alles ins Stocken. Zudem existiert Insiderwissen in Köpfen, nicht in Dokumentationen oder Skripten.

1.2.3 Notwendige Schritte um in dieser Organisation DevOps einzuführen

Fragestellung: *Beschreibe die notwendigen Schritte um in dieser Organisation DevOps (bis einschließlich Continuous Delivery) einzuführen.*

Einen DevOps-Kurs abstecken DevOps ist keine Wunderpille - es ist eine Sammlung von Praktiken, die Entwicklung und Betrieb durch Automatisierung, geteilte Verantwortung und kürzere Feedback-Schleifen verbindet. Unser Ziel: Jeden Commit durch eine automatisierte CI-Pipeline schieben, Tests durchführen und dann auf Abruf in Staging und Produktion ausliefern (Continuous Delivery).

1.2.4 Reihenfolge der Schritte

Fragestellung: *Welche Schritte sind in welcher Reihenfolge notwendig?* Nachfolgend unser **Schritt für Schritt - Plan für eine ABC AD Tech Pipeline** und glücklichere Devs- und Engineers.

- **Shift Left: Build- und Test-Automatisierung**-Phase 1 (Wochen 1-4)

  - Einführung eines CI-Servers (z.B. Azure Pipelines, Bitbucket, AWS CodePipeline,GitLab).

  - Automatisierung von Builds bei jedem Push auf main oder Feature-Branches.

  - Automatische Ausführung von Unit- und Integrationstests.

- **Artefakt-Repository und Versionierung**-Phase 2 (Wochen 3-6)

  - Deployment eines Artefakt-Repositorys (z.B. Nexus, Azure Artifacts).

- Build-Ergebnisse direkt in versionierte Pakete übertragen.

- **Automatisiertes Deployment in die Staging-Umgebung**-Phase 3 (Wochen 5-8)

  - Skriptgesteuerte Deployments (Infrastructure as Code) auf einen Staging-Cluster.

  - Workflows durch automatisierte Smoke-Tests absichern.

- **Feedback-Loop-Integration**- Phase 4 (Wochen 7-10)

  - Integration von Testberichten und Metriken in Dashboards (z.B. Grafana).

  - Weiterleitung von Staging-Ergebnissen zurück an Ad-Dev via Slack oder Teams.

- **Continuous Delivery in die Produktion**- Phase 5 (Wochen 9-12)

  - Freigaben in der Pipeline für Produktiv-Pushes aktivieren.

  - Feature-Flag-gesteuerte Rollouts für sichere Canary-Deployments einrichten.

Jede Phase baut auf der vorherigen auf - kein großer Knall, sondern inkrementelle Erfolge.

1.2.5 Benötigte Tools

Fragestellung: *Welche Tools werden hierzu benötigt?* Nachfolgend sind Beispiel des **Werzeugkasten** der Pipline zu finden.

- **Versionskontrolle / Code-Review:** Git (Branches, Hooks, PR/MR-Workflows)
- **CI/CD-Plattform:** Jenkins, GitHub Actions, GitLab CI oder Azure Pipelines
- **Artefakt-Management:** Nexus, Artifactory oder Azure Artifacts
- **Infrastructure as Code:** Terraform, Azure Resource Manager (ARM)-Templates oder Bicep
- **Konfigurationsmanagement:** Ansible oder Azure Automation
- **Monitoring und Feedback:** Prometheus + Grafana, Azure Monitor oder ELK-Stack
- **Zusammenarbeit** Slack/MS Teams-Integrationen, Confluence oder SharePoint-Dokumente

Diese Tools arbeiten Hand in Hand, um Builds, Tests und Deployments zu automatisieren - und beseitigen das manuelle Kopieren und Einfügen.

1.2.6 Wesentliche Stakeholder und Argumente

Fragestellung: *Beachte das solche Änderung Schritt für Schritt eingeführt werden müssen damit diese erfolgreich sein können. Ebenso müssen die wesentlichen Stakeholder überzeugt werden. Identifiziere die wesentlichen Stakeholder und liefere ihnen Argumente, die sie davon überzeugen das die Einführung von DevOps Praktiken vorteilhaft ist.*

Im nächsten Abschnitten schauen wir uns an **Wen betrifft es? - Stakeholder und unsere Win-Win-Argumente**

- **CTO / Leiter Engineering** wird besonders hellhörig, wenn es um messbare Geschäftsvorteile geht. Hier punktet die DevOps-Einführung mit schnellerer *Time-to-Market*, was im werbegetriebenen Geschäft von ABC Ad Tech unmittelbar wettbewerbsrelevant ist. Zudem sinkt das Risiko kostspieliger fehlgeschlagener Releases erheblich, während gleichzeitig eine bessere Ressourcenverteilung möglich wird - Ingenieure können sich auf wertschöpfende Tätigkeiten konzentrieren, statt in manuellen Prozessen festzustecken.

- **Dev-Lead** für ihn bietet die Transformation eine willkommene Entlastung vom *ständigen Feuerlöschen*. Das unmittelbare Feedback durch automatisierte Tests gibt seinen Entwickler:innen die Sicherheit, dass ihre Änderungen funktionieren, bevor sie in größere Codestränge einfließen. Die Verringerung von Merge-Konflikten durch häufigere Integration spart wertvolle Entwicklungszeit, die sonst für konfliktlösende Meetings verloren ginge.

- **Ops-Lead** gewinnt durch vorhersehbare, vollständig dokumentierte Deployments erheblich an Betriebsstabilität. Besonders wertvoll ist für ihn die Selbstbedienungsfähigkeit - keine frustrierenden Wartezeiten mehr, bis jemand die richtigen Artefakte auf der Netzwerkfreigabe ablegt. Seine Teams können sich auf strategischere Aufgaben konzentrieren, statt repetitive manuelle Einspielungen durchzuführen.

- **QA-Lead** bekommt durch DevOps einen Quantensprung (die Metapher,nicht die physikalische diskrete Zustandsänderungen im Quantensystemen) in der Testqualität. Konsistente Testumgebungen, die automatisch und identisch aufgesetzt werden, eliminieren das lästige *Bei mir funktioniert es aber*-Syndrom. Die frühzeitige Fehlererkennung direkt nach einem Commit verkürzt die Fehleranalyse dramatisch, während die durchgängige Nachverfolgbarkeit vom Code bis zum Deployment endlich Klarheit bei auftretenden Problemen schafft.

- **Produktmanager** sieht den größten Gewinn in den verkürzten Feedback-Schleifen. Statt alle drei Monate auf eine große Release zu warten, können neue Features schneller validiert und bei Bedarf angepasst werden. Diese Agilität führt zu zufriedeneren Kunden, die neue Funktionen schneller nutzen können und dadurch eine höhere Bindung an die SaaS-Lösung von ABC Ad Tech entwickeln.

Indem DevOps als Reihe inkrementeller Verbesserungen statt als radikaler Umbau dargestellt wird, sieht jeder Stakeholder einen direkten Gewinn in Produktivität, Qualität oder Agilität.

Zusammenfassung Der Wechsel von *handkopierten*-Artefakten zu einer vollautomatisierten CI/CD-Pipeline dreht sich weniger um ausgefallene Werkzeuge und mehr um eine Denkweise - geteilte Verantwortung, Automatisierung des Mundanen und Verkürzung von Feedback-Schleifen von Wochen auf Minuten. Fangen wir klein an, messen wir unsere Erfolge und bauen durch diese Erfolge Dynamik auf. Bevor wir es merken, wird der vierteljährliche Kraftakt wie eine ferne Erinnerung erscheinen, und wir werden in Cloud-Geschwindigkeit ausliefern - und lernen.

Abbildung 3: To the moon

Literaturverzeichnis

[Abdelaziz et al., 2015] Abdelaziz, A. A., El-Tahir, Y., and Osman, R. (2015). Adaptive software development for developing safety critical software. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*. IEEE.

[Ahmad et al., 2018] Ahmad, M. O., Dennehy, D., Conboy, K., and Oivo, M. (2018). Kanban in software engineering: A systematic mapping study. *J. Syst. Softw.*, 137:96–113.

[Al-Sabbagh and Gren, 2018] Al-Sabbagh, K. W. and Gren, L. (2018). The connections between group maturity, software development velocity, and planning effectiveness. *J. Softw. (Malden)*, 30(1):e1896.

[Alnoukari et al., 2008] Alnoukari, M., Alzoabi, Z., and Hanna, S. (2008). Applying adaptive software development (ASD) agile modeling on predictive data mining applications: ASD-DM methodology. In *2008 International Symposium on Information Technology*. IEEE.

[Alshayeb and Li, 2006] Alshayeb, M. and Li, W. (2006). An empirical study of relationships among extreme programming engineering activities. *Inf. Softw. Technol.*, 48(11):1068–1072.

[Ambler, 2010] Ambler, S. W. (2010). *The object primer*. Cambridge University Press, Cambridge, England, 3 edition.

[Atlassian, 2024] Atlassian (2024). Code review best practices. Zuletzt abgerufen am 27. April 2025.

[Ayewah and Pugh, 2008] Ayewah, N. and Pugh, W. (2008). Using static analysis to find bugs. In *Proceedings of the 2008 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 321–330. IEEE.

[Bacchelli and Bird, 2013] Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721. IEEE Press.

[Bache and Bache, 2014] Bache, E. and Bache, G. (2014). Specification by example with GUI tests - how could that work? In *Lecture Notes in Business Information Processing*, Lecture notes in business information processing, pages 320–326. Springer International Publishing, Cham.

[Beck et al., 2001] Beck, K. et al. (2001). Manifest für agile softwareentwicklung.

[Beecham et al., 2021] Beecham, S., Clear, T., Lal, R., and Noll, J. (2021). Do scaling agile frameworks address global software development risks? an empirical study. *J. Syst. Softw.*, 171(110823):110823.

[Beller et al., 2016] Beller, M., Gousios, G., and Zaidman, A. (2016). Static analysis: A critical evaluation. *IEEE Software*, 33(4):51–59.

[Binamungu et al., 2020] Binamungu, L. P., Embury, S. M., and Konstantinou, N. (2020). Characterising the quality of behaviour driven development specifications. In *Lecture Notes in Business Information Processing*, Lecture notes in business information processing, pages 87–102. Springer International Publishing, Cham.

[Blasquez and Leblanc, 2017] Blasquez, I. and Leblanc, H. (2017). Specification by example for educational purposes. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA. ACM.

[Chowdhury and Huda, 2011] Chowdhury, A. F. and Huda, M. N. (2011). Comparison between adaptive software development and feature driven development. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*. IEEE.

[Christou et al., 2010] Christou, I., Ponis, S., and Palaiologou, E. (2010). Using the agile unified process in banking. *IEEE Softw.*, 27(3):72–79.

[Cockburn, 2004] Cockburn, A. (2004). Crystal clear a human-powered methodology for small teams.

[Cohen, 2010] Cohen, M. (2010). The business value of code review. Technical report, SmartBear Software. Zuletzt abgerufen am 27. April 2025.

[Conboy and Carroll, 2019] Conboy, K. and Carroll, N. (2019). Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Softw.*, 36(2):44–50.

[Crispin and Gregory, 2009] Crispin, L. and Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 1 edition.

[Downs, 2011] Downs, G. (2011). Lean-agile acceptance test-driven development. *Softw. Eng. Notes*, 36(4):34–34.

[Ebert and Paasivaara, 2017] Ebert, C. and Paasivaara, M. (2017). Scaling agile. *IEEE Softw.*, 34(6):98–103.

[Fagan, 1976] Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

[Fojtik, 2011] Fojtik, R. (2011). Extreme programming in development of specific software. *Procedia Computer Science*, 3:1464–1468. World Conference on Information Technology.

[GitHub, 2024] GitHub (2024). About git.

[Granulo and Tanovic, 2019] Granulo, A. and Tanovic, A. (2019). Comparison of SCRUM and KANBAN in the learning management system implementation process. In *2019 27th Telecommunications Forum (TELFOR)*. IEEE.

[Ibrahim et al., 2020] Ibrahim, Aftab, Bakhtawar, Ahmad, Iqbal, Aziz, Javeid, and Ihnaini (2020). Exploring the agile family: A survey. *Int. J. Comput. Sci. Netw. Secur.*, 20(10):163–179.

[Janes, 2015] Janes, A. (2015). A guide to lean software development in action. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE.

[Jonsson et al., 2013] Jonsson, H., Larsson, S., and Punnekkat, S. (2013). Synthesizing a comprehensive framework for lean software development. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE.

[Kasauli et al., 2021] Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., and de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. *J. Syst. Softw.*, 172(110851):110851.

[Kaur and Singh, 2019] Kaur, S. and Singh, P. (2019). How does object-oriented code refactoring influence software quality? research landscape and challenges. *J. Syst. Softw.*, 157(110394):110394.

[Larman and Basili, 2003] Larman, C. and Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer (Long Beach Calif.)*, 36(6):47–56.

[Lehtinen et al., 2014] Lehtinen, T. O. A., Virtanen, R., Viljanen, J. O., Mäntylä, M. V., and Lassenius, C. (2014). A tool supporting root cause analysis for synchronous retrospectives in distributed software teams. *Inf. Softw. Technol.*, 56(4):408–437.

[Li and Wang, 2010] Li, J. and Wang, X. (2010). Research and practice of agile unified process. In *2010 2nd International Conference on Software Technology and Engineering*. IEEE.

[Mahnič and Hovelja, 2012] Mahnič, V. and Hovelja, T. (2012). On using planning poker for estimating user stories. *J. Syst. Softw.*, 85(9):2086–2095.

[McDonough, 2000] McDonough, III, E. F. (2000). Investigation of factors contributing to the success of cross-functional teams. *J. Prod. Innov. Manage.*, 17(3):221–235.

[Miranda, 2011] Miranda, E. (2011). Time boxing planning. *Softw. Eng. Notes*, 36(6):1–5.

[Rademacher et al., 2018] Rademacher, F., Sorgalla, J., and Sachweh, S. (2018). Challenges of domain-driven microservice design: A model-driven perspective. *IEEE Softw.*, 35(3):36–43.

[Rigby et al., 2013] Rigby, P. C., Storey, M.-A., and Zimmermann, T. (2013). A survey of modern code review: Benefits, challenges and best practices. Technical report, Microsoft Research. Zuletzt abgerufen am 27. April 2025.

[ShuiYuan et al., 2009] ShuiYuan, H., LongZhen, D., Jun, X., JunCai, T., and GuiXiang, C. (2009). A research and practice of agile unified requirement modeling. In *2009 International Symposium on Intelligent Ubiquitous Computing and Education*. IEEE.

[SonarSource, 2023] SonarSource (2023). *SonarQube Documentation*. Zuletzt abgerufen am 27. April 2025.

[Stray et al., 2016] Stray, V., Sjøberg, D. I. K., and Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *J. Syst. Softw.*, 114:101–124.

[Svensson et al., 2019] Svensson, R. B., Gorschek, T., Bengtsson, P.-O., and Widerberg, J. (2019). BAM - backlog assessment method. In *Lecture Notes in Business Information Processing*, Lecture notes in business information processing, pages 53–68. Springer International Publishing, Cham.

[Wautelet et al., 2017] Wautelet, Y., Heng, S., Kiv, S., and Kolp, M. (2017). User-story driven development of multi-agent systems: A process fragment for agile methods. *Comput. Lang. Syst. Struct.*, 50:159–176.