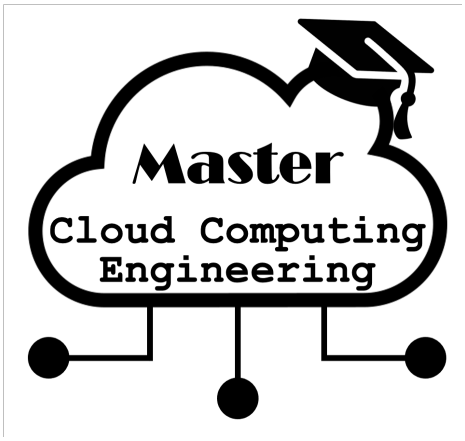


Aufgabenstellung 1 Taucher

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

24. April 2025

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

Inhaltsverzeichnis

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. Agile Software Entwicklung setzt sich aus an Agilität ausgerichteten Methoden und agilen Prozessen zusammen.

1.1.1 Vorgehensmodelle

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

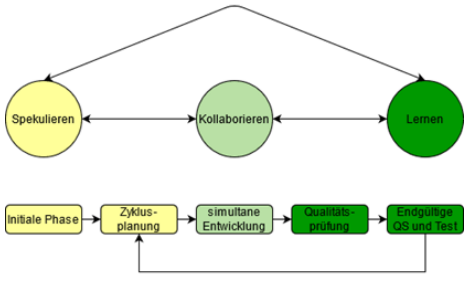
Die Leitwerte des **XP** sind Kommunikation, Einfachheit, Feedback und Mut und sollen durch die Praktiken: Vor-Ort-Kunde, Planspiel, Metapher, einfaches Design, kleines Release, Pair Programming, Testen, Refactoring, kontinuierliche Integration, 40-Stunden-Woche, Coding Standard und kollektive Verantwortung 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 nur aus zwei Hauptphasen. Der Entdeckungsphase 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 (WIP)
- Arbeitsfluss messen und verwalten
- Richtlinien für den Arbeitsprozess definieren und explizit sichtbar machen
- Nutzung von Modellen aus der Praxis oder Theorie, um Verbesserungsmöglichkeiten im Arbeitsablauf zu erkennen.

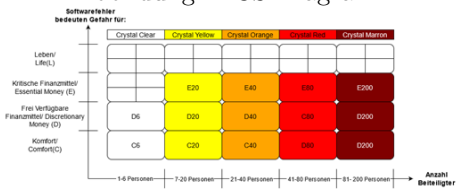
Die Visualisierung erfolgt durch das sogenannte Kanban-Board, das sowohl die Begrenzung der Arbeitslast bzw. des Arbeitspakets, als auch die Priorisierung der Aufgabe und das Eingreifen bei Engpässen im Arbeitsfluss ermöglicht. [Ahmad et al., 2018, Granulo and Tanovic, 2019]
In **ASD** wird der Fokus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

Abbildung 2: CS-Diagramm



- **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 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

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.

Scrum

Extreme Programming

Test-Driven Development

Kanban

1.1.2 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.

1.1.3 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

1.1.4 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

1.1.5 Qualitätssteigernde Maßnahmen

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

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 heruntergekopiert 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

1.2.3 Notwendige Schritte um in dieser Organisation DevOps einzuführen

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

1.2.4 Reihenfolge der Schritte

Fragestellung: Welche Schritte sind in welcher Reihenfolge notwendig?

1.2.5 Benötigte Tools

Fragestellung: Welche Tools werden hierzu benötigt?

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.

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.

[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.

[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.

[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.

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

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

[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.

[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.

[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.