Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Untersuchung der Abdeckung vom Software-Lebenszyklus durch den gewählten Entwicklungsprozess

Examination of the shaping of the software lifecycle with the choosen development process

Bachelorarbeit

im Studiengang Informatik

von

Lukas Dreyer

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Dr. Jil Klünder

Hannover, 13.07.2022

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 13.07.2022

Lukas Dreyer

Zusammenfassung

In Software-Projekten werden zunehmend hybride Entwicklungsmethoden verwendet, statt nur einen bestimmten Ansatz oder Methode zu nutzen. Dabei sollen mehrere Methoden, wie z.B. Kanban und Scrum, dann zusammen eingesetzt werden und somit die jeweiligen Vorteile von diesen kombiniert werden und somit die Phasen des Software-Lebenszyklus besser und vollständiger abgedeckt werden. Neben Methoden, können aber auch Praktiken in solchen hybriden Ansätzen kombiniert werden. Dabei wird nicht immer berücksichtigt, welche Phasen bei einer Kombination von Methoden und Praktiken nun wirklich abgedeckt werden und wie sinnvoll das Kombinieren von diesen Methoden und Praktiken aufgrund dessen wirklich ist.
Ziel dieser Arbeit ist, mittels einer geeigneten Studie, weiterer Literatur und Untersuchungen, die Abdeckung der Phasen des Software-Lebenszyklus von bestimmten Methoden und Praktiken zu untersuchen und für jede Methode und Praktik jeder Phase des SW-Lebenszyklus eine Stufe der Abdeckung zuzuordnen. Dabei gibt es für die Abdeckung insgesamt drei Stufen.
Mit diesen Ergebnissen wird dann eine Anwendung mit einer grafischen Benutzeroberfläche entwickelt, welche die Abdeckung der Phasen des Software-Lebenszyklus für eine ausgewählte Kombination von Methoden und Praktiken berechnet und grafisch in einem Diagramm darstellt. Die Stufe der Abdeckung einer Phase durch die Kombination der Methoden und Praktiken in dem Diagramm wird dabei mit der Helligkeit und Stärke der Farbe dieser dargestellt.
Mittels der Daten der HELENA-Studie werden die Abdeckungen der verschiedenen Kombinationen von den Methoden und Praktiken in der Anwendung dann mit den Nutzungen dieser in der Industrie verglichen.

Abstract

Examination of the shaping of the software lifecycle with the choosen development process

In software-projects, hybrid development methods are more often used then the use of only one single approach or method. In hybrid approaches, multiple methods should be used together. The benefits of them can be combined. Consequently, the software-lifecycle would be better and more completely shaped. Practices can also be combined in such a hybrid approach. It is not often considered, which phases really are shaped by a combination of methods and practices, so if it is sensible to combine them in such an approach.
The goal of this work is, with a certain appropriate study, other literature and other studies, to examine the shaping of the phases of the software-lifecycle for a defined set of methods and practices and to assign to every phase a shaping level for each method and practice. Overall, there are three shaping levels.
With these results, an application with a graphical user interface will be developed, which calculates the shaping of the phases of the software-lifecycle for a selected combination of methods and practices and displays it graphical in a diagram. The shaping level of a phase in the diagram will be represented by the density and brightness of the colour of the filling of the phase.
With the data of the HELENA-study, the shapings of the different combinations of the methods and practices in the application are compared with the application of them in industry.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Problemstellung

Bei der Entwicklung von Software spielt der gewählte Entwicklungsprozess häufig eine sehr wichtige Rolle, da dieser für die erfolgreiche Bereitstellung der Software maßgeblich entscheidend sein kann. Dabei steht vor allem im Vordergrund, welche Phasen des Software-Lebenszyklus abgedeckt werden und welche eher weniger. Nutzt man zum Beispiel einen Entwicklungsprozess, welcher lediglich die Phase Implementierung abdeckt, aber weniger oder gar nicht die Phase Anforderungsanalyse, dann sind dies keine zuversichtlichen Voraussetzungen für eine erfolgreiche Entwicklung und Bereitstellung der geplanten Software. Jedoch ist bis heute weder einheitlich definiert, welche Phasen des Software-Lebenszyklus von einem bestimmten Entwicklungsprozess oder einer Kombination von Methoden und Praktiken abgedeckt werden, noch ist festgelegt, welche Methoden oder Praktiken kombiniert alle Phasen abdecken können und somit zusammen genutzt werden sollten.

1.2 Lösungsansatz

Um für ausgewählte Entwicklungsprozesse bzw. Methoden und Praktiken die Abdeckung der Phasen des Software-Lebenszyklus zu analysieren und auswerten zu können, soll auf Basis einer bereits erfolgten Studie und weiterer Literatur sowie Untersuchungen eine Anwendung entwickelt werden, welche die Abdeckung der Phasen bei Auswahl bestimmter Methoden und Praktiken berechnet und dann grafisch übersichtlich in einem Diagramm darstellt. Somit kann dann vor der Entwicklung von Software mittels dieser Anwendung analysiert werden, ob sich die geplanten Methoden oder Praktiken dafür eignen.

1.3 Struktur der Arbeit

Diese Arbeit ist wie folgt strukturiert:
In Kapitel 2 werden die Grundlagen von den für die Arbeit ausgewählten Methoden und Praktiken in der Software-Entwicklung und der Bedeutung von dem Software-Lebenszyklus und dem hybriden Entwicklungsprozess erläutert.
Daraufhin wird in Kapitel 3 auf die Anforderungen und die Stakeholder der zu entwickelnden Anwendung eingegangen.
In Kapitel 4 wird die Interpretation und Nutzung der für die Entwicklung der Anwendung gegebenen Werte der Studie erläutert. Also wie mit diesen Werten den Phasen des Software-Lebenszyklus für jede der Methoden und Praktiken eine bestimmte Abdeckung zugeordnet wird. Dazu wird neben den gegebenen Werten auch auf andere Studien und Literatur zurückgegriffen. Es werden dabei auch die möglichen Ursachen für die gegebenen Werte bezüglich der Abdeckung diskutiert.
In Kapitel 5 wird die gewählte Programmiersprache und das gewählte Framework sowie andere Erweiterungen beschrieben, welche für die Entwicklung genutzt wurden. Darauf folgend wird die Architektur der Anwendung erläutert und danach auf die Gestaltung und Bedienung dieser eingegangen.
In Kapitel 6 wird dann die von der Anwendung ausgegebene Abdeckung der Phasen bei bestimmten Kombinationen von Methoden evaluiert und dabei mit der tatsächlichen Nutzung dieser Kombinationen in der Industrie verglichen.
In Kapitel 7 werden die Ergebnisse interpretiert und darauf eingegangen, inwiefern sich die Erwartungen bei der Evaluation der Anwendung bestätigt haben oder nicht. Außerdem werden die Limitierungen dieser Arbeit beschrieben.
In Kapitel 8 werden verwandte Arbeiten vorgestellt und von dieser Arbeit abgegrenzt.
In Kapitel 9 wird eine kurze Zusammenfassung und abschließend ein Ausblick für die Arbeit und Anwendung gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen des Software-Lebenszyklus, der untersuchten Methoden und Praktiken und der hybriden Software-Entwicklung dargelegt.

2.1 Software-Lebenszyklus und Phasen

Jedes Softwaresystem durchläuft einen Software-Lebenszyklus, dieser beginnt bei der Idee oder Planung der Software und endet entweder mit der Abschaltung, Ablösung oder Weiterentwicklung der Software [20][44][115]. Dabei besteht der Software-Lebenszyklus aus mehreren Phasen. Typischerweise sind diese wie folgt:
In der ersten Phase werden die Anforderungen an die Software ermittelt, in der zweiten Phase wird die Architektur der Software entworfen, in der dritten Phase wird diese dann implementiert und die Anforderungen umgesetzt, in der vierten Phase findet die Installation (auch als Auslieferung bezeichnet) auf dem System statt und in der fünften Phase dann die Wartung und der Betrieb der Software [20][44][115]. Zwischen der Implementierung und Installation oder Auslieferung der Software kann auch eine Phase stattfinden, in welcher das Testen oder die Erprobung der Software vollzogen wird [44]. Zusätzlich gibt es noch weitere Phasen im Software-Projekt, wie das Projektmanagement, das Risikomanagement, das Anforderungsmanagement, das Qualitätsmanagement, das Änderungsmanagement und das Konfigurationsmanagement [65][39].

2.2 Methoden

2.2.1 Scrum

Scrum ist ein Vorgehensmodell bzw. Rahmenwerk um komplexe Aufgabenstellungen angehen und Produkte mit hohem Wert ausliefern zu können [124][117]. Das Fundament von Scrum besteht aus drei Säulen, nämlich Transparenz, Überprüfung und Anpassung [99][124][117]. Diese Säulen werden erst dann lebendig, wenn die fünf Scrum-Werte Commitment, Mut, Offenheit, Fokus und Respekt durch die Team-Mitglieder aktiv verkörpert werden [99][124][117]. Scrum besteht aus den Scrum-Teams mit ihren Rollen, den Artefakten, Ereignissen und den Regeln [99][124][117]. Die Regeln bestimmen die Beziehungen zwischen den Artefakten, Ereignissen sowie den Rollen der Teams [99][124][117].

Rollen

Zu den Rollen innerhalb eines Scrum-Teams zählen der Product-Owner, die Entwickler und der Scrum-Master [99]. Der Product-Owner organisiert den Product-Backlog und die sich in diesem befindenden Anforderungen oder User Stories [99].
Der Scrum-Master ist für die Unterstützung der Kommunikation und Zusammenarbeit innerhalb der Entwickler-Teams zuständig [99][124][117]. Dieser beseitigt eventuelle Hindernisse in der Zusammenarbeit, ist aber keinem vorgesetzt oder besitzt irgendeine Weisungsbefugnis gegenüber den Entwicklern oder gar dem Product-Owner [99][124][117]. Die Entwickler sind für die Umsetzung und Implementierung der Anforderungen im Sprint-Backlog, möglichst innerhalb eines Sprints, zuständig [124][117].

Artefakte

Zu den Artefakten oder Erzeugnissen von Scrum gehören der Product Backlog, der Sprint Backlog und das Inkrement[124][117]. Der Product Backlog enthält die gesamten Anforderungen für das zu entwickelnde Produkt. Dieser kann während der Entwicklung kontinuierlich anwachsen und wird ausschließlich vom Product Owner verwaltet und gepflegt [124][117]. Der Sprint Backlog enthält alle Backlog Items, also vor allem Anforderungen, welche in dem (kommenden) Sprint implementiert und fertiggestellt werden sollen. Ausschließlich die Entwickler selbst entscheiden, welche Backlog Items für den Sprint Backlog ausgewählt werden sollen [124][117]. Ein Inkrement bezeichnet eine neue Version oder einen weiteren Schritt des entwickelten Produkts bis zum fertigen Produkt(-ziel). Innerhalb einem Sprint werden Anforderungen implementiert und diese Version vom Produkt am Ende, ist dann ein (weiteres) Inkrement. Ein Inkrement enthält dabei alle vorherigen Inkremente, diese bauen also immer aufeinander auf [124][117].

Ereignisse

Zu den Ereignissen von Scrum zählen der Sprint, das Daily Scrum Meeting, die Retrospektive, der Sprint Review und das Sprint Planning [99][124][117]. Der Sprint, auch als Iteration bezeichnet, ist quasi das wichtigste Ereignis von Scrum. In diesem findet die Entwicklung und Implementierung der Anforderungen für das Produkt statt. Dieser kann ein bis maximal vier Wochen lang sein. Nach jedem Sprint sollte ein ausführbares Inkrement ausgeliefert und hergestellt worden sein [99, p. 44]. Beim Sprint Planning wird diskutiert und geplant, was und wie in dem kommenden Sprint entwickelt werden soll [99][124][117]. Es wird innerhalb des Ereignisses also vor allem festgelegt und entschieden, welche Backlog Items aus dem Product Backlog ausgewählt werden sollen. Das Sprint Planning findet vor jedem Sprint einmal statt [99][124][117].
Bei dem Daily Scrum Meeting finden sich die Entwickler jeden Tag morgens einmal zusammen, um dann den Plan für diesen Tag zu besprechen und gegebenfalls die Kommunikation zwischen den Entwicklern verbessern [99][124][117]. An diesem Meeting nehmen alle Entwickler und der Scrum-Master teil und gegebenenfalls der Product Owner und andere Stakeholder nur als Zuhörer [99][124][117].
Im Sprint Review wird das Ergebnis des aktuellen Sprints begutachtet. Dabei wird auch diskutiert, ob an dem Product Backlog bestimmte Änderungen vorgenommen werden sollen [99][124][117]. Auch das Präsentieren des Ergebnisses des Sprints vor bestimmten Stakeholdern kann zu diesem Ereignis dazugehören. Der Sprint Review findet immer am Ende eines Sprints statt [99][124][117].
Die Retrospektive findet direkt nach dem Sprint Review statt und dient vorrangig der Reflexion und Verbesserung der Effizienz und der Qualität der Zusammenarbeit während des Sprints. An diesem nimmt das ganze Scrum-Team teil [99][124][117].

2.2.2 DevOps

DevOps bezeichnet einen Ansatz bzw. eine Projektmanagement-Methode, die Entwicklung (Development) und den Betrieb (Operations), die Unternehmensleitung und die Qualitätssicherung zusammenzubringen, damit diese gemeinsam an einer kontinuierlichen Softwareentwicklung arbeiten [112][39]. Dieser Ansatz basiert dabei auf Lean-Prinzipien (Lean Development) und Agilen-Methoden [112]. DevOps ist jedoch im Gegensatz zu anderen Frameworks (wie z.B. Scrum) kein fest definierter Prozess, manche betrachten DevOps zum Beispiel auch nur als Philosophie [85]. Aus diesem Grund haben viele Organisationen oder Zertifizierer auch unterschiedliche Vorstellungen von DevOps [85].

2.2.3 Kanban

Kanban ist als Vorgehensmodell der schlanken Software-Entwicklung weit verbreitet [49, p. 34-36]. Dabei werden die sieben Werte der schlanken Software-Entwicklung bei Kanban durch vier bestimmte, charakterisierende Elemente realisiert [49, p. 53-54]. Diese charakterisieren Kanban in der Software-Entwicklung, somit geht dann eine bestimmte projektindividuelle Kultur und Arbeitsweise im Software-Projekt hervor [49, p. 20]. Die folgenden vier charakterisierenden Elemente von Kanban sind also maßgebend:

- **Pull**

Arbeit und Aufgaben werden nicht gegeben und geschoben, also von einer Phase in die nächste Phase, sondern von dem Team oder einem Team-Mitglied aus der "Empfänger"-Phase gezogen (en. pull), sofern dieses sich für die Bearbeitung einer nächsten Aufgabe zur Verfügung sieht. Somit entsteht z.B. keine Überlastung von dem Entwickler-Team, indem dieses (ständig) mit neuen Aufgaben zugeworfen wird [49, p. 54-56].

- **Limitierte Mengen**

Die gleichzeitig zu bearbeitenden Aufgaben in einer Phase (und in der gesamten Wortschöpfungskette) sind limitiert. Es besteht somit ein Work-in-Progress-Limit (WIP-Limit). Dadurch wird also die Anzahl der gleichzeitig zu bearbeitenden Aufgaben verringert und damit auch die Durchlaufzeit, somit werden die Aufgaben also (im Schnitt) schneller bearbeitet und abgeschlossen [49].

- **Transparente Information**

Damit auch jedes Team-Mitglied eigenverantwortlich und selbstorganisiert arbeiten kann, soll jedes Team-Mitglied über Informationen über die Phasen der Wertschöpfungskette, über die Aufgaben in diesen Phasen, über die an diesen Aufgaben arbeitenden Personen, die Limitierung der Anzahl von Aufgaben für jede Phase und die Projektkennzahlen verfügen [49].

- **Kontinuierliche Verbesserung**

In Kanban wird das Streben nach kontinuierlicher Verbesserung gefordert. Dabei gilt es, Verbesserungen mit der Beteiligung aller Team-Mitglieder durchzuführen. Verbesserungen werden durch folgende Vorgänge und Handlungen durchgeführt und geebnet: Durch die Einführung neuer Werte, Elemente oder Techniken, durch das Hinterfragen bestehender Werte, Elemente oder Techniken oder das Aufgeben nicht länger benötigter Werte, Elemente oder Techniken. Durch die Vergangenheit und dadurch erlernter Erfahrung soll also für die Zukunft gelernt werden und dies kontinuierlich. Kanban soll so zu einem empirischen Vorgehensmodell werden [49].

2.3 Praktiken

Coding Standards
Mit diesen Richtlinien wird die Gestaltung des Codes vorgeschrieben, wie zum Beispiel "Camel Case" für die Benennung von Variablen zu nutzen [39][53].

Code Review
Mit dieser Praktik wird der geschriebene Code von einem oder mehrerer anderer Team-Mitglieder auf Mängel oder Fehler hin untersucht [39].

Daily Stand-Ups
Bei dieser Praktik wird täglich ein 15-minütiges Treffen der Entwickler im Stand abgehalten, wobei diese die Arbeit für den aktuellen oder folgenden Tag kurz besprechen und planen [117][39].

Continuous Integration
Mittels dieser Praktik wird der geschriebene Code der Entwickler kontinuierlich in den bestehenden Code integriert [39].

Release Planning
Bei dieser Praktik werden zu entwickelnde Anforderungen und Features bestimmten Releases der Software zugeordnet. Dies wird anhand der Prioritäten und dem Aufwand für die Implementierung der Features entschieden [108].

User Stories
Über User Stories werden die Anforderungen an die Software nicht-technisch festgehalten [39]. Oft wird für diese auch der Aufwand geschätzt und somit das Software-Projekt geplant [39].

Burn Down Charts
Mit dieser Praktik wird die noch verbleibende Arbeit im Verhältnis zur verbleibenden Zeit auf einem Diagramm grafisch dargestellt und analysiert [39].

Backlog Management
Diese Praktik richtet sich auf das Ordnen und Verwalten der Backlog Items im Backlog. Außerdem ist die Verständlichkeit dieser Backlog Items ein wichtiger Aspekt des Backlog Managements [117].

Iteration Planning
Bei dieser Praktik werden nach bestimmten Kriterien die Backlog Items ausgewählt, die in der nächsten Iteration bearbeitet werden [6].

Automated Unit Testing
Die Praktik beinhaltet das Erstellen und Durchführen von automatischen Tests für die kleinsten Einheiten der Software [136][94].

2.4 Hybrider Entwicklungsprozess

Ein Großteil der Projekt-Teams und Unternehmen in der Software-Entwicklung nutzen kombinierte Entwicklungsmethoden, welche dann als hybride Methoden bzw. Entwicklungsprozesse bezeichnet werden [65][134]. Solche dienen dazu, den Nutzen von einer bestimmten Methode explizit da anzuwenden, wo die anderen Methoden diesen nicht bieten können, somit sollten also hybride Methoden eigentlich alle Projekt-Disziplinen im Software-Projekt sinnvoll abdecken [65][134].

2.5 Bedeutung: Abdeckung einer Phase im SW-Lebenszyklus

Kann in einer Phase oder Projekt-Disziplin des Software-Lebenszyklus eine Methode oder Praktik besonders genutzt werden oder bringt diese allein durch ihre Anwendung in dieser Phase einen besonderen Nutzen hervor, dann deckt diese Methode oder Praktik diese Phase ab [65][39]. Allein die Anwendung einer Methode oder Praktik in bzw. für eine Phase ohne weiteren Nutzen oder ohne weitere Dienlichkeit reicht dagegen nicht aus, um diese Phase als abgedeckt zu bezeichnen. Je höher der Nutzen der Anwendung der Methode oder Praktik in der Phase ist, desto höher ist auch die Abdeckung. Dies spiegelt sich in der Zahl an Nutzungen einer Methode oder Praktik für die Phase wieder [65][39].

Kapitel 3

Anforderungsanalyse

In diesem Kapitel werden alle erfassten Stakeholder und Anforderungen für die zu entwickelnde Anwendung aufgelistet und jeweils beschrieben. Dabei wird bei den Anforderungen zwischen den funktionalen und den nicht-funktionalen Anforderungen unterschieden. Die funktionalen Anforderungen sind mit einem „R“ und die nicht-funktionalen Anforderungen mit einem „NR“ gekennzeichnet.

3.1 Stakeholder

Die Stakeholder für die Anwendung sind einerseits das FG SE und andererseits die möglichen späteren Nutzer der Anwendung, welche sich zum Beispiel für einen bestimmten Entwicklungsprozess bzw. für bestimmte Methoden für ihr Software-Projekt entscheiden müssen und sich dabei an der Abdeckung des gesamten Software-Lebenszyklus orientieren. Dies können zum Beispiel IT-Projektleiter sein.

3.2 Funktionale Anforderungen

R01: *Die Anwendung soll dem Nutzer die Funktion bieten, eine oder mehrere Methoden auswählen zu können*

Die Hauptfunktion der Anwendung liegt in der Ermittlung und Ausgabe der Abdeckung der Phasen bzw. der Projekt-Disziplinen durch die Nutzung bestimmter Methoden und Praktiken als hybrider Ansatz. Diese müssen natürlich im Vorhinein ausgewählt werden. Um eine gewisse Struktur zu erhalten, werden die Methoden bei der Auswahl von den Praktiken getrennt.

R02: *Die Anwendung soll dem Nutzer die Funktion bieten, eine oder*

mehrere Praktiken auswählen zu können

Diese Anforderung spiegelt die Anforderung oben wider, nur mit Praktiken statt Methoden.

R03: *Die Anwendung soll dem Nutzer die Funktion bieten, eine ausgewählte Methode wieder von der Auswahl zu entfernen*

Da die Auswahl der Methoden für den Nutzer dynamisch möglich sein soll, da dieser eventuell mehrere mögliche Kombinationen von Methoden zum Beispiel direkt nacheinander untersuchen möchte, muss auch das Entfernen bzw. Deselektieren von Methoden möglich sein, ohne die Anwendung erneut starten zu müssen.

R04: *Die Anwendung soll dem Nutzer die Funktion bieten, eine ausgewählte Praktik wieder von der Auswahl zu entfernen bzw. zu deselektieren*

Diese Anforderung spiegelt die Anforderung oben wider, nur mit Praktiken statt Methoden.

R05: *Die Anwendung soll die Abdeckung aller Phasen des Software-Lebenszyklus anhand der ausgewählten Methoden und Praktiken ermitteln und ausgeben können*

Nach dem Selektieren bestimmter Methoden und Praktiken soll mittels der Zuordnungen der Abdeckungen zu den Phasen für jeweils alle Methoden und Praktiken, für die ausgewählten Methoden und Praktiken, die gesamte Abdeckung ermittelt und anschließend auf der Benutzeroberfläche der Anwendung ausgegeben werden.

R06: *Die Anwendung soll den Software-Lebenszyklus auf der Benutzeroberfläche in Form eines Diagramms darstellen, wobei die Phasen jeweils als Rechtecke dargestellt werden*

So kann die Abdeckung besser überblickt werden, als wenn dies z.B. mittels textueller Darstellung geschieht.

R07: *Die Anwendung soll die Abdeckung der einzelnen Phasen im Diagramm auf der Benutzeroberfläche jeweils mit der Füllung einer der Farben Hellblau, Blau oder Dunkelblau darstellen*

Die Abdeckung einer Phase durch eine oder mehrere Methoden oder Praktiken kann mit den gegebenen Werten nicht besonders genau, also z.B. prozentual, ermittelt werden. Die Abdeckung kann jedoch in z.B.

drei Stufen eingeteilt werden. Also von leicht/wenig über mittel/mäßig nach stark/intensiv. Diese Stufen sollen dann als Farben gekennzeichnet werden. Damit auch Nutzer mit einer Rot-Grün-Schwäche die Farben auseinanderhalten können, wurde hier die blaue Farbe gewählt.

R08: *Die Anwendung soll dem Nutzer die Funktion bieten, nach Auswahl und Ausgabe der Abdeckung der Phasen im Software-Lebenszyklus, zusätzlich eine oder mehrere weitere Methoden oder Praktiken auswählen zu können*

Auch nach der Ausgabe der Abdeckung soll der Nutzer die Möglichkeiten haben, weitere Methoden oder Praktiken auswählen und die Abdeckung erneut ausgeben lassen zu können.

R09: *Die Anwendung soll dem Nutzer die Funktion bieten, nach Auswahl der Methoden und Praktiken und anschließender Ausgabe der Abdeckung, die gesamte Auswahl und Abdeckung zurückzusetzen und damit zum Anfangszustand zurückzukehren*

Bei den Methoden hält sich die Anzahl begrenzt, bei den Praktiken kann die Anzahl jedoch um ein Vielfaches höher sein. Deshalb soll dem Nutzer die Möglichkeit geboten werden, die gesamten ausgewählten Methoden und Praktiken und die Abdeckung direkt zurückzusetzen.

R10: *Die Anwendung soll nach der Ausgabe der Abdeckung für eine ausgewählte Kombination von Methoden und Praktiken, die Methoden und Praktiken ausgeben (falls vorhanden), welche zu der bisherigen Kombination von den Methoden und Praktiken hinzugefügt werden können und eine bei allen Phasen mindestens mäßige Abdeckung gewährleisten*

Deckt eine Kombination von Methoden und Praktiken den Software-Lebenszyklus nicht vollständig oder unzureichend ab, dann sollte diese mit anderen Methoden oder Praktiken kombiniert werden können, sodass eine entsprechende Abdeckung erreicht werden kann. Dazu soll die Anwendung genau solche Methoden sowie Praktiken ermitteln und ausgeben, nachdem die Abdeckung für eine zuvor ausgewählte bestimmte Kombination von Methoden und Praktiken ausgegeben und berechnet wurde.

3.3 Nicht-funktionale Anforderungen

NR01: *Das Diagramm des Software-Lebenszyklus auf der Benutzeroberfläche sollte möglichst übersichtlich und groß dargestellt werden*

Die wichtigste Funktion der Anwendung besteht in der Ausgabe der Abdeckung der Phasen des SW-Lebenszyklus, aus diesem Grund müssen diese Phasen auch entsprechend übersichtlich dargestellt werden.

NR02: *Die Farben für die Stufen der Abdeckung der Phasen sollen eindeutig unterscheidbar sein, sodass eine Verwechslung für den Nutzer der Anwendung nicht möglich ist*

Es ist wichtig, dass der Nutzer die angezeigten Abdeckungen der Phasen eindeutig und leicht erkennen kann. Deshalb ist von großer Bedeutung, dass der Nutzer die Farben direkt einer Stufe der Abdeckung zuordnen kann, also z.B. die dunkle blaue Farbe direkt der stärksten Stufe der Abdeckung zuordnen kann.

NR03: *Die Anwendung sollte einfach zu installieren und auf jedem durchschnittlichen, gängigen Betriebssystem ausführbar sein*

Die Anwendung könnte von Nutzern angewandt werden, welche nicht nur ein bestimmtes Betriebssystem verwenden (wie z.B. Microsoft Windows), sondern auch andere häufig genutzte Betriebssysteme. Beispielsweise werden Unix-/Linux-Systeme von einem großen Teil der IT-affinen Personen genutzt. Dementsprechend muss auch die Wahl der Programmiersprache getroffen werden.

NR04: *Die Anwendung sollte eine Benutzeroberfläche mit einem schlichten Design und einer schlichten Gestaltung besitzen, sodass alle Buttons und Schaltflächen klar erkennbar sind*

Die Anwendung sollte nur wenige Buttons und Schaltflächen beinhalten bzw. nicht zu viele unnötige davon, sodass jedoch aber alle Funktionen abdeckt werden. Dies steigert die Übersichtlichkeit.

NR05: *Die Anwendung sollte, bezogen auf den Programmcode und die Architektur, leicht erweiterbar und wartbar sein*

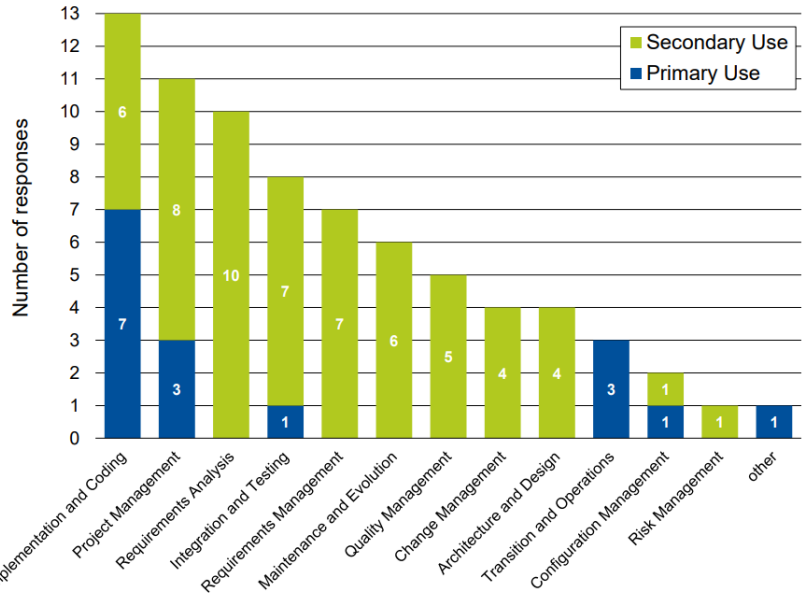
Man könnte die Anwendung noch in unzählige Richtungen erweitern, präzisieren oder verbessern. Aus genau diesem Grund sollte die Architektur der Anwendung so gewählt und konstruiert werden, dass spätere Änderungen leicht implementierbar sowie integrierbar sind.

Kapitel 4

Konzept und Umsetzung

4.1 Genutzte Werte und Statistiken

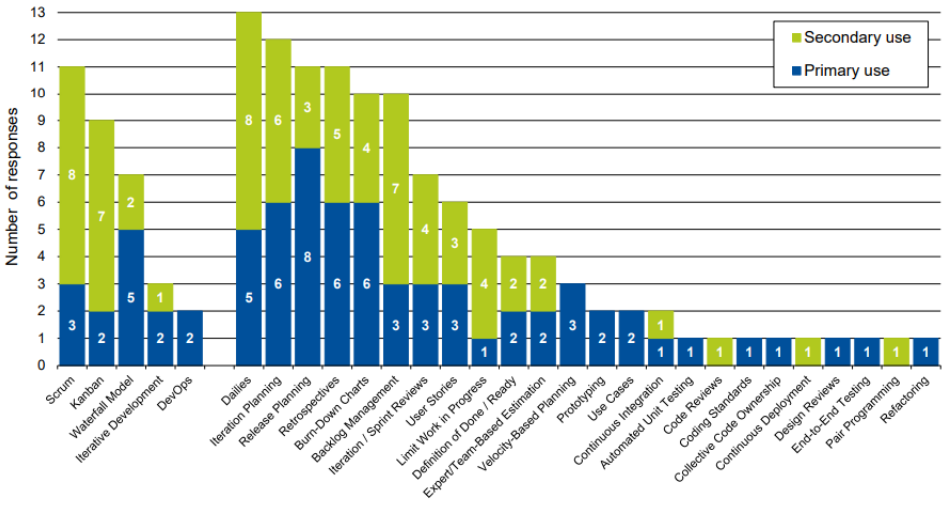
Maßgeblich für den Entwurf des Tools zur Analyse der Abdeckung des Software-Lebenszyklus durch unterschiedliche Methoden und Praktiken sind die Werte aus dem Paper "Towards Shaping the Software Lifecycle with Methods and Practices" und der Bachelorarbeit "Umfragestudie zur Nutzung von Methoden und Praktiken in unterschiedlichen Softwareprojektdisziplinen", denn diese Nutzen die Ergebnisse einer bestimmten Studie/Umfrage zu dem Nutzen von bestimmten Methoden und Praktiken in den jeweiligen Phasen des Software-Lebenszyklus [65][39]. Diese Werte werden dann für die Untersuchungen in Abschnitt 4.2 hauptsächlich genutzt. Den Werten der Diagramme des erstgenannten Papers kommen dabei zunächst eine besondere Bedeutung zu. Zusätzlich werden noch andere Paper und Studien zu den jeweiligen Methoden und Praktiken miteinbezogen, um die Werte der genannten Umfrage zu diskutieren und gegebenenfalls die Sinnhaftigkeit dieser abzuwägen. Es könnte nämlich vorkommen, dass bei einer Angabe zur Nutzung einer Methode oder Praktik für eine Phase der Teilnehmer die Phase falsch gedeutet hat. Um solche Missverständnisse im weiteren Verlauf vermeiden zu können werden also zusätzliche Quellen und Studien verwendet, um die Abdeckung einer Phase durch eine Methode oder Praktik zu untersuchen. Die Abbildungen 4.1 und 4.2 zeigen Ausschnitte aus den Diagrammen aus dem erstgenannten Paper oben und sind im Anhang unter A.2 und A.3 vollständig enthalten. Diese stellen die oben beschriebenen Werte der Umfrage dar. In dem Diagramm der Abbildung 4.1 werden die Nutzungen der drei am häufigsten verwendeten Methoden und Praktiken für jeweils alle Phasen gezeigt. In dem Diagramm in Abbildung 4.2 werden für die vier Phasen *Implementation and Coding*, *Project Management*, *Integration and Testing* und *Quality Management* die Anteile der Teilnehmer, welche in den Phasen eine Methode oder Praktik eingesetzt haben, gezeigt.



(a) Use of Scrum per project discipline (n=16)

Abbildung 4.1: Ausschnitt aus dem Diagramm für die Nutzung der drei häufigsten Methoden und Praktiken für alle Phasen des SW-Lebenszyklus [65]

Wurde bei den beiden genannten Diagrammen eine Methode oder Praktik für eine Phase eingesetzt, dann wurde dabei zwischen “Primary Use” und “Secondary Use” unterschieden. Unter “Primary Use” fällt die Verwendung der Methode oder Praktik, welche hauptsächlich und vorrangig gegenüber allen anderen Methoden oder Praktiken für die Phase erfolgte. “Secondary Use” bezieht sich dagegen auf die gegenüber den anderen Methoden oder Praktiken nachrangige Nutzung der Methode oder Praktik für die Phase. Die Werte des Diagramms aus Abbildung 4.2 beziehen sich nur auf die vier am häufigsten erwähnten und hervortretendsten Phasen des SW-Lebenszyklus. Um dennoch die Abdeckung der restlichen Phasen des SW-Lebenszyklus durch die gewählten Methoden und Praktiken zu analysieren, wird das im gleichen Paper enthaltene Diagramm in Abbildung 4.1 verwendet. Bei diesem wird die Nutzung der drei am häufigsten verwendeten Methoden und Praktiken für alle Phasen dargestellt. Um jedoch auch für alle Praktiken die Nutzung in allen Phasen des SW-Lebenszyklus zu untersuchen, wird auf das Diagramm in Abbildung 4.3 des zweitgenannten Papers zurückgegriffen. Dieses kennt kein Primary-/Secondary-Use, sondern ordnet der Nutzung für eine Phase nur die Ränge von 1 bis max. 12 zu, da es bei der Untersuchung ja auch nur 12 Phasen gab. Je kleiner der Rang, desto größer die Nutzung.



(b) Methods and practices used for Project Management

Abbildung 4.2: Ausschnitt aus dem Diagramm für die Nutzung der Methoden/Praktiken für die vier wichtigsten Phasen des SW-Lebenszyklus [65]

	Methode/ Praktik	n	#andere/ keine Angabe	Project Management	Quality Management	Risk Management	Configuration Management	Change Management	Requirements Analysis	Requirements Management	Architecture and Design	Implementation/Coding	Integration and Testing	Transition and Operation	Maintenance and Evolution
Methoden	DevOps	12	2	4.	6.		5.		7.		5.	2.	**1.**	3.	5.
	Iterative Development	11	1	7.	8.	11.	10.	6.	4.	5.	3.	**1.**	2.	12.	9.
	Kanban	13	2	**1.**	6.	11.	12.	10.	5.	3.	8.	2.	4.	7.	9.
	Scrum	17	1	2.	6.	12.	11.	9.	3.	5.	8.	**1.**	4.	10.	7.
	Wasserfallmodell	12	1	**1.**	5.	10.	8.	6	3.	9.	4.	2.	7.	12.	11.
Praktiken	Automated Unit Testing	16	1	4.	3.			5.			6.	**1.**	2.		
	Backlog Management	15	1	**1.**	6.	5.	10.	3.	4.	2.	8.	3.	6.		9.
	Burndown Charts	15	2	**1.**	5.	3.			7.	8.		2.	4.	6.	
	Code Reviews	18	1	10	2.	5.	7.	5.	6.	5.	4.	**1.**	3.	8.	4.
	Coding Standards	17	1	5.	2.		8.	5.	7.	6.	4.	**1.**	3.		5.
	Continuous Integration	16	1	5.	3.	4.	7.			6.	6.	**1.**	2.	5.	6.
	Daily Standup	19	2	**1.**	8.	10.	8.	9.	4.	6.	5.	2.	6.	7.	3.
	Definition of Done/Ready	14	1	4.	3.	9.	11.	6.	7.	5.	8.	**1.**	2.	10.	
	End-to-End Testing	11	1	4.	2.	5.		5.				3.	**1.**	4.	
	Team/expert based estimation	11	1	3.	6.	5.	8.	9.	3.	2.	7.	**1.**	4.	11.	10.
	Iteration Planning	16	1	**1.**	4.	5.	9.	3.	10.	8.	7.	2.	6.	11.	
	Iteration/Sprint Reviews	13	1	2.	3.	6.		10.	9.	7.	4.	**1.**	5.	8.	
	Limit Work-in-Progress	10	1	2.		7.	8.	9.	10.	5.	6.	**1.**	4.		3.
	Pair Programming	10	1	4.	2.			4.	3.		3.	**1.**	2.		
	Prototyping	12	1	2.	4.	4.			2.	3.	2.	**1.**	5.		
	Refactoring	13	1	6.	3.		7.				2.	**1.**	5.		4.
	Release Planning	16	1	**1.**	2.	9.	10.	8.	11.	7.	12.	4.	5.	3.	6.
	Retrospectives	15	2	**1.**	3.	7.	10.	6.	9.	11.	4.	2.	5.	8.	9.
	Use Case Modeling	10	2	3.	7.			6.	**1.**	2.	4.	5.			
	User Stories	16	2	3.	4.		10.	7.	3.	**1.**	6.	2.	5.	8.	9.

Abbildung 4.3: Nutzung aller Methoden/Praktiken für alle Phasen des SW-Lebenszyklus [39]

4.2 Nutzung/Interpretation der gegebenen Werte und Diagramme

In dem zu entwickelnden Tool soll die Abdeckung der Phasen des Software-Lebenszyklus durch farbige Phasen dargestellt werden.
Dabei stellt die Hellblaue-Phase eine sehr geringe bis geringe Abdeckung, die Blaue-Phase eine mäßige und die Dunkelblaue-Phase eine intensive Abdeckung einer Phase des Software-Lebenszyklus dar. Diese Farben repräsentieren dann also, der Helligkeit nach in Stufen geordnet, die Abdeckung für die Phasen des SW-Lebenszyklus.
Dazu muss also für jede Methode und Praktik jeweils für jede Phase des SW-Lebenszyklus eine entsprechende Stufe und Farbe (Hellblau, Blau oder Dunkelblau) zugeordnet werden. Diese Zuordnung erfolgt dann mittels einer individuellen Analyse der gegebenen Werte der entsprechenden Diagramme (in Abschnitt 4.1) unter Einbeziehung anderer Studien und Literatur.
Die Bewertung der Abdeckung der Phasen erfolgt deshalb in einem Drei-Stufen-Modell und nicht einfach in einem Zwei-Stufen-Modell, weil die Abdeckung mittels der gegebenen Werte der Diagramme (siehe Abschnitt 4.1) und unter Einbeziehung der anderen Quellen nicht immer als eindeutig abgedeckt oder nicht abgedeckt bewertet werden kann und deshalb eine Zwischenstufe nötig ist, um die Abdeckung zuverlässiger und genauer zu bewerten.
Bei den nachfolgenden Analysen der Abdeckungen für jede der Phasen für die Methoden und Praktiken wird dabei meistens zuerst auf die Werte der vollständigen Diagramme der Ausschnitte 4.1 und 4.2 eingegangen und dann auf die Erkenntnisse anderer Arbeiten oder Studien.
Wurde eine Methode oder Praktik also von z.B. 30 % der Teilnehmer ausschließlich als Primary Use bewertet und ist nach den weiteren Quellen eine teilweise Abdeckung dieser Phase durch diese Methode oder Praktik festzustellen, dann wird die Phase von dieser Methode oder Praktik z.B. als mäßig abgedeckt bewertet.

Folgende Bedeutungen kommen den Farben nun in Bezug auf die Abdeckung der Phasen des SW-Lebenszyklus zu:

Abdeckung/Farbe	Abdeckung der Phase durch die Methode/Praktik
Dunkelblau	sehr stark oder vollständig
Blau	teilweise oder zweitrangig
Hellblau	gar nicht oder sehr gering

Tabelle 4.1: Farben und die Bedeutung der Abdeckung

4.2.1 Abdeckung der Phasen: Scrum

Starke Abdeckung

In der Phase *Implementation and Coding* wurde Scrum mit ca. 81 % der gesamten Scrum-Nutzer am häufigsten eingesetzt, von diesen ca. 54 % als Primary Use [65]. Scrum hat während der Entwicklung den Vorteil, dass die Motivation innerhalb des Teams gesteigert wird [29]. Abgesehen davon wird auch die Teamfähigkeit bzw. Zusammenarbeit gestärkt. [82] Zusätzlich unterstützen die Daily-Standup Meetings von Scrum den schnellen und flexiblen Entwicklungs- bzw. Arbeitsfluss [107, p. 24]. Durch Scrum werden durch die erhöhte geforderte Zusammenarbeit zwischen den Team-Mitgliedern sowie den Kunden die persönlichen Beziehungen und somit auch das Vertrauen in den Gegenüber gestärkt [107, p. 6]. Die Phase *Implementation and Coding* wird von Scrum also stark abgedeckt.
Auch die Phase *Project Management* wird von Scrum stark abgedeckt, denn 68,75 % der Scrum-Nutzer verwendeten dieses in dieser Phase und davon ca. 27,27 % als Primary Use [65]. Dies lässt sich damit erklären, dass Scrum selbst auch ein Vorgehensmodell für das (agile) Projektmanagement ist [47, p. 61][24]. Außerdem wurde in einer Umfrage von dem Report "The State of Scrum: Benchmarks and Guidelines" festgestellt, dass 39 % der Teilnehmer Scrum für das Projektmanagement bzw. als Praktik für das Projektmanagement einsetzten [7]. Zusätzlich wird in "Der Ultimative Scrum Guide 2.0" darauf eingegangen, dass Scrum trotz des agilen Ansatzes dennoch über eine geplante Entwicklung verfügt, da ein Release-Plan in Form des Product Backlog besteht und zur langfristigen Orientierung bei Scrum die Produktvision hilft. Diese unterstützen entsprechend das Projektmanagement in Scrum [51].

Mäßige Abdeckung

In der Phase *Integration and Testing* wurde Scrum zwar von 50 % der Scrum-Nutzer eingesetzt, jedoch davon nur ca. 13 % als Primary Use [65]. In Scrum gibt es keinen definierten Test-Manager [80], auch sagt der Scrum-Guide bzw. die Scrum-Definition von Schwaber und Sutherland nichts über das Testen der Software [135]. Dennoch kann mit Scrum getestet werden, dazu sollte das Testen innerhalb eines Sprints stattfinden [80]. Jedoch hat Scrum beim Testen auch seine Nachteile [126][92], denn wenn die Tests mit der Implementierung innerhalb eines Sprints stattfinden, muss dafür auch eine entsprechende zusätzliche Zeit eingeplant werden [92]. Dabei müssen die Entwickler dann also auch ein entsprechendes Zeit-Management verantworten, also zwischen der Qualität des aktuellen Inkrements und dem Aufwand für das Testen von diesem abwägen [92]. Dabei kann dann also das Zeitlimit für einen Sprint zum Problem werden [92], denn ein Sprint ist nur einige Wochen lang und somit kann es bei komplexeren Anforderungen und

Features knapp werden, noch entsprechende Tests für diese innerhalb dieses Sprints zu implementieren [92].
Ein anderes Problem kann durch das nachträgliche Ändern der Anforderungen (nach einem Sprint) entstehen, da auch das Testen für diese geplant werden muss und somit komplexe Situationen entstehen können [92], wenn also auch die Tests angepasst oder geändert werden müssen.
Ein weiteres Problem ist, dass das Schätzen des Aufwands (bzw. der Zeit) für die Entwicklung und das Testen in Scrum nicht immer zuverlässig möglich ist bzw. sich schwierig gestaltet [92].
Abgesehen von diesen (möglichen) Problemen in Scrum, gibt es einige Ansätze, das Testen in Scrum zu ergänzen bzw. mit anderen Ansätzen zu kombinieren, welche das Testen besser ermöglichen (z.B. Ansätze/Prinzipien der schlanken Software-Entwicklung) [92].
In der Phase *Change Management* wurde Scrum von 25 % der Scrum-Nutzer eingesetzt und dies nur als Secondary Use [65]. *Change Management* wird durch Scrum zugelassen bzw. kann durch Scrum unter Berücksichtigung bestimmter Bedingungen sogar besser gefördert werden [96]. Abgesehen davon wird Scrum selbst auch als *Change Managemenent* Ansatz aufgeführt, um die bestmögliche Software zu entwickeln, unter Einbeziehung und Berücksichtigung der gegebenen Zeit, dem Budget, der Funktionalität und der Qualität [106][116]. Das *Change Management* wird von Scrum also mäßig abgedeckt.
In der Phase *Architecture and Design* wurde Scrum von 25 % der Scrum-Nutzer verwendet und dies nur als Secondary Use [65]. In dem Scrum-Guide wird über den Software-Entwurf nichts gesagt [117], dies könnte auch der Grund sein, warum Scrum in dieser Phase nicht ein einziges mal als Primary Use eingesetzt wurde. Eine Umfrage mit einem achtköpfigen Scrum-Team, bestehend aus dem Scrum-Master und den Entwicklern, hat jedoch ergeben, dass sich alle acht Mitglieder des Scrum-Teams für die Anwendung von Scrum, statt dem traditionellen Ansatz, bei dem Design Prozess bzw. für die Design Phase ausgesprochen haben und Scrum also gegenüber dem traditionellen Ansatz für diese Phase bevorzugen würden [129]. Scrum deckt die Phase *Architecture and Design* also mäßig ab.
In der Phase *Transition and Operations* wird Scrum von ca. 19 % der Scrum-Nutzer verwendet und dies ausschließlich als Primary Use [65]. In Scrum spielt das Release-Management nämlich eine wichtige Rolle [102]. Dabei gibt es in Scrum jedoch keine Position oder Person, welche für das Release-Management verantwortlich ist, also keinen Release-Manager [102]. Damit das Release-Management am effektivsten mit Scrum realisiert werden kann, wird am Ende zusätzlich ein Sprint durchgeführt, für die nötigen Release-Aktivitäten [102]. Diese werden dann nicht nur von einem Operations-Team, sondern von dem Scrum-Team und dann wiederum Einzelpersonen aus dem Operations-Team zusammen durchgeführt [102]. So kann diese Phase mit Scrum realisiert werden. Scrum deckt die Phase *Transition and Operations*

also mäßig ab.

Keine Abdeckung

In der Phase *Maintenance and Evolution* wurde Scrum von ca. 38 % der Scrum-Nutzer eingesetzt und dies ausschließlich als Secondary-Use [65]. Scrum eignet sich nicht für diese Phase, da sich die Wartung oder der Wartungsbetrieb nicht in Sprints teilen lässt, in welchen ja ein bestimmtes Ziel wie in der Entwicklung verfolgt wird [17]. In einer Befragung mit Software-Entwicklern aus der Industrie, haben nahezu 80 % der Teilnehmer zugestimmt, dass Scrum sich nicht effektiv für die Software-Wartung und -Support einsetzen lässt [91]. Dabei gibt es jedoch Möglichkeiten, Scrum als Modell so zu erweitern, dass es dennoch effizient für diese Phase verwendet werden kann [91]. Diese Phase wird von Scrum also dennoch nicht abgedeckt.
Die Phase *Risk Management* wird von Scrum nicht wirklich abgedeckt, dies kann vor allem daran geschlussfolgert werden, dass Scrum keine Mittel und Prozesse anbietet, welche für das Risk Management gedacht sind [132]. Trotzdem wird in Scrum in bestimmten Bereichen das *Risk Management* unterstützt bzw. Risiken direkt oder auch indirekt vermieden, zum Beispiel durch die Priorisierung der Product Backlog-Einträge, wobei Einträge mit einem höheren Risiko geringer priorisiert werden und Einträge, welche das Risiko hingegen vermindern, weiter oben priorisiert werden [51, p. 47]. Außerdem werden bei dem iterativen Ansatz von Scrum, implementierungsspezifische Risiken in dem Sprint Review und prozess-spezifische Risiken in der Sprint Retrospektive früh erkannt und ggf. entsprechende Maßnahmen eingeleitet um diese zu beheben [51]. Zusätzlich können in den Daily Scrum Meetings ggf. auch noch Risiken erkannt, diskutiert und besprochen werden [51]. Auch durch den Scrum Master sollten und können Risiken vermieden oder zumindest erkannt werden, da eine wesentliche Aufgabe des Scrum-Masters ja die Beseitigung von Hindernissen ist [51]. Jedoch nutzten nur 6,25 % der Scrum-Nutzer dieses für das *Risk Management* und dies auch nur als Secondary Use [65]. Unter Berücksichtigung von diesem sehr geringen Anteil und der Tatsache, zu Beginn oben aufgeführt, dass Scrum selbst direkt keine Mittel für diese Phase besitzt, wird die Abdeckung der Phase *Risk Management* durch Scrum also als nicht vorhanden festgelegt.
Die Phase *Requirements Engineering* bzw. die beiden Phasen *Requirements Analysis* und *Requirements Management* werden von Scrum nicht direkt abgedeckt, da diese im Scrum-Modell bzw. -Zyklus nicht wirklich enthalten sind [145]. Aus diesem Grund wird bezüglich der Phase *Requirements Analysis* vor dem eigentlichen Scrum-Prozess (z.B.) eine Vorbereitungsphase vorgeschaltet, in welcher dann die Anforderungen für den Product Backlog erst einmal ermittelt und erstellt werden [145], da der Scrum-Prozess davon ausgeht, dass bereits am Anfang ein vollständig gefüllter Product Backlog zur Verfügung steht [145]. Das *Requirements Management* wird

hingegen von Scrum mehr angegangen und deutlicher abgedeckt, da die Anforderungen nicht nur einfach im Product Backlog gesammelt werden, sondern schließlich auch nach bestimmten Kriterien (z.B. Wert für den Kunden, Risiko oder Abhängigkeit) priorisiert und der Aufwand für diese geschätzt wird [99, p. 123-135][47, p. 62]. Der Aufwand für diese kann mit unterschiedlichen Praktiken geschätzt werden. Ein Beispiel ist die Praktik Planning Poker [99, p. 123-135]. Dennoch verfügt zum Beispiel die Methode Kanban über ein deutlich besseres *Requirements Management*, da dort zum einen die Anforderungen über den gesamten Verlauf der Entwicklung visualisiert und einsehbar sind und zum anderen, die Anforderungen für die jeweiligen Phasen direkt limitiert sind (Work-In-Progress Limit bzw. "Limitierte Mengen" Element) und nicht nur indirekt für jeden Sprint wie bei Scrum [54][12] (siehe Abschnitt 2.2.3).
In der Phase *Quality Management* wurde Scrum von ca. 31 % der Scrum-Nutzer eingesetzt und dies ausschließlich als Secondary Use [65]. Dass Scrum in dieser Phase kein einziges Mal als Primary Use angewandt wurde, ist nicht besonders unerwartet, da das *Quality Management* generell eine große Schwäche von Scrum ist [28], denn dieses lässt bei dem *Quality Management* zu viele Aspekte und Standpunkte offen, vor allem was das Testen und Verifizieren betrifft [28][62]. Zudem kann Scrum auch Zeitdruck auf die Entwickler ausüben und diesen somit erschweren, bestimmte Qualitätssicherungs-Techniken wie Refactoring auszuführen [62]. Jedoch gibt es auch einige wenige Ansätze und Möglichkeiten, Scrum mit dem Qualitätsmanagement zu kombinieren oder dieses sinnvoll einzuführen [62]. Dennoch ist die Qualitätssicherung und das Qualitätsmanagement mit Scrum nicht wirklich oft untersucht worden und es gibt nur wenige Studien zu diesem Thema [62].
Die Phase *Quality Management* wird von Scrum also nicht abgedeckt.
In der Phase *Configuration Management* wurde Scrum von ca. 13 % der Scrum-Nutzer verwendet, davon 50 % als Primary Use [65]. Scrum selbst per Definiton im Scrum-Guide sagt nichts zum *Configuration Management* [117], jedoch wurde auch festgestellt, dass Scrum den Pratiken des *Configuration Management* nicht widerspricht oder diese behindert [24].

Abdeckung durch Scrum	Phase/Projekt-Disziplin
	Integration and Testing
	Implementation and Coding
	Requirements Management
	Configuration Management
	Risk Management
	Project Management
	Requirements Analysis
	Quality Management
	Architecture and Design
	Maintenance and Evolution
	Change Management
	Transition and Operations

Tabelle 4.2: Abdeckung der Phasen durch Scrum

4.2.2 Abdeckung der Phasen: DevOps

Starke Abdeckung

In der Phase *Implementation and Coding* wurde DevOps von ca. 33 % der DevOps-Nutzer eingesetzt und davon 50 % als Primary Use [65]. Es ist somit die zweithäufigste Phase, in welcher DevOps eingesetzt wurde. DevOps bringt in dieser Phase den Vorteil mit sich, dass auch die Entwickler vor dem Festlegen der Anforderungen und User Stories, mit dem Kontext des zu entwickelnden Produktes oder Features vertraut gemacht werden [52, p. 125-127].
Abgesehen davon wird DevOps insgesamt am häufigsten bzw. von mindestens einem Drittel der DevOps-Nutzer, für die Phasen *Integration and Testing*, *Implementation and Coding* und *Transition and Operations* eingesetzt [65]. Diese Werte für diese drei Phasen sind für DevOps auch nicht ungewöhnlich, sondern eher zu erwarten. Eines der Hauptziele von DevOps ist nämlich die Entwicklung (Development) und den Betrieb (Operations) zusammenzubringen und zu vereinen (siehe Abschnitt 2.2.2).
Vor allem die Phase *Integration and Testing* wird durch die von DevOps eingeführten automatisierten Tests neben der Entwicklung weitreichend abgedeckt [63, p. 117-121][93]. Die beiden Praktiken Continous Integration und Continous Deployment sind eng mit DevOps verbunden und für dieses auch charakterisierend [69]. Dabei wurden diese beiden Praktiken in der Studie, nach der Praktik Automated Unit Testing, am häufigsten für die Phase *Integration and Testing* verwendet und dies auch als Primary Use (siehe vollständiges Diagramm der Abbildung 4.2). Somit ist zu erwarten, dass damit für die Phase *Integration and Testing* auch eine gewisse Abdeckung mit dem Nutzen von DevOps einhergeht. Die Phase *Integration and Testing* wird von DevOps also stark abgedeckt.
Auch die starke Abdeckung der Phase *Transition and Operations* ist zu erwarten, da mit DevOps auch explizit die Nutzung von Tools für das *Deployment Management* und *Release Management* einhergeht [19]. Abgesehen davon hat DevOps sowieso einen besonderen Fokus auf das Deployment, da der DevOps-Zyklus die eigenständige Phase "Deploy" enthält [60][93]. Wesentlich für das Deployment in DevOps ist, abgesehen davon, die Praktik Continous Deployment. Mittels dieser werden Änderungen schnell und kontinuierlich von der Entwicklungs- in die Produktiv-Umgebung integriert und überliefert [60][139][93]. Zusätzlich werden durch die Anwendung solcher automatisierter, kontinuierlicher Deploy-Vorgänge mit entsprechenden Tools und Praktiken das Risiko fehlerhafter Deployments reduziert [60][93]. Außerdem führen die in DevOps eingeführten Feedback-Schleifen dazu, dass die Produktivbereitschaft von dem entwickelten Code verbessert wird und somit auch die Produktiv-Deployments sicherer und stabiler werden [63, p. 234]. Die Phase *Transition and Operations* wird von

DevOps also stark abgedeckt.
Bei der Phase *Project Management* hingegen, wird DevOps ausschließlich als Primary Use eingesetzt [65]. Dabei wurde DevOps jedoch nur von einem Sechstel der gesamten DevOps-Nutzer für das *Project Management* eingesetzt [65]. Der Grund, dass DevOps überhaupt für diese Phase verwendet werden kann, liegt daran, dass DevOps den gesamten Software-Lebenszyklus auffassen und unterstützen kann [88], da die Teams eine Ende-zu-Ende-Verantwortung für diesen Zyklus besitzen, also nach den Mottos "you build it, you run it", "you break it, you fix it" und "you destroy it, you rebuild it later" ausgerichtet werden [88]. Außerdem ist DevOps selbst sowieso auch eine Methode für das Projektmanagement [39][56]. Die Abdeckung der Phase *Project Management* durch DevOps ist deshalb stark.

Mäßige Abdeckung

In der Phase *Configuration Management* wurde DevOps jedoch von nur insgesamt 16,67 % der DevOps-Nutzer verwendet, davon je 50 % als Primary Use und je 50 % als Secondary Use [65]. Dabei ist das *Configuration Management* ein zentraler und wichtiger Bestandteil von DevOps, da z.B. die Installation, Konfiguration oder Aktualisierung von Systemen oder Anwendungen mithilfe von bestimmten Tools in DevOps automatisiert durchgeführt wird [98][101][93]. Die Abdeckung dieser Phase durch DevOps ist somit also mäßig.
In der Phase *Maintenance and Evolution* wurde DevOps, wie bei *Configuration Management*, ebenfalls nur von 16,67 % der DevOps-Nutzer verwendet, davon je 50 % als Primary Use und Secondary Use [65]. Dies steht der Definition oder dem Motto von DevOps, in welcher ja die Wartung und die Optimierung von dieser als zentrale Bestandteile aufgegriffen werden [75], entgegen. DevOps ist abgesehen davon auch eine wichtige Praxisumgebung und zukünftiger Betriebstrend für die Software-Entwicklung und Wartung (Maintenance) [21]. Jedoch könnten die geringen Werte für die Abdeckung von *Maintenance and Evolution* durch DevOps (in der Umfrage) auch an der Tatsache liegen, dass ja durch DevOps die Entwicklungsphase bis in die Wartungsphase erweitert oder expandiert wird und somit die Phase *Implementation and Coding* die Phase *Maintainance and Evolution* quasi überlagert [22]. Abgesehen davon gibt es generell den Mythos, dass DevOps die IT-Operations (den Betrieb oder die Wartung) eliminiert [63, p. 17], dies entspricht aber keinesfalls der Wahrheit, denn DevOps verlagert den Betrieb, wie bereits oben erwähnt, nur weiter in die Entwicklung und ist an dieser auch beteiligt [63, p. 17]. Somit könnten es also sein, dass die DevOps-Nutzer in der Umfrage lediglich den Nutzen von DevOps für *Implementation and Coding* angegeben haben, aber dafür nicht für *Maintenance and Evolution*, aber genau diese Phase dennoch gleichzeitig mit eingeschlossen haben, da

sich ja beide Phasen in DevOps (mindestens zu einem Teil) überlagern. Die Abdeckung dieser Phase ist somit mäßig.
Für die Phase *Architecture and Design* wurde DevOps von ca. 17 % der DevOps-Nutzer verwendet, davon 50 % als Primary Use [65]. Die Design-Phase ist in den meisten Unternehmen mit der Entwicklungsphase vermischt [52, p. 109-125], denn diese ist als eigene Phase, ganz im Gegensatz zum Wasserfall-Modell, nicht als solche in dem DevOps-Lebenszyklus aufgeführt [88]. Jedoch sollte diese in DevOps dennoch von der Entwicklungsphase getrennt werden und vorher stattfinden [52, p. 109-125]. Diese sollte also zwischen der Planungs- und Entwicklungsphase des DevOps-Lebenszyklus stattfinden. Wichtig und charakterisierend für DevOps ist bei dieser Phase, dass die Entwickler sich aktiv an dem Entwurf beteiligen und nicht, wie z.B. in Silos, von dem Design-Team getrennt arbeiten [52, p. 109-125]. Folgendes kann und sollte sich aus der Nutzung von DevOps für den Design-Prozess ebenfalls ergeben und ist für die Nutzung von DevOps für diese Phase vorausgesetzt [52, p. 109-125]: Durch die Nutzung von DevOps ist auch eine gewisse Flexibilität und Wartungsfreundlichkeit während der Design-Phase gegeben und vorausgesetzt, es sind also technische Änderungen sowie vor allem Code-Änderungen wesentlich einfacher und schneller durchzuführen [52, p. 109-125]. Durch den DevSecOps-Ansatz innerhalb von DevOps wird die Software nicht mehr, wie ja sonst bisher, erst am Ende des SW-Lebensyzklus auf die Sicherheit hin überprüft, sondern bereits am Anfang ab dem Planungs- und Designprozess [52, p. 109-125][138].
Die Phase *Architecture and Design* wird von DevOps also mäßig abgedeckt.
Für die Phase *Requirements Analysis* wurde DevOps lediglich von 8,33 % der DevOps-Nutzer eingesetzt und auch nur als Secondary Use [65]. Dies obwohl die Produktanforderungen bei DevOps einen großen Stellenwert einnehmen, um alle an einen "Tisch" zu bringen und so eine gemeinsame Vision zu etablieren und entwickeln [52, p. 95-107]. Je mehr Stakeholder nämlich in den Prozessbeginn miteinbezogen werden, desto geringer ist die Wahrscheinlichkeit, dass später Änderungen vorgenommen werden müssen [52, p. 95-107]. Für das Ermitteln der Anforderungen gibt es bei DevOps passende Techniken und Praktiken, wie z.B. User Stories, das Erstellen eines Produktanforderungsdokuments (PRD), das Entwickeln eines Minimum Variable Product (MVP) und das Erstellen von Personas [52, p. 95-107]. Diese sind in Verbindung mit DevOps nützlich für das Herauskitzeln von Anforderungen [52, p. 95-107]. Auch die Priorisierung der zu übernehmenden Funktionen und Anforderungen für das (Software-)Produkt gehört bei DevOps dazu [52, p. 95-107]. Diese Techniken können hinzugefügt werden, sind jedoch kein direkter fester Bestandteil von DevOps [52]. DevOps deckt das *Requirements Analysis* also mäßig ab.

Keine Abdeckung

Für die Phase *Quality Management* wurde DevOps lediglich von ca. 8,33 % der DevOps-Nutzer genutzt und dies ausschließlich als Secondary Use [65]. Dabei ist eine der wichtigsten Benefits und Nützlichkeiten die DevOps besitzt, eine verbesserte Qualität der entwickelten Software [104][121]. Einer der Gründe ist z.B. das automatisierte und vor allem vielfachere Testen von Code-Änderungen, bevor diese weiter übernommen und ausgeliefert werden [104][121]. Jedoch bezieht sich die Phase *Quality Management* nicht nur auf die bloße Qualitätssicherung, sondern auch z.B. auf die Verbesserung der Qualität des Entwicklungsprozesses. Dafür könnte DevOps eventuell nicht oder nur in geringem Umfang genutzt worden sein. Die Abdeckung der Phase *Quality Management* ist also nicht wirklich vorhanden.
In der Phase *Change Management* wurde DevOps nicht ein einziges Mal genutzt [65]. Bei der Phase *Change Management* gibt es dafür unter anderem einen erklärbaren Grund, denn das Change Management ist darauf ausgelegt, Änderungen in längeren Zeitabschnitten durchzuführen, während nach DevOps ja gerade umgekehrt Änderungen besonders schnell und kurz durchgeführt werden sollen [73]. Außerdem dauern Änderungen im Change Management normalerweise sowieso länger als Änderungen bezüglich der Software [73]. Es ist nämlich sowieso der Fall, dass die Change-Management Frameworks im Kern für großangelegte Änderungen ausgelegt sind und DevOps hingegen für kleine und hochfrequente Änderungen [87]. Somit kann Change Management in DevOps die Software-Release-Zyklen verlangsamen und verzögern [25][10]. Somit ist es durchaus eine große Herausforderung DevOps mit dem *Change Management* zu integrieren und kombinieren.
In der Phase *Requirements Management* wurde DevOps ebenfalls nicht ein einziges Mal, also von 0 % der DevOps-Nutzer, eingesetzt [65]. Dieses so schon beinahe eindeutige Ergebnis wird dadurch gestützt, dass in sämtlichen untersuchten Arbeiten mit Bezug zu DevOps, nichts konkretes über das Nutzen von *Requirements Management* genannt wurde. In einer Studie hingegen wurde auch festgestellt, dass bei der Einführung von DevOps, während der intensiven Zusammenarbeit zwischen dem Entwicklungs- und Betriebsteam andere Teile und Reste der Teams wiederum nicht mit dieser neuen "Kultur" (von DevOps) vertraut sind, dies waren unter anderem auch das *Requirements Management*-Team [48]. Außerdem wurde in zwei Büchern über (die Einführung von) DevOps für die Software-Entwicklung, wie bei den Papern oben, nichts konkretes über das *Requirements Management* mit DevOps gesagt [63][52], jedoch einiges über *Requirements Analysis* mit DevOps. Dazu ist es so, dass in einem dieser Bücher das Visualisieren der Aufgaben und Anforderungen innerhalb von DevOps mit dem Kanban-Board durchgeführt werden kann und empfohlen wurde [63, p. 14]. Also einer Praktik, welche quasi symbolisch für die Methode Kanban steht [144][59, p. 33][49, p. 115]. Damit wird also quasi auch indirekt auf das

Nutzen einer Kanban-Praktik für das *Requirements Management* verwiesen. Dies könnte darauf hindeuten, dass DevOps selbst weder eine Praktik noch eine Möglichkeit bietet, das *Requirements Management* zu betreiben. Mit allem insgesamt, deckt DevOps diese Phase also nicht ab.
In der Phase *Risk Management* wurde DevOps kein einziges Mal eingesetzt [65]. Dagegen ist z.B. DevSecOps ein, wie der Namensteil "Sec" bereits impliziert, verbreiteter Ansatz zur zusätzlichen Integration von Sicherheitsmaßnahmen in die Verschmelzung und Vereinigung von der Entwicklung (Development) und dem Betrieb (Operations), welcher mit dem *Risk Management* auch sinnvoll und sehr gut kombiniert werden kann [46]. Zusätzlich können auch "Scanning Tools" zur Verbesserung und Unterstützung des *Risk Management* genutzt werden, um dieses soweit wie möglich zu automatisieren [68]. Dagagen ist die (Software-)Sicherheit auch nur ein Teil von dem gesamten *Risk Management* [127]. Somit wird dieser Phase hier keine Abdeckung durch DevOps zugeordnet, da die Ergebnisse der Studie, welche oben aufgeführt wurden, hier als überzeugend betrachtet werden. Dabei kann übrigens auch sein, dass den Teilnehmern der Studie Ansätze, wie z.B. DevSecOps, nicht bekannt waren oder einfach nicht genutzt wurden.

Abdeckung durch DevOps	Phase/Projekt-Disziplin
	Integration and Testing
	Implementation and Coding
	Requirements Management
	Configuration Management
	Risk Management
	Project Management
	Requirements Analysis
	Quality Management
	Architecture and Design
	Maintenance and Evolution
	Change Management
	Transition and Operations

Tabelle 4.3: Abdeckung der Phasen durch DevOps

4.2.3 Abdeckung der Phasen: Kanban

Starke Abdeckung

Für die Phase *Project Management* wurde Kanban am häufigsten eingesetzt, denn 75 % der Kanban-Nutzer haben Kanban in dieser Phase verwendet, davon ca. 22,22 % als Primary Use [65]. Mit den vier charakterisierenden Elementen von Kanban (siehe Abschnitt 2.2.3) fördert diese Methode diese Phase wie folgt: Durch das Element "Pull" wird jeder Person im Team eine selbstorganisierte und eigenverantwortliche Arbeit ermöglicht [49]. Es wird darauf vertraut, dass jede Person im Team die Entwicklung einer Anforderung selbst auch wirklich übernimmt und dies nach bestem Gewissen tut [49].
Durch das Element "Limitierte Mengen" kann das Projektmanagement den Fluss von Anforderungen auch über alle Phasen hinweg kontinuierlich überprüfen und regulieren [49].
Durch das Element "Transparente Information" werden wichtige Informationen, Kennzahlen und auch Ziele von dem Projekt an alle beteiligten Personen weitergegeben [49]. Dazu gehören neben den Team-Mitgliedern auch z.B. der Auftraggeber [49]. Dadurch wird das Vertrauen und die Zusammenarbeit gefördert und gestärkt, da allen Beteiligten der aktuelle Stand des Projektverlaufes bekannt gemacht wird [49].
Durch das Element "Kontinuierliche Verbesserung" wird eine (vertrauensvolle) Atmosphäre geschaffen, in welcher jede Person des Teams einen Beitrag zur kontinuierlichen Verbesserung leisten kann und dazu auch motiviert wird [49, p. 79-82].
Abgesehen davon ist Kanban selbst auch per Definition eine Methode für das Projektmanagement [39].
Somit wird das *Project Management* durch Kanban stark abgedeckt.
In der Phase *Requirements Management* wurde Kanban von ca. 41,67 % der Kanban-Nutzer verwendet, davon 20 % als Primary Use [65]. Begründen lässt sich dies mit den Benefits und Vorteilen von Kanban in dieser Phase, denn durch drei der vier charakterisierenden Elemente von Kanban (siehe Abschnitt 2.2.3) wird das *Requirements Management* unterstützt. Dabei werden die Anforderungen nach dem "Pull-Prinzip" der Entwicklung bereitgestellt, ganz im Gegensatz zu dem "Push-Prinzip" [49, p. 70-74]. Zudem werden die Anforderungen, welche sich gleichzeitig in der Entwicklung befinden oder zu dieser bereitgestellt werden, durch das Element "Limitierte Mengen" (siehe Abschnitt 2.2.3) begrenzt und somit verhindert, das es zu Engpässen oder Überarbeitungen kommt [49, p. 70-74]. Das Element "Transparente Informationen" bezieht sich auf das Erklären, Deuten und Einbetten der Anforderungen in einen fachlichen Kontext [49, p. 70-74]. Zusätzlich werden die Anforderungen in Kanban individuell priorisiert, z.B. anhand der Dringlichkeit oder dem Risiko der Anforderung [49, p. 70-74].

Diese Prioritäten bestimmen dann die Reihenfolge der Bereitstellung der Anforderungen an die Entwicklung [49, p. 70-74]. Somit kann dann neben der Verwaltung der Anforderungen, auch der Anforderungsfluss gesteuert und optimiert werden. Kanban deckt die Phase *Requirements Management* also stark ab.
Die Phase *Implementation and Coding* kann ebenfalls mit Kanban abgedeckt werden und dies spiegelt sich auch in den Werten der Nutzung von Kanban für diese Phase wieder, denn von ca. 41,67 % der Kanban-Nutzer wurde Kanban für diese Phase verwendet und von 40 % dieser als Primary Use [65]. Es wirken sich die charakterisierenden Elemente von Kanban (siehe Abschnitt 2.2.3) unterstützend auf die Entwicklung aus, dabei führt das Element "Pull" von Kanban dazu, dass selbstorganisiert und eigenverantwortlich mit der Entwicklung von Anforderungen begonnen wird und fertiggestellte Anforderungen der Qualitätssicherung bereitgestellt werden [49, p. 74-76].
Durch das Element "Limitierte Mengen" werden die gleichzeitig zu entwickelnden Anforderungen limitiert und beschränkt, somit werden z.B. Engpässe verhindert [49, p. 74-76].
Mit dem Element "Transparente Information" werden die Probleme oder Fortschritte der zu entwickelnden Anforderungen klar angesprochen und kommuniziert [49, p. 74-76].
Ein Profit von Kanban ist außerdem, dass die Motivation der Entwickler erhöht wird und Probleme (wie Bugs) leichter gelöst werden können [13]. Kanban deckt die Phase *Implementation and Coding* also stark ab.

Mäßige Abdeckung

In der Phase *Maintenance and Evolution* wurde Kanban von nur 25 % der Kanban-Nutzer verwendet und dies ausschließlich als Secondary Use [65]. Dabei gibt es einige Vorteile von Kanban die zur Wartung (der Software) beitragen können: In einer Studie wurden die beiden Methoden Scrum und Kanban bezüglich der Nützlichkeit und Eignung für die Software-Wartung (Maintenance) untersucht [12]. Durch das Kanban-Element "Limitierte Mengen" (siehe Abschnitt 2.2.3) werden während der Wartung nicht zu viele Wartungs-Tickets/-Aufgaben gleichzeitig bearbeitet oder angefangen. Dies erhöht den Durchsatz und Effizienz bei der Bearbeitung dieser Wartungs-Tickets/-Aufgaben [12]. Da Kanban ja eine evolutionäre Methode ist, unterstützt diese gerade Situationen wie die Wartung, in welchen sich die Aufgaben und Ziele ständig ändern (siehe drei Change-Management-Prinzipien von Kanban [105][37]) [12]. Kanban sorgt mit dem Element "Transparente Information" (siehe Abschnitt 2.2.3) für eine Visualisierung der Wartungs-Tickets/-Aufgaben (z.B. mit dem Kanban-Board), damit geht eine bessere Übersichtlichkeit des gesamten Wartungsprozesses einher [12]. Kanban deckt das *Maintenance and Evolution* also mäßig ab.

Für das *Change Management* wurde Kanban von lediglich 17 % der Kanban-Nutzer eingesetzt, davon zu je 50 % als Primary-Use und je 50 % als Secondary-Use [65]. Dabei ist Kanban "eine neue Change-Management-Methode, um in IT-Organisationen kleinschrittige Veränderungen mit hoher Akzeptanz aller Beteiligter durchzuführen", so David J. Anderson [16]. Dieser beschreibt eine Methode für evolutionäres Change-Management, welche das Pull-System bzw. -Element wie bei Kanban verwendet und Optimierungen an den Prozessen der Software-Entwicklung und dem IT-Betrieb mit minimalem Widerstand durch die beteiligten Personen gewährleistet [16]. Dazu verfügt Kanban über insgesamt sechs Prinzipien, von welchen drei zu den Change-Management-Prinzipien zählen [37][105]. Das erste dieser Prinzipien lautet "Beginne mit dem, was du gerade tust", dabei geht es darum, die bestehenden Prozesse und Abläufe und die gesamte bestehende Organisation nicht von Grund auf umstellen zu müssen, um Kanban nutzen und damit Änderungen und Verbesserungen der Prozesse durchzuführen zu können [37][105]. Das zweite Prinzip lautet "Vereinbare, dass evolutionäre Veränderung verfolgt wird", damit ist gemeint, dass es nie einen perfekten, idealen Zustand gibt, welcher erreicht werden kann oder soll. Stattdessen sollen die Prozesse Schritt für Schritt kontinuierlich optimiert und angepasst werden [37][105]. Das dritte Prinzip lautet "Fördere Führung auf allen Ebenen der Organisation – angefangen beim einzelnen Mitarbeiter bis zur Geschäftsleitung", damit ist wiederum gemeint, das Prinzip und Mindset von Kanban und der kontinuierlichen Verbesserung nicht nur bei einzelnen Teams oder Abteilungen einzuführen, sondern unternehmensweit in der gesamten Organisation, also auch bei den Managern und anderen Führungskräften [37][105].
Die Phase *Change-Management* wird also mäßig von Kanban abgedeckt.
In der Phase *Quality Management* wurde Kanban nur von 25 % der gesamten Kanban-Nutzern angewendet und dies auch nur ausschließlich als Secondary Use [65]. Dabei kann Kanban auch besonders in der Qualitätssicherung sinnvoll angewendet werden [49, p. 77-78, 100-103]. Dabei wirkt das charakterisierende Element "Pull" von Kanban (siehe Abschnitt 2.2.3), wodurch qualitätsgesicherte Anforderungen oder Features an den Auftraggeber bereitgestellt werden und implementierte Anforderungen oder Features eigenverantwortlich von den Personen in der Qualitätssicherung abgenommen und bearbeitet werden [49, p. 77-78, 100-103].
Durch das Element "Limitierte Mengen" von Kanban (siehe Abschnitt 2.2.3), werden nur in entsprechend limitierter Anzahl fertig entwickelte Anforderungen qualitätsgesichert. Durch das Element "Transparente Information" von Kanban (siehe Abschnitt 2.2.3), werden Probleme, Fortschritte oder Details über die Anforderungen angesprochen [49, p. 77-78, 100-103]. Innerhalb von Kanban gibt es z.B. zwei Techniken, Abnahmekriterien und automatisierte Tests, welche für die Qualitätssicherung mit Kanban eingesetzt werden können [49, p. 77-78, 100-103]. Durch Abnahmekriterien wird für eine Anforderung festgelegt, wann die Qualitätssicherung dieser

abgeschlossen ist und diese dann (endgültig) bereitgestellt werden kann [49, p. 77-78, 100-103]. Diese Technik realisiert bzw. basiert auf dem Element "Transparente Information" von Kanban, da die Kriterien für den Abschluss der Qualitätssicherung einer Anforderung öffentlich sind und sein sollten [49, p. 77-78, 100-103]. Mit automatisierten Tests können die Abnahmekriterien von (implementierten) Anforderungen schnell und in hoher Frequenz geprüft werden [49, p. 77-78, 100-103]. Dies ermöglicht und unterstützt das Entwickeln von Anforderungen in kleinen Schritten [49, p. 77-78, 100-103]. Diese Technik wiederum realisiert bzw. basiert auf dem Element "Pull" von Kanban, da die Tests für die automatisierte Ausführung bereitgestellt werden und dann eigenverantwortlich genutzt und ausgeführt werden können [49, p. 77-78, 100-103].
Kanban deckt die Phase *Quality Management* also mäßig ab.
Für die Phase *Risk Management* gibt es von Kanban keine klaren Vorgaben oder Anwendungsregeln und somit geht mit der Nutzung von Kanban nicht gleich sofort eine vollständige Abdeckung von dem *Risk Management* einher [137]. Dennoch wurde Kanban im *Risk Management* von ca. 16 % der Kanban-Nutzer verwendet und je 50 % als Primary Use und Secondary Use [65]. In einer Studie wurde untersucht, wie oft die agilen Methoden Scrum, Kanban und XP mit dem *Risk Management* kombiniert werden und dieses in diese Methoden integriert wird. Dabei wurden mittels einem SLM (Systematic Literature Mapping) alle Studien untersucht und gesucht, welche den Nutzen von einer agilen Methode mit dem *Risk Management* untersuchen [141]. Es wurde festgestellt, dass nur ca. 6 % der gesamten Studien über (die Integration von) *Risk Management* in Kanban handeln. Dagegen liegt z.B. der Anteil von Scrum bei ca. 44 % [141]. Dazu muss aber noch berücksichtigt werden, dass ca. 7 % der Unternehmen in der Software-Industrie Kanban hauptsächlich als agile Methode nutzen [141][57]. Jedoch gibt es außerdem einige Anwendungsmöglichkeiten von Kanban, um mit diesem (richtiges) *Risk Management* zu betreiben [141].
Das *Risk Management* wird also mäßig abgedeckt.
Das Testen in der Phase *Integration and Testing* ist bei Scrum und Kanban relativ ähnlich [23].
Abgesehen davon kann in dieser Phase das Kanban-Board die Spalte "Test" (zwischen den Spalten "Development" und "Ready to Deploy") enthalten [83], in welcher dann alle (soweit) fertig entwickelten Anforderungen von der Spalte "Development" abgelegt und für das Testen freigegeben werden [83].
Die Phase *Integration and Testing* wird von Kanban also mäßig abgedeckt.
In der Phase *Architecture and Design* wurde Kanban von ca. 17 % der gesamten Kanban-Nutzer verwendet, davon je 50 % als Primary Use und je 50 % als Secondary Use [65]. Es gibt einige Ansätze das Kanban-Board, also das Kern-Werkzeug von Kanban, für das Design zu verwenden [64][59, p. 36-43]. Einerseits gibt es den Ansatz, im Kanban-Board direkt eine "Design"-Spalte zu integrieren, in welcher die zu entwickelnden Anforderungen erst

mal entworfen werden und erst danach für die Entwicklung freigegeben werden [64][59, p. 36-43]. Dazu können innerhalb dieser Spalte noch zwei Unterspalten eingeführt werden, "In Arbeit" und "Fertig". Ist also ein Feature oder eine Anforderung fertig "designed" bzw. entworfen worden, wird dieses von der ursprünglichen "In Arbeit"-Spalte in die "Fertig"-Spalte geschoben. Somit kann diese dann nach dem Pull-Prinzip von Kanban von der Entwicklung bei freien Kapazitäten direkt genommen und bearbeitet werden [59, p. 36-43].
Andererseits gibt es den Ansatz für die Integration von einem Human-Centered-Design-Prozess (HCD-Prozess) in den bestehenden Prozess mit einem Kanban-Board [144], indem ein weiteres Konzeptions-Board vor das eigentliche Kanban-Board bzw. Entwicklungsboard, mit den zu implementierenden Anforderungen, geschaltet wird [144]. Dieses dient dann dem Konzeptionsteam, konzeptionelle Aufgaben oder Anforderungen zu verwalten und zu bearbeiten [144]. Der Kanban-Prozess wird dann also quasi um ein weiteres Kanban-Board nach "vorne" verlagert [144]. Somit werden konzeptionelle Aufgaben, für eine positive User-Experience, auf dem Konzeptionsboard genauso behandelt, wie die Entwicklungsaufgaben auf dem Entwicklungsboard [144]. Wurde eine Konzeptionsaufgabe auf dem Konzeptionsboard abgeschlossen, wird diese auf das Entwicklungsboard transferiert und auf dieses in die "To Do"-Spalte übertragen [144]. Die Phase *Architecture and Design* wird von Kanban also mäßig abgedeckt.
In der Phase *Requirements Analysis* wurde Kanban von 25 % der Kanban-Nutzer eingesetzt, davon 33 % als Primary Use [65]. In Kanban können für das Herauskitzeln und Ermitteln der Anforderungen sogenannte Kanban-Stakeholder-Interviews verwendet werden [49, p. 89-96]. Die Anforderungen können dann wie bei Scrum in User Stories formuliert werden und dann mittels Planning Poker der Aufwand dieser geschätzt werden [49, p. 89-96]. Diese beiden Techniken lassen sich gut mit Kanban umsetzen [49, p. 89-96]. Häufig tritt in Software-Projekten der Fall auf, dass erstmal lediglich eine Epic, also eine sehr umfassende und weit gefächerte Anforderung, statt einer schlanken, kurzen und eindeutigen User-Story definiert wird [59, p. 31-43]. Auch für diesen Fall liefert Kanban mittels der Anpassung des Kanban-Boards (bzw. der Visualisierung) eine Möglichkeit damit umzugehen [59, p. 31-43]. Dazu werden an das Kanban-Board die beiden Spalten "Epic" und "Analysiert" vorne hinzugefügt [59, p. 31-43]. Eine Epic bekommt eine ganze Reihe oder Zeile des Kanban-Boards und wird in einer Zelle der "Epic"-Spalte abgelegt, in der Zelle daneben (also der "Analysiert"-Spalte), werden dann nach und nach die User Stories abgelegt, welche durch die Zerkleinerung der Epic hervorgegangen sind [59, p. 31-43]. Die aus der Epic abgeleiteten User Stories werden dann regulär das Kanban-Board bzw. -Prozess durchlaufen und implementiert und bereitgestellt [59, p. 31-43]. Kanban deckt die Phase *Requirements Analysis* also mäßig ab.
In der Phase *Transition and Operations* wurde Kanban von ca. 17 % der

Kanban-Nutzer eingesetzt, davon 50 % als Primary Use [65]. Tatsächlich eignet sich Kanban für das Release Management und diese diese Phase. Dabei wirken die Kanban-Prinzipien wie folgt: Nach dem Kanban-Prinzip "visualization of the workflow" können Releases mittels dem Kanban-Board sichtbar gemacht werden, dadurch kann auf einen Überblick erfasst werden, welche Releases zu bearbeiten sind oder bereitstehen [61]. Wenn ein Projekt also bereit ist, released zu werden, dann würde für dieses ein Release-Work-Item (als Karte) zu dem Kanban-Board hinzugefügt werden und dieses durchläuft dann auf dem Kanban-Board sichtbar den gesamten Release-Prozess [61]. Nach der Kanban-Praktik "limitation of the work in progress WIP" werden die Releases in jedem Zustand limitiert und so verhindert, dass das Release-Team überfordert wird [61]. Nach dem Kanban-Prinzip "measurement and control of the lead time" wird die Durchlaufzeit kontrolliert, somit können zeitintensive Zustände und somit auch Engpässe im Release-Prozess identifiziert werden [61].
Somit wird die Phase *Transition and Operations* von Kanban mäßig abgedeckt.

Keine Abdeckung

In der Phase *Configuration Management* wurde Kanban von gerade einmal ca. 8,33 % der Kanban-Nutzer eingesetzt und dies auch nur als Secondary Use [65]. Im Rahmen dieser Untersuchung konnte in sämtlichen Papern zu Kanban nichts (konkretes) über den Einfluss von dieser Methode auf das *Configuration Management* gefunden werden. Ebenso findet diese Phase im (offiziellen) Kanban-Guide der Kanban University keine Erwähnung [9]. Aus diesen Gründen wird geschlussfolgert, dass Kanban das *Configuration Management* nicht abdeckt.

Abdeckung durch Kanban	Phase/Projekt-Disziplin
	Integration and Testing
	Implementation and Coding
	Requirements Management
	Configuration Management
	Risk Management
	Project Management
	Requirements Analysis
	Quality Management
	Architecture and Design
	Maintenance and Evolution
	Change Management
	Transition and Operations

Tabelle 4.4: Abdeckung der Phasen durch Kanban

4.2.4 Abdeckung der Phasen bei Praktiken

Code Reviews

- Starke Abdeckung

In den Phasen *Implementation and Coding* und *Quality Management* werden Code Reviews mit Abstand am häufigsten genutzt. In der Phase *Implementation and Coding* haben 71,43 % der Code Reviews-Nutzer diese angewendet, davon jeweils 60 % als Primary Use [65]. Dieses aussagekräftige Ergebnis lässt sich für diese Phase schon alleine aufgrund der Tatsache nicht anzweifeln, dass der Kern von Code Reviews darin besteht, neuen oder veränderten Code vor der Auslieferung und Bereitstellung, von einem oder mehreren anderen Entwicklern zu prüfen [113].
Für die Phase *Quality Management* nutzen ca. 38,10 % der Code Reviews-Nutzer diese Praktik, davon 62,50 % als Primary Use [65]. Bei einem Code Review kann der Entwickler oder Reviewer neben typischen Fehlern auch Mängel in der Codequalität identifizieren und ggf. beheben. Somit können Code Reviews also auch für das *Quality Management* ein großer Gewinn sein [113].

- Mäßige Abdeckung

Tatsächlich wird manchmal außer Acht gelassen, dass auch der Code von den Tests und Testfällen ebenfalls geprüft werden sollte, da auch dieser eine gewisse Qualität aufweisen sollte [125]. Deshalb deckt die Praktik Code Reviews die Phase *Integration and Testing* oder zumindestens einen Teil des Testens dieser Phase mäßig ab.
Code Reviews können auch von Sicherheitsspezialisten für das Sicherheits- bzw. Risikomanagement genutzt werden, indem diese den Code auf Schwachstellen (wie z.B. SQL-Injections) hin untersuchen [113]. Aus diesem Grund ist die Abdeckung von *Risk Managment* durchaus mäßig, obwohl Code Reviews in dieser Phase nur von ca. 14 % der Code Reviews-Nutzer eingesetzt wurden [65].
Auch in der Phase *Maintenance and Evolution* können Code Reviews effektiv und sinnvoll eingesetzt werden [113][149]. Dies liegt daran, dass auch bereits etablierter Code untersucht werden kann und sollte, falls dieser geändert wurde [149]. Ca. 14 % der Code Reviews-Nutzer nutzten diese Praktik in dieser Phase, davon ca. 33 % als Primary Use [65].

- Keine Abdeckung

In den Phasen *Change Management*, *Requirements Management*,*Architecture and Design*, *Project Management* und *Transition and Operations* und *Configuration Management* wurden Code Reviews mit weniger als 15 % sehr gering eingesetzt und auch dann nur als Secondary Use [65]. Zwischen

dem Nutzen von Code Reviews und diesen Phasen besteht kaum ein Bezug [113][125][149]. Aus diesen Gründen ist die Abdeckung dieser Phasen durch Code Reviews sehr gering oder kaum vorhanden. Auch das *Requirements Analysis* wird kaum abgedeckt, da Code Reviews natürlich erst dann zur Anwendung kommen können, wenn Code zur Verfügung steht [39]. Das *Requirements Analysis* hingegen findet vor der Entwicklung (von Code) statt [44].

Abdeckung durch Code Reviews	Phase/Projekt-Disziplin
	Integration and Testing
	Implementation and Coding
	Requirements Management
	Configuration Management
	Risk Management
	Project Management
	Requirements Analysis
	Quality Management
	Architecture and Design
	Maintenance and Evolution
	Change Management
	Transition and Operations

Tabelle 4.5: Abdeckung der Phasen durch Code Reviews

Daily Stand-Up Meetings

- Starke Abdeckung

In der Phase *Project Management* wurden Daily Stand-Up Meetings mit ca. 62 % am häufigsten eingesetzt, davon jedoch nur ca. 38 % als Primary Use [65]. Ein Grund für das Nutzen dieser Praktik für das *Project Management* ist, dass durch die täglichen Meetings ein Überblick über die Aktivitäten der anderen Entwickler und Team-Mitglieder aufgebaut wird und erhalten bleibt [128]. Somit bekommt man also auch einen Überlick über das gesamte Projekt und die gesamte Entwicklung.
Für die Phase *Implementation and Coding* wurden Daily Stand-Up Meetings im Vergleich zu anderen Phasen besonders stark als Primary Use eingesetzt. Der Anteil von Primary Use liegt bei ca. 48 % und ingesamt bei ca. 52 %, somit wurde die Praktik in dieser Phase zu ca. 91 % nur als Primary Use eingesetzt [65]. Dies liegt z.B. daran, dass durch diese Praktik die Zusammenarbeit zwischen den Team-Mitgliedern und den Entwicklern verbessert wird [148]. In Daily Stand-Up Meetings können außerdem Hindernisse und Risiken für das Erreichen der (Entwicklungs-)Ziele identifiziert werden [142]. Abgesehen davon haben solche Gesicht-zu-Gesicht Meetings

den Vorteil gegenüber anderen Kommunikations-Formen, wie z.B. Anrufe oder Chats, dass diese geplant sind und man andere Team-Mitglieder somit nicht während der Entwicklung unterbricht oder stört, indem man diese bei Fragen also z.B. spontan anschreibt oder anruft, anstatt diese Fragen im Daily Stand-Up Meeting zu klären [128].

- Mäßige Abdeckung

In der Phase *Requirements Management* wurde die Praktik Daily Stand-Up Meetings das einzige Mal, abgesehen von den Phasen *Project Management* und *Implementation and Coding*, als Primary Use verwendet. Insgesamt wurde diese Praktik in dieser Phase mit einem Anteil von ca. 10 % verwendet [65]. Bei dieser Praktik ist von Vorteil, dass die Daily Stand-Up Meetings auch mit dem Kunden bzw. On-Site Customer gehalten werden, wobei dieser dann die Anforderungen priorisiert oder auch andere Funktionalitäten spezifiziert [148]. Dies könnte sich auch positiv auf das Ermitteln von Anforderungen auswirken. In der Phase *Requirements Analysis* wurden Daily Stand-Up Meetings nur als Secondary Use eingesetzt, dafür aber von ca. 14 % der Nutzer dieser Praktik [65].
In der Phase *Maintenance and Evolution* wurden Daily Stand-Up Meetings von ca. 14 % eingesetzt und dies nur als Secondary Use [65]. Daily Stand-Up Meetings sind im *Maintenance and Evolution* nicht immer hilfreich, weil es den Team-Mitgliedern der Wartung manchmal an Motivation fehlt, einerseits anwesend zu sein oder andererseits, dem Präsentierenden genug Aufmerksamkeit zu schenken [12]. Jedoch helfen Daily Stand-Up Meetings generell einen Überblick über die Backlog-Items und die zu erledigende Arbeit zu behalten [95], so könnte dies dann auch in der Wartung, ähnlich wie im *Project Management*, hilfreich sein.

- Keine Abdeckung

Trotz der (mäßigen) Nutzbarkeit von Daily Stand-Up Meetings in Phasen wie z.B. dem *Requirements Management* oder *Implementation and Coding*, ist diese Praktik dennoch (eigentlich) für das *Project Management* vorgesehen [39]. Es ist also z.B. kein Werkzeug für die Implementierung, kann in dieser Phase jedoch eingesetzt werden, da über die Inhalte der Implementierung oder den Aufgaben und Anforderungen gesprochen wird [39]. Die folgenden Phasen hingegen, finden in der grundlegenden Definition von Daily Stand-Up Meetings keine Erwähnung [51, p. 46][99, p. 80-83] und für diese ist diese Praktik auch nicht vorgesehen, sondern lediglich für das *Project Management* [39]. Außerdem wurden Daily Stand-Up Meetings für jede dieser Phasen in der Studie nur mit einem Anteil von ca. 5 % verwendet und dies nur als Secondary Use [65]. Diese Phasen sind *Change Management*, *Architecture and Design*, *Transition and Operations*, *Configuration Management*, *Risk Managment*, *Integration and Testing* und *Quality Management*. Diese werden also alle kaum bis gar nicht von Daily Stand-Up Meetings abgedeckt.

Abdeckung durch DSMs	Phase/Projekt-Disziplin
	Integration and Testing
	Implementation and Coding
	Requirements Management
	Configuration Management
	Risk Management
	Project Management
	Requirements Analysis
	Quality Management
	Architecture and Design
	Maintenance and Evolution
	Change Management
	Transition and Operations

Tabelle 4.6: Abdeckung der Phasen durch DSMs

Coding Standards

- Starke Abdeckung

In der Phase *Implementation and Coding* wurden Coding Standards von 65 % der Coding Standards-Nutzer genutzt, davon ca. 62 % als Primary Use [65]. Dies ist zu erwarten, da Coding Standards in dieser Phase viele Vorteile mit sich bringen, denn durch Coding Standards wird einerseits die Teamarbeit gefördert, da dadurch der Code von anderen Entwicklern leichter gelesen und verstanden werden kann [131], anderseits wird durch Coding Standards die Geschwindigkeit der Entwicklung erhöht, da die Entwickler sich so nicht immer alleine für die anzuwendenden Programmierstile oder Prinzipien (neu) entscheiden müssen [131].
In der Phase *Quality Management* wurden Coding Standards von 25 % der Coding Standards-Nutzer eingesetzt und davon 40 % als Primary Use [65]. Dies ist ebenfalls nicht überraschend, denn durch Coding Standards wird die Code-Qualität des Codes erhöht. Dies liegt daran, dass dadurch die Entwickler ermutigt und getrieben werden, direkt in einer konsistenten Weise zu entwickeln [131][118, p. 13].

- Mäßige Abdeckung

In der Phase *Integration and Testing* wurden Coding Standards von 15 % der Coding Standards-Nutzer eingesetzt und davon ca. 67 % als Primary Use [65]. Coding Standards haben einen erheblichen indirekten positiven Einfluss auf die Testphase, denn durch die Anwendung von sicheren oder fehler-vermeidenden Coding Standards werden Probleme und Fehler im Code verhindert [50], bevor diese erst überhaupt in der Testphase auftreten. Somit

werden die Kosten in der Testphase vermindert [97].
In der Phase *Maintenance and Evolution* wurden Coding Standards zwar nur von 5 % der Coding Standards-Nutzer eingesetzt, jedoch dann auch nur als Primary Use [65]. Coding Standards können helfen, bestehenden Code zu warten, da durch diese die Übersichtlichkeit und Lesbarkeit des Codes erhalten bleibt und gesteigert wird [41].
In der Phase *Architecture and Design* wurden Coding Standards von 10 % der Coding Standards-Nutzer eingesetzt, je 50 % als Primary und Secondary Use [65]. Es ist generell schwierig, den Design Stil (also das Entwerfen) und den Coding Stil vollständig voneinander zu trennen [131, p. 11]. Dementsprechend könnte es also sein, dass der Nutzen von Coding Standards in dieser Phase nur auf den vorgegebenen Coding Stil (der Coding Standards) zurückgeht.

- Keine Abdeckung

In den Phasen *Requirements Analysis*, *Requirements Management* wurden Coding Standards mit einem Anteil von 5 % eingesetzt und dies nur als Secondary Use [65]. Dies liegt nahe, denn Coding Standards beziehen sich ja nur auf den Code oder andersherum gesagt, ohne Code sind Coding Standards zwecklos [76]. Somit kann es vor allem in der *Requirements Analysis* eigentlich keinen Nutzen dieser Praktik geben, da diese ja frühestens erst eine Phase später im Entwurf angewendet wird, da erst da Code mit ins "Spiel" kommen könnte [76].
In den Phasen *Transition and Operations* und *Configuration Management* wurden Coding Standards von nicht mal mehr als 5 % eingesetzt [65]. Dies lässt sich mit der Definition dieser Praktik erklären und bestätigen, denn diese bezieht sich ja nur auf den Code [76], somit kann diese für die zuletzt genannten Phasen auch keine Unterstützung bieten, denn diese haben mit dem Code hingegen wenig zu tun.
Die Phasen *Change Management* und *Project Management* werden durch Coding Standards ebenfalls nicht unterstützt, trotz der jeweils einmaligen Angabe des Nutzens dieser Praktik als Primary Use [65]. Dies müsste auf ein falsches Verständnis der Phasen zurückzuführen sein, denn wie oben bereits beschrieben, beziehen sich Coding Standards nur auf den Code [76].
Für die Phase *Risk Managment* wurden Coding Standards nicht ein einziges Mal verwendet [65], wobei Coding Standards die Sicherheit des Codes stärken können, da z.B. in der Programmiersprache C durch das Anwenden von sicheren Programmierregeln bzw. -stilen (vor allem typische) Sicherheitslücken verhindert werden können [118, p. 13][72]. Dabei können bestimmte Coding Standards in C z.B. Möglichkeiten für Pufferüberläufe effektiv verhindern, welche ja meistens ein Risiko für die Sicherheit und Verfügbarkeit darstellen [118, p. 230-231].

Abdeckung durch Coding Standards	Phase/Projekt-Disziplin
	Integration and Testing Implementation and Coding Requirements Management Configuration Management Risk Management Project Management Requirements Analysis Quality Management Architecture and Design Maintenance and Evolution Change Management Transition and Operations

Tabelle 4.7: Abdeckung der Phasen durch Coding Standards

Im weiteren Verlauf der Arbeit wurden zusätzlich sieben weitere Praktiken hinsichtlich ihrer Abdeckung der Phasen des Software-Lebenszyklus untersucht. Dazu wurden vorrangig die Werte des Diagramms in Abbildung 4.3 genutzt, da dieses auch Werte für diese sieben zusätzlichen Praktiken besitzt und mit dieser auch entsprechende Aussagen über die Nutzung dieser Praktiken in allen Phasen des SW-Lebenzyklus getroffen werden können. Dabei werden hier jedoch nur die Ergebnisse, also die Zuordnungen der Phasen zu den Stufen der Abdeckung und die entsprechenden Quellen sowie Referenzen für diese Zuordnungen präsentiert. Dennoch wurde die Untersuchung dieser sieben weiteren Praktiken genau nach dem gleichen Verlauf und Schema wie bei den drei zuvor untersuchten Praktiken durchgeführt.

Automated Unit Testing

Abdeckung	Phasen
stark	*Implementation and Coding* [94][35][79][136][150][39], *Integration and Testing* [136][150][110][39]
mäßig	*Quality Management* [27][130][89][31][39]
keine	*Risk Management*, *Transition and Operations*, *Configuration Management*, *Architecture and Design*, *Requirements Analysis*, *Requirements Management*, *Maintenance and Evolution*, *Change Management*, *Project Management* [39]

Tabelle 4.8: Abdeckung der Phasen durch Automated Unit Testing

Continuous Integration

Abdeckung	Phasen
stark	*Implementation and Coding* [45][52, p. 152-153][39], *Integration and Testing* [45][52, p. 152-153][39]
mäßig	*Quality Management* [38][111][39], *Configuration Management* [45][39] *Risk Management* [45][84][39]
keine	*Architecture and Design* [39][122], *Requirements Analysis* [39][122], *Requirements Management* [122] [39] (finden im SW-Lebenszyklus vor der Anwendung von CI statt) *Project Management*, *Maintenance and Evolution*, *Change Management*, *Transition and Operations*[39]

Tabelle 4.9: Abdeckung der Phasen durch CI

Iteration Planning

Abdeckung	Phasen
stark	*Project Management* [6][39], *Implementation and Coding* [78][39]
mäßig	*Change Management* [11][39], *Quality Management* [33][39]
keine	*Requirements Management*, *Requirements Analysis*, *Quality Management*, *Architecture and Design*, *Transition and Operations*, *Configuration Management*, *Quality Management*, *Maintenance and Evolution* [39]

Tabelle 4.10: Abdeckung der Phasen durch Iteration Planning

Release Planning

Abdeckung	Phasen
stark	*Project Management* [15][39], *Transition and Operations* [109][39]
mäßig	*Quality Management* [34][39], *Requirements Management* [34] [15][39], *Integration and Testing* [147][108][1][39]
keine	*Implementation and Coding*, *Configuration Management*, *Maintenance and Evolution*, *Change Management*, *Risk Management*, *Architecture and Design*, *Requirements Analysis* [39]

Tabelle 4.11: Abdeckung der Phasen durch Release Planning

User Stories

Abdeckung	Phasen
stark	*Requirements Analysis* [40][90][43][39], *Requirements Management* [36] [81][39]
mäßig	*Project Management* [26][143][39], *Implementation and Coding* [81][39], *Quality Management* [100][81][39], *Integration and Testing* [100][39]
keine	*Maintenance and Evolution*, *Change Management*, *Risk Management*, *Architecture and Design*, *Configuration Management*, *Transition and Operations* [39]

Tabelle 4.12: Abdeckung der Phasen durch User Stories

Backlog Management

Abdeckung	Phasen
stark	*Project Management* [119][58][39], *Implementation and Coding* [18][120][39], *Requirements Management* [117][55][39]
mäßig	*Change Management* [14][58][39]
keine	*Risk Management*, *Architecture and Design*, *Requirements Analysis*, *Quality Management*, *Maintenance and Evolution*, *Configuration Management*, *Integration and Testing*, *Transition and Operations* [39]

Tabelle 4.13: Abdeckung der Phasen durch Backlog Management

Burn-Down-Charts

Abdeckung	Phasen
stark	*Project Management* [140][30][32][39], *Implementation and Coding* [146][114][39], *Risk Management* [103][32][74][39]
mäßig	*Integration and Testing* [86][39]
keine	*Transition and Operations*, *Maintenance and Evolution*, *Architecture and Design*, *Quality Management*, *Change Management*, *Requirements Analysis*, *Requirements Management*, *Configuration Management* [39]

Tabelle 4.14: Abdeckung der Phasen durch Burn-Down-Charts

4.3 Finale Abdeckung einer Phase

Bisher wurde die Abdeckung einer Phase durch nur jeweils eine der Methoden und Praktiken untersucht und festgelegt, jedoch werden bei hybriden Entwicklungsprozessen ja mehrere Methoden miteinander kombiniert. Dadurch wird die Abdeckung einer Phase somit auch durch mehrere Methoden oder Praktiken beeinflusst und bestimmt. Entsprechend der zu entwickelnden Anwendung, wird die Abdeckung einer Phase ja mit drei Stufen bewertet: starke Abdeckung, mäßige Abdeckung und keine Abdeckung.
Falls nun mehrere Methoden in Form eines hybriden Entwicklungsprozesses miteinander kombiniert werden, dann werden die Abdeckungsstufen der Phasen bei den Methoden nicht einfach addiert, also aus zwei mäßigen Abdeckungen wird keine starke Abdeckung. Diese werden sondern nur ergänzt, also bei einer mäßigen Abdeckung und einer starken Abdeckung, bleibt es bei einer starken Abdeckung. Bei zwei mäßigen Abdeckungen wird es somit auch bei einer mäßigen Abdeckung bleiben und diese nicht zu einer starken Abdeckung werden.
In der Tabelle 4.15 unten sind für alle dreier Kombinationen der Abdeckungsstufen die jeweiligen resultierenden Abdeckungsstufen aufgelistet.

Abdeckungskombination	Finale resultierende Abdeckung
keine - keine - keine	keine
mäßig - keine - keine	mäßig
mäßig - mäßig - keine	mäßig
mäßig - mäßig - mäßig	mäßig
stark - keine - keine	stark
stark - stark - keine	stark
stark - stark - stark	stark
mäßig - stark - keine	stark
stark - stark - mäßig	stark
mäßig - stark - mäßig	stark

Tabelle 4.15: Finale resultierende Abdeckung bei mehreren Abdeckungen

Kapitel 5

Implementierung

5.1 Programmiersprache

Für die Entwicklung der Anwendung wurde die Programmiersprache Java und die IntelliJ IDE verwendet. Dabei wurde ergänzend auf das Framework JavaFX zur Umsetzung und Entwicklung der grafischen Benutzeroberfläche mitsamt der Schaltflächen, Buttons, Textfelder und Auswahlflächen zurückgegriffen. Java wurde vor allem aufgrund der besonderen Plattformunabhängigkeit gewählt, da eine wichtige Qualitätsanforderung für diese Anwendung ja gerade dies voraussetzt (siehe Kapitel 3). JavaFX als zusätzliches Framework wurde daraufhin gewählt, da dieses alle benötigten Interaktionselemente für die grafische Benutzeroberfläche anbietet, also z.B. Buttons [5]. Die Anwendung wurde mit dem Tool Maven in der IntelliJ IDE als JAR-Datei exportiert und dann mittels dem Tool Launch4j in eine auf Windows-Systemen ausführbare Datei konvertiert [3][4][2].

5.2 Architektur

Die Anwendung ist mit dem Design Pattern Model-View-Controller (MVC) entworfen und implementiert worden. Dabei decken die drei Komponenten View, Model und Controller unterschiedliche Bereiche der Anwendung ab. View ist ausschließlich für die Präsentation und Benutzerinteraktion zuständig, während Model hingegen für das Speichern und Verwalten der Daten verantwortlich ist. Die hauptsächliche Logik und Steuerung von den Komponenten Model sowie View wird von der Komponente Controller verantwortet. Folgendes UML-Diagramm in Abbildung 5.1 veranschaulicht die Architektur der Anwendung. Die Funktionen und Attribute bei den drei Klassen im UML-Diagramm sind, der Übersichtlichkeit wegen, nur auf die wichtigsten beschränkt.

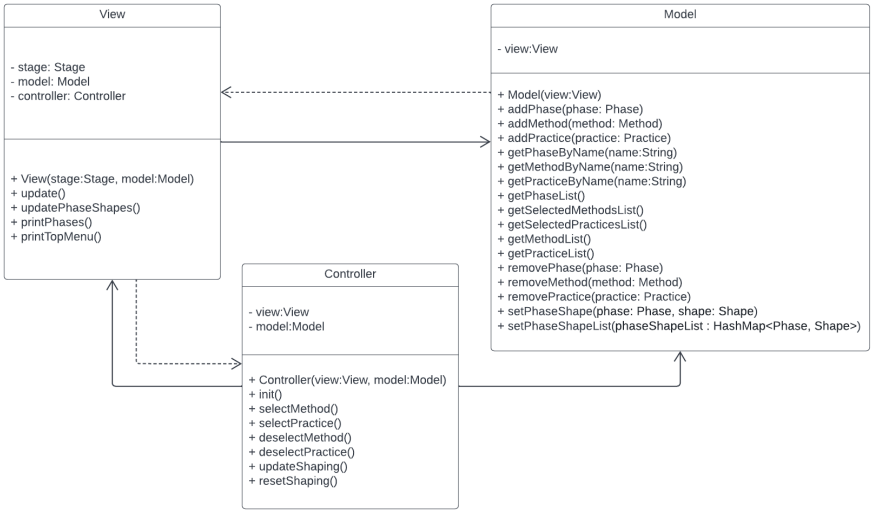


Abbildung 5.1: Vereinfachtes UML-Diagramm der Anwendung

5.2.1 Modell

Diese Komponente wird durch die Klasse Model umgesetzt und verwaltet die Datenstrukturen, in welchen unter anderem die ausgewählten und die zur Auswahl stehenden Praktiken und Methoden gespeichert sind. Auch die Phasen werden von dieser Komponente verwaltet. Dazu stellt diese Komponente/Klasse entsprechende Funktionen zum Festlegen, Ändern sowie Löschen der Daten bereit.

5.2.2 View

Diese Komponente wird durch die Klasse View umgesetzt und stellt alle nötigen Funktionen zum Anzeigen und Darstellen der Phasen auf dem Diagramm sowie dem Erstellen der gesamten Menüs, mit ihren Schaltflächen und Buttons, bereit.

5.2.3 Controller

Diese Komponente wird durch die Klasse Controller umgesetzt und übernimmt die logische Verarbeitung der Eingaben sowie die Berechnung der Abdeckung des Software-Lebenszyklus und der nötigen zu hinzufügenden Methoden und Praktiken für eine vollständige Abdeckung.

5.3 Gestaltung und Bedienung

In der Abbildung 5.2 unten ist die Benutzeroberfläche der Anwendung mit ihren Buttons und anderen Schaltflächen abgebildet. Dabei befindet sich die Anwendung da noch in dem initialen Zustand, denn es wurden noch keine Methoden oder Praktiken ausgewählt.

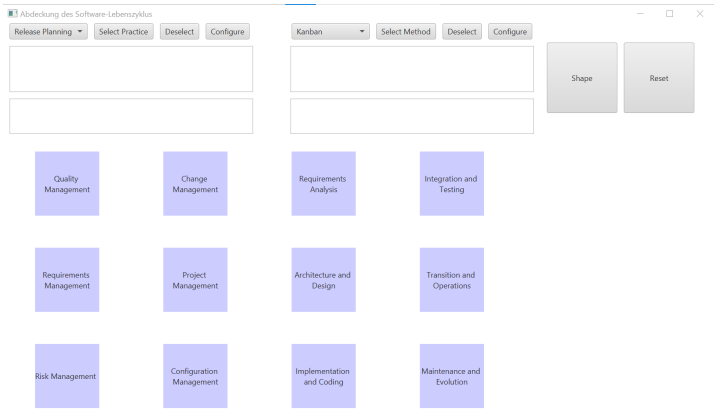


Abbildung 5.2: Anwendung ohne einer ausgewählten Kombination

In der Abbildung 5.3 hingegen, wurden z.B. die Praktiken *Release Planning* und *Continuous Integration* sowie die Methode *Kanban* ausgewählt und anschließend auf den Button *Shape* geklickt. Mit einem Klick auf den Button *Reset* wird dann wieder in den initialen Zustand, wie in der Abbildung 5.2 oben, zurückgekehrt.

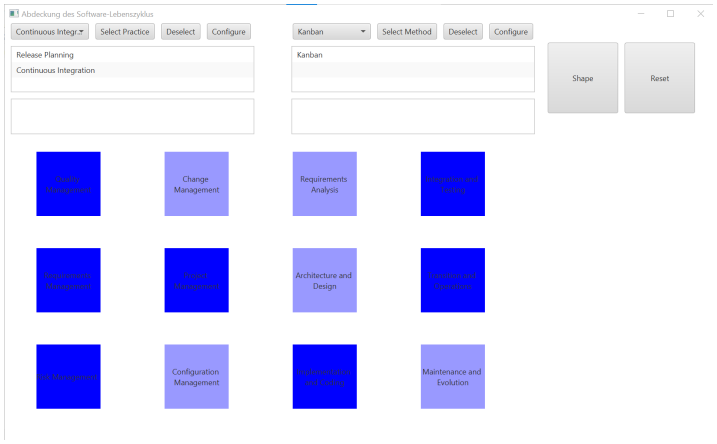


Abbildung 5.3: Anwendung mit einer ausgewählten Kombination

Kapitel 6

Evaluation

Um die Anwendung mit ihren Ergebnissen für die Abdeckung des Software-Lebenszyklus bei den verschiedenen Kombinationen von Methoden und Praktiken zu evaluieren, werden zuerst alle Kombinationen von den drei Methoden Scrum, Kanban und DevOps ausgewählt. Die Abdeckungen all dieser Kombinationen und die Phasen, welche nicht abgedeckt werden, werden dann mithilfe der Anwendung bestimmt. Infolgedessen wird die Nutzung dieser Kombinationen in der Industrie untersucht. Dies erfolgt dann mithilfe der Daten der HELENA-Studie (siehe Kapitel 8). Dabei wird unter anderem auch untersucht, welche Praktiken bei einer Kombination noch zusätzlich genutzt wurden. Es wird dann untersucht und geschlussfolgert, inwiefern diese zusätzlich genutzten Praktiken die bei der Kombination nicht ausreichend abgedeckten Phasen zusätzlich abdecken.
Hat eine Kombination an sich also nur eine geringe Abdeckung, wird jedoch in der Industrie oft mit einer anderen weiteren Praktik kombiniert, dann wird dahingehend untersucht, in wie weit diese Praktik zu einer vollständigeren Abdeckung des Software-Lebenszyklus beitragen könnte. Insgesamt wird in diesem Kapitel also der Frage nachgegangen, ob die von der Anwendung berechneten Abdeckungen auch mit der Nutzung dieser Kombinationen in der Industrie bestätigt werden können.

6.1 Zu untersuchende Kombinationen

In der Tabelle unten sind die ausgewählten Kombinationen von den drei Methoden Scrum, Kanban und DevOps aufgeführt, welche untersucht werden sollen. Dabei wurden alle möglichen Kombinationen dieser ausgewählt, bis auf die Kombination *kein Scrum, kein Kanban, kein DevOps*.
In den zwei Spalten “mäßige Abdeckung” und “keine Abdeckung” sind dazu für die entsprechenden Kombinationen die Phasen aufgeführt, welche nur mäßig oder gar nicht von diesen abgedeckt werden.

Diese Phasen wurden dabei mittels der Anwendung ermittelt, indem die Abdeckung des SW-Lebensyzklus für diese bestimmt wurde.
Dabei sind vor allem die weniger abgedeckten Phasen dieser Kombinationen wichtig für die Evaluation der Anwendung. Anhand dieser kann eher die Abdeckung in der Anwendung beurteilt werden, denn eine fehlende Abdeckung könnte z.B. durch das Nutzen von bestimmten Praktiken ausgeglichen werden, welches sich dann mit den Daten der HELENA-Studie feststellen lassen könnte. An dieser HELENA-Studie haben weltweit ca. 1500 Unternehmen teilgenommen. Die Teilnehmer wurden dabei hinsichtlich der von ihnen eingesetzten Methoden und Praktiken für die Software-Entwicklung befragt und in welchem Ausmaß diese in ihrem Unternehmen genutzt werden [71][70].
Die Praktiken wurden hier bei der Auswahl der Kombinationen also vorerst noch nicht berücksichtigst, sondern erst im nächstem Unterkapitel, wenn die Nutzung der Kombinationen dieser Methoden in der Industrie mit den Daten der HELENA-Studie untersucht wird.

Kombination	Mäßige Abdeckung	Keine Abdeckung
Scrum kein DevOps kein Kanban	ChangeM, IntTest, ReqAn, ArchDes, TranOp	QM, ReqM, RiskM, ConfM, MaintEvo
Scrum DevOps kein Kanban	ChangeM, ConfM, MaintEvo	QM, ReqM, RiskM
DevOps kein Scrum kein Kanban	ReqAn, ArchDes, MaintEvo, ConfM	QM, ReqM, RiskM
Scrum Kanban kein DevOps	QM, RiskM, MaintEvo	ConfM
Scrum Kanban DevOps	ConfM, QM, RiskM	-
Kanban kein Scrum kein DevOps	ChangeM, IntTest, ReqAn, ArchDes, TranOp, QM, RiskM, MaintEvo	ConfM
Kanban DevOps kein Scrum	ChangeM, QM, RiskM, ConfM	-

Tabelle 6.1: Kombinationen und die weniger abgedeckten Phasen

6.2 Vergleich mit der Nutzung in der Industrie

Zuerst wurde für jede der oben genannten Kombinationen die gesamte Abdeckung des SW-Lebenszyklus mit der Anwendung bestimmt und dann vor allem die wenig und nicht abgedeckten Phasen für diese Kombination im weiteren Verlauf genauer untersucht und mit der Nutzung in der Industrie verglichen. Dazu wurden die Daten der HELENA-Studie genutzt.

6.2.1 Datenbasis

Um die Nutzung der Kombinationen von Methoden und Praktiken in der Industrie zu untersuchen und mit der von der Anwendung berechneten Abdeckung zu vergleichen, wurde die HELENA-Studie genutzt. In dieser internationalen Studie wurden circa 1500 teilnehmende Unternehmen hinsichtlich der Nutzung ihrer Methoden und Praktiken befragt [65][70][71]. Dabei konnte jeder Teilnehmer für jede der aufgeführten Methoden und Praktiken Werte angeben, inwiefern diese genutzt wurden. Für die Methoden und Praktiken können diese Werte wie folgt übersetzt werden: 7-3, immer bis gar nicht genutzt, 2, Nutzen unbekannt, 1, Methode oder Praktik unbekannt, -9, nicht beantwortet.

6.2.2 Datenselektion

Innerhalb der Evaluation der Anwendung wurde die Nutzung von Kombinationen von Methoden und Praktiken untersucht. Dabei wurde festgelegt, dass eine Methode oder Praktik in den Daten der HELENA-Studie dann als "genutzt" betrachtet wird, wenn der Nutzen mit 6 oder 7 bewertet wurde. Ansonsten gilt die diese als nicht genutzt, da ab dem Wert 5 abwärts bis 3 die Methode oder Praktik nur manchmal bis gar nicht genutzt wurde und bei 2, 1 oder -9 der Nutzen der Methode oder Praktik auch nicht eindeutig hervorgeht.

6.2.3 Datenanalyse

Um die Daten auszuwählen und zu bestimmen, bei welchen eine bestimmte Kombination von Methoden genutzt wurde, wurden also die Datensätze herausgefiltert, bei welchen für jede der Methoden dieser Kombination jeweils zutrifft, dass diese "genutzt" wurde oder nicht. Bei allen Datensätzen der Kombination *Scrum - kein DevOps - kein Kanban* trifft also zu, dass die Werte für Scrum nur 6 oder 7 betragen und für DevOps und Kanban nur 1-5 oder -9.
Um die relative Häufigkeit zu untersuchen, mit der eine Praktik nun neben einer Kombination genutzt wurde, wurde der Anteil bei den insgesamt

ausgewählten Datensätze für diese bestimmte Kombination untersucht, bei welchen für diese Praktik die Werte 6 oder 7 angegeben wurden von insgesamt allen Datensätzen für diese Kombination, bei welchen für diese Praktik die Werte 1-7 oder -9, also alle möglichen Werte, angegeben wurden.

6.2.4 Ergebnisse

Zuerst wurde untersucht, wie oft die Kombinationen der Methoden in der HELENA-Studie im Verhältnis genutzt wurden:

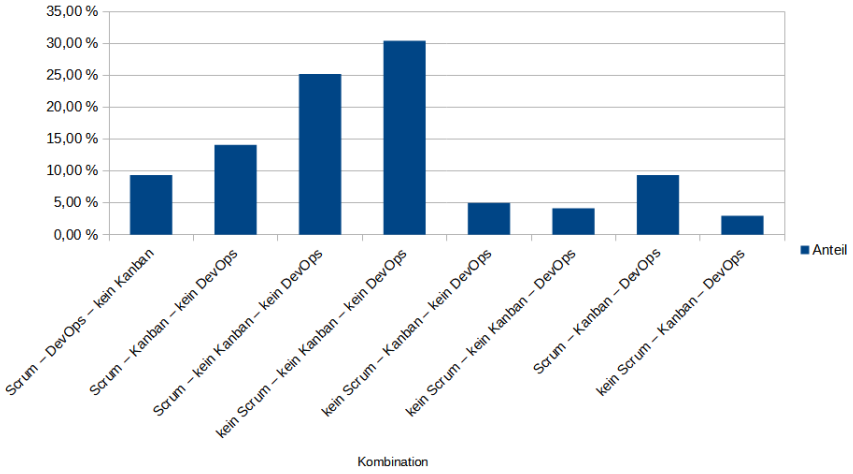


Abbildung 6.1: Kombinationen und deren Nutzung

Die Kombination *Scrum - kein Kanban - kein DevOps* wurde von allen Kombinationen, bei welchen mindestens eine Methode genutzt wurde, am häufigsten eingesetzt. Dagegen werden von allen Kombinationen in der Anwendung bei dieser am meisten bzw. sechs Phasen nicht abgedeckt.
Die Kombination *Scrum - Kanban - DevOps* deckt in der Anwendung hingegen alle Phasen ab, genutzt wird diese jedoch nur von etwas weniger als 10 % der Teilnehmenden, die Kombination *Scrum - Kanban - kein DevOps* dagegen wird von etwas weniger als 15 % genutzt. Allerdings ist bei dieser Kombination die Abdeckung geringer, denn z.B. die Phase *Configuration Management* wird bei dieser nicht abgedeckt oder das *Integration and Testing* nur mäßig, während dieses von *Scrum - Kanban - DevOps* stark abgedeckt wird.
Kanban - DevOps - kein Scrum, *kein Scrum - Kanban - kein DevOps* und *kein Scrum - kein Kanban - DevOps* werden am wenigsten verwendet, denn nur ca. 2,96 % bzw. 4,97 % bzw. 4,14 % nutzen diese. Dabei decken diese mehr Phasen ab als *Scrum - kein Kanban - kein DevOps* mit 25,21 %.

Kombinationen mit Scrum

Bei allen Kombinationen mit Scrum wurden die genannten Praktiken (außer Pair Programming und Use Case Modeling) überdurchschnittlich oft eingesetzt. Es könnte sein, dass die Praktiken generell gut zu Scrum passen. Außerdem verändert sich die Nutzung von den untersuchten Praktiken bei den Kombinationen nicht besonders, wenn bei allen Scrum genutzt wurde. Dies könnte darauf hindeuten, dass Scrum an erster Stelle oder dominanter eingesetzt wird, wenn dieses mit Kanban oder DevOps kombiniert wird.
Bei den Kombinationen mit Scrum wurden Burn Down Charts mindestens ca. 20 Prozentpunkte über dem Durchschnitt eingesetzt, außer bei *Scrum - kein Kanban - kein DevOps*, da nur mit ca. 10 Prozentpunkten. Gerade diese Praktik deckt das *Risk Management* stark ab, welches gerade von Scrum gar nicht abgedeckt wird. Somit könnte es sein, dass diese Praktik das fehlende *Risk Management* abdeckt oder aber auch einfach gut zu Scrum passt.
Bei allen Kombinationen mit DevOps, mit/ohne Scrum, wurde Continuous Integration erhöht verwendet, ohne DevOps wurde CI nämlich maximal von ca. 65 % und mit DevOps von mindestens ca. 75 % aufwärts verwendet. Dies könnte daran liegen, dass CI generell mit DevOps genutzt wird [69].
Während *Scrum - Kanban - kein DevOps* wurde *Continuous Integration* mit ca. 10 Prozentpunkten überdurchschnittlich genutzt. Dies könnte daran liegen, dass bei der Kombination das *Configuration Management* nicht abgedeckt wird und durch die Praktik hingegen mäßig.
Die meisten Praktiken werden bei *Scrum - Kanban - DevOps* sehr überdurchschnittlich verwendet. In der Anwendung ist die Abdeckung dieser Kombination auch am höchsten. Code Reviews werden bei *Scrum - kein Kanban - kein DevOps* und *Scrum - Kanban - kein DevOps* mit ca. 61 % bei allen Kombinationen mit Scrum am geringsten verwendet. Bei *Scrum - DevOps - kein Kanban* werden diese hingegen mit ca. 81 % am stärksten verwendet, bei dieser Kombination ist die Abdeckung vom *Quality Management* auch nicht vorhanden. Jedoch ist diese auch bei *Scrum - kein Kanban - kein DevOps* nicht vorhanden.
Coding Standards wurden bei allen Kombinationen von ca. 71-75 % und somit ca. 5-10 Prozentpunkte überdurchschnittlich verwendet. Außer bei *Scrum - Kanban - DevOps*, da von ca. 85 %. Bei dieser Kombination werden aber auch die meisten Praktiken überdurchschnittlich verwendet.
Das Backlog Management und die Daily Stand-Up Meetings wurden ebenfalls überdurchschnittlich angewandt, diese könnten also das nicht abgedeckte *Requirements Management* bzw. *Maintenance and Evolution* in Scrum abdecken, aber gehören auch sowieso zu dieser Methode [117]. Die Daily Stand-Up Meetings werden auch dann überdurchschnittlich verwendet, wenn Scrum mit Kanban/DevOps kombiniert wird, welche das *Maintenance and Evolution* hingegen mäßig abdecken, dies könnte wieder bestätigen, dass Scrum in hybriden Ansätzen dominanter oder erstrangig genutzt wird.

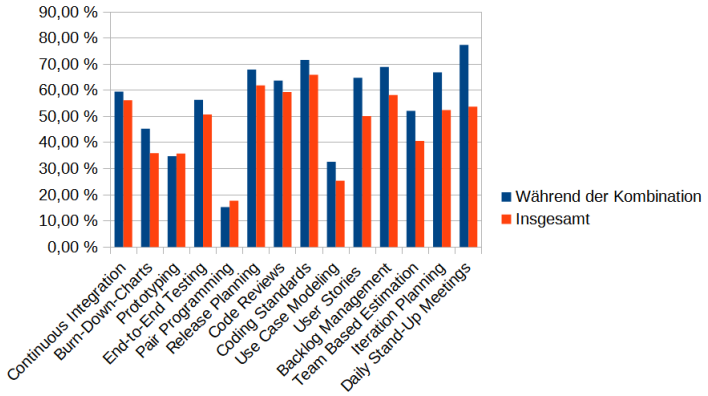


Abbildung 6.2: Zusätzliche Praktiken bei Scrum - kein DevOps - kein Kanban

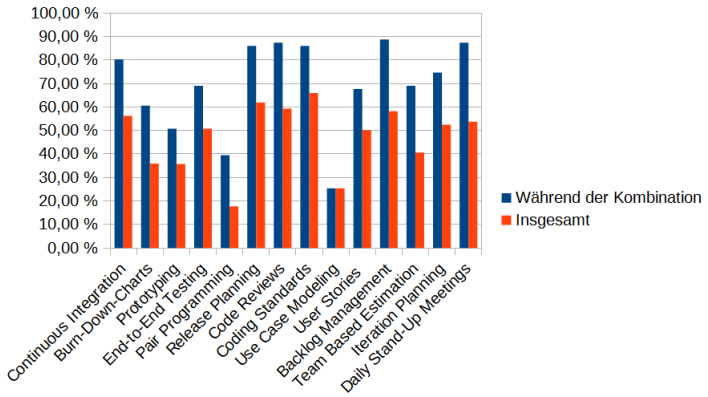


Abbildung 6.3: Zusätzliche Praktiken bei Scrum - Kanban - DevOps

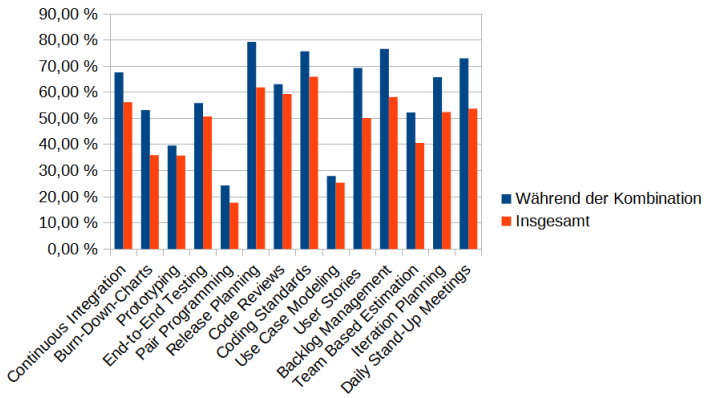


Abbildung 6.4: Zusätzliche Praktiken bei Scrum - Kanban - kein DevOps

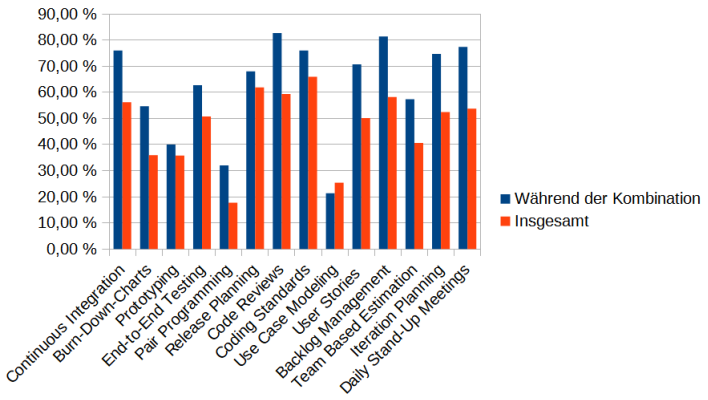


Abbildung 6.5: Zusätzliche Praktiken bei Scrum - DevOps - kein Kanban

Kombinationen ohne Scrum

Bei *Kanban - DevOps - kein Scrum* und *DevOps - kein Scrum - kein Kanban* werden, im Vergleich zu insgesamt allen anderen Kombinationen, die untersuchten Praktiken am unterdurchschnittlichsten genutzt. In der Anwendung deckt *Kanban - DevOps - kein Scrum* hingegen z.B. alle Phasen mindestens mäßig ab. Dies könnte zeigen, dass gerade deshalb keine zusätzlichen Praktiken nötig sind.
Bei *Kanban - kein Scrum - kein DevOps* wurden die Praktiken durchschnittlich verwendet. Diese Kombination deckt in der Anwendung alle Phasen bis auf das *Configuration Management* mindestens mäßig ab.
Bei *DevOps - kein Scrum - kein Kanban* und *Kanban - kein Scrum - kein DevOps* wurden Daily Stand-Up Meetings unterdurchschnittlich oft genutzt, dies kann darauf hindeuten, dass die Kombinationen selbst bereits schon die Phase *Maintenance and Evolution* mäßig abdecken. Während der Kombination *Kanban - kein Scrum - kein DevOps* wird die Phase *Configuration Management* gar nicht abgedeckt. Dennoch wurde festgestellt, dass das *Continuous Integration* nur mit ca. einem Prozent über dem Durchschnitt während dieser Kombination eingesetzt wurde und gerade dieses deckt das *Configuration Management* im Gegensatz zu den anderen Praktiken noch mäßig ab.
Außerdem wurden das *Iteration Planning* und *Backlog Management* leicht unterdurchschnittlich eingesetzt, dies könnte auch daran liegen, dass gerade das *Change Management* durch Kanban als Change-Management Methode abgedeckt wird und deshalb keine weitere Abdeckung durch diese beiden Praktiken notwendig ist oder andererseits diese Praktiken vielmehr ein fester Bestandteil von Scrum sind [117], welches bei diesen Kombinationen ja nicht verwendet wird.

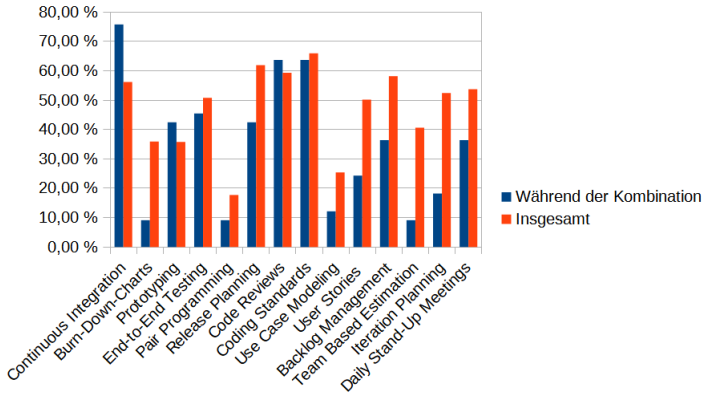


Abbildung 6.6: Zusätzliche Praktiken bei DevOps - kein Scrum - kein Kanban

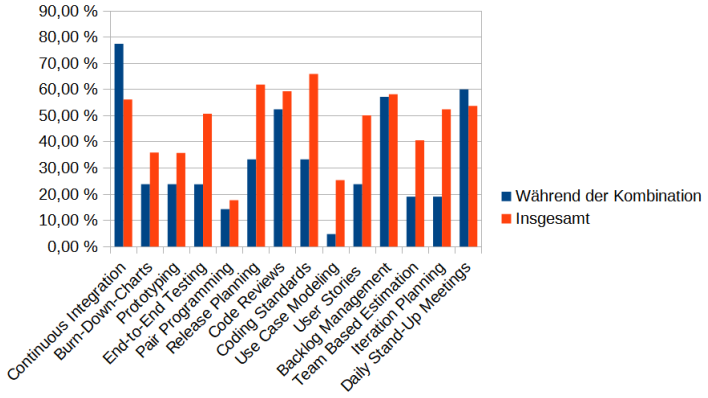


Abbildung 6.7: Zusätzliche Praktiken bei Kanban - DevOps - kein Scrum

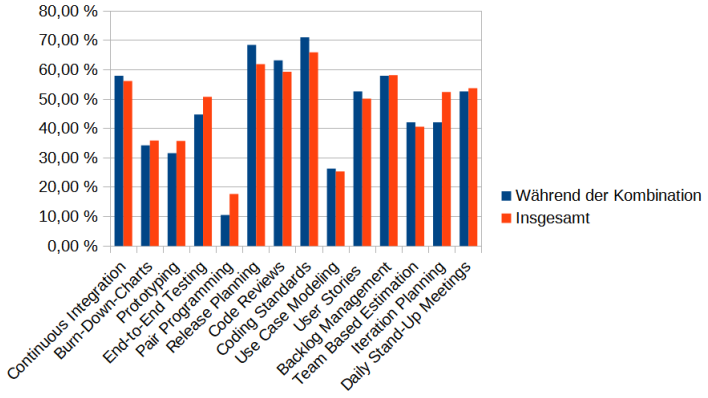


Abbildung 6.8: Zusätzliche Praktiken bei Kanban - kein Scrum - kein DevOps

Kapitel 7

Diskussion

7.1 Zusammenfassung der Ergebnisse

Die Ergebnisse zeigen, dass die Abdeckungen des SW-Lebenszyklus durch die Methoden bzw. Kombinationen von Methoden durchaus verschieden sind und bei einigen Kombinationen von Methoden bestimmte Phasen auch gar nicht abgedeckt werden.
Eine weitere Erkenntnis ist, dass manche Kombinationen, welche im Vergleich zu den anderen Kombinationen wenig Phasen des SW-Lebenszyklus abdecken, hingegen wesentlich häufiger verwendet werden als andere, welche wiederum durchaus mehr Phasen abdecken.
Eine andere Erkenntnis ist, dass bei Kombinationen, bei welchen bestimmte Phasen nicht oder wenig abgedeckt werden, die Praktiken, welche genau diese Phasen stärker abdecken, auch nicht direkt häufiger oder intensiver genutzt wurden um die Abdeckung also auszugleichen.

7.2 Interpretation der Ergebnisse

Einige Kombinationen, welche im Vergleich zu den anderen Kombinationen weniger Phasen abdecken, wurden also dennoch zum Teil häufiger verwendet. Ein Beispiel ist *Scrum - kein Kanban - kein DevOps*, den Ergebnissen nach deckt diese Kombination die wenigsten Phasen ab und wird dennoch im Vergleich zu *Scrum - Kanban - kein DevOps* oder *Scrum - DevOps - kein Kanban* wesentlich häufiger verwendet.
Aber auch umgekehrt wurden bestimmte Kombinationen von Methoden im Verhältnis besonders selten genutzt, welche hingegen, den Ergebnissen nach, die meisten Phasen abdecken. Ein Beispiel ist die Kombination *Kanban - DevOps - kein Scrum*, denn diese wurde, von allen Kombinationen mit nur ca. 2,5 % deutlich am seltensten verwendet.

Hingegen gibt es auch Kombinationen, welche diesen Erwartungen eher entsprechen, z.B. wurde die Kombination *Scrum - Kanban - DevOps*, welche den Ergebnissen nach alle Phasen abdeckt, auch von knapp ca. 10 % verwendet. Die Erwartung bezüglich den Abdeckungen der Kombinationen, mit deren Nutzungen in der Industrie verglichen, bestätigt sich also nur teilweise. Es wurde also nicht festgestellt, dass eine Kombination mit einer sehr hohen Abdeckung auch direkt besonders häufig in der Industrie verwendet wird oder andersherum eine Methode mit einer geringen Abdeckung direkt besonders selten verwendet wird.
Die Ergebnisse der Untersuchung der Nutzung der Praktiken während den Kombinationen, verglichen mit den nicht abgedeckten Phasen dieser Kombinationen, zeigen, dass die Praktiken, welche eigentlich diese nicht abgedeckten Phasen stark abdecken, nicht immer übermäßig, wie erwartet, eingesetzt wurden. Dies könnte daran liegen, dass Praktiken nur als Ergänzung zu den Methoden genutzt werden und nicht um die Abdeckungen von bestimmten Phasen auszugleichen. Dass Praktiken als Ergänzung zu den Methoden genutzt werden, zeigen die Ergebnisse hingegen, beispielsweise wird die Praktik *Continuous Integration* während DevOps übermäßig genutzt und die Praktiken *Iteration Planning*, *Backlog Management* und *Daily Stand-Up Meetings* während Scrum, denn *Continuous Integration* wird mit DevOps generell genutzt und die letzten drei genannten Praktiken gehören per Definition, im Scrum-Guide, zu Scrum [69][117].
Dazu passend ist die Aussage “practices are the (real) bulding blocks of development approches (hybrid methods)”, welche in einigen wissenschaftlichen Arbeiten getroffen wird [66][133]. Diese kann durch unsere Ergebnisse ebenfalls gestützt werden, denn wir haben ja eine Abhängigkeit der Praktiken von den Methoden-Kombinationen festgestellt. Man könnte also argumentieren, dass die Praktiken als Bausteine die Methoden an sich aufbauen und sich z.B. bereits anhand der Nutzung der Praktiken ein Bild einer Methode machen lässt, siehe Diagramme der Evaluation.
Abgesehen davon sind die Nutzungen der Praktiken während allen Kombinationen mit Scrum weitgehend gleich verteilt, dies könnte daran liegen, dass auch wenn Scrum in einem hybriden Ansatz verwendet wird, dennoch “dominanter” genutzt wird und die anderen Methoden “verdrängt” werden. Wenn also z.B. Scrum und Kanban kombiniert werden, könnte es sein, dass dennoch Scrum im Vordergrund genutzt wird und Kanban nur teilweise im Hintergrund.

7.3 Beschränkungen

Bei den Untersuchungen und den anschließenden Zuordnungen der drei Stufen der Abdeckungen zu den Phasen des SW-Lebenszyklus für jede der Methoden und Praktiken gab es einige Beschränkungen und Limitierungen.

Eine dieser Beschränkungen ist die Tatsache, dass zu nicht abgedeckten Phasen von einer Methode wenig Literatur und wissenschaftliche Untersuchungen vorliegen, welche dies bestätigen. Es war nicht immer möglich, Nachweise zu finden, welche also eine fehlende Abdeckung durch eine Methode oder Praktik bestätigen.
Außerdem wurde die Kombinierbarkeit der Methoden zu einem hybriden Ansatz bei dieser Arbeit nicht berücksichtigt, es wurde angenommen, dass alle Methoden genau gleich miteinander kombiniert werden können. Dabei kann es sein, dass z.B. Kanban mit DevOps weniger passend zusammen genutzt werden kann, wie z.B. Scrum und Kanban. Auf die resultierende Abdeckung hatte dies in der Arbeit jedoch keine Auswirkung. Es wurde also nicht die Kombinierbarkeit der Methoden und Praktiken selbst untersucht und beachtet.
Außerdem wurden die Stufen der Abdeckungen nicht addiert, sondern nur ergänzt. Wenn also zwei Methoden eine Phase mäßig abdecken, dann wird die Phase immer noch mäßig abgedeckt. Hingegen könnte es auch sein, dass mehrere mäßige Abdeckungen zu einer starken Abdeckung führen könnten, wenn z.B. durch eine Methode ein Bereich der Phase stärker abgedeckt wird und ein anderer von der anderen Methode. Dies wird hier jedoch nicht weiter betrachtet, da solche Untersuchungen den Rahmen dieser Arbeit deutlich überschreiten würden.
Abgesehen davon wurde das Maß der Abdeckung nur auf drei Stufen begrenzt, aber theoretisch könnte man die Abdeckungen durch die Methoden und Praktiken auch näher untersuchen und mit vier Stufen bewerten.
Weitere Beschränkungen gab es auch bei der Evaluation. Zum einen wurden nur die Praktiken neben der Nutzung der Kombinationen untersucht, welche die weniger abgedeckten Phasen, insgesamt von allen untersuchten Kombinationen, stärker abdecken. Es wurden also nicht alle Praktiken, welche aber in der HELENA-Studie mit untersucht wurden, ausgewählt.
Es könnte jedoch sein, dass sich bei der Untersuchung aller gesamten Praktiken ein anderes Bild über die Nutzung dieser Praktiken ergibt.
Zum anderen wurden die Daten bei der Auswertung nur nach einem Verfahren untersucht, denn in den Daten der HELENA-Studie wurden für die Methoden und Praktiken jeweils die Werte 1-7 oder -9 angegeben. Dabei repräsentiert jeder Wert wie häufig die Methode genutzt wurde, ob diese überhaupt bekannt ist oder bekannt ist, dass diese Methode oder Praktik genutzt wurde. Bei der Untersuchung wurde eine Methode oder Praktik entweder als genutzt oder nicht genutzt bewertet, es gab also nichts dazwischen (wie z.B. bei der Abdeckung der Phasen). Dabei standen ausschließlich die Werte 6 (oft genutzt) und 7 (immer genutzt) für genutzt und ansonsten, für alle anderen Werte, galt eine Methode oder Praktik als nicht genutzt. Dabei stehen die Werte 4 und 5 für eine seltene bzw. gelegentliche Nutzung und somit könnte man diese also auch als eine mäßige Nutzung werten.

Kapitel 8

Verwandte Arbeiten

Das Paper "Towards Shaping the Software Lifecycle with Methods and Practices" stellt die Ergebnisse einer Umfrage vor, welche mit insgesamt 27 Teilnehmern (Firmen) aus der Branchen Software-Entwicklung, IT-Consulting und Training and Services durchgeführt wurde [39]. Bei diesen hat es sich sowohl um große, mittelgroße sowie kleine Unternehmen gehandelt. In dieser wurden die Teilnehmer für jede Projekt-Disziplin gefragt, welche Methoden und Praktiken diese für diese verwenden [65].

In der HELENA-Studie wurde untersucht, welche Methoden und Praktiken von den Unternehmen genutzt und miteinander kombiniert werden. Die Studie besteht aus insgesamt drei Stages, wobei während der Stage 2 die Daten von ca. 1500 teilnehmenden Unternehmen gesammelt wurden. Zu diesen zählt der Nutzungsgrad von 24 Methoden und 35 Praktiken. [70][71]. Diese Daten werden auch im Rahmen dieser Arbeit während der Evaluation verwendet, um die Abdeckung des SW-Lebenszyklus in der Anwendung von den ausgewählten Kombinationen von Methoden und Praktiken mit dem Nutzen in der Industrie zu vergleichen.

In dem Paper "What are Hybrid Development Methods Made Of? An Evidence-based Characterization" werden hybride Entwicklungsprozesse untersucht. Es liegt bei der Untersuchung der Fokus darauf, welche Methoden und Praktiken in einem hybriden Entwicklungsansatz wie oft miteinander kombiniert werden. Das Paper basiert dabei auf den Daten der Stage 2 der HELENA-Studie. Es wurde z.B. festgestellt, dass bestimmte Praktiken (Code Reviews, Coding Standards und Release Planning) am häufigsten und akzeptiertesten neben den verschiedenen hybriden Kombinationen von Methoden (z.B. Scrum und Kanban) genutzt werden [134].

In der Bachelorarbeit "Umfragestudie zur Nutzung von Methoden und Praktiken in unterschiedlichen Softwareprojektdisziplinen" werden die Ergebnisse

einer durchgeführten Umfrage zu der Anwendung von den Methoden und Praktiken in den unterschiedlichen Disziplinen des Software-Projekts vorgestellt und diskutiert [39]. Dabei wird auch Bezug zu dem Paper "Determining Context Factors for Hybrid Development Methods with Trained Models" genommen, in welchem untersucht wird, welche Faktoren die Kombination mehrerer Methoden und Praktiken zu einer hybriden Methode in der Software-Entwicklung beeinflussen [67].

In der Studie "15th Annual State Of Agile Report" wird untersucht, welche Unternehmen in der Softwareentwicklung-Branche welche (agilen) Methoden und Praktiken nutzen und welche Ursachen dafür in Frage kommen könnten. Aus dieser geht z.B. hervor, dass nur sechs Prozent der Unternehmen Kanban hauptsächlich bzw. am ehesten auf der Team-Ebene einsetzen, was im Verglich zu Scrum relativ gering ist, denn diese Methode nutzen 66 Prozent der Unternehmen. Zudem wurde festgestellt, dass 87 Prozent der Unternehmen bei den agilen Techniken/Praktiken Daily Stand-Ups nutzen und 83 Prozent Retrospektiven. Bei den agilen Planungs- und Auslieferungstechniken waren das Kanban-Board mit 77 Prozent und das Task-Board mit 67 Prozent die am häufigsten genutzten Tools [8].

In dem Paper "Software Development Life Cycle AGILE vs Traditional Approaches" werden die Unterschiede sowie Vor- und Nachteile zwischen dem traditionellen und agilen Ansatz in der Software-Entwicklung in Bezug auf den Software-Lebenszyklus untersucht und dargelegt. Zusätzlich wird aufgezeigt, wie die agile Entwicklung, hinsichtlich der Eignung für das Projekt-Management, verbessert werden kann [77].

In dem Paper "Agile Processes and Methodologies: A Conceptual Study" werden die Vorteile sowie vor allem aber auch Nachteile der agilen Entwicklung in der Software-Entwicklung dargelegt und der agile Ansatz mit anderen Modellen für den Software-Lebenszyklus hinsichtlich bestimmter Aspekte (wie z.B. der Testphase oder der Flexibilität) verglichen. Diese Modelle sind das Spiral- sowie das RAD-Modell. Es wird z.B. dargelegt, dass das Spiral-Modell, im Gegensatz zu dem agilen Prozess oder dem RAD-Modell, den Bereich Risk Analysis abdeckt [123].

In dem Paper "Agile practices in practice: a mapping study" wird entsprechend den Ergebnissen einer durchgeführten Studie dargelegt, welche agilen Praktiken in der Industrie für welche Bereiche (Domains) oder Projekt-Typen genutzt werden und vor allem auch mit welcher Intensität und Häufigkeit. Es wurde festgestellt, dass bestimmte Praktiken (z.B. Time Boxing) stets häufiger genutzt werden als andere und auch die Domain einen Einfluss auf die Nutzung von agilen Praktiken hat, so wurden z.B. die meisten agilen Praktiken in der Telekommunikation verwendet (von den ausgemachten Domains). Die wenigsten hingegen eher z.B. in der Medizin [42].

Kapitel 9

Zusammenfassung und Ausblick

In diesem Kapitel soll ein Überblick über die Inhalte und Ergebnisse dieser Arbeit gegeben werden. In Abschnitt 9.1 werden die Inhalte und Ergebnisse zusammengefasst und in Abschnitt 9.2 werden mögliche Erweiterungen der Arbeit und Anwendung beschrieben.

9.1 Zusammenfassung

In der Praxis werden vermehrt mehrere Methoden und Praktiken miteinander kombiniert, wenn diese geeignet erscheinen und dann zusammen als hybrider Prozess für die Software-Entwicklung genutzt.
Jedoch werden die Methoden und Praktiken nicht immer so eingesetzt wie man diese auch einsetzen sollte. Dies kann dann negative Auswirkungen auf die Abdeckung vom Software-Lebenszyklus haben, welcher ja eigentlich möglichst vollständig abgedeckt sein sollte.
In dieser Arbeit wurde die Abdeckung des Software-Lebenszyklus durch ausgewählte Methoden und Praktiken untersucht und für jede Methode und Praktik jeder Phase des SW-Lebenszyklus eine Stufe der Abdeckung zugeordnet. Insgesamt wurden drei Stufen der Abdeckung (stark, mäßig, keine) festgelegt.
Für die Auswahl der zu untersuchenden Methoden und Praktiken und die Einstufung der Abdeckungen der Phasen des SW-Lebenszyklus durch die Methoden und Praktiken war eine innerhalb der Arbeit "Umfragestudie zur Nutzung von Methoden und Praktiken in unterschiedlichen Softwareprojektdisziplinen" durchgeführte Umfrage zu der Nutzung von den bekanntesten Methoden und Praktiken für die Software-Entwicklung, maßgebend. In dieser wurden IT-Unternehmen befragt, welche Methoden und Praktiken diese

nutzen und für welche Phasen des SW-Lebenszyklus sie eingesetzt werden. Zusätzlich wurde für die Einstufung der Abdeckungen noch andere Studien und Literatur hinzugezogen.
Mittels der getroffenen Zuordnungen der Abdeckungen zu den Phasen, für die ausgewählten Methoden und Praktiken, wurde dann eine Anwendung entwickelt. Diese bietet dem Nutzer die Funktion, eine bestimmte Kombination von diesen Methoden und Praktiken auszuwählen und dann die gesamte Abdeckung des SW-Lebenszyklus zu berechnen und grafisch in einem Diagramm darzustellen. Außerdem wurde in der Anwendung anschließend die Funktion integriert, dass nach der Auswahl einer Kombination von Methoden und Praktiken und der anschließenden Ausgabe der Abdeckung, weitere bestimmte Methoden oder Praktiken berechnet und aufgelistet werden, welche die aktuelle Abdeckung des SW-Lebenszyklus vervollständigen würden, wenn diese also mit der aktuellen Auswahl von Methoden und Praktiken kombiniert werden würden.
Abschließend wurden die von der Anwendung berechneten Abdeckungen der Kombinationen mit der Nutzung dieser in der Industrie verglichen.

9.2 Ausblick

Die Arbeit kann in einige Richtungen erweitert werden. Einerseits können weitere Methoden und Praktiken und deren Abdeckungen des SW-Lebenszyklus untersucht und dann zu der Anwendung hinzugefügt werden. Andererseits gibt es bei einigen Kombinationen noch Differenzen zwischen der Abdeckung in der Anwendung und der Nutzung der Praktiken dieser Kombinationen in der Industrie. Zum Beispiel werden bei einigen Kombinationen von Methoden bestimmte Phasen nicht abgedeckt. In der Industrie werden dahingegen während diesen Kombinationen auch die Praktiken nicht direkt vermehrt und häufiger genutzt, welche genau diese Phasen stärker abdecken könnten, obwohl ja gerade dies zu erwarten wäre.
Andererseits wurde in der Arbeit nicht berücksichtigt, inwieweit die untersuchten Methoden selbst überhaupt miteinander kombiniert werden können. Also inwieweit Kanban zum Beispiel mit DevOps zusammen als hybrider Entwicklungsansatz verwendet werden kann oder nicht. In der Arbeit wurde nur davon ausgegangen, dass diese genauso wie alle anderen miteinander zusammenpassen und als Kombinationen genau gleich miteinander genutzt werden können und es dahingehend keine Unterschiede gibt. Die Kombination *Kanban - DevOps - kein Scrum* zum Beispiel, deckt neben *Kanban - DevOps - Scrum* in der Anwendung den SW-Lebenszyklus am vollständigsten ab, wird jedoch mit einem Anteil von nur circa 2,96 % am seltensten verwendet. Dahingehend könnte untersucht werden, inwiefern die beiden Methoden Kanban und DevOps zum Beispiel überhaupt miteinander in einem hybriden Ansatz kombiniert werden können.

Anhang A

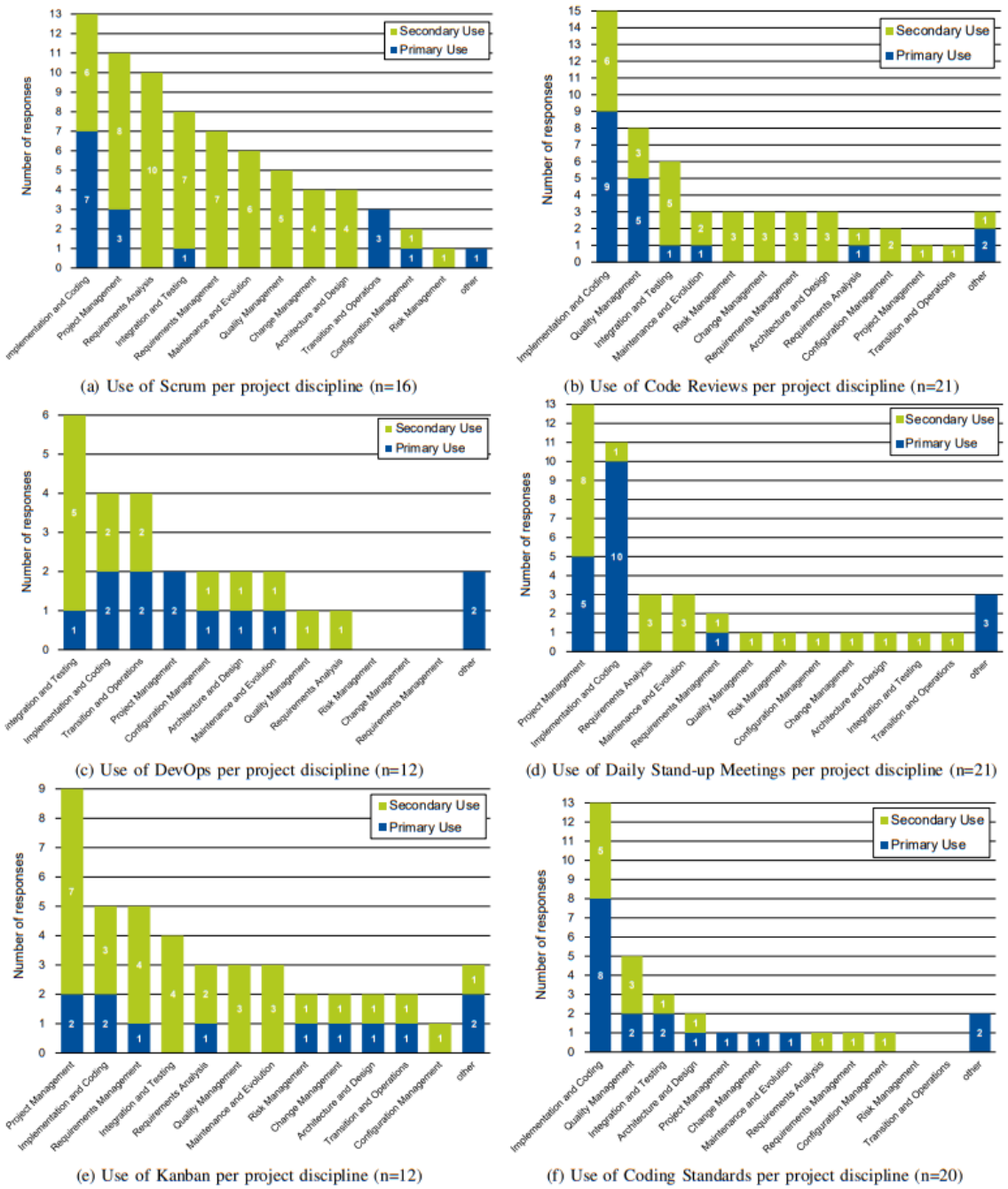
Anhang

A.1 Vollständiges Diagramm der Nutzung aller Methoden/Praktiken für alle Phasen mit Rangfolgen bewertet

	Methode/ Praktik	n	#andere/ keine Angabe	Project Management	Quality Management	Risk Management	Configuration Management	Change Management	Requirements Analysis	Requirements Management	Architecture and Design	Implementation/Coding	Integration and Testing	Transition and Operation	Maintenance and Evolution
Methoden	DevOps	12	2	4.	6.		5.		7.		5.	2.	**1.**	3.	5.
	Iterative Development	11	1	7.	8.	11.	10.	6.	4.	5.	3.	**1.**	2.	12.	9.
	Kanban	13	2	**1.**	6.	11.	12.	10.	5.	3.	8.	2.	4.	7.	9.
	Scrum	17	1	2.	6.	12.	11.	9.	3.	5.	8.	**1.**	4.	10.	7.
	Wasserfallmodell	12	1	**1.**	5.	10.	8.	6	3.	9.	4.	2.	7.	12.	11.
Praktiken	Automated Unit Testing	16	1	4.	3.			5.			6.	**1.**	2.		
	Backlog Management	15	1	**1.**	6.	5.	10.	3.	4.	2.	8.	3.	6.		9.
	Burndown Charts	15	2	**1.**	5.	3.			7.	8.		2.	4.	6.	
	Code Reviews	18	1	10	2.	5.	7.	5.	6.	5.	4.	**1.**	3.	8.	4.
	Coding Standards	17	1	5.	2.		8.	5.	7.	6.	4.	**1.**	3.		5.
	Continuous Integration	16	1	5.	3.	4.	7.			6.	6.	**1.**	2.	5.	6.
	Daily Standup	19	2	**1.**	8.	10.	8.	9.	4.	6.	5.	2.	6.	7.	3.
	Definition of Done/Ready	14	1	4.	3.	9.	11.	6.	7.	5.	8.	**1.**	2.	10.	
	End-to-End Testing	11	1	4.	2.	5.		5.				3.	**1.**	4.	
	Team/expert based estimation	11	1	3.	6.	5.	8.	9.	3.	2.	7.	**1.**	4.	11.	10.
	Iteration Planning	16	1	**1.**	4.	5.	9.	3.	10.	8.	7.	2.	6.	11.	
	Iteration/Sprint Reviews	13	1	2.	3.	6.		10.	9.	7.	4.	**1.**	5.	8.	
	Limit Work-in-Progress	10	1	2.		7.	8.	9.	10.	5.	6.	**1.**	4.		3.
	Pair Programming	10	1	4.	2.			4.	3.		3.	**1.**	2.		
	Prototyping	12	1	2.	4.	4.			2.	3.	2.	**1.**	5.		
	Refactoring	13	1	6.	3.		7.				2.	**1.**	5.		4.
	Release Planning	16	1	**1.**	2.	9.	10.	8.	11.	7.	12.	4.	5.	3.	6.
	Retrospectives	15	2	**1.**	3.	7.	10.	6.	9.	11.	4.	2.	5.	8.	9.
	Use Case Modeling	10	2	3.	7.			6.	**1.**	2.	4.	5.			
	User Stories	16	2	3.	4.		10.	7.	3.	**1.**	6.	2.	5.	8.	9.

Abbildung A.1: Diagramm für die Nutzung aller Methoden und Praktiken für alle Phasen mit Rangfolgen bewertet [39]

A.2 Vollständiges Diagramm der Nutzung der drei häufigsten Methoden und Praktiken



(a) Use of Scrum per project discipline (n=16)

(b) Use of Code Reviews per project discipline (n=21)

(c) Use of DevOps per project discipline (n=12)

(d) Use of Daily Stand-up Meetings per project discipline (n=21)

(e) Use of Kanban per project discipline (n=12)

(f) Use of Coding Standards per project discipline (n=20)

Abbildung A.2: Diagramm für die Nutzung der drei häufigsten Methoden und Praktiken für alle Phasen des SW-Lebenszyklus [65]

A.3 Vollständiges Diagramm der Nutzung aller Methoden und Praktiken für die vier häufigsten Phasen

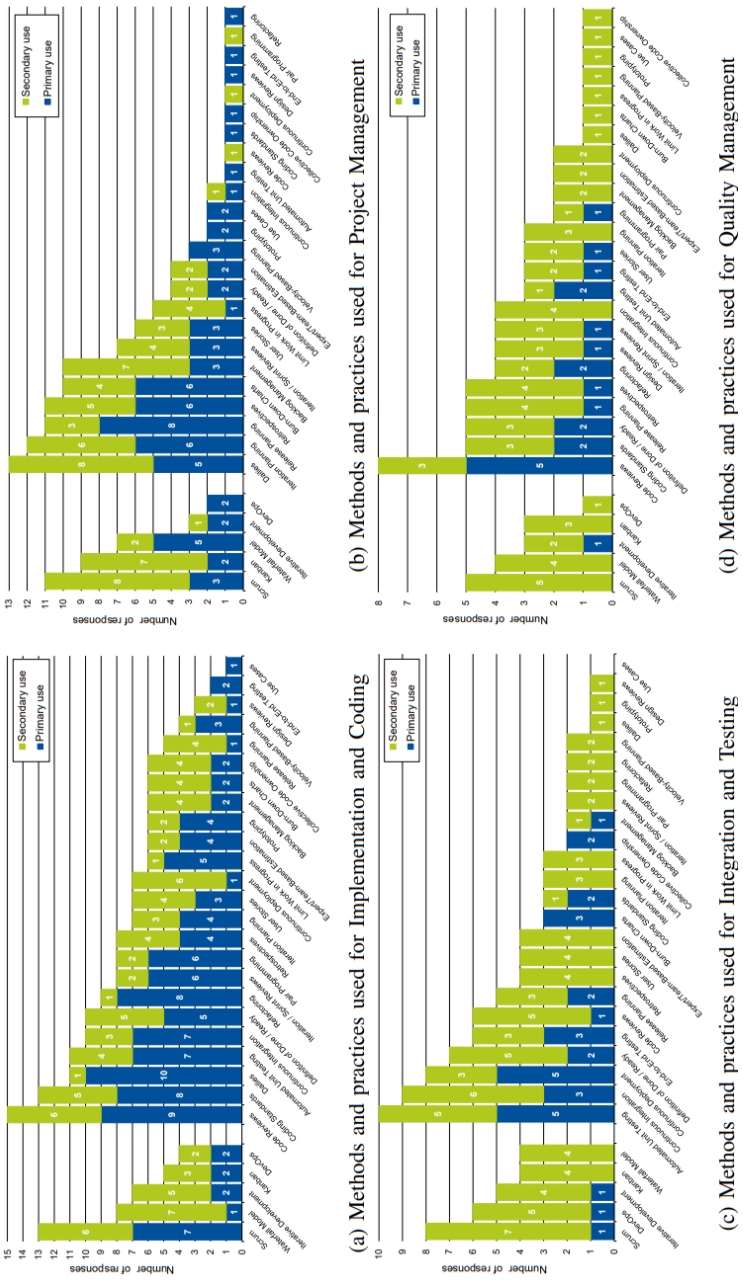


Abbildung A.3: Diagramm für die Nutzung aller Methoden und Praktiken für die vier häufigsten Phasen des SW-Lebenszyklus [65]

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] Agile Release Planning Methods & Tools. *inflectra*. Zuletzt abgerufen am 01.07.2022: https://www.inflectra.com/SpiraTeam/Highlights/Release-Planning.aspx.

[2] Intellij IDEA. https://www.jetbrains.com/idea/.

[3] Launch4j. http://launch4j.sourceforge.net/.

[4] Maven. https://maven.apache.org.

[5] Overview (JavaFX 8). https://docs.oracle.com/javase/8/javafx/api/index.html. Zuletzt abgerufen am: 2022-05-14.

[6] Iteration Planning Guide. *Rally Software Development*, 2013. Zuletzt abgerufen am 29.05.2022: https://www.projectmanagement.com/pdf/IterationPlanningGuide.pdf.

[7] The State of Scrum: Benchmarks and Guidelines. *Scrum Alliance*, 2013.

[8] 15th Annual State Of Agile Report. Zuletzt abgerufen am 15.05.2022: https://digital.ai/resource-center/analyst-reports/state-of-agile-report, 2021.

[9] THE OFFICIAL GUIDE TO THE KANBAN METHOD. Zuletzt abgerufen am 26.05.2022: https://kanban.university/kanban-guide/, 2021.

[10] How to close the gap between DevOps and Change Management. *Plat4mation*, 2022. Zuletzt abgerufen am 12.06.2022: https://plat4mation.com/blog/how-to-close-the-gap-between-devops-and-change-management/.

[11] S. AGILE. Iteration Planning. Zuletzt abgerufen am 29.05.2022: https://www.scaledagileframework.com/iteration-planning/, 2021.

[12] M. O. Ahmad, P. Kuvaja, M. Oivo, and J. Markkula. Transition of Software Maintenance Teams from Scrum to Kanban. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 5427–5436, 2016.

[13] M. O. Ahmad, J. Markkula, and M. Oivo. Kanban in software development: A systematic literature review. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 9–16, 2013.

[14] A. M. Alsalemi and E.-T. Yeoh. A survey on product backlog change management and requirement traceability in agile (Scrum). In *2015 9th Malaysian Software Engineering Conference (MySEC)*, pages 189–194, 2015.

[15] D. Ameller, C. Farré, X. Franch, D. Valerio, and A. Cassarino. Towards continuous software release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 402–406, 2017.

[16] D. J. Anderson. *Kanban: Evolutionäres Change Management für IT-Organisationen*. dpunkt.verlag, 2011.

[17] S. Ashraf and S. Aftab. IScrum: An Improved Scrum Process Model. *International Journal of Modern Education & Computer Science*, 9(8), 2017.

[18] Z. Azham, I. Ghani, and N. Ithnin. Security backlog in Scrum security practices. In *2011 Malaysian Conference in Software Engineering*, pages 414–417, 2011.

[19] Z. Babar, A. Lapouchnian, and E. Yu. *Modeling DevOps Deployment Choices Using Process Architecture Design Dimensions*. 11 2015.

[20] H. Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Spektrum Akademischer Verlag, Heidelberg, 2011.

[21] L. Barolli, F. Xhafa, N. Javaid, and T. Enokido. In *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2018)*, Advances in Intelligent Systems and Computing, page 565. Springer International Publishing, 2018.

[22] L. Barolli, F. Xhafa, N. Javaid, and T. Enokido. Innovative mobile and internet services in ubiquitous computing: Proceedings of the 12th international conference on innovative mobile and internet services in ubiquitous computing (imis-2018). Advances in Intelligent Systems and Computing, page 566. Springer International Publishing, 2018.

[23] Barraood, Samera Obaid and Mohd, Haslina and Baharom, Fauziah. A Comparison Study of Software Testing Activities in Agile Methods. In *Knowledge Management International Conference (KMICe) 2021*, 2021.

[24] A. Bergström. *Software Configuration Management in Scrum Projects*. Department of Computer Science, Lund University, 2008.

[25] J. Bloomberg. DevOps Versus ITIL: How to Win the Battle Over Change Management. *Techstrong Group, Inc.*, 2020. Zuletzt abgerufen am 12.06.2022: `https://devops.com/devops-versus-itil-how-to-win-the-battle-over-change-management/`.

[26] K. Buffardi. *Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories*, page 650–656. Association for Computing Machinery, New York, NY, USA, 2020.

[27] L. Bulej, T. Bureš, V. Horkỳ, J. Kotrč, L. Marek, T. Trojánek, and P. Tma. Unit testing performance with stochastic performance logic. *Automated Software Engineering*, 24(1):139–187, 2017.

[28] E. Caballero, J. A. Calvo-Manzano, and T. San Feliu. Introducing scrum in a very small enterprise: A productivity and quality analysis. In *European Conference on Software Process Improvement*, pages 215–224. Springer, 2011.

[29] Cardozo, Eliza SF and Araújo Neto, J Benito F and Barza, Alexandre and França, A César C and da Silva, Fabio QB. SCRUM and productivity in software projects: a systematic literature review. In *14th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–4, 2010.

[30] H. F. Cervone. Understanding agile project management methods using Scrum. *OCLC Systems & Services: International digital library perspectives*, 2011.

[31] J. Chan, R. Kwan, and E. Wong. *Quality Management: A New Era*. World Scientific Publishing Company, 2005.

[32] D. J. Chaudhuri and A. Chaudhuri. Agile burndown chart deviation-predictive analysis to improve iteration planning. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 1. Citeseer, 2011.

[33] Y.-h. Chen, Y.-h. Cao, F.-s. Qiu, and G.-d. Wang. Iteration planning of product concurrent design process based on activity item. In *2011 IEEE 18th International Conference on Industrial Engineering and Engineering Management*, volume Part 1, pages 543–546, 2011.

[34] R. Colomo Palacios, P. Soto Acosta, F. J. García Peñalvo, Á. García Crespo, et al. A study of the impact of global software development in packaged software release planning. 2012.

[35] E. Daka and G. Fraser. A Survey on Unit Testing Practices and Problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 201–211, 2014.

[36] F. Dalpiaz and S. Brinkkemper. Agile Requirements Engineering with User Stories. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 506–507, 2018.

[37] David J. Anderson, Andy Carmichael. *Die Essenz von Kanban.* kompakt. dpunkt.verlag, 2018.

[38] A. Debbiche, M. Dienér, and R. Berntsson Svensson. Challenges when adopting continuous integration: A case study. In *International Conference on Product-Focused Software Process Improvement*, pages 17–32. Springer, 2014.

[39] N. Dehn. *Umfragestudie zur Nutzung von Methoden und Praktiken in unterschiedlichen Softwareprojektdisziplinen.* Institut für Praktische Informatik der Leibniz Universität Hannover, 2020. Zuletzt abgerufen am 15.05.2022: `https://www.pi.uni-hannover.de/fileadmin/pi/se/Stud-Arbeiten/2020/Dehn2020.pdf`.

[40] M. Delen, F. Dalpiaz, and K. Cooper. Bakere: A serious educational game on the specification and analysis of user stories. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 369–374, 2019.

[41] P. Devanbu, T. Zimmermann, and C. Bird. Belief amp; Evidence in Empirical Software Engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 108–119, 2016.

[42] P. Diebold and M. Dahlem. Agile practices in practice: A mapping study. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[43] S. Dimitrijević, J. Jovanović, and V. Devedžić. A comparative study of software tools for user story management. *Information and Software Technology*, 57:352–368, 2015.

[44] R. Dumke. *Software Engineering Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools.* Friedr. Vieweg und Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 2013.

[45] P. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Pearson Education, 2007.

[46] O. Díaz and M. Muñoz. Reinforcing DevOps approach with security and risk management: An experience of implementing it in a data center of a mexican organization. In *2017 6th International Conference on Software Process Improvement (CIMPS)*, pages 1–7, 2017.

[47] Eckhart Hanser. *Agile Prozesse: Von XP über Scrum bis MAP*. Springer Berlin Heidelberg, 2010.

[48] F. Elberzhager, T. Arif, M. Naab, I. Süß, and S. Koban. From Agile Development to DevOps: Going Towards Faster Releases at High Quality – Experiences from an Industrial Context. In D. Winkler, S. Biffl, and J. Bergsmann, editors, *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies*, pages 33–44, Cham, 2017. Springer International Publishing.

[49] T. Epping. *Kanban für die Softwareentwicklung (Informatik im Fokus)*. Informatik im Fokus. Springer Heidelberg Dordrecht London New York, 2011.

[50] X. Fang. Using a coding standard to improve program quality. In *Proceedings Second Asia-Pacific Conference on Quality Software*, pages 73–78, 2001.

[51] M. Foegen. *DER ULTIMATIVE SCRUM GUIDE 2.0*. wibas, 2014.

[52] E. Freeman. *DevOps für Dummies*. Wiley, 2020.

[53] F. Hermans. *The Programmer's Brain: What every programmer needs to know about cognition*. Manning, 2021.

[54] D. Huchthausen. SCRUM VS. KANBAN – HOW TO SELECT THE BEST AGILE METHODOLOGY FOR YOU. `https://cloudogu.com/en/blog/scrum-vs-kanban_en`. Zuletzt abgerufen am: 2022-05-13.

[55] L. B. Hvatum and R. Wirfs-Brock. Patterns to Build the Magic Backlog. In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, EuroPLoP '15, New York, NY, USA, 2015. Association for Computing Machinery.

[56] M. Hüttermann. *DevOps for Developers*. Apress, 09 2012.

[57] M. Ikonen, E. Pirinen, F. Fagerholm, P. Kettunen, and P. Abrahamsson. On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 305–314, 2011.

[58] Indeed Editorial Team. What Is a Product Backlog? (With Definition and Guide). Zuletzt abgerufen am 29.05.2022: `https://www.indeed.com/career-advice/career-development/product-backlog`, 2021.

[59] S. Kaltenecker and K. Leopold. *Kanban in der IT - Eine Kultur der kontinuierlichen Verbesserung schaffen.* Carl Hanser Verlag, München, 2018.

[60] I. Karamitsos, S. Albarhami, and C. Apostolopoulos. Applying DevOps practices of continuous automation for machine learning. *Information*, 11(7):363, 2020.

[61] N. Kerzazi and P. N. Robillard. Kanbanize the release engineering process. In *2013 1st International Workshop on Release Engineering (RELENG)*, pages 9–12, 2013.

[62] T. Khalane and M. Tanner. Software quality assurance in Scrum: The need for concrete guidance on SQA strategies in meeting user expectations. In *2013 International Conference on Adaptive Science and Technology*, pages 1–6, 2013.

[63] Kim, Humble, Debois. *Das DevOps Handbuch - Teams, Tools und Infrastrukturen erfolgreich umgestalten.* O'Reilly Media, Inc., 2017.

[64] P. Klipp. Getting Started with Kanban. *Amazon Digital Services*, 2014.

[65] J. Klünder, M. Busch, N. Dehn, and O. Karras. Towards Shaping the Software Lifecycle with Methods and Practices. In *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, pages 1–11. IEEE, 2021.

[66] J. Klünder, D. Karajic, P. Tell, O. Karras, C. Münkel, J. Münch, S. G. MacDonell, R. Hebig, and M. Kuhrmann. Determining Context Factors for Hybrid Development Methods with Trained Models. In *Proceedings of the International Conference on Software and System Processes*, ICSSP '20, page 61–70, New York, NY, USA, 2020. Association for Computing Machinery.

[67] J. Klünder, D. Karajic, P. Tell, O. Karras, C. Münkel, J. Münch, S. MacDonell, R. Hebig, and M. Kuhrmann. Determining context factors for hybrid development methods with trained models. In A. Koziolek, I. Schaefer, and C. Seidl, editors, *Software Engineering 2021*, pages 65–66, Bonn, 2021. Gesellschaft für Informatik e.V.

[68] O. Kolisnichenko, M. Kolomytsev, and S. Nosok. Software security risk management in DEVOPS methodology. *Theoretical and Applied Cybersecurity*, 3(1), 2021.

[69] M. Krief. *Learning DevOps: The complete guide to accelerate collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps*. Packt Publishing, 2019.

[70] M. Kuhrmann, P. Tell, J. Klünder, R. Hebig, S. Licorish, and S. MacDonell. HELENA Stage 2 Results. Zuletzt abgerufen am 29.05.2022: `https://www.researchgate.net/publication/329246439_HELENA_Stage_2_Results`, 11 2018.

[71] M. Kuhrmann, P. Tell, J. Klünder, R. Hebig, and S. MacDonell. Complementing Materials for the HELENA Study (Stage 2), 11 2018.

[72] R. Kumar, S. K. Pandey, and S. I. Ahson. Security in Coding Phase of SDLC. In *2007 Third International Conference on Wireless Communication and Sensor Networks*, pages 118–120, 2007.

[73] Köykkä, Kaiser. The role of change management in DevOps. unpublished thesis, `https://lutpub.lut.fi/handle/10024/156626`, 2018.

[74] S.-T. Lai, H. Susanto, and F.-Y. Leu. Project Management Mechanism Based on Burndown Chart to Reduce the Risk of Software Project Failure. In L. Barolli, editor, *Advances on Broad-Band Wireless Computing, Communication and Applications*, pages 197–205, Cham, 2022. Springer International Publishing.

[75] M. Lal. *Knowledge Driven Development: Bridging Waterfall and Agile Methodologies*. Cambridge IISc Series. Cambridge University Press, 2018.

[76] P. Laplante. *What Every Engineer Should Know about Software Engineering*. What Every Engineer Should Know. CRC Press, 2007.

[77] Y. B. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan. Software development life cycle AGILE vs traditional approaches. In *International Conference on Information and Network Technology*, volume 37, pages 162–167, 2012.

[78] D. Leffingwell. Mastering the iteration: An agile white paper. *Rally Software Development Corporation*, 2007.

[79] Z. Li, W. Sun, Z. B. Jiang, and X. Zhang. BPEL4WS unit testing: framework and implementation. In *IEEE International Conference on Web Services (ICWS'05)*, pages 103–110 vol.1, 2005.

[80] T. Linz. In *Testing in Scrum: A Guide for Software Quality Assurance in the Agile World*, pages 864–869. Rocky Nook, Inc., 2014.

[81] G. Lucassen, F. Dalpiaz, J. M. Van der Werf, and S. Brinkkemper. The Use and Effectiveness of User Stories in Practice. In *Requirements*

Engineering: Foundation for Software Quality 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016, Proceedings, pages 205–222, 03 2016.

[82] W. Mahmood, N. Usmani, M. Ali, and S. Farooqui. Benefits to Organizations after migrating to Scrum. In *29th International Business Information Management Association Conference*, 2017.

[83] V. Mahnic. Kanban in software engineering education: An outline of the literature. *World Transactions on Engineering and Technology Education*, 17:23–28, 01 2019.

[84] D. Meedeniya, I. Rubasinghe, and I. Perera. Software artefacts consistency management towards continuous integration: a roadmap. *International Journal of Advanced Computer Science and Applications*, 10(4):100–110, 2019.

[85] C. Meister. *DevOps - Erfolgreich Entwicklung und IT-Betrieb verbinden: Grundlagen und Werkzeuge für eine erfolgreiche DevOps-Implementierung.* Books on Demand, 2021.

[86] E. Miranda and P. Bourque. Agile monitoring using the line of balance. *Journal of Systems and Software*, 83(7):1205–1215, 2010. SPLC 2008.

[87] C. Mobile. DevOps Change Management In The Enterprise World. Zuletzt abgerufen am 12.06.2022: `https://clearbridgemobile.com/devops-change-management-in-the-enterprise-world/`, 2020.

[88] J. Mulder. *Enterprise DevOps for Architects: Leverage AIOps and DevSecOps for secure digital transformation.* Packt Publishing, 2021.

[89] Y. Muto, K. Okano, and S. Kusumoto. Improvement of a Visualization Technique for the Passage Rate of Unit Testing and Static Checking and Its Evaluation. In *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 279–284, 2011.

[90] S. Nasiri, Y. Rhazali, and M. Lahmer. Towards a generation of class diagram from user stories in agile methods. In *Advancements in Model-Driven Architecture in Software Engineering*, pages 135–159. IGI Global, 2021.

[91] R. Naz, M. Khan, and M. Aamir. Scrum-based methodology for product maintenance and support. *International Journal of Engineering and Manufacturing (IJEM)*, 6(1):10, 2016.

[92] P. Nidagundi and L. Novickis. Introducing Lean Canvas Model Adaptation in the Scrum Software Testing. *Procedia Computer Science*, 104:97–103, 2017. ICTE 2016, Riga Technical University, Latvia.

[93] P. Nielsen, T. Winkler, and J. Nørbjerg. Closing the IT Development-Operations Gap: The DevOps Knowledge Sharing Framework. 08 2017.

[94] M. Olan. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2):319–328, 2003.

[95] D. Pauly, B. Michalik, and D. Basten. Do Daily Scrums Have to Take Place Each Day? A Case Study of Customized Scrum Principles at an E-commerce Company. In *2015 48th Hawaii International Conference on System Sciences*, pages 5074–5083, 2015.

[96] R. Pawar and K. Mahajan. Implementation of Change Management in Software Development by using Scrum Framework. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7:400, 07 2017.

[97] S. Popić, G. Velikić, H. Jaroslav, Z. Spasić, and M. Vulić. The Benefits of the Coding Standards Enforcement and it's Influence on the Developers' Coding Behaviour: A Case Study on Two Small Projects. In *2018 26th Telecommunications Forum (TELFOR)*, pages 420–425, 2018.

[98] Prof. Dr. Sachin S. Vernekar, Sanjay Kumar Dhar. IMPACT OF DevOps SKILLS ON PROJECT MANAGEMENT OUTCOME – AN EMPIRICAL STUDY. *European Journal of Molecular Clinical Medicine*, 7(8):2, 2020.

[99] R. Dräther, H. Koschek, C. Sahling. *Scrum - kurz & gut.* O'Reillys Taschenbibliothek. O'Reilly Verlag GmbH Co. KG, 2013.

[100] I. K. Raharjana, D. Siahaan, and C. Fatichah. User stories and natural language processing: A systematic literature review. *IEEE Access*, 9:53811–53826, 2021.

[101] Ravi Teja Yarlagadda. DevOps and Its Practices. *International Journal of Creative Research Thoughts (IJCRT)*, 9(3):2, 2021.

[102] S. Resnick, A. Bjork, and M. de la Maza. *Professional Scrum with Team Foundation Server 2010.* Wrox guides. Wiley, 2011.

[103] Rice, John F. Adaptation of porter's five forces model to risk management. Technical report, DEFENSE ACQUISITION UNIV FT BELVOIR VA, 2010.

[104] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen, and T. Männistö. DevOps Adoption Benefits and Challenges in Practice:

A Case Study. In P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, editors, *Product-Focused Software Process Improvement*, pages 590–597, Cham, 2016. Springer International Publishing.

[105] F. K. Roman Simschek. *Kanban - Der agile Klassiker einfach erklärt.* UVK Verlag, 2021.

[106] J. Rubart and F. Freykamp. *Supporting Daily Scrum Meetings with Change Structure.* HT '09. Association for Computing Machinery, New York, NY, USA, 2009.

[107] K. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Addison-Wesley signature series. Addison-Wesley, 2012.

[108] G. Ruhe. *Product Release Planning: Methods, Tools and Applications.* CRC Press, 2010.

[109] G. Ruhe and M. Saliu. The art and science of software release planning. *IEEE Software*, 22(6):47–53, 2005.

[110] P. Runeson. A survey of unit testing practices. *IEEE Software*, 23(4):22–29, 2006.

[111] I. Saidani, A. Ouni, M. W. Mkaouer, and F. Palomba. On the impact of continuous integration on refactoring practice: An exploratory study on travistorrent. *Information and Software Technology*, 138:106618, 2021.

[112] B. C. Sanjeev Sharma. *DevOps für Dummies®, 2. limitierte Auflage von IBM.* John Wiley Sons, Inc., 2015.

[113] D. Schadow. *Java-Web-Security: Sichere Webanwendungen mit Java entwickeln.* dpunkt.verlag, 2014.

[114] J. Schild, R. Walter, and M. Masuch. ABC-Sprints: Adapting Scrum to Academic Game Development Courses. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, page 187–194, New York, NY, USA, 2010. Association for Computing Machinery.

[115] K. Schneider. *Die Informatik AG — Telekooperation.* Vieweg+Teubner Verlag, 2013.

[116] K. Schwaber. *Agile Project Management with Scrum.* Microsoft Press, 2004.

[117] K. Schwaber and J. Sutherland. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. 2020. Zuletzt abgerufen am 12.06.2022: `https://www.scrum.org/resources/scrum-guide`.

[118] Seacord, Robert C. *The CERT® C Coding Standard, Second Edition: 98 Rules for Developing Safe, Reliable, and Secure Systems*. Addison-Wesley Professional, 2nd edition, 2014.

[119] T. Sedano, P. Ralph, and C. Péraire. The Product Backlog. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 200–211, 2019.

[120] T. Sedano, P. Ralph, and C. Péraire. The Product Backlog. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 200–211, 2019.

[121] M. Senapathi, J. Buchan, and H. Osman. DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. *CoRR*, abs/1907.10201, 2019.

[122] M. Shahin, M. Ali Babar, and L. Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, 2017.

[123] S. Sharma, D. Sarkar, and D. Gupta. Agile processes and methodologies: A conceptual study. *International journal on computer science and Engineering*, 4(5):892, 2012.

[124] Simschek, R. and Kaiser, F. *SCRUM: Das Erfolgsphänomen einfach erklärt*. UVK Verlag, 2018.

[125] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli. When Testing Meets Code Review: Why and How Developers Review Tests. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 677–687, 2018.

[126] A. Srivastava, S. Bhardwaj, and S. Saraswat. SCRUM model for agile methodology. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 864–869, 2017.

[127] G. Stoneburner, A. Goguen, A. Feringa, et al. Risk management guide for information technology systems. *Nist special publication*, 800(30):800–30, 2002.

[128] V. Stray, D. I. Sjøberg, and T. Dybå. The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software*, 114:101–124, 2016.

[129] T. Streule, N. Miserini, O. Bartlomé, M. Klippel, and B. G. de Soto. Implementation of Scrum in the Construction Industry. *Procedia Engineering*, 164:269–276, 2016. Selected papers from Creative Construction Conference 2016.

[130] B. Sun, Y. Shao, and C. Chen. Study on the Automated Unit Testing Solution on the Linux Platform. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 358–361, 2019.

[131] H. Sutter and A. Alexandrescu. *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. C++ In-Depth Series. Pearson Education, 2004.

[132] Tavares, Breno Gontijo and da Silva, Carlos Eduardo Sanches and de Souza, Adler Diniz. Risk management analysis in Scrum software projects. *International Transactions in Operational Research*, 26(5):1884–1905, 2019.

[133] P. Tell, J. Klünder, S. Küpper, D. Raffo, S. MacDonell, J. Münch, D. Pfahl, O. Linssen, and M. Kuhrmann. Towards the statistical construction of hybrid development methods. *Journal of Software: Evolution and Process*, 33(1):e2315, 2021.

[134] P. Tell, J. Klünder, S. Küpper, D. Raffo, S. G. MacDonell, J. Münch, D. Pfahl, O. Linssen, and M. Kuhrmann. What are Hybrid Development Methods Made Of? An Evidence-Based Characterization. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 105–114, 2019.

[135] S. Tockey. *How to Engineer Software: A Model-Based Approach*. Wiley, 2019.

[136] B.-Y. Tsai, S. Stobart, N. Parrington, and B. Thompson. Iterative design and testing within the software development life cycle. *Software Quality Journal*, 6(4):295–310, 1997.

[137] J. Turner. *Kanban - 3 Books in 1 - the Ultimate Beginner's, Intermediate and Advanced Guide to Learn Kanban Step by Step*. Publishing Factory LLC, 2020.

[138] U.S. - Department of Defense. DevSecOps Fundamentals Guidebook: DevSecOps Tools Activities. Zuletzt abgerufen am 30.05.2022: `https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsTools-ActivitiesGuidebook.pdf`, 2021.

[139] S. Vadapalli. *DevOps: Continuous Delivery, Integration, and Deployment with DevOps: Dive into the core DevOps strategies*. Packt Publishing, 2018.

[140] A. Vajda. PROJECT MONITORING AND CONTROL USING BURNDOWN CHARTS. In *The International Conference Interdisciplinarity in Engineering INTER-ENG*, page 196. Elsevier Limited, 2009.

[141] Vieira, Marcel and C. R. Hauck, Jean and Matalonga, Santiago. How explicit risk management is being integrated into agile methods: Results from a systematic literature mapping. In *19th Brazilian Symposium on Software Quality*, SBQS'20, New York, NY, USA, 2020. Association for Computing Machinery.

[142] J. Watkins. *Agile Testing: How to Succeed in an Extreme Testing Environment*. Cambridge University Press, 2009.

[143] H. Wells. How effective are project management methodologies? An explorative evaluation of their benefits in practice. *Project management journal*, 43(6):43–58, 2012.

[144] D. Winter, E.-M. Schön, J. Uhlenbrok, and J. Thomaschewski. User Experience in Kanban. In H. Brau, A. Lehmann, K. Petrovic, and M. C. Schroeder, editors, *Tagungsband UP13*, pages 220–224, Stuttgart, 2013. German UPA e.V.

[145] M. Winteroll. *Requirements Engineering für Dummies*. WILEY-VCH GmbH, Weinheim, 2021.

[146] C. J. Woodward, A. Cain, S. Pace, A. Jones, and J. F. Kupper. Helping students track learning progress using burn down charts. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 104–109, 2013.

[147] I. Workshop, R. Conradi, W. on Software Configuration Management (7 : 1997 : Boston), I. C. on Software Engineering, and SCM. *Software Configuration Management: ICSE'97 SCM-7 Workshop, Boston, MA, USA, May 18-19, 1997 Proceedings*. Number Bd. 7 in ACM conference proceedings series. Springer, 1997.

[148] X. Yu and S. Petter. Understanding agile software development practices using shared mental models theory. *Information and Software Technology*, 56(8):911–921, 2014.

[149] Yu, Liguo and Batzinger, Robert and Ramaswamy, Srini. A Comparison of the Efficiencies of Code Inspections in Software Development and Maintenance. In *Software Engineering Research and Practice*, pages 460–468, 2006.

[150] Z. Zakaria, R. Atan, A. A. A. Ghani, and N. F. M. Sani. Unit Testing Approaches for BPEL: A Systematic Review. In *2009 16th Asia-Pacific Software Engineering Conference*, pages 316–322, 2009.