BACHELORARBEIT

SOFTWARE METRIKEN UND MECHANISMEN

zu Bewertung agiler Teams

ausgeführt an der

am Studiengang

Wirtschaftsinformatik

Von: Harald Beier

Personenkennzeichen: 1810319001

Graz, am …

.....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

KURZFASSUNG

In einer digitalisierten Welt, in der immer mehr Menschen als Wissensarbeiter arbeiten, ist es wichtig, Metriken zu haben, um den Output zu bewerten. Diese Bachelorarbeit hat das Ziel, die Grundlagen zu schaffen, um die Entwicklung von Teams und Einzelpersonen in einer agilen Softwareentwicklungsumgebung zu unterstützen. Es wird eine vergleichende Literaturanalyse durchgeführt mit dem Ziel, die existierenden Metriken in der agilen Softwareentwicklung aufzuzeigen, einen Produktivitätsbegriff zu definieren und diesen für die Team- und Mitarbeiterentwicklung zu nutzen. Die Ergebnisse der Meta-Analyse legen nahe, dass ein vielschichtiger Ansatz mit Analyse der Ursache für Regressionen, Qualitätsmanagement durch Einhaltung von Dokumentations- und Coding-Standards, kombiniert mit Story-Points als Messinstrument für Velocity, das beste Ergebnis liefert. Die Schlussfolgerung ist, dass es für diese Kombination von Metriken als Messwerkzeug für die Produktivität zwingend notwendig ist, ein Testframework zu etablieren und gut zu überwachen. Im nächsten Schritt ist eine empirische Studie mit mehreren Wissensarbeitern als Teilnehmer angeraten, um herauszufinden, ob diese Metriken für den langfristigen Einsatz geeignet sind.

ABSTRACT

In a digitalized world with more and more people working as knowledge workers it is important to have metrics to evaluate the output. This bachelor thesis has the aim to establish the groundwork to help to develop teams and individuals in an agile software development environment. A comparative literature analysis will be carried out with the aim to pinpoint the existing metrics in agile software development, to define a concept of productivity and how to use this for team and employee development. The results of the meta-analysis suggest that a multifaceted approach with analysis of the reason for regressions, quality management through adherence to documentation and coding standards combined with Story points as measurement tool for velocity delivers the best result. The conclusion is that it is imperative that for this combination of metrics as a measuring tool of productivity a testing framework is established and well monitored. In the next step an empirical study with multiple knowledge workers as participants is advised to find out if these metrics will be suitable for long-term use.

INHALTSVERZEICHNIS

1 EINLEITUNG

" Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen (Beck, et al., 2001).“

Agilität hat sich als Schlagwort des jungen 21ten Jahrhunderts etabliert und hat unter unterschiedlichen Gesichtern Einzug gehalten, sowohl in der Software Entwicklung, als auch im Projektmanagement und in der Organisationsentwicklung.

Insbesondere in der Software Entwicklung hat sich Agiles Software Management als defacto Standard etabliert. Als Software Manager*In, Produktmanager*In oder Teamleiter*In ist es eine integrale Aufgabe das Team zu fordern und zu fördern sowie weiterzuentwickeln. Es ist dafür notwendig, gezielt Maßnahmen für jedes einzelne Teammitglied zu definieren, um die weitere professionelle Entwicklung sowie die Steigerung der notwendigen Fähigkeiten zu fördern. Scrum, als eine der meistverbreiteten Methoden in der Software Entwicklung, bietet über die Velocity bereits ein Werkzeug, um die Team-Performance darzustellen. Es stellt sich nun aber die Frage, inwieweit können Metriken etabliert werden, um die Leistungsfähigkeit einzelner Personen zu bewerten und inwiefern ist die Velocity als alleiniger Faktor eine ausreichende Metrik?

Diese Arbeit beschäftigt sich daher mit der immer drängenderen Frage, ob es andere Methoden gibt, mit der die Qualität/Produktivität von Wissensarbeiter*Innen (speziell Softwareentwickler) bewertet werden kann. Welche Ansätze sind hierfür schon im Angloamerikanischen Raum entwickelt worden?

1.1 Ausgangssituation

Aktuell adaptieren immer mehr Firmen agile Praktiken und Frameworks, um damit Softwareentwicklung durchzuführen. Im Zuge der Umstellung werden für Firmen Metriken benötigt, um die neuen Prozesse und Leistungen der Teams und Teammitglieder zu evaluieren und zu bewerten. Eine klare Definition eines Frameworks an Metriken und Mechanismen sind notwendig, um Prozess- und Qualitätsmanagement in der Software Entwicklung und in der Arbeit von Wissensarbeiter*Innen zu ermöglichen.

1.2 Ziele

Diese Arbeit beschäftigt sich mit der Forschungsfrage „*Welche Metriken und Mechanismen existieren und wie werden diese angewandt, um agile Teams zu bewerten und fördern?“*. Ziel ist die Erstellung einer Liste von Metriken und Mechanismen, die in der agilen Software Entwicklung zur

Produktivitätsmessung herangezogen werden. Diese dienen als Ausgangsgrundlage für die weitere Bearbeitung der Forschungsfrage. Es folgt die Analyse der Verwendung der verschiedenen Metriken in unterschiedlichen Projekten. Anhand dieser wird aufzeigt, was die Kostentreiber in der Softwareentwicklung sind und welche Qualitätsmaßnahmen sowie Controlling Möglichkeiten vorhanden sind. Für die weitere Bearbeitung der Forschungsfrage ist es notwendig, den Begriff der Produktivität genauer zu untersuchen und eine für diese Arbeit gültige Definition zu erstellen. Abschließend muss eine Klarstellung der fachlichen Fähigkeiten, Kommunikationsmuster und Arbeitsformen erfolgen, die zu Verbesserung von agilen Teams und Teammitgliedern führen.

1.3 Motivation

Grund für die Auswahl dieses Themas der Bachelorarbeit für den Autor ist die Schaffung von transparenten und validen Methoden, um Teammitglieder zu evaluieren und weiterzuentwickeln. Diese Methoden sollen theoriegestützt sein und durch wissenschaftlichen Hintergrund eine Bewertung rein aus dem Bauch heraus verhindern. Es gilt diese Arbeit auch für die tägliche Praxis als Grundlage heranzuziehen, um die Teamperformance von agilen Teams außerhalb der bekannten Metriken zu erhöhen und zusätzlich immanenten Mehrwert zu generieren.

Somit soll in bester wissenschaftlicher Tradition eine Umsetzung der Theorie in die Praxis erfolgen. Weiters soll es dem Autor als Teamleiter ermöglichen, die Teamperformance zu messen und die Teammitglieder im Sinne des Servant Leading zu fordern und zu fördern sowie Entscheidungen anhand wissenschaftlicher Grundlagen zu fällen.

1.4 Vorgehensweise

Für die vorliegende Arbeit wird die Methode der Literaturrecherche in den Portalen von ACM, IEEE *Xplore* digitalen Bibliothek, Springer, ScienceDirec, Scopus, Emerald und ResearchGate, sowie in der Campus02 Bibliothek und den zur Verfügung gestellten Online Ressourcen durchgeführt. Außerdem wurden Onlineressourcen, die durch die Universität Wien zugänglich sind, auch genutzt.

Im Zuge der Literaturanalyse werden die unterschiedlichen wissenschaftlichen Artikel und Studien auf die Parameter und Blickwinkel hinsichtlich der Metriken, agile Praktiken und Mechanismen analysiert.

1.5 Aufbau und Struktur

Diese Arbeit gliedert sich in sechs Kapitel. Nach der Einleitung in Kapitel 1 beschäftigt sich Kapitel 2 mit den agilen Methoden und Frameworks. Anhand des agilen Frameworks Scrum werden verschiedene Metriken für agile Teams definiert.

Der Autor beschäftigt sich in Kapitel 3 damit, einen allgemeingültigen Begriff für Produktivität in der Softwareentwicklung anhand der Literaturanalyse zu erstellen, der für weitere Untersuchungen und Einordnung von Metriken verwendet werden kann. Es wird dabei zuerst eine allgemeine Definition verschiedener Parameter von Produktivität untersucht und beschrieben, bevor ein spezifisches Modell mit unterschiedlichen Parametern für die Software Entwicklung präsentiert wird.

Die Synergie von Produktivität und Softwaremetriken sowie die Möglichkeiten diese anhand von bestimmten Indikatoren zu überprüfen, ist das Hauptaugenmerk des 4 Kapitel.

In Kapitel 5 werden zuerst verschiedene Faktoren innerhalb von agilen Teams und deren Auswirkung auf die Leistungsfähigkeit präsentiert und dann verschiedene Ideen und Konzepte zur Weiterentwicklung von agilen Teams.

Zum Abschluss findet sich in Kapitel 6 eine Zusammenfassung der gefunden Faktoren und ein Ausblick auf die Entwicklung eines gewichteten Kriterien- und Metrik-Kataloges zur Entwicklung von agilen Teams.

2 BEWERTUNGSMETRIKEN FÜR AGILE TEAMS

Die Agile Softwareentwicklung[1], (Siepermann, 2020) in der Art und Weise wie Sie heute praktiziert und rezitiert wird, basiert auf dem **Manifest für Agile Softwareentwicklung**. (Beck, et al., 2001). Agile Software Entwicklung setzt sich aus an Agilität ausgerichteten Methoden und agilen Prozessen zusammen.

2.1 Agile Methoden und Praktiken

Es existiert eine große Vielfalt an unterschiedlichen Methoden. Zur Vereinfachung werden diese kurz in Englisch angeführt, da diese Methoden für die Bewertungsmetriken von besonderer Bedeutung sind:

Tabelle 1: Tabellarische Auflistung agiler Methoden ©Harald Beier

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

[1] Für einen schnellen Überblick ist hier das Gabler Wirtschaftslexikon empfehlenswert (Siepermann, 2020)

ATTD: Hierbei handelt es sich um eine Methode, welche großen Wert auf die Kommunikation der Kund*Innen, Entwickler*Innen und Tester*Innen legt. Wichtig sind hierbei die Implementierung von Akzeptanztests bevor die Entwicklungsteams mit dem coden beginnen. Diese Tests sollen in der Sprache der Geschäftsdomäne geschrieben werden. Eine Anforderung bzw. User Story für welche kein Test vor Anforderungsumsetzung geschrieben wurde, ist schlecht implementiert. (Gause & Weinberg, 1989; Pugh, 2011)

AM: Dies ist eine Methode für Modellierung und Dokumentation von Softwaresystemen, die auf unterschiedlichen Kernprinzipien beruht. (Ambler, 2004)

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

BL sind eine Liste von Arbeitspaketen, die geordnet sind nach Priorität und Reifegrad der einzelnen Artefakte. Die bekanntesten Varianten der Arbeitspaketformate sind User Stories oder Use Cases. Es werden Features, Problemlösungen und andere nicht technische Anforderungen dokumentiert, um ein Softwareprodukt liefern zu können.(Svensson et al., 2019)

BDD ist eine agile Testtechnik, deren Ziel es ist, Software-Anforderungen als Beispielinteraktionen mit dem System zu spezifizieren, wobei eine strukturierte natürliche Sprache verwendet wird. Während die Beispiele (in der Theorie) von nicht-technischen Stakeholdern gelesen werden können, können sie auch gegen die Codebasis ausgeführt werden, um Verhaltensweisen zu identifizieren, die noch nicht korrekt implementiert sind. (Binamungu et al., 2020; Carvalho et al., 2013)

CI: Kontinuierliche Integration ist ein Teil der kontinuierlichen Praktiken. Ziel ist eine regelmäßige Lieferung und Bereitstellung von neuen Anwendungen, Funktionen, Features und neuer Software zu erreichen. (Shahin et al., 2017)

CFT: Hierbei handelt es sich um Teams aus unterschiedlichen Fachbereichen einer Firma. Ziel ist es, die Wissenssilos innerhalb der eigenen Organisation aufzubrechen und damit die Kundenbedürfnisse an die erste Stelle zu bringen. Innerhalb der Softwareentwicklung bedeutet dies oft, dass Frontend, Backend, Devops, Tester*Innen, Interface Designer*Innen und Personen, die für die Geschäftslogik zuständig sind, zusammenarbeiten. (Khalil et al., 2013; McDonough, 2000)

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

DDD ist eine Vorgehensweise, in der versucht wird, das Domänenwissen für den Softwareentwurf korrekt abzufragen, zu erfassen und modellbasiert darzustellen. Ziel ist, die Beziehungen unterschiedlicher Domänenkonzepte zu identifizieren. DDD findet oft Anwendung beim Entwurf von

Microservice-Architekturen, da es eine Schnittmenge zwischen den einzelnen funktionalen Microservices und den verschiedenen Geschäftsfeldern gibt. (Rademacher et al., 2018)

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

PPr: Pair Programming ist eine Technik aus dem Extreme Programming. Hier müssen zwei Programmierer gleichzeitig am selben Artefakt arbeiten. Es wird zwischen einem designierten Fahrer, der aktiv den Code erstellt, und dem Nicht-Fahrer unterschieden, der die ständige Überprüfung des Artefakts vornimmt. Nach einer bestimmten Zeitperiode werden die Rollen getauscht. Nur Code, der von beiden Partnern erstellt wurde, wird akzeptiert und in das Software Projekt integriert. (McDowell et al., 2002)

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

RF ist der Prozess, in dem der Quellcode so umstrukturiert wird, dass die interne Codequalität und Struktur der Software verbessert wird, dabei aber das externe Verhalten, das für die Benutzer ersichtlich ist, gleichbleibt. Ziel ist es, die Wartungskosten zu verringern und die Lebensdauer der Software zu verlängern. (Kaur et al., 2021; Kaur & Singh, 2019)

RS sind Meetings bzw. Aktivitäten zur Erfahrungssicherung der Teammitglieder. Oft auch als Post-Mortems bezeichnet, in der die Probleme sowie Ursachen eines abgeschlossenen Projektes oder einer Softwareiteration analysiert werden. Es sollen durch diese Untersuchung sowohl die Erfolgsfaktoren als auch die Auslöser von Problemen erkannt werden. (Dingsøyr, 2005; Lehtinen et al., 2014)

SBE ist ein Leitfaden oder eine kollaborative Methode, deren oberstes Ziel die Entwicklung von Software ist, welche die Kundenanforderungen erfüllt. Es wird hier mit Workshops gearbeitet, in denen die verschiedenen Rollen und Standpunkte vertreten sind und die gemeinsam spezifischen Szenarien der Nutzung anhand von Beispielen entwerfen. Mit Hilfe dieser Szenarien sollen folgend die automatisierten und funktionalen Tests entwickelt werden. Ziel ist es, eine Dokumentation zu erstellen, die in der Sprache der jeweiligen Fachdomäne geschrieben wurde und auch von Personen aus den Fachabteilungen, die keine Programmierer sind, gelesen werden kann. (Bache & Bache, 2014; Blasquez & Leblanc, 2017; Qattous et al., 2010)

Im Gegensatz zu anderen Arten der objektorientierten Modellierung wird im **SDM** die Struktur nicht durch statische Klassendiagramme und der Beziehung dieser Klassen zueinander Wert gelegt, sondern auf die Erstellung von spezifischen Beispielen, die Szenarien in der Nutzung abbilden. Zudem sollen die daraus entstehenden Objektdiagramme sich im Laufe des Szenarios und der Ausführung der Software weiterentwickeln. (Wautelet et al., 2017; Zündorf, 2008)
Automatisierte Tests und kleine schnelle Iterationen stehen im Mittelpunkt von **TDD**. Es besteht aus den Komponenten Testen, Driven und Development. Es wird hier zuerst der Test entwickelt, welcher die darauffolgende Codezeilen validieren und überprüfen soll, dann wird nur so viel programmiert, bis der Test positiv bestanden wird. Im Rahmen der Softwareentwicklung soll es immer wieder zum Refactoring sowohl des Codes als auch der automatisierten Tests kommen. (Janzen & Saiedian, 2005)
TB ist die Definition eines Zeitrahmens oder einer Zeitphase, die innerhalb der Projektplanung für gewisse Vorgänge genutzt werden darf. Es wird hier ein Zeitbudget für die jeweiligen Aufgaben erstellt. Insbesondere im Scrum und der agilen Softwareentwicklung wird Timeboxing für einzelne Vorgänge wie Meetings und Iterationszyklen verwendet.(Jalote et al., 2004; Miranda, 2011)
Die Velocity ist die Maßeinheit der geleisteten Arbeit innerhalb eines bestimmten Iterationszyklus während der agilen Softwareentwicklung. Das **VT** ist die Messung und Darstellung der Team Performanz. (Al-Sabbagh & Gren, 2018; Olsen, 1995). Zwei für die Darstellung verwendete Methoden sind Burn-Down und Burn-Up Charts, die in Kapitel 2.1 näher beschrieben werden.
Bei diesen verschiedenen Praktiken ist ersichtlich, dass einige bereits die Produktivität (z.B. CI, IID, VT, PlP, BL), Ergebnisorientierung (z.B.: RF, TDM, TB) sowie die Weiterentwicklung der Teams (z.B.: RF, PP, AT, AM) innerhalb der agilen Softwareentwicklung mitgedacht bzw. angedacht haben. Betreffend der Verbreitung bestimmter Methoden wird hier kurz auf das Ergebnis aus der aktuellen Studie Status Quo (Scaled) Agile 2019/20 (Komus et al., 2020) verwiesen.

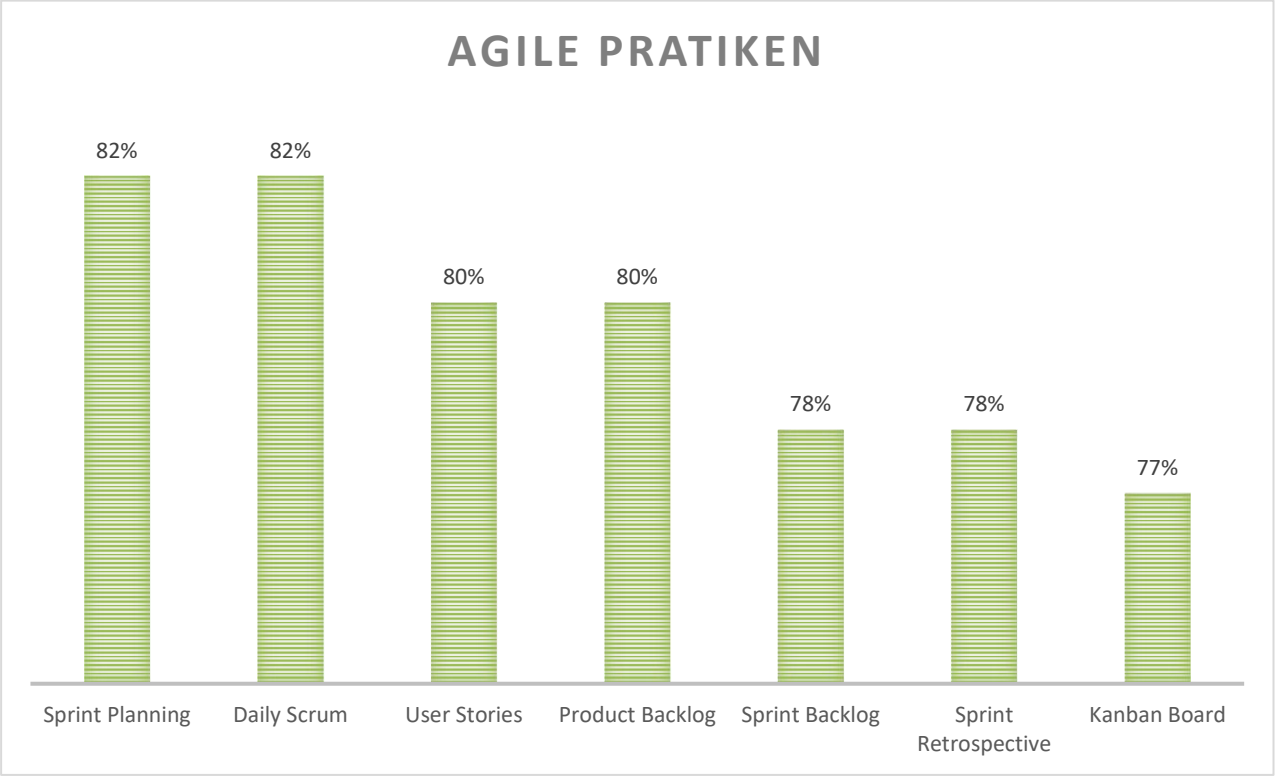

Abbildung 1: Diagramm der Top 7 Agilen Praktiken basierend auf (Komus et al., 2020, S. 84) ©

Das Kanban Board wird als Methode des Kanban in der IT in Kapitel 2.2 kurz beschrieben. User Stories und ihre speziellen Parameter werden in Kapitel 2.3 erläutert. Auf das Sprint Planning und die möglichen Metriken, die hier zur Anwendung kommen können, wird in Kapitel 2.4 noch näher eingegangen. Da diese Begrifflichkeiten einem bestimmten agilen Framework zugeordnet werden können, müssen einige agile Frameworks im nächsten Kapitel exemplarisch beschrieben werden.

2.2 Agile Frameworks

Aufbauend auf das agile Manifest haben sich eine Vielzahl von Frameworks für das Management der agilen Prozesse entwickelt. Diese sind mit Fokus auf die Software Entwicklung entstanden, haben aber ihren Weg auch schon in andere Bereiche angetreten. Eines der am häufigsten verwendeten Frameworks ist **Scrum**. Für den Zweck dieser Arbeit wird hier ein Fokus auf das Framework Scrum gelegt. Es sollen aber einige ausgewählte Frameworks entsprechend kurz präsentiert werden.

Tabelle 2: Agile Prozessmanagement Frameworks der Software Entwicklung ©Harald Beier

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

Die Leitwerte des **XP** sind Kommunikation, Einfachheit, Feedback und Mut und sollen durch die Praktiken: Vor-Ort-Kunde, Planspiel, Metapher, einfaches Design, kleines Release, Pair Programming, Testen, Refactoring, kontinuierliche Integration, 40-Stunden-Woche, Coding Standard und kollektive Verantwortung für die Entstehung eines optimalen Produktes sorgen. Einige dieser Praktiken bzw. Methoden werden auch in anderen agilen Frameworks genutzt, im **XP** wird aber stark auf die Synergieeffekte zwischen den Leitwerten und Praktiken wert gelegt. (Fojtik, 2011; Tolfo & Wazlawick, 2008) Mohammad Alshayeb und Wie Li untersuchten die spezifischen Praktiken und stellten im XP folgende Tätigkeiten als Hauptaktivitäten der Entwickler fest: Refactoring, neues Design, Beheben von Fehlern und Implementieren von Unit- sowie Funktionstests, um die Funktionstüchtigkeit des Produktes sicherzustellen. (Alshayeb & Li, 2006)

Das **FDD** hat sich aus der Coad Methode von Peter Coad entwickelt und besteht nur aus zwei Hauptphasen. Der Entdeckungsphase und der Implementierungsphase. Hauptaugenmerk wird dabei auf die Entdeckungsphase gelegt, da hier sowohl die Liste der Features erstellt wird, als auch die UML-Diagramme der spezifischen Kundendomäne. Die Mitwirkung des Kunden ist besonders wichtig, damit die Wartbarkeit und Erweiterbarkeit des Codes im weiteren Projektverlauf gewährleistet werden kann. Die verwendete Sprache sollte von sowohl von Entwickler*Innen,- als auch von Kundenseite verstanden werden. (Chowdhury & Huda, 2011; “Feature-Driven Development,” 2006)

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

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

Die Visualisierung erfolgt durch das sogenannte Kanban-Board, das sowohl die Begrenzung der Arbeitslast bzw. des Arbeitspakets, als auch die Priorisierung der Aufgabe und das Eingreifen bei Engpässen im Arbeitsfluss ermöglicht. (Ahmad et al., 2018; Anderson, 2010; Granulo & Tanovic, 2019)

In **ASD** wird der Fokus auf ein zyklisches Weiterentwicklungsmodell gelegt, das die unterschiedlichen Phasen im Lebenszyklus der Software widerspiegelt. (Alnoukari et al., 2008)

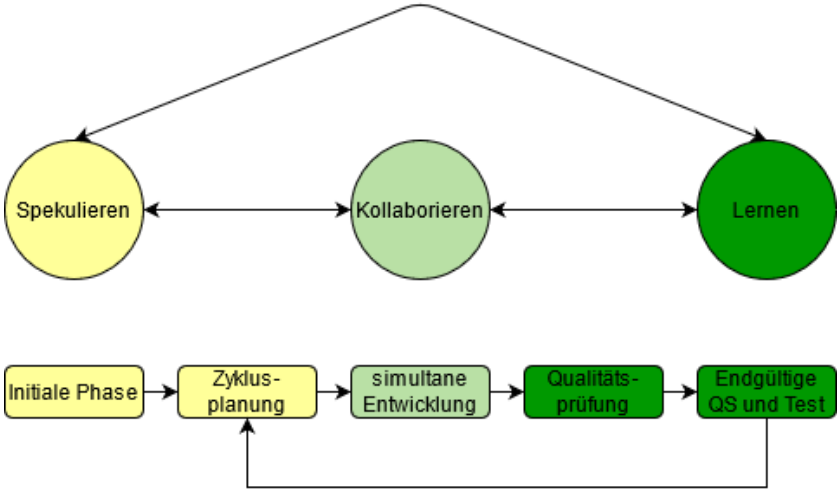

Abbildung 2: Darstellung ASD basierend auf (Abdelaziz et al., 2015) ©Harald Beier

- **Spekulieren** ersetzt planen, um hier mehr Raum für Innovation und Ungewissheit bei komplexen Problemen zu schaffen.
- **Kollaborieren/Zusammenarbeiten:** Bei komplexen Software-Anwendungen und sich ändernden Anforderungen ist der Informationsfluss nur durch die Zusammenarbeit im Team schaffbar.
- **Lernen**: In dieser Phase können die technischen Parameter und Kundenwünsche regelmäßig nach den abgeschlossenen einzelnen Iterationen überprüft werden. (Abdelaziz et al., 2015)

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

- Menschen
- Interaktionen
- Gemeinschaft
- Fertigkeiten
- Talente und Kommunikation (Ibrahim Muhammad et al., 2020)

Die Methoden von Crystal werden nach Farben benannt und nach der Größe des Projektes und der Anzahl der Projektmitarbeiter*Innen eingeteilt.

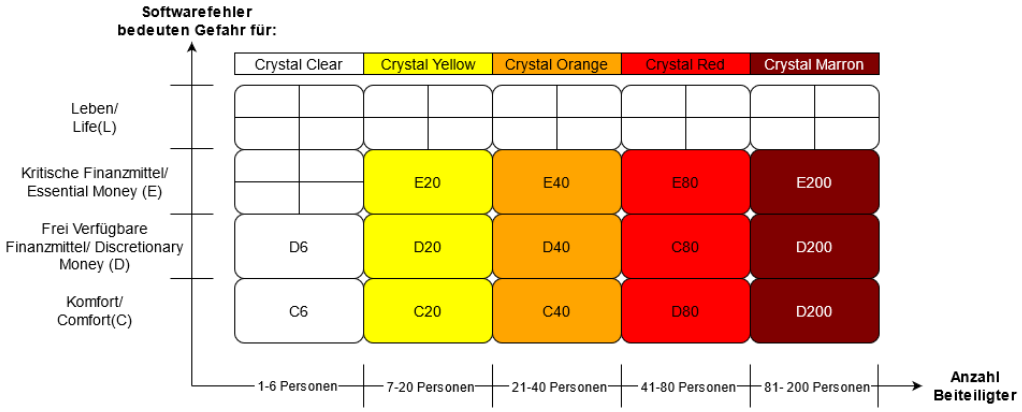

Abbildung 3: Darstellung der Crystal Methoden basierend auf (Cockburn, 2005) ©Harald Beier

Auf der Y-Achse werden die vier Level des Gefahrenlevels definiert und auf der X-Achse die Anzahl der Beteiligten. Hier angezeigt werden die Crystal Methoden, die noch nicht das Risikolevel Leben abdecken können. Crystal Clear sollen Projekte mit fixiertem Preis und klaren Parameter sein und sobald Projekte Gefahr für Leib und Leben beinhalten, handelt es sich dabei um Crystal Diamond und Crystal Sapphire. (Ibrahim Muhammad et al., 2020)

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

Die meisten Menschen werden keine detaillierte Dokumentation lesen. Sie werden jedoch hin und wieder Anleitung und Schulung benötigen.

Das Projekt sollte einfach mit wenigen Seiten beschrieben werden.

Die AUP entspricht den von der Agile Alliance beschriebenen Werten und Prinzipien.

Das Projekt muss sich darauf konzentrieren, einen wesentlichen Wert zu liefern und nicht unnötige Funktionen.

Die Entwickler müssen die Freiheit haben, Werkzeuge zu verwenden, die für die jeweilige Aufgabe am besten geeignet sind, und nicht, um eine Vorschrift zu erfüllen.

AUP lässt sich über gängige HTML-Editierwerkzeuge leicht anpassen. (Christou et al., 2010)

Es werden dann vier seriell ablaufende Phasen definiert:

Inception/Beginn: Ziel ist es, den anfänglichen Umfang des Projekts und eine potenzielle Architektur für Ihr System festzulegen, sowie die anfängliche Projektfinanzierung und die Akzeptanz der Stakeholder zu erhalten.

Elaboration/Ausarbeitung: Die Architektur des Systems wird überprüft.

Construction/Konstruktion: Es soll regelmäßig und schrittweise funktionierende Software erstellt werden, welche die Anforderungen der Projektbeteiligten mit höchster Priorität erfüllt.

Transition/Übergabe: Das Ziel ist die Validierung und der Einsatz ihres Systems in ihrer Produktionsumgebung. (Li & Wang, 2010; ShuiYuan et al., 2009)

LSD hat als Vorbild das Toyota Produktionssystem und definiert sieben Prinzipien und 22 Praktiken für die Umsetzung im Rahmen der agilen Softwareentwicklung. Die sieben Prinzipien sind:

Verschwendung vermeiden

Lernen unterstützen

So spät entscheiden wie möglich

So früh ausliefern wie möglich

Verantwortung an das Team geben

Integrität einbauen

Das Ganze sehen (Janes, 2015; Jonsson et al., 2013)

Unterschiedliche **LSAF** sind entwickelt worden, um agile Praktiken auf große Projekte und Softwareentwicklung mit weltweit verteilten Teams anzuwenden. Zu den bekanntesten Modellen zählen:

Scrum of Scrums (SoS)

Scaled Agile Framework (SAFe)

Large-Scale Scrum (LeSS)

Disciplined Agile Delivery (DAD)

Lean Scalable Agility for Engineering (LeanSAFE) (Beecham et al., 2021; Ebert & Paasivaara, 2017)

Kieran Conboy und Noel Carroll definieren ausgehend von einer 15-jährigen Retrospektive folgende Herausforderungen bei der Implementation von Large Scale Agile Frameworks:

Definieren von Konzepten

Vergleich und Gegenüberstellung von Frameworks

Bereitschaft und Appetit auf Veränderung

Abgleich von Organisationsstruktur und Rahmenbedingungen

Top-down- versus Bottom-up-Implementierung

Überbetonung der 100%igen Einhaltung der Regeln des Frameworks gegenüber dem Mehrwert für die Firma

Fehlender evidenzbasierter Einsatz

Erhaltung der Autonomie der Entwickler und Entwicklerteams

Fehlende Abstimmung zwischen Kundenprozessen und Frameworks (Conboy & Carroll, 2019; Kasauli et al., 2021)

Anhand der kurzen Zusammenfassung ist ersichtlich, dass die unterschiedlichen Frameworks eine differenzierte Gewichtung auf verschiedene Parameter und Faktoren legen. Es werden hier für einige Kriterien die Ergebnisse aus der Studie „Status Quo (Scaled) Agile 2019/20“ kurz präsentiert: (Komus et al., 2020)

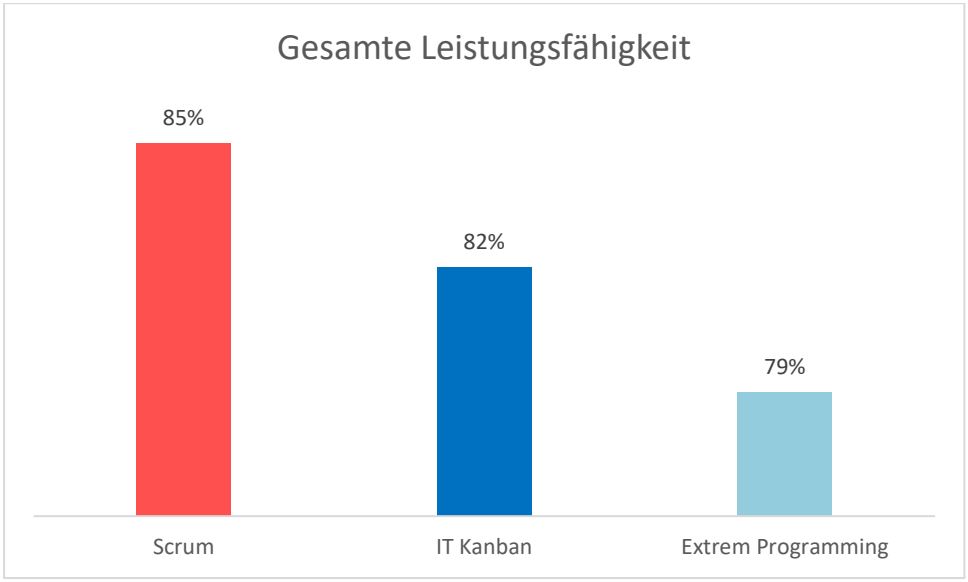

Abbildung 4: Diagramm der Top 3 im Kriterium Leistungsfähigkeit basierend auf (Komus, et al., 2020, S. 79) ©Harald Beier

Hier ist erkennbar, dass Scrum von den in der Studie befragten Personen die größte Leistungsfähigkeit im Rahmen der Software Entwicklung beigemessen wird. Da im Rahmen dieser Arbeit Software Metriken und Mechanismen für die Bewertung und Weiterentwicklung von agilen Teams und Einzelpersonen analysiert und verglichen werden, muss ein Hauptaugenmerk auf das agile Framework Scrum gelegt werden. Als nächster Parameter wird nun die wichtige Komponente Teamwork aus der Studie präsentiert.

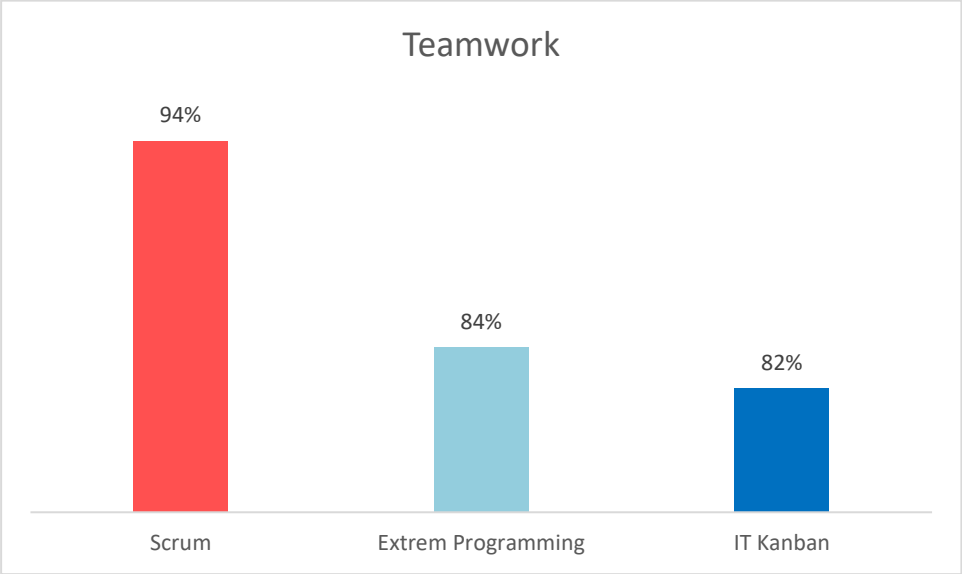

Abbildung 5: Diagramm der Top 3 im Kriterium Teamwork basierend auf (Komus, et al., 2020, S. 79) ©Harald Beier

Hier ist erkennbar, dass Scrum die größte Zustimmung bei den befragten Personen hinsichtlich positiver Auswirkung für das Teamwork hat.

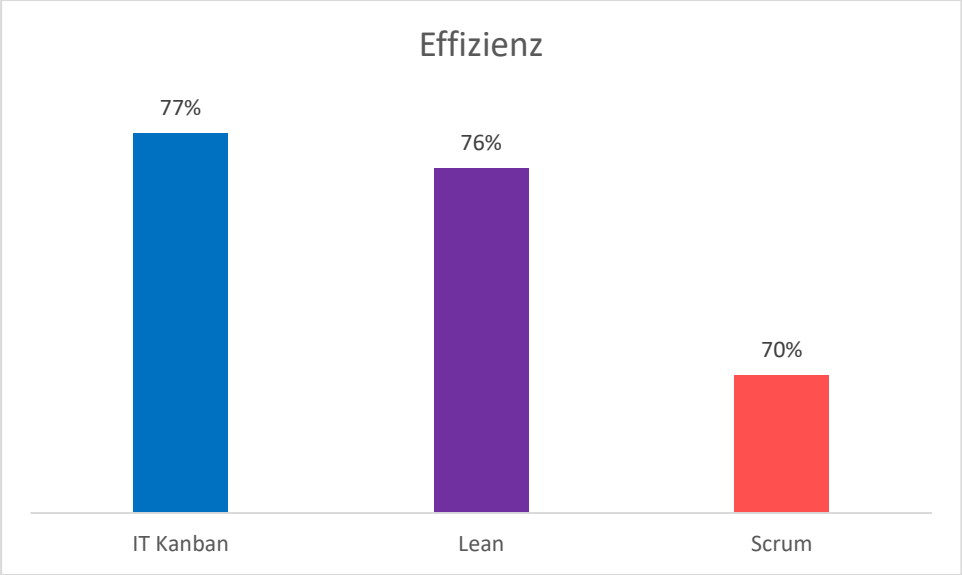

Abbildung 6: Diagramm der Top 3 im Kriterium Effizienz basierend auf (Komus, et al., 2020, S. 79) ©Harald Beier

Im Bereich Effizienz belegt Scrum nur den dritten Platz, hier ist die Nummer 1 das Framework IT Kanban. Aufgrund dieser Verbreitungsergebnisse und da Scrum sowohl bei Leistungsfähigkeit und Teamwork an erster Stelle steht, wird hier auf das Scrum-Framework und die vorhandenen Bewertungsmetriken näher eingegangen. Der Parameter der wahrgenommenen geringeren Effizienz darf für die nachfolgende Kapitel aber nicht außer Acht gelassen werden.

2.3 Scrum und seine Bewertungsmetriken

Ken Schwaber und Jeff Sutherland bieten auf der Homepage https://www.scrumguides.org/ den Scrum Guide sowohl online als auch in unterschiedlichen Sprachen an, wo Fragmente des Frameworks folgendermaßen definiert sind (Schwaber & Sutherland, 2020):

- Scrum-Team:
  - Product Owner
  - Entwicklungsteam
  - Scrum Master
- Scrum-Ereignisse:
  - Der Sprint
  - Das Sprint Planning
  - Der Daily Scrum
  - Der Sprint Review
  - Die Sprint Retrospektive
- Scrum-Artefakte:
  - Der Product Backlog
  - Der Sprint Backlog
  - Das Inkrement

Für die Transparenz der Artefakte definieren Ken Schwaber und Jeff Sutherland die **„Definition of Done“.** Damit die erbrachten Leistungen der Entwicklungsteams gemessen werden können und es einen Rahmen für die Bewertung gibt, ist es notwendig, bestimmte Metriken anzuwenden. Verschiedene Metriken werden benötigt, um innerhalb der Firma Features zu priorisieren und die Planung des nächsten Sprints als Iteration zu planen. (Hodgkins & Hohmann, 2007)
Eine Standardmethode für die Aufwandsschätzung in Scrum sind sogenannte Story Points. Diese Methodik zur Einschätzung des Aufwandes ist relativer Natur und basiert oft auf der Komplexität und dem Risiko der Umsetzung. (Hill, 2011) Mike Cohn beschreibt in seinem Werk „User Stories Applied“ einen Story Point als den Arbeitsaufwand, der in einem idealen Arbeitstag erledigt werden kann, und dass die Schätzung des Aufwandes innerhalb des Teams erfolgen muss. Oft werden zudem Fibonacci-Folgen genutzt. (Cohn, 2013, S. 87–95)
Für die bereits beschriebene Umsetzung und Nutzung von Story Points ist es notwendig, auf die drei Kernprinzipien der agilen Software Entwicklung zurückzugreifen:

- **User Stories:** Hierbei handelt es sich um den Versuch, Software Anforderungen in Alltagssprache zu dokumentieren. Oft werden dafür bestimmte Aufbaukriterien genutzt, die als Satzbaukasten funktionieren. Wie z. B. **As a…, I want to…, So that…** (Cohn, 2013, S. 3–10)
- **Definition of Ready:** Eine Liste von Parametern und Kriterien in der agilen Software Entwicklung, die erfüllt sein müssen, damit das Inkrement z. B. die User Story in der nächsten Iteration bearbeitet werden kann. (Power, 2014)

- **Definition of Done:** Hierbei handelt es sich um Qualitätsstandard für ein Inkrement. Es sind verschiedene Arbeitsschritte, die durchgeführt und positiv abgeschlossen werden müssen, damit die zukünftigen BenutzerInnen des Produktes den gesamten Funktionsumfang nutzen können. (Schwaber & Sutherland, 2020)

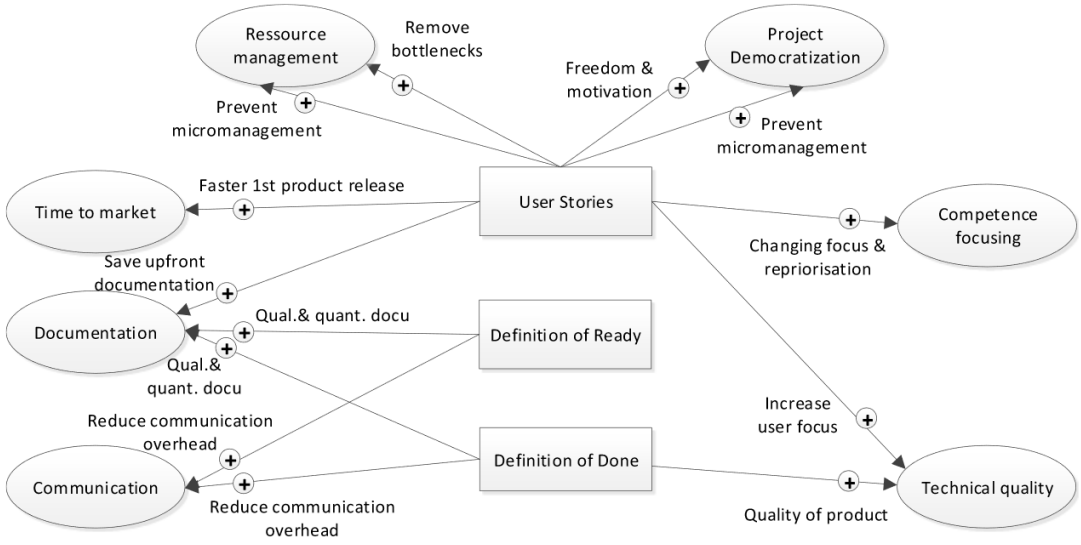

Abbildung 7: Auswirkungen von User Stories, Definition of Ready und Definition of Done basierend auf "Agile Practice Impact Model" ©Harald Beier

Abbildung 7 zeigt die von Philipp Diebold und Thomas Zehler beschriebenen Verbindungen bzw. die positiven Auswirkungen die „*User Stories*“, „*Definition of Ready*“ sowie „*Definition of Done*“ als Metriken auf die Software Entwicklung haben. (Diebold & Zehler, 2015, S. 92–96)
Visualisierungsmöglichkeiten, die oft zum Einsatz kommen, sind Burn Charts. Dabei handelt es sich um Darstellungsformen, mit denen der Fortschritt innerhalb eines Projektes präsentiert werden kann. Es werden dabei entweder die bereits abgeschlossenen oder noch zu erledigenden Arbeitspakete aufgezeigt. Es kann zwischen zwei typischen Formen unterschieden werden: (Dinwiddie, 2009)

- **Burn-down Charts:** Diese Variante zeigt die noch offenen Arbeitspakete bzw. Story Points an, die noch bis zum Erreichen der Deadline des Projektes bearbeitet werden müssen. Entscheidend für die korrekte Nutzung ist, dass es zu keiner Änderung der zu erfüllenden Arbeitsmenge während der Projekt- bzw. Iterationsphase kommt. (Kocurek, 2011)
- **Burn-up Charts:** Anstatt die offenen Arbeitspakete zu überprüfen, kann es abhängig vom gewünschten Projekt oder der Zielsetzung der Iteration besser sein, die kumulative Arbeitsleistung, welche innerhalb einer bestimmten Zeitperiode erledigt wurde, aufzuzeigen. (Dinwiddie, 2009)

2.4 Metriken für die Planung von Scrum Sprints

Für die Bewertung gibt es unterschiedliche Metriken, die herangezogen werden können. Eine sehr einfache und lange in Verwendung befindliche Methode ist „*Source Lines of Code*“ (SLOC). Hier werden nur die Zeilen des Codes bzw. der verwendeten Programmiersprache betrachtet. Es lässt zwar einen Rückschluss auf den Umfang und den verwendeten Zeiteinsatz zu, aber nicht auf die abgebildeten Funktionen und die Qualität. SLOC ist daher weniger geeignet als Maßeinheit, um Effizienz oder Produktivität zu messen. (McConnell, 2009)
Eine zweite Variante einer Metrik zur Messung wäre die „*Function Point Analysis*“. Diese bietet fünf unterschiedliche Faktoren an, um die Business Funktion für den Endnutzer zu gewichten. Es handelt sich dabei um:

External Inputs (EIs)

External Outputs (EOs)

External Inquiries (EQs)

Internal Logical Files (ILFs)

External Interface Files (EIFs)

Diese Art, den Aufwand und die Leistung von Projekten zu bewerten, wird durch die „*Use Case Points*“- Methode erweitert. (Park, 1992) Andere Metriken, wie Story Points, wurden bereits im vorangegangenen Kapitel beschrieben.
Die Bedeutung der unterschiedlichen Praktiken und Metriken zur Feststellung des Aufwandes werden sehr gut im Paper **„***Effort Estimation in Agile Software Development: A Survey on the State of the Practice***“** dargestellt und summiert. Zusammengefasst sollen hier die Top drei präsentiert werden: (Usman et al., 2015)

Tabelle 3: Aufwandseinschätzung in der agilen Software Entwicklung ©Harald Beier basierend auf (Usman et al., 2015)

Parameter	Genutzte Varianten
Einschätzung des Aufwandes	1. Planning Poker
	2. Estimation by Analogy
	3. Expert Judgement
Maßeinheiten für den Aufwand	1. Story Points
	2. Function Points
	3. Use Case Points

Metriken, Visualisierungen, Schätzmethoden und Kernprinzipien sind notwendig, um Planung in der agilen Software Entwicklung überhaupt zu ermöglichen. Daher ist es wichtig, dass für die

unterschiedlichen Ebenen, wie Releaseplanung, Sprintplanung oder aktuelle Tagesplanung, verlässliche Werkzeuge vorliegen, welche eine einfache Umsetzung ermöglichen.
Eine Analyse findet im Artikel „*Software Metrics Classification for Agile Scrum Process: A Literature Review*“ statt. Es führt Metaanalyse von 34 Metriken durch´, ordnet diese Scrum Events zu und erstellt dann eine Empfehlung für die jeweiligen Metriken: (R. Kurnia et al., 2018)

- **Sprint-Planning:** Hier wird empfohlen sowohl Story Points, als auch Velocity als Grundlage für die Planung zu nutzen. Velocity dient der Messung der Zuverlässigkeit des Teams. Story Points dienen der Abschätzung der Projektgröße und der Komplexität der einzelnen User Stories im Backlog.
- **Daily**: Sprint- und Release Burn-Charts dienen der kontinuierlichen Überprüfung der Teamaktivitäten und der Fortschritte. Anhand dessen kann der Projektfortschritt überprüft werden.
- **Sprint**-**Review**: Für die Validierung der geleisteten Arbeit und deren Qualität definieren die Autoren den gelieferten Geschäftswert und die Kundenzufriedenheit als die Hauptindikatoren für die Messung im Sprint-Review.
- **Sprint**-**Retrospektive**: Dieses Event dient dazu den letzten Sprint zu analysieren. Als Indikatoren werden hierzu eine Leistungswertanalyse EVM mit den zentralen Kennwerten des Kosteneffizienz sowie Zeiteffizienz

$$Kosteneffizien = \frac{Fertigstellungswert}{Istkosten}$$

$$Zeiteffizient = \frac{Fertigstellungswert}{Plankosten}$$

für den Erfolg aus Sicht des Managements und die Abfrage der Arbeitszufriedenheit des Teams für die Organisationsweiterentwicklung empfohlen. (R. Kurnia et al., 2018)

3 DEFINITION VON PRODUKTIVITÄT IN DER AGILEN SOFTWARE ENTWICKLUNG

Damit agile Teams korrekt bewertet werden und auch die zuvor definierten Metriken in einen Bezug gebracht werden können, ist es notwendig, eine für diese Arbeit gültige Definition zu erstellen.

3.1 Geschichte der Produktivität in der Software Entwicklung

Laut Stefan Wagner und Florian Deissenböck gibt es bereits seit über 4 Jahrzehnten eine Diskussion über den Begriff der Produktivität in der Softwareentwicklung. Beide beschreiben, dass zu Anfang diese Diskussion hauptsächlich durch die Geschichten und Anekdoten von Fachexpert*Innen und anerkannten Wissenschaftler*Innen aufgezeigt wurden.(Sadowski & Zimmermann, 2019)

3.2 Begriffsdefinition in der Literatur

Produktivität wird in unterschiedlichen Fachgebieten und Wissenschaftszweigen untersucht und definiert. Es gibt hier verschiedene Abstraktionsebenen und für die Software Entwicklung fehlt eine einheitliche Definition. (Sadowski & Zimmermann, 2019)

Ausgehend von Stefan Tangens Artikel „*Demystifying productivity and performance*“ lassen sich folgende Faktoren für Performanz als weiteres Feld als die reine Produktivität definieren. (Tangen, 2005)

3.2.1 Produktivität

Stefan Wagner und Florian Deissenböck beschreiben, dass es, obwohl es keine generell anerkannte Definition für Produktivität gibt, doch einen Konsens zu folgender Formel gibt:

$$Produktivität = \frac{Output}{Input}$$

Und das diese Formel insbesondere im zweiten Sektor, dem sogenannten Sekundärsekt bzw. industrieller Produktion einfach umzusetzen ist.(Chew, 1988; Sadowski & Zimmermann, 2019)

Im Bereich der Wissensarbeit merken aber verschiedene Autor*Innen an, dass es schwierig ist, die Produktivität klar zu messen und, dass unterschiedliche Parameter mit einbezogen werden müssen.

3.2.2 Rentabilität

Die Rentabilität ist gerade in der Software Entwicklung eine wichtige Komponente, um die Performanz eines Einzelnen oder Teams im Rahmen einer Firma zu messen. Die Rentabilität und Produktivität sind laut Wagner und Deissenböck zwei stark miteinander verflochtene Parameter. Sie beschreiben dafür die generelle Formel: (Sadowski & Zimmermann, 2019)

$$Rentabilität = \frac{Umsatz}{Kosten}$$

3.2.3 Performance

Wagner und Weissenböck beschreiben, dass der Begriff Performance noch weiter gefasst ist, als jener der Produktivität und Rentabilität. Die unterschiedlichen Steuerungs- und Messinstrumente betrachten verschiedene Faktoren, um die Performance des Unternehmens zu messen. (Sadowski & Zimmermann, 2019)

Ein weitverbreitetes klassisches Performancemesssystem ist die sogenannte Balanced Scorecard, die vier Perspektiven mit unterschiedlichen Zielen und Messgrößen in Bezug vergleicht.

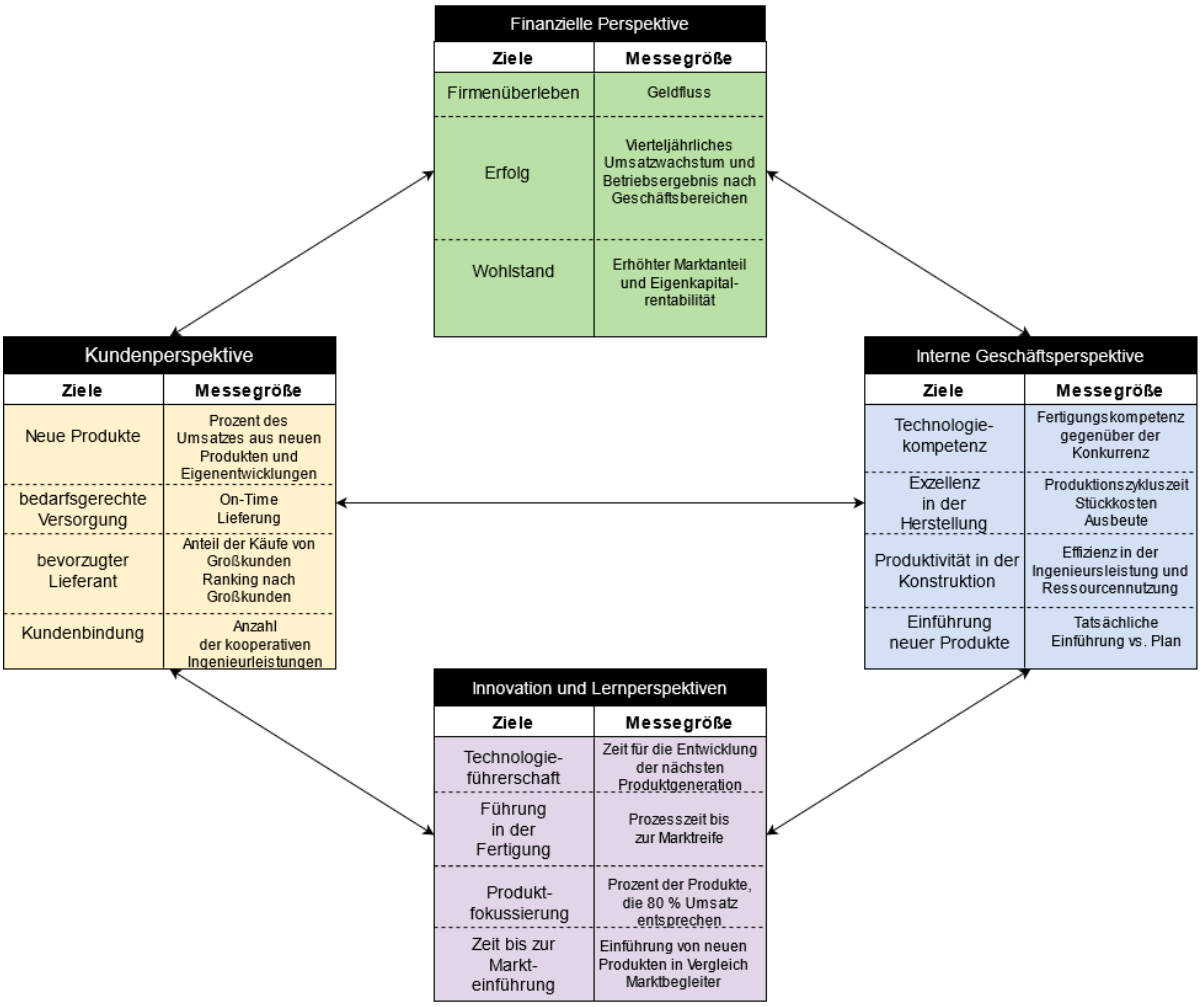

Abbildung 8: Balance Scorecard basierende auf (Kaplan & Norton, 1992) ©Harald Beier

Im Artikel "*Pointing out the gap between academic research and supporting software tools in the domain of the performance measurement management of engineering projects*" werden unterschiedliche Systeme zur Messung von Performance in zwei Perioden unterteilt: jene von 1989-2001 und jene ab 2002 bis jetzt. Die erste Phase war geprägt durch die Entwicklung von Systemen mit einer Balance zwischen finanziellen Messparametern und nicht finanziellen. Fünf der erfolgreichsten Modelle neben der Balanced Scorecard werden in der Tabelle 4 dargestellt. (Zheng et al., 2016)

Tabelle 4: Tabelle der klassischen PMSs basierend auf (Zheng et al., 2016, S. 1562)

PMSs: Klassische Performance Messsysteme (1989-2021)					
Name des PMS Modell und Framework	**PPM**:	**PPS**:	**RDF**	**DPMS**	**PP**
Perspektive	Extern/Kosten; Extern/Nicht-Kosten; Intern/Kosten; Intern/Nicht-Kosten.	Vision; Markt, Finanzen; Kundenzufriedenheit, Flexibilität, Produktivität; Qualität, Lieferung, Durchlaufzeit, Abfall.	Ergebnisse--Wettbewerbsfähigkeit, finanzielle Leistung; Determinanten--Qualität, Flexibilität, Ressourcen und Innovation.	Ein externes Überwachungssystem; Ein internes Überwachungssystem; Ein Überprüfungssystem; Ein internes Einsatzsystem.	Stakeholder-Zufriedenheit; Strategien; Prozesse; Fähigkeiten; Stakeholder-Beitrag.
Hauptparameter	1.Leistungskennzahlen müssen aus der Strategie abgeleitet werden; 2.Leistungskennzahlen sind vertikal und horizontal integriert. 3.Leistungskennzahlen unterstützen das multidimensionale Umfeld. 4.Leistungskennzahlen basieren auf Kostenbeziehungen und Verhalten.	1.Unternehmensvision in den Fokus zu stellen. 2.Unternehmensstrategie mit dem operativen Geschäft zu verknüpfen. 3.Richtige Ausrichtung durch die vertikale und horizontale Ausrichtung sicherzustellen.	1.Berücksichtigung, dass Ergebnisse nachlaufende Indikatoren sind. 2.Determinanten sind vorlaufende Indikatoren. 3.Sorgfältige Definition der Leistungsindikatoren, die zum Erreichen des Leistungsziels erforderlich sind.	1.Annahme einer breiteren Definition für die Leistungsmessung. 2.Regelkreis, der Korrekturmaßnahmen einschließt. 3.Zahlreiche, miteinander verbundene Leistungsmessungen. 4.Überprüfungsmechanismus.	1.Idenfitikation Stakeholder 2. Strategie um die Stakeholder zufriedenzustellen. 3.Setzen Sie die Prozesse ein, um die Strategien umzusetzen. 4.Identifizieren Sie die Fähigkeiten, um die Prozesse zu betreiben. 5.Schlagen Sie die Wünsche und Bedürfnisse der Stakeholder vor.
Gemeinsame Merkmale	Ausgewogen, integriert, strategieorientiert, multiperspektivisch, dynamisch und Stakeholder-orientiert				

Im Artikel wird beschrieben, dass es ab 2002 eine breitere Ausrichtung der verschiedenen Performance Messsysteme mit einem Fokus auf Disziplinenübergreifends, Fallanalysen und Erweiterung der traditionellen Balance Scorecard, die bereits in Abbildung 8 ausführlich dargestellt wurde. Für die Erstellung einer nachvollziehbaren Matrix, werden die Modelle und deren Parameter in einer Tabelle dargestellt. Dazu wird differenziert zwischen den wissenschaftlichen Hauptbeiträgen pro Modell:

- **Visuelles Performance Measurement Management VPMM**: : Visuelle Strategie- und Leistungsmessungstechniken für Organisationen wurden durch den Artikel „*A system dynamics-based balanced scorecard to support strategic decision making***“** (Bititci et al., 2016) dem VPMM hinzugefügt. In der Tabelle werden die dazugehörigen Merkmale mit A gekennzeichnet. Identifikation der visuellen Managementfunktion wurden durch das Werk „*Visual management in construction“* (Tezel et al., 2010) dem VPMM hinzugefügt. In der Tabelle werden die dazugehörigen Merkmale mit B gekennzeichnet.
- **Systeme zur Messung der Projektleistung PPMSs**: Ein mehrdimensionales System zur Messung der Projektleistung wurde durch den Artikel „*Towards a multi-dimensional project Performance Measurement System***“** (Lauras et al., 2010) entwickelt.
- **Supply-Chain Leistungsmessung Management SCPMM** : Das SCPMM wird um den Aspekt des mehrdimensionales Systems zur Messung der Projektleistung durch den Artikel **„***A framework for supply chain performance measurement*“ (Gunasekaran et al., 2004) erweitert. In der Tabelle wird das dazugehörige Merkmal mit A gekennzeichnet. Die Untersuchung des grünen Lieferkettenmanagements auf Leistungsfähigkeit wurde durch den Artikel **„***Green supply chain management practices: Impact on performance*“ (Green et al., 2012) dem SCPMM hinzugefügt. In der Tabelle wird das dazugehörige Merkmal mit B gekennzeichnet.
- **Quantitative Modelle für PMSs QM-PMSs** : Der Blickwinkel der Leistungsverbesserung basierend auf einer Choquet-Integral-Aggregation wurde durch den Artikel „*Monitoring the improvement of an overall industrial performance based on a Choquet integral aggregation*“ (Berrah et al., 2008) für das QM-PMSs entwickelt.
- **IT-PMSs Implementierung:** Die Komponenten der Überwachung des erweiterten Unternehmensbetriebs wurden mit Hilfe von KPIs und der Entwicklung eines Performance Dashboard (Busi & Strandhagen, 2004) in den IT-PMSs hinzugefügt. In der Tabelle wird das dazugehörige Merkmal mit A gekennzeichnet. Überprüfung der Faktoren bei der Einführung von Performanzmanagementsysteme und die Überprüfung von Schlüsselfaktoren währenddessen wird durch „*Performance plumbing: Installing performance management systems to deliver lasting value*“ (Meekings et al., 2009) dem SCPMM hinzugefügt. In der Tabelle werden die dazugehörigen Merkmale mit B gekennzeichnet.

Die weiteren Merkmale der unterschiedlichen PMSs werden zu leichterer Übersichtlichkeit als Tabelle dargestellt:

Tabelle 5: Tabelle der modernen PMSs basierend auf (Zheng et al., 2016, S. 1562–1563)

Modell	Merkmale
VPPM	A. Durchgängige visuelle Strategie- und Performance-Management-Ansätze werden für Fallunternehmen vorgeschlagen und als effektiv befunden.
	B. Basierend auf der Identifizierung der Hauptfunktionen des visuellen Managements in verschiedenen Disziplinen wird eine Idee zur Vervollständigung eines visuellen Management-Rahmens für Bauorganisationen vorgeschlagen.
PPMSs	Es konzentrierte sich auf 3 besondere Achsen für die Analyse der Projektleistung: Projektaufgabe, Leistungsindikator-Kategorien und eine Aufschlüsselung des Leistungs-Triptychons (Effektivität, Effizienz, Relevanz).
SCPMM	A. Es berücksichtigt die vier Hauptaktivitäten der Lieferkette: Planen, Beschaffen, Herstellen/Montieren und Liefern jede Aktivität besteht aus Metriken, die auf strategischer, taktischer und operativer Ebene klassifiziert sind.
	B. Es wird ein umfassendes Leistungsmodell für grüne Lieferkettenmanagementpraktiken vorgeschlagen und empirisch bewertet;
QM-PMSs	Es wurde eine Methode zur Quantifizierung der kausalen Beziehung zwischen den verschiedenen Kriterien entwickelt, die auf einem Choquet-Integral-Aggregationsoperator basiert.
IT-PMSs Implementierung	A. Es kombinierte die Konzepte von KPIs, Dashboards und IKT zur Unterstützung eines erweiterten Enterprise Performance Managements für selbst entwickelte Software.
	B. Es umfasst 4 Schlüsselelemente - Performance-Architektur, Performance-Einblicke, Performance-Fokus und Performance-Aktion mit Suggesting Commodity Software zur Unterstützung der Implementierung des Performance Measurement Framework.

3.2.4 Effizienz und Effektivität

Die Begriffe Effizienz und Effektivität sind zwei Begriffe, die oft im normalen Sprachgebrauch falsch verwendet werden und miteinander verwechselt (Tangen, 2005) Die informelle Unterscheidung, die am meisten angewandt wird, ist hierbei „*Effizienz ist, Dinge richtig zu tun*“ und „*Effektivität ist, die richtigen Dinge zu tun*“ (Sink & Tuttle,

1995). Exemplarisch hier zwei Varianten Effizienz und Effektivität messbar zu machen, sind zum einen die Variante von Jackson: (Jackson, 2000)

$$Effizienz = \frac{ideale\ systemabhängige\ Zeit}{Gesamtzeit}$$

$$Effektivität = \frac{Wertschöpfungszeit}{ideale\ systemabhängige\ Zeit}$$

Eine weitere etwas ältere Definition, die bereits den Ablauf Charakter berücksichtig, ist jene von Sink und Turtle, nachfolgend als Abbildung dargestellt. (Sink & Tuttle, 1995)

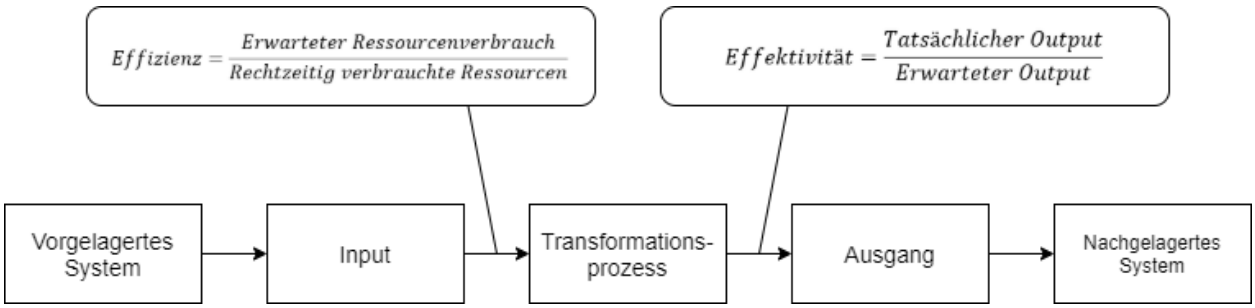

Abbildung 9: Prozessmodell mit Effizienz- und Effektivität-Definition und Einordnung basierend auf (Sink & Tuttle, 1995) ©Harald Beier

Eine Definition, die in unterschiedlichen Ausformungen in der wissenschaftlichen Literatur vorliegt, ist, dass sich die Effizienz auf die Ausnutzung der Ressourcen bezieht und insbesondere den erforderlichen Einsatz der Produktivitätsverhältnisse in den Fokus nimmt. Effektivität hat als Hauptaugenmerk die Betrachtung Nützlichkeit und Angemessenheit des Outputs, da sie direkte Auswirkungen auf den Kunden hat. (Sadowski & Zimmermann, 2019)

3.2.5 Qualität

Laut Wagner und Deissenboeck sollte Qualität der primäre Faktor für die Bewertung der Produktivität von Wissensarbeiter*Innen sein. Es sollte die Prima Causa der Bewertung sein und erst danach die Frage, welche Quantität abgearbeitet wurde. Sie stellen zudem fest, dass in den Disziplinen, die nicht mit Softwareentwicklung zusammenhängen, die Frage nach der Qualität für die Entwicklung von Produktivitätskennzahlen oft vernachlässigt wird. Sie betonen, dass es noch kein tragfähiges und allgemein anerkanntes operationalisierbares Konzept gibt, die Qualität in der Bestimmung der Produktivität einzubeziehen. (Sadowski & Zimmermann, 2019)

3.3 Produktivität in der Softwareentwicklung

Für die Messung der Software-Produktivität ist eine Messung von Input und Output eines Softwareprojekts notwendig. Der Input ist der Aufwand, der für seine Entwicklung benötigt wird. Der Output ist der Wert der Software für ihre Benutzer*Innen oder Kund*Innen. Diese Messung sollte

anhand des Geschäftsziels erfolgen und die Frage beantworten, wie gut erfüllt die Software ihren Zweck im Bezug auf funktionale und nicht-funktionale Anforderungen? Ein Beispielsmodell wird hier von Wagner und Deissenboeck entwickelt:

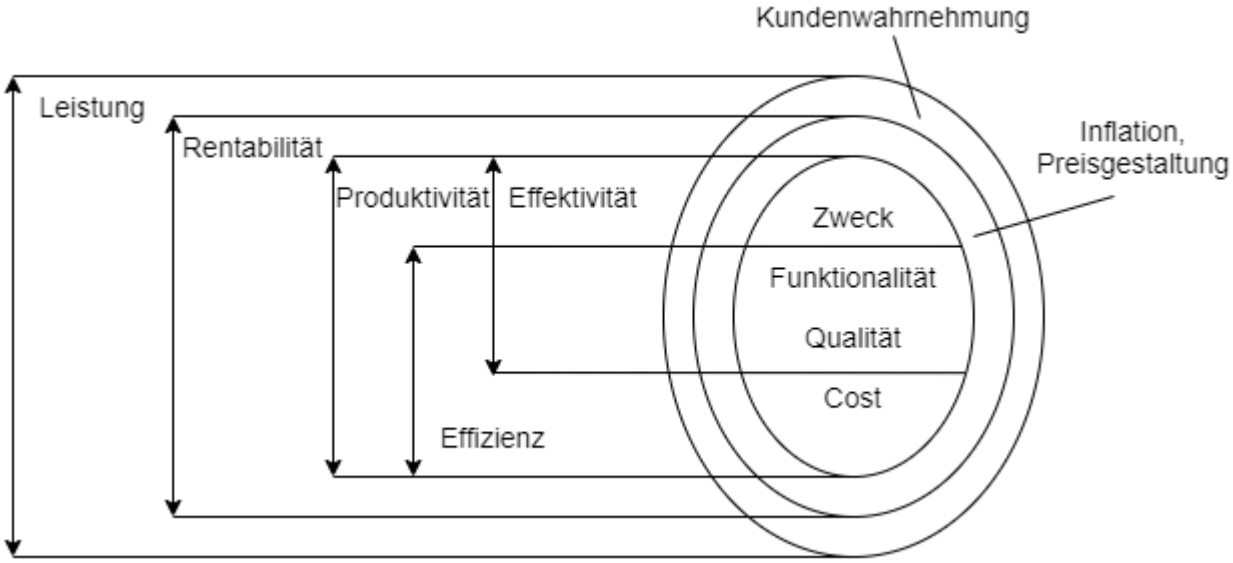

Abbildung 10: Leistungsmessungs-Modell basierend auf (Sadowski & Zimmermann, 2019) ©Harald Beier

Der Vergleich der idealen mit der tatsächlich produzierten Funktionalität und Qualität zeigt die Effektivität der Softwareentwicklungsaktivitäten; das Verhältnis des idealen zum tatsächlichen Aufwand ergibt die Effizienz. (Sadowski & Zimmermann, 2019)

Im Artikel "*Towards an evidence-based theoretical framework on factors influencing the software development productivity*" führen die Autoren eine Metastudie und erstellen folgende Liste von Parametern, die laut Ihrer Analyse mit der Produktivität in der Softwareentwicklung verknüpft sind: (Chapetta & Travassos, 2020)

- Generell gilt: Je signifikanter die Zunahme der LOC-basierten Größe eines Produkts im Laufe der Zeit, desto größer ist der positive Einfluss auf die Produktivität der Softwareentwicklung.
- Die Volatilität der Anforderungen wirkt sich negativ auf die Produktivität der Softwareentwicklung aus.
- Ein höheres Maß an personeller Kompetenz wirkt sich positiv auf die Produktivität der Softwareentwicklung aus.
- Die Erfahrung der Entwickler im Softwareentwicklungsprozess hat einen schwachen positiven Einfluss auf die Produktivität der Softwareentwicklung.
- Das Domänenwissen des Entwicklers wirkt sich positiv auf die Produktivität der Softwareentwicklung aus.
- Neuere Programmiersprachen tragen positiv mehr zur Produktivität der Softwareentwicklung bei als ältere Sprachen.
- Die Verfügbarkeit und Strategien für die adäquate Zuweisung von Entwicklern wirken sich positiv auf die Softwareproduktivität aus.
- Kommunikation unter Entwicklern wirkt sich positiv auf die Produktivität der Softwareentwicklung aus und ist für die Sicherstellung der Produktivität notwendig.

- Höhere Reifegrade von Softwareprozessen können effektiver zur Produktivität der Softwareentwicklung beitragen als niedrigere Reifegrade.
- Je dynamischer und neuer die Technologien im jeweiligen Geschäftsfeld sind, für die eine Software erstellt werden soll, desto geringer ist die Produktivität der Softwareentwicklung.
- Die Bearbeitung komplexer Artefakte während der Software-Entwicklung beeinträchtigt deren Produktivität nur geringfügig oder gar nicht.
- Die verstärkte Wiederverwendung von Anforderungen, Analyse und Designmustern führt zu Produktivitätsgewinnen in der Softwareentwicklung.
- Der kombinierte Einsatz von Methoden und Techniken zur Verifikation, Validierung, zum Testen oder zu anderen Aspekten der Qualitätssicherung hat positive Auswirkungen auf die Software-Produktivität.
- Die Erhöhung der Anzahl der Entwickler im Laufe der Zeit hat einen leicht negativen Effekt auf die Produktivität der Softwareentwicklung.

Die Autoren stellen aber fest, dass es sich nur um einen ersten Versuch handelt, um einen klaren Rahmen für die Definition der Produktivität von Softwareentwicklung zu erstellen und ausgehend davon sollen weitere Untersuchungen stattfinden. (Chapetta & Travassos, 2020)

Ausgehend von COCOMO werden im Artikel "*An Analysis of Trends in Productivity and Cost Drivers over Years"* 341 Projekte auf die 22 Parameter untersucht und diese nach Ihrer Auswirkung auf die Fertigstelltung gelistet (Nguyen et al., 2011):

Tabelle 6: Korrelationskoeffizienten zwischen dem Fertigstellungsjahr und Kostentreibern (sortiert nach Grad der Korrelation) basierend auf (Nguyen et al., 2011)

Faktoren	Kostentreiber	Kürzel	Kendall`s τ	p-Wert
Projekt-Faktoren	Einsatz von Software-Tools.	TOOL	-0.37/	2.20E-16
Skalenfaktoren	Prozess-Reifegrad.	PMAT	-0.30	1.22E-13
Plattform-Faktoren	Hauptspeichereinschränkung.	STOR	-0.29	1.31E-11
Plattform-Faktoren	Ausführungszeit-Beschränkung.	TIME	-0.26	6.62E-10
Personal-Faktoren	Plattform Erfahrung.	PLEX	-0.17	1.98E-05
Plattform-Faktoren	Plattform-Volatilität.	PVOL	-0.18	2.04E-05
Personal-Faktoren	Erfahrung mit gleichen Applikationen.	APEX	0.17	4.88E-05
Personal-Faktoren	Erfahrung mit den verwendeten Tools und Softwaresprache.	LTEX	-0.15	2.84E-04
Produkt-Faktoren	Datenbankgröße.	DATA	0.13	1.81E-03
Produkt-Faktoren	Notwendiger Verlässlichkeits-/Ausfallsgrad der Software.	RELY	-0.10	1.42E-02
Produkt-Faktoren	Produktkomplexität.	CPLX	-0.10	1.58E-02
Skalenfaktoren	Erstmalige Programmierung der Applikation.	PREC	-0.09	2.13E-02
Personal-Faktoren	Analysekapazität.	ACAP	0.08	4.87E-02
Projekt-Faktoren	Erforderlicher Zeitplan für die Entwicklung	SCED	-0.09	5.49E-02

Produkt-Faktoren	Anpassung der Dokumentation an den/ die Anforderungen im Produkt/Softwarelebens-zyklus.	DOCU	-0.07	8.29E-02
Produkt-Faktoren	Entwickelt für Wiederverwendbarkeit.	RUSE	-0.07	8.81E-02
Skalenfaktoren	Risikoauflösung.	RESL	-0.06	1.64E-01
Personal-Faktoren	Programmierer Fähigkeit.	PCAP	0.05	2.40E-01
Projekt-Faktoren	Multi-Site-Entwicklung.	SITE	-0.04	3.52E-01
Skalenfaktoren	Flexibilität in der Entwicklung.	FLEX	-0.04	3.78E-01
Skalenfaktoren	Zusammenhalt des Teams.	TEAM	0.01	7.66E-01
Personal-Faktoren	Kontinuität im Personalbereich.	PCON	0.01	7.93E-01

Tabelle 6 bildet die Kendall's Rank Korrelationskoeffizienten zwischen dem Fertigstellungszeitpunkt und den Kostentreibern ab. Die Kostentreiber sind in der Reihenfolge ihres Korrelationsgrades mit dem Fertigstellungszeitpunkt dargestellt. Der p-Wert zeigt den Test der Nullhypothese, dass die Schätzung des Korrelationskoeffizienten statistisch gleich Null ist. Der p-Wert zeigt den Test der Nullhypothese, dass die Schätzung des Korrelationskoeffizienten statistisch gleich Null ist. Das Vorzeichen des Kendall'schen τ-Wertes impliziert die Richtung der Korrelation. Dreizehn der zweiundzwanzig Kostentreiber wiesen eine statistisch signifikante Korrelation mit dem Fertigstellungszeitpunkt auf (pWert < 0,05). Dazu gehören ein Projektfaktor (TOOL), zwei Skalenfaktoren (PMAT, PREC), drei Produktfaktoren (DATA, CPLX, RELY), vier Personalfaktoren (PLEX, APEX, LTEX, ACAP) und alle drei Plattformfaktoren (TIME, STOR, und PVOL). Von diesen hat TOOL die stärkste Korrelation ($|\tau| = 0{,}37$), gefolgt von PMAT, STOR und TIME ($|\tau| \geq 0{,}26$). Alle außer APEX, DATA und ACAP haben negative Korrelationen mit dem Jahr der Fertigstellung, was auf die Abnahme der gesamten numerischen Bewertungsskalen dieser Kostentreiber bei den Projekten im Laufe des Jahres hinweist. Die Autoren halten fest, dass es Limitationen der Aussagekraft gibt, durch den Fokus auf die Kostentreiber nach dem COCOMO Modell, und, dass dadurch möglicherweise Faktoren nicht berücksichtigt wurden. Da es sich hierbei um eine Studie mit Daten seit 1970 handel,t hat diese Studie dennoch durch die Größe des Datensets und der den langen historischen Querschnitt starke Aussagekraft. (Nguyen et al., 2011) Für die Berechnung der Kosten eines Software Projekts unter der Berücksichtigung der Kostentreiber des COCOMO II hat Ray Madachy ein Onlinekalkulatorentwickelt, das unter der Homepage http://softwarecost.org/tools/COCOMO/ genutzt werden kann, es ist zudem möglich hier die in Kapitel 2.4 beschriebenen Function Points oder SLOC zu nutzen. (Ray Madachy, 2021)

4 SYNERGIE VON PRODUKTIVITÄT UND SOFTWAREMETRIKEN

Die vorhergehenden Kapitel haben gezeigt, dass Produktivität in der Software Entwicklung noch keine allgemeingültige Definition hat, deswegen wird für den Rahmen dieser Arbeit auf das Modell von Wagner und Deissenboeck (Sadowski & Zimmermann, 2019) zurückgegriffen, unter der Berücksichtigung der vorhanden Erweiterungen und Implekationen von Chapetta & Travassos (Chapetta & Travassos, 2020). Da hier von den agilen Frameworks das Hauptaugenmerk auf Scrum gelegt wurde und der Fokus dieser Arbeit der Bewertung und Weiterentwicklung von agilen Teams ist, bietet es sich an, Software Metriken mit den Scrum Events der Sprint Reviews und der Retrospektive zu koppeln. Aufschluss dazu gibt die Arbeit **„***Software Metrics Classification for Agile Scrum Process***“** die Autoren stellen hier folgende Parameter zu tabellarischer Aufschlüsselung zu Verfügung (R. Kurnia et al., 2018):

Tabelle 7: Sprint Review Softwaremetriken basierend auf (R. Kurnia et al., 2018)

Software Metrik	Messprozess	Beschreibung
Anzahl der beim Systemtest gefundenen Fehler	Eine Gesamtanzahl an Fehlern, die während des Systemtests gefunden wurden, nachdem diese durch Fehlerbeseitigungstechniken reduziert wurden, wie z. B.: testgetriebene Entwicklung, Pair Programming, Peer Review.	Eine im Systemtest gefundene Gesamtanzahl an Fehlern zur Messung der Codequalität, die an den Systemtestprozess geliefert wird.
Fehlerkorrekturzeit vom neuen zum geschlossenen Zustand	Nicht klar definiert in der Primärstudie.	Zeit, die benötigt wird, um die Fehler zu lösen, die während einem Sprint entdeckt wurden.
Gelieferter Geschäfts-/Mehrwert	Gemessen in Form von Story Points, Anzahl der Storys oder einer anderen abstrakten Maßeinheit zur Beschreibung des erreichten Geschäftswerts.	Messung, welchen Wert das Unternehmen einer einzelnen Funktion oder Story beimisst.
Kundenzufriedenheit	Basierend aufgrund der Produkt-/Funktionsdemonstration.	Qualitative Befragung der Kundenzufriedenheit basierend auf vorher definierten Indikatoren.
Aufgeschobene Mängel	Gemessen anhand der Anzahl an Mängel bzw. Fehler, die vom Kunden bei der Nutzung entdeckt wurden.	Die Anzahl an Mängeln, die bis zum Release entdeckt aber nicht gelöst wurden.
Mängel/Fehler pro Iteration	Anzahl aller Mängel/Fehler, die während eines Sprints entstanden sind.	Mängel/Fehler, die gefunden wurden und im nächsten Sprint nicht mehr auftauchen sollten
Fristgerechte Lieferung	Das Verhältnis von Funktionen, die innerhalb der geplanten Zeitspanne geliefert wurden.	Diese Softwaremetrik zeigt den Kundenmehrwert anhand der gelieferten Softwarefunktionen.

Mängel/Fehlerdichte	Anzahl an Fehler, die im Sprint gefunden wurde, Anzahl an Fehler, die von den Endnutzer nach dem Release gemeldet wurden, Gesamte Anzahl an fehlerhaften Codezeilen (LOC).	Die Mängel/Fehlerdichte wird anhand der Summe aller Produktbacklog Einträge (PBI), die im Sprint bzw. Release enthalten waren, erstellt.
Faktor „Fokus“	$Fokus = \frac{Velocity}{Arbeitskapazität}$	Das Verhältnis zwischen Velocity und Arbeitskapazität. Ein gutes Verhältnis ist hier für Teams circa 80%.
Erfüllung des Lieferumfangs	Das Verhältnis zwischen der Anzahl der tatsächlich implementierten PBIs und der Anzahl der zugesagten PBIs.	Zeigt, wie stark das Team den vereinbarten Lieferumfang während des Sprint Planning erfüllt hat.
Anzahl der Stories	Diese Metrik wird als einfache Zählung oder Gewichtung nach Story-Komplexität berechnet, z. B. einfach, mittel und komplex sowie der Anzahl der Stories im Sprint.	Dient der Messung des Projektfortschrittes basierend auf der Anzahl der akzeptierten User Stories.
Index für den Schweregrad offener Mängel/Fehler	Wird am Ende des Sprints gemessen und dann in der Retrospektive besprochen, um dies in Zukunft zu vermeiden.	Messen der Qualität bei jedem Feature innerhalb einer Iteration.
Prozentsatz der zusätzlich angenommenen Arbeitspakete	$\frac{\sum Originalschätzungen\ der\ ang.Arbeitspaket}{Schätzungsprognose\ für\ den\ Sprint}$	Angenommene Arbeit ist die neue Arbeit, die aus dem Product Backlog zum Sprint hinzugefügt wird, weil das Team bereits den Sprint Backlog bearbeitet hat.
Zusätzlich gefundene Arbeitspakete	$\frac{\sum Orginalschätzung\ der\ gefunden\ Arbeitspakte}{Schätzungsprognose\ für\ den\ Sprint}$	Zusätzliche Arbeit sind Arbeitspakete, die nicht erwartet wurden und während eines Sprints dem Sprint Backlog hinzugefügt werden.
Fortschrittsdiagramm (Scrum-Board)	Zeigt den Projektfortschritt nach Aufgabenstatus (nicht gestartet, in Bearbeitung und abgeschlossen) am Scrum Board an.	Ein Werkzeug, um den Fortschritt der Teammitglieder zu verfolgen.
Unit-Test-Abdeckung für entwickelten Code	Die Messung zeigt, wieviel Prozent des im Sprint entstandenen Codes bzw. der Softwarefunktionen mit Unit-Test abgedeckt sind.	Bezieht sich durch die Testabdeckung auf die Prüfung der Relation zwischen erwarteter Funktion und gelieferten Ergebnissen.
Arbeitskapazität	Der Wert der Arbeitskapazität sollte gleich oder größer sein als die Velocity.	Die Anzahl der Aufgaben, die während des Sprints beendet wurden, in Bezug auf den fertiggestellte Funktionsumfang bzw. ob diese nicht fertiggestellt wurden.

Anhand dieser Tabelle ist ersichtlich, dass in den letzten Jahren auch in Scrum unterschiedliche Metriken neben der Velocity Einzug gehalten haben und von anderen agilen Frameworks Inspiration bezogen wird. Insbesondere im Bereich des Testens und der Testabdeckung ist hier ein immer stärkeres Bedürfnis nach qualitativer Softwareentwicklung mit geringer Fehlermarge erkennbar. Im nächsten Kapitel werden zusätzliche Arbeiten von Autor*Innen bezüglich Agile Performance Indikatoren für Teams untersucht.

4.1 Agile Performance Indikatioren für agile Teams

Die Autor*Innen von "*Agile Performance Indicators for Team Performance Evaluation in a Corporate Environment"* definieren als agile Metriken (Ertaban et al., 2018):

- Produktion – Velocity
- Produktion – Durchlaufzeit und Zykluszeit
- Produktion – Vermeidung von Prozess"abfällen"
- Qualität – Anzahl von Mängeln/Fehlern
- Qualität – Fehlerdichte
- Kundenzufriedenheit – Anzahl von Kunden

Hier ist bereits eine große Schnittmenge mit den in Kapitel 4 am Anfang angeführten Metriken erkennbar. Sie schlagen zudem vor, Ziele für die Teams und die Individuen zu erstellen, zur Bewertung der einzelnen Entwickler*Innen. Die vorgeschlagene Berechnung erfolgt folgendermaßen:

- X sind Teamziele mit einer Gewichtung von 70% mit einer Skala von 1-10
- Y sind Individualziele mit einer Gewichtung von 30% mit einer Skala von 1-10

$$Kombiniertes\ Ergebnis = (X * 0{,}7) + (Y * 0{,}3)$$

Diese Berechnung legt einen Fokus auf Teamwork und sollte regelmäßig durchgeführt werden. (Ertaban et al., 2018)

"*Measuring Productivity in Agile Software Development Process: A Scoping Study*" definiert neun Produktivitätsdimensionen (Umfang, Kosten, Zeitdisziplin, Autonomie, Effizienz, Qualität, Effektivität, Projekterfolg, Kundenzufriedenheit) die wiederum sehr ähnlich jener der vorherigen Autoren sind. Sie definieren zudem Produktivitätsmetriken, die in der Literatur vorkommen und ordnen diese der jeweiligen Gruppengröße der Wissenarbeiter*Innen zu (Shah et al., 2015):

Tabelle 8: Produktivitätsmetriken basierend auf (Shah et al., 2015)

Produktivitätsmetrik	Wissenarbeiterorganisation
$Zeilen\ ausführbaren\ Code/MitarbeiterIn - Tag$	Team
$Function\ Points/MitarbeiterIn - Monat$	Team
$LOC/Personenstunden$	Team
$LOC/Stunden$	Team
Durchschnittliche Anzahl von nicht angepassten Functionpoints, die pro Zeiteinheit abgeschlossen werden	Entwicklungsteam von 2 Entwickler*Innen
$Anzahl\ der\ abgeschlossenen\ Aufgaben/Monat$	Pro Entwickler*In
$Funktionale\ Größe/Aufwand$	Team
$Function\ points/Monate$	Pro Entwickler*In

Im Artikel **"***Use of Software Metrics in Agile Software Development Process***"** identifizieren die Autoren 10 verschiedene Metriken, die sowohl im traditionellen Softwareentwicklungsprozess (TSD) genutzt wurden, als auch in der Agilen Softwareentwicklung (ASD) geeignet sind, im iterativen und inkrementellen Entwicklungsprozess als Metriken verwendet zu werden. Ausgehend von dieser Analyse erstellen sie eine Empfehlung von Metriken für ASD-Prozesse. Diese Metriken können Ihrer Analyse auch entweder direkt oder indirekt in JIRA oder Greenhopper abgebildet werden. (Padmini et al., 2015)

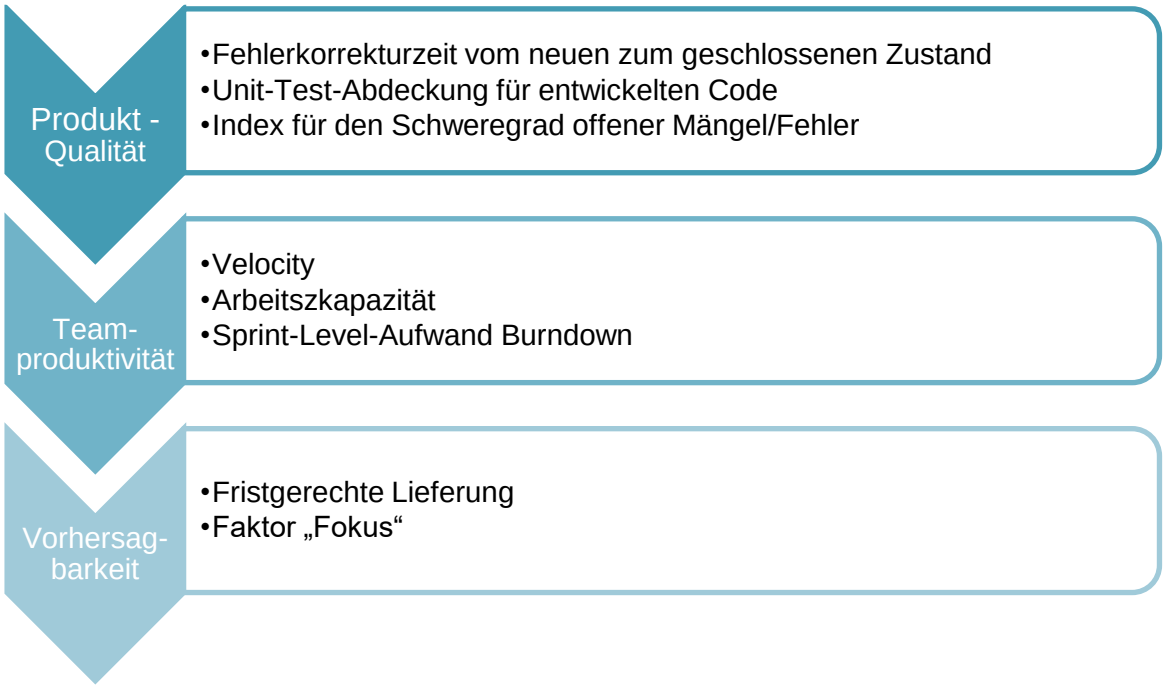

Abbildung 11: Empfohlene Metriken zur Verwendung im ASD-Prozess basierend auf (Padmini et al., 2015)

Da die Darstellung Padmini sowohl auf einer Metastudie der verwendeten Metriken als auch einer Befragung von Anwender basiert, fasst sie Produktivitätsmetriken und deren Dimension gut zusammen, die für den Zweck dieser Arbeit als Basis verwendet wird.

4.2 Monitoring Methoden für Produktivität

Nachdem verschiedene Metriken etabliert wurden, ist es wichtig zu definieren wie die Metriken am einfachsten und übersichtlichsten im Blick behalten werden können. Die Autoren des Artikels "*Effective Monitoring of Progress of Agile Software Development Teams in Modern Software Companies - An Industrial Case Study*" haben zu diesem Zweck gemeinsam mit 9 agilen Teams ein Dashboard entwickelt und die verschiedenen Metriken für Ihren Nutzen des Monitoring überprüft. Folgende Parameter sind laut den Autoren für das Monitoring am besten geeignet (Meding, 2017):

- Defect Backlog: gemessen als Anzahl der offenen Mängel/Fehler
- Feature-Backlog: gemessen als Burn-down und Burn-up-Chart
- Integration: gemessen als Anzahl der geplanten und erfolgreich ausgeführten Tests und der Integrationsgeschwindigkeit.

5 POTENZIELLE MASSNAHMEN ZU WEITERENTWICKLUNG AGILER TEAMS UND TEAMMITGLIEDER

Die vorhergehenden Kapitel haben klar gemacht, dass Software-Produktivität ein mehrdimensionales Konzept ist, dass als Indiz für den Projekterfolg verwendet werden kann und soll. Für Software-Teams ist die Produktivität daher ein kritischer Aspekt, der berücksichtigt werden muss. Unterschiedliche Faktoren werden in der Literatur, wie oben dargestellt, für die Produktivität in der Software Entwicklung genannt. Für die Bedeutung der Faktoren innerhalb der Teams und der Teamarbeit wird hier die Untersuchung der Autoren im Artikel An „*Empirical Analysis of the Effect of Agile Teams on Software Productivity*“ herangezogen. Diese haben 52 agile Software Entwicklungsunternehmen in Pakistan untersucht. Folgende Ergebnisse wurden dabei erhoben: (Institute of Electrical & Electronics Engineers, 2019)

Tabelle 9: Korrelation zwischen agilen Teamfaktoren und Produktivität basierend auf (Institute of Electrical & Electronics Engineers, 2019)

Faktor	Unterfaktoren	Korrelations-koeffizient (r)	Signifikanz (p-Wert < .05)
Teammitglieder	Die Rolle der Teammitglieder.	0.02	0.87
Teamleiter*In (TL)	TL leitet Besprechungen.	-0.14	0.31
	TL weist Aufgaben zu.	0.16	0.24
Beziehungen innerhalb des Teams	Zusammenarbeit im Team.	0.08	0.56
	Teammitglieder.	0.21	0.12
	Ermächtigung/Befähigung des Teams.	0.31*	ß-ß2
	Selbstorganisation des Teams	0.10	0.46
	Teamübergreifende Koordination der Zuständigkeiten für die Zielerreichung.	0.34*	0.01
	Teameffektivität.	0.39*	0.00
	Team kennt den Product Owner*In.	0.22	0.11
	Anforderungen als User Stories.	0.38*	0.00

Behandlung von Anforderungen durch das Team	Beteiligung bei der Erhebung der Anforderungen.	0.42*	0.00
	Funktionen sind klar beschrieben und entwicklungsbereit. Erfüllen DOR.	0.35*	0.00
Team Velocity	Die Team Velocity.	0.20	0.14
	Messung der Velocity für jede User Story/Iteration.	0.23	0.08
Qualitätskonformität durch das Team	Testfälle werden im Vorfeld mit den Anforderungen/der User Story geschrieben.	0.33*	0.01
	Automatisierte Unit-Tests.	-0.15	0.28
	Automatisierte Build- und Regressionstests.	-0.12	0.28
	Alle Code-Änderungen sind reversibel, und es ist möglich, einen Release jederzeit durchzuführen.	0.13	0.34
	Integrationstests über den gesamten Software-Lebenszyklus.	0.30*	0.02
Teamvision	Erfasste Hindernisse in der Arbeit.	0.02	0.87
	DONE bedeutet auslieferbar an den Kunden.	0.14	0.30
	Das Team verwendet ein Whiteboard für den Fortschritt.	0.15	0.26

Die Autoren verwenden für Ihre Studie eine Spearman´s Rangordnungs-Korrelation, um die Faktoren von agilen Teams auf die Produktivität zu untersuchen. Die Ergebnisse zeigen eine positive Korrelation für fast alle agilen Teamfaktoren. Einige der kategorisierten Sub-Faktoren sind signifikant mit der Produktivität korreliert, wie Teamermächtigung. Auch die Faktoren Testfälle (r = 0,33 % und p = 0,01) und Integrationstests (r = 0,30 % und p = 0,02) sind signifikant mit der Produktivität korreliert. (Institute of Electrical & Electronics Engineers, 2019)

5.1 Teamweiterentwicklung

Die Weiterentwicklung von agilen Teams in der Software Entwicklung ist ein fortlaufendes Unterfangen. Im Artikel "*Characterizing Software Engineering Work with Personas Based on Knowledge Worker Actions"* beschreiben die Autor*Innen, dass eine bessere Leistungsfähigkeit der agilen Teams erreicht werden kann, wenn nicht ein One-Size-Fits-All-Modell verwendet wird, sondern Software Entwickler*Innen entsprechend ihrer Fähigkeiten und Arbeitsstile eingesetzt werden. Sie beschreiben eine Liste von 12 Tätigkeiten (Lernen, Analysieren, Autorenschaft, Co-Autorenschaft, Veröffentlichung von Wissen, Experten-Suche, Feedback, Informationsorganisation, Informationssuche, Überwachung, Netzwerken, Service Suche) und führen mit 21 Personen ein qualitatives Interview und erstellen einen Fragebogen. Dieser Fragebogen wird von 868 Personen ausgefüllt. Anhand der Neigung zu den 12 Tätigkeiten entwickeln sie 7 Cluster und Personas (Ford et al., 2017). Bei der Weiterentwicklung von Teams und der Nutzung der zuvor beschriebenen Software- und Produktivitätsmetriken erscheint es dem Autor sinnvoll, diese Neigungen von Softwareentwickler*Innen mit einzubeziehen.

Für die Weiterentwicklung von agilen Teams ist es sinnvoll, auch die Perspektive der Entwickler*Innen miteinzubeziehen. Die Autoren von „*Software Developers' Perceptions of Productivity*" haben dazu eine Befragung mit 379 Software Entwickler*Innen durchgeführt und definieren anhand der Ergebnisse folgende Top 5 Parameter, wie Produktivität wahrgenommen und wie Sie von Software Entwickler*Innen am liebsten gemessen wird (Meyer et al., 2014) :

Tabelle 10: Tabelle der Metriken für persönliche Produktivität basierend auf (Meyer et al., 2014)

Gründe für einen produktiven Arbeitstag		
Grund	Anzahl Befragter	% der Befragten
Abgeschlossene Aufgabe/Erreichtes Ziel	192	53.2
Wenige Unterbrechungen und Störungen	182	50.4
Keine Meetings	79	21.9
Klare Ziele und Anforderungen sind definiert	72	19.9
Vorgeplanter Arbeitstag	62	17.2
Produktive Aktivitäten		
Aktivität	Anzahl Befragter	% der Befragten
Coding (Neue Funktionen, Testen, Bugfixen, Reviews)	236	71.5
Meeting	57	17.3
Planung	25	7.6
Lesen/Schreiben von Dokumentation und Berichten	22	6.7
Design oder Modellierung von Software Architektur	18	5.5

Unproduktive Aktivitäten		
Aktivität	Anzahl Befragter	% der Befragten
Meetings	191	57.9
Lesen/Schreiben von Emails	62	18.8
Ungeplante Arbeiten (z.B. Lösung von Problemen)	58	17.6
Coding (Testing,Debugging,Maintenance)	47	14.2
Lesen/Schreiben von Dokumentation und Berichten	25	7.6
Was die Teilnehmer*Innen messen wollen		
Metrik	Anzahl Befragter	% der Befragten
Aktivitäten (wie viel Zeit wurde womit verbracht)	67	27.0
Leistungen (tatsächlich geleistete Arbeit, Fortschritt)	44	17.7
Wert (Erfolg der Funktion, Wert für den Kunden)	41	16.5
Zeit pro Aufgaben	39	15.7
Anzahl der Kontextwechsel und Ablenkungen	36	14.5

Die Autoren halten fest, dass Entwickler*Innen eine hohe Anzahl von Wechseln bei ihrer Arbeit erleben, indem sie alle 6,2 Minuten die Aufgaben und alle 1,6 Minuten die Aktivitäten wechseln. Entwickler*Innen fühlen sich trotzdem durch den unterschiedlichen Arbeitsaufwand, der mit diesen unterschiedlichen Formen von Kontextwechseln verbunden sind, produktiv. (Meyer et al., 2014)

6 ZUSAMMENFASSUNG UND AUSBLICK

Das abschließende Kapitel fasst die Ergebnisse der gefundenen Metriken und Mechanismen für die Bewertung der Produktivität von agilen Teams zusammen, um die Forschungsfrage „*Welche Metriken und Mechanismen existieren und wie werden diese angewandt, um agile Teams zu bewerten und fördern?“* zu beantworten. Des Weiteren findet sich im zweiten Abschnitt ein Ausblick für eine weitere Arbeit.

6.1 Zusammenfassung

Unterschiedliche agile Metriken und Frameworks haben zur Erweiterung der Metriken und Mechanismen agiler Teams beigetragen. Insbesondere XP mit seinem Fokus auf Qualität, IT-Kanban mit der Spezialisierung der Weiterentwicklung des Arbeitsprozesses und Lean Software Development mit der Gewichtung der Vermeidung von unnötigen Arbeitsschritten und Abfall haben zusätzliche Gesichtspunkte bei den Metriken gebracht.

Die Definition der Produktivität in der Softwareentwicklung benötigt einen multifaktoriellen Ansatz, der unterschiedliche Parameter wie Effizienz, Effektivität und Leistung berücksichtigt. Das Modell von Wagner und Deissenboeck, das in Kapitel 3.3 vorgestellt wurde, erscheint hier vielversprechend.

Abhängig vom gewählten agilen Framework und den vorhandenen Kapazitäten macht es Sinn sich bei der Messung von Produktivität zu entscheiden, welcher Detailgrad innerhalb der Organisation benötigt wird. Entweder es sind die konzentrierten Metriken und Dimensionen in Abbildung 11 ausreichend, oder es werden die umfassenderen auf Scrum-Ereignisse bezogenen Metriken, die in Tabelle 7 dargestellt werden, benötigt.

Bei der Weiterentwicklung von Teams und Individuen sind die Parameter der persönlichen Präferenz, sowie der Vermeidung von Ablenkungen und Kontextwechsel zu berücksichtigen. Es ist außerdem sinnvoll, das Hauptaugenmerk auf die Entwicklung der Parameter mit der höchsten Korrelation in Tabelle 9 zu legen.

6.2 Ausblick

Basierend auf den gewonnenen Ergebnissen der Literaturrecherche und Analysesollte ein gewichteter Kriterienkatalog zu Bewertung von Team- und individuellen Leistungen von Software Entwickler*Innen erstellt werden. Dieser Fragebogen sollte Anhand qualitativer Interviews von Softwareleiter*Innen erstellt und die Berechnung und Gewichtung der verschiedenen Faktoren anhand der Erfahrungswerte und Rückmeldungen definiert werden. Anhand des Erfüllungsgrades der einzelnen Kriterien kann dann eine Formel für die Produktivitäts- bzw. Leistungserfassung erstellt werden.

ABKÜRZUNGSVERZEICHNIS

A

ACM
Association for Computing Machinery 2

AM
Agile Modeling 4

ASD
Adaptive Software Development 9
Agile Software Development 10, 31

AT
Agile Testing 4

ATDD
Acceptance test-driven development 4

AUP
Agile Unified Process 9

B

BDD
Behavior-driven development 4

BL
Backlogs 4

C

CF
Crystal Family 9

CFT
Cross-Functional Team 4, 5

CI
Continous integration 4

COCOMO
Constructive Cost Model 26

D

DAD
Disciplined Agile Delivery 12

DDD

Domain-driven development 4

Devops

Development und IT Operations 5

DoT

Definition of Test 5

DPMS

Dynamic PMS 21

DST

Daily Stand-Up 4

E

EVM

Earned Value Management 18

F

FDD

Feature Driven Development 8

I

IEEE

Institute of Electrical and Electronics Engineers 2

IID

Iterative and incremental development 4

IT-Kanban

Kanban in der IT 8

L

LeanSAFE

Lean Scalable Aigility for Engineering 12

LeSS

Large-Scale Scrum 12

LOC

Line of Code 28

LSAF

Large Scale Agile Frameworks 9

LSD

Lean Software Development 9

P

PBI
Product Backlog Item 28

PIP
Planning Poker 4

PMSs
Performance Measurement Systems 21

PP
Performance Prism 21

PPM
Performance Measurement Matrix 21

PPMSs
Project Performance Measurement Systems 22

PPr
Pair Programming 4

PPS
Performance Pyramid System 21

Q

QM-PMSs
Quantitative Models for Performance Measurment Systems 22

R

RDF
Result and Determinants Framework 21

RF
Refactoring 4

RS
Retrospective 4

RUP
Rational Unified Process 11

S

SAFe
Scaled Agile Framework 12

SBE
Specification by example 4

SCPMM
Supply-Chain Performance Measurment Management 22

SDM

Story-driven modeling 4

SLOC

Source Lines of Code 17

SoS

Scrum of Scrums 12

T

TB

Timeboxing 4

TDD

Test-driven modeling 4

TL

TeamleiterIn 32

TSD

Traditional Software Development 31

V

VPMM

Visual Performance Measurement Management 22

VT

Velocity Tracking 4

W

WIP

Work in Progress 9

X

XP

Extreme Programming 8

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Abdelaziz, A. A., El-Tahir, Y. & Osman, R. (2015). Adaptive Software Development for developing safety critical software. In R. A. Saeed & R. A. Mokhtar (Hg.), *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE): 7th-9th September 2015, Khartoum, Sudan : proceedings* (S. 41–46). IEEE. https://doi.org/10.1109/ICCNEEE.2015.7381425

Ahmad, M. O., Dennehy, D., Conboy, K. & Oivo, M. (2018). Kanban in software engineering: A systematic mapping study. *Journal of Systems and Software*, *137*, 96–113. https://doi.org/10.1016/j.jss.2017.11.045

Alnoukari, M., Alzoabi, Z. & Hanna, S. (2008). Applying adaptive software development (ASD) agile modeling on predictive data mining applications: ASD-DM methodology. In H. B. Zaman (Hg.), *Cognitive informatics: bridging natural and artificial knowledge: Proceedings ; International Symposium of Information Technology 2008, [ITSim '08], Kuala Lumpur Convention Centre, Malaysia, August 26 - 29, 2008* (S. 1–6). IEEE. https://doi.org/10.1109/ITSIM.2008.4631695

Al-Sabbagh, K. W. & Gren, L. (2018). The connections between group maturity, software development velocity, and planning effectiveness. *Journal of Software: Evolution and Process*, *30*(1), e1896. https://doi.org/10.1002/smr.1896

Alshayeb, M. & Li, W. (2006). An empirical study of relationships among extreme programming engineering activities. *Information and software technology*, *48*(11), 1068–1072. https://doi.org/10.1016/j.infsof.2006.01.005

Ambler, S. W. (2004). *The object primer: Agile model-driven development with UML 2.0* (3rd ed.). Cambridge Univ. Press. http://www.loc.gov/catdir/samples/cam041/2004040400.html

Anderson, D. J. (2010). *Kanban: Successful evolutionary change for your technology business*. Blue Hole Press.

Bache, E. & Bache, G. (2014). Specification by Example with GUI Tests - How Could That Work? In G. Cantone (Hg.), *Lecture Notes in Business Information Processing: Bd. 179, Agile processes in software engineering and extreme programming: 15th International Conference, XP 2014, Rome, Italy, May 26-30, 2014 ; proceedings* (S. 320–326). Springer.

Beecham, S., Clear, T., Lal, R. & Noll, J. (2021). Do scaling agile frameworks address global software development risks? An empirical study. *Journal of Systems and Software*, *171*, 110823. https://doi.org/10.1016/j.jss.2020.110823

Berrah, L., Mauris, G. & Montmain, J. (2008). Monitoring the improvement of an overall industrial performance based on a Choquet integral aggregation. *Omega*, *36*(3), 340–351.

Binamungu, L. P., Embury, S. M. & Konstantinou, N. (2020). Characterising the Quality of Behaviour Driven Development Specifications. In V. Stray, R. Hoda, M. Paasivaara & P. Kruchten (Hg.), *Lecture Notes in Business Information Processing, Agile Processes in Software Engineering and Extreme Programming: 21st International Conference on Agile Software*

Development, XP 2020, Copenhagen, Denmark, June 8–12, 2020, Proceedings (1. Aufl., S. 87–102). Springer International Publishing; Imprint: Springer.

Bititci, U., Cocca, P. & Ates, A. (2016). Impact of visual performance management systems on the performance management practices of organisations. *International Journal of Production Research*, *54*(6), 1571–1593. https://doi.org/10.1080/00207543.2015.1005770

Blasquez, I. & Leblanc, H. (2017, Juli). Specification by Example for Educational Purposes. In R. Davoli (Hg.), *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (S. 212–217). Association for Computing Machinery. https://doi.org/10.1145/3059009.3059039

Busi, M. & Strandhagen, J. O. (2004). Monitoring extended enterprise operations using KPIs and a performance dashboard. In *Second World Conference on POM and 15th Annual POM Conference, Cancun, Mexico*.

Carvalho, R. A. de, Carvalho e Silva, F. L. de, Manhães, R. S. & Oliveira, G. L. de (2013). Implementing Behavior Driven Development in an Open Source ERP. In G. Poels (Hg.), *Lecture Notes in Business Information Processing: Bd. 139, Enterprise information systems of the future: 6th IFIP WG 8.9 Working Conference, CONFENIS 2012, Ghent, Belgium, September 19 - 21, 2012 ; revised selected papers* (S. 242–249). Springer.

Chapetta, W. A. & Travassos, G. H. (2020). Towards an evidence-based theoretical framework on factors influencing the software development productivity. *Empirical Software Engineering*, *25*(5), 3501–3543. https://doi.org/10.1007/S10664-020-09844-5 (Empirical Software Engineering).

Chew, B. W. (1988). No-Nonsense Guide to Measuring Productivity. *Harvard Business Review*(66), 110–115. https://hbr.org/1988/01/no-nonsense-guide-to-measuring-productivity

Chowdhury, A. F. & Huda, M. N. (2011). Comparison between Adaptive Software Development and Feature Driven Development. In *2011 International Conference on Computer Science and Network Technology (ICCSNT 2011): Harbin, China, 24 - 26 December 2011* (S. 363–367). IEEE. https://doi.org/10.1109/ICCSNT.2011.6181977

Christou, I., Ponis, S. & Palaiologou, E. (2010). Using the Agile Unified Process in Banking. *IEEE Software*, *27*(3), 72–79. https://doi.org/10.1109/MS.2009.156

Cockburn, A. (2005). *Crystal clear: A human-powered methodology for small teams*. *The agile software development series*. Addison-Wesley.

Cohn, M. (2013). *User stories applied: For agile software development* (18. Aufl.). *Addison-Wesley signature series*. Addison-Wesley.

Conboy, K. & Carroll, N. (2019). Implementing Large-Scale Agile Frameworks: Challenges and Recommendations. *IEEE Software*, *36*(2), 44–50. https://doi.org/10.1109/MS.2018.2884865

Crispin, L. & Gregory, J. (2009). *Agile testing: A practical guide for testers and agile teams* (1. Aufl.). *A Mike Cohn signature book*. Addison-Wesley.

Diebold, P. & Zehler, T. (2015). The agile practices impact model: idea, concept, and application scenario. In *ICSSP 2015, Proceedings of the 2015 International Conference on Software*

and System Process (S. 92–96). Association for Computing Machinery. https://doi.org/10.1145/2785592.2785609

Dingsøyr, T. (2005). Postmortem reviews: purpose and approaches in software engineering. *Information and software technology*, *47*(5), 293–303. https://doi.org/10.1016/j.infsof.2004.08.008

Dinwiddie, G. (2009). Feel the Burn: Getting the Most out of Burn Charts. *Better Software,* 11,(5), S. 26–31. http://idiacomputing.com/pub/BetterSoftware-BurnCharts.pdf

Ebert, C. & Paasivaara, M. (2017). Scaling Agile. *IEEE Software*, *34*(6), 98–103. https://doi.org/10.1109/MS.2017.4121226

Ertaban, C., Sarikaya, E. & Bagriyanik, S. (2018). Agile Performance Indicators for Team Performance Evaluation in a Corporate Environment. In *XP '18, Proceedings of the 19th International Conference on Agile Software Development: Companion.* Association for Computing Machinery. https://doi.org/10.1145/3234152.3234156

Feature-Driven Development. (2006). In J. Hunt (Hg.), *Agile software construction* (S. 161–182). Springer. https://doi.org/10.1007/1-84628-262-4_9

Fojtik, R. (2011). Extreme Programming in development of specific software. *Procedia Computer Science*, *3*, 1464–1468. https://doi.org/10.1016/j.procs.2011.01.032

Gause, D. C. & Weinberg, G. M. (1989). *Exploring requirements: Quality before design*. Dorset House Pub.

Granulo, A. & Tanovic, A. (2019). Comparison of SCRUM and KANBAN in the Learning Management System implementation process. In *TELFOR 2019: 27. Telekomunikacioni Forum : Beograd, 26. i 27. novembar 2019. godine, Sava Ts̆entar = TELFOR 2019 : 27th Telecommunications Forum : Belgrade, 26 and 27 November 2019, the SAVA Center* (S. 1–4). Telecommunications Society; Academic Mind. https://doi.org/10.1109/TELFOR48224.2019.8971201

Green, K. W., Zelbst, P. J., Meacham, J. & Bhadauria, V. S. (2012). Green supply chain management practices: impact on performance. *Supply Chain Management: An International Journal*, *17*(3), 290–305. https://doi.org/10.1108/13598541211227126

Gunasekaran, A., Patel, C. & McGaughey, R. E. (2004). A framework for supply chain performance measurement. *International Journal of Production Economics*, *87*(3), 333–347. https://doi.org/10.1016/j.ijpe.2003.08.003

Hill, P. R. (Hg.). (2011). *Practical software project estimation: A toolkit for estimating software development effort & duration*. McGraw-Hill.

Hodgkins, P. & Hohmann, L. (2007). Agile Program Management: Lessons Learned from the VeriSign Managed Security Services Team. In J. Eckstein (Hg.), *Agile 2007: 13 - 17 August 2007, Washington, D.C* (S. 194–199). IEEE Computer Soc. https://doi.org/10.1109/AGILE.2007.11

Ibrahim Muhammad, Aftab Shabib, Bakhtawar Birra, Ahmad Munir, Iqbal Ahmed, Aziz Nauman, Javeid Muhammad Sheraz & Ihnaini Dr. Baha Najim Salman (2020). Exploring the Agile

Family: A Survey. *International Journal of Computer Science and Network Security*, *20*(10), 163–179. https://doi.org/10.22937/IJCSNS.2020.20.10.22

Institute of Electrical & Electronics Engineers, a. (2019). *An empirical analysis of the effect of agile teams on software productivity*. Institute of Electrical and Electronics Engineers.

Jackson, M. (2000). *An analysis of flexible and reconfigurable production systems: An approach to a holistic method for the development of flexibility and reconfigurability*. Zugl.: Linköping, Univ., Diss., 2000. *Linköping studies in science and technology Dissertations: Bd. 640*. Univ.

Jalote, P., Palit, A., Kurien, P. & Peethamber, V. T. (2004). Timeboxing: a process model for iterative software development. *Journal of Systems and Software*, *70*(1-2), 117–127. https://doi.org/10.1016/S0164-1212(03)00010-4

Janes, A. (2015). A guide to lean software development in action. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation workshops (ICSTW 2015): Graz, Austria, 13 - 17 April 2015 ; [proceedings* (S. 1–2). IEEE. https://doi.org/10.1109/ICSTW.2015.7107412

Janzen, D. & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer*, *38*(9), 43–50. https://doi.org/10.1109/MC.2005.314

Jonsson, H., Larsson, S. & Punnekkat, S. (2013). Synthesizing a Comprehensive Framework for Lean Software Development. In O. Demirors (Hg.), *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA): 4 - 6 Sept. 2013, Santander, Spain* (S. 1–8). IEEE. https://doi.org/10.1109/SEAA.2013.64

Kaplan, R. S. & Norton, D. P. (1992). The balanced scorecard: measures that drive performance. *Harvard Business Review*, *83*(7), 71–79. https://hbr.org/1992/01/the-balanced-scorecard-measures-that-drive-performance-2

Kasauli, R., Knauss, E., Horkoff, J., Liebel, G. & Oliveira Neto, F. G. de (2021). Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software*, *172*, 110851. https://doi.org/10.1016/j.jss.2020.110851

Kaur, S., Kaur, A. & Dhiman, G. (2021). Deep analysis of quality of primary studies on assessing the impact of refactoring on software quality. *Materials Today: Proceedings.* Vorab-Onlinepublikation. https://doi.org/10.1016/j.matpr.2020.11.217

Kaur, S. & Singh, P. (2019). How does object-oriented code refactoring influence software quality? Research landscape and challenges. *Journal of Systems and Software*, *157*, 110394. https://doi.org/10.1016/j.jss.2019.110394

Khalil, C., Fernandez, V. & Houy, T. (2013). Can Agile Collaboration Practices Enhance Knowledge Creation between Cross-Functional Teams? In P.-J. Benghozi, D. Krob & F. Rowe (Hg.), *Advances in intelligent systems and computing: Bd. 205, Digital enterprise design and management 2013: Proceedings of the First International Conference on Digital Enterprise Design and Management DED&M 2013* (S. 123–133). Springer.

Kocurek, D. (2011). Understanding of Burndown Chart. *Methods & Tools*, *19*(4), 25–34.

Komus, A., Kuber, M., Schmidt, S., Rost, L., Koch, C.-P., Bartnick, S., Graf, E., Keller, M., Linkenbach, Felix, Linkenbach, F., Pieper, C. & Weiß, L. (2020). *Study Status Quo (Scaled) Agile 2019/20.* Hochschule Koblenz, Koblenz.

Larman, C. & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer*, *36*(6), 47–56. https://doi.org/10.1109/MC.2003.1204375

Lauras, M., Marques, G. & Gourc, D. (2010). Towards a multi-dimensional project Performance Measurement System. *Decision Support Systems*, *48*(2), 342–353. https://doi.org/10.1016/j.dss.2009.09.002

Lehtinen, T. O., Virtanen, R., Viljanen, J. O., Mäntylä, M. V. & Lassenius, C. (2014). A tool supporting root cause analysis for synchronous retrospectives in distributed software teams. *Information and software technology*, *56*(4), 408–437. https://doi.org/10.1016/j.infsof.2014.01.004

Li, J. & Wang, X. (2010). Research and practice of agile Unified Process. In H. Kettani (Hg.), *2010 2nd International Conference on Software Technology and Engineering (ICSTE 2010): San Juan, Puerto Rico, USA, 3 - 5 October 2010.* IEEE. https://doi.org/10.1109/ICSTE.2010.5608794

Mahnič, V. & Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of Systems and Software*, *85*(9), 2086–2095. https://doi.org/10.1016/j.jss.2012.04.005

McConnell, S. (2009). *Code Complete* (2nd ed.). O'Reilly Media Inc. http://gbv.eblib.com/patron/FullRecord.aspx?p=488632

McDonough, E. F. (2000). Investigation of Factors Contributing to the Success of Cross-Functional Teams. *Journal of Product Innovation Management*, *17*(3), 221–235. https://doi.org/10.1111/1540-5885.1730221

McDowell, C., Werner, L., Bullock, H. & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. In J. Gersting, H. M. Walker & S. Grissom (Hg.), *Proceedings of the 33rd SIGCSE technical symposium on Computer science education - SIGCSE '02* (S. 38). ACM Press. https://doi.org/10.1145/563340.563353

Meding, W. (2017). Effective monitoring of progress of agile software development teams in modern software companies. In *IWSM Mensura '17, Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement* (S. 23–32). Association for Computing Machinery. https://doi.org/10.1145/3143434.3143449

Meekings, A., Povey, S. & Neely, A. (2009). Performance plumbing: installing performance management systems to deliver lasting value. *Measuring Business Excellence*, *13*(3), 13–19. https://doi.org/10.1108/13683040910984284

Miranda, E. (2011). Time boxing planning. *SIGSOFT Softw. Eng. Notes*, *36*(6), 1–5. https://doi.org/10.1145/2047414.2047428

Nguyen, V., Huang, L. & Boehm, B. (2011). An analysis of trends in productivity and cost drivers over years. In T. Menzies (Hg.), *Proceedings of the 7th International Conference on Predictive Models in Software Engineering* (S. 1–10). ACM Digital Library. https://doi.org/10.1145/2020390.2020393

Olsen, N. C. (1995). Survival of the fastest: improving service velocity [software products]. *IEEE Software*, *12*(5), 28–38. https://doi.org/10.1109/52.406754

Padmini, K. V. J., Dilum Bandara, H. M. N. & Perera, I. (2015). Use of software metrics in agile software development process. In A. G. B. P. Jayasekara (Hg.), *2015 Moratuwa Engineering Research Conference (MERCon 2015): Moratuwa, Sri Lanka, 7-8 April 2015* (S. 312–317). IEEE. https://doi.org/10.1109/MERCon.2015.7112365

Park, R. (1992). *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-020). Pittsburgh, PA. Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11689

Power, K. (2014). Definition of Ready: An Experience Report from Teams at Cisco. In G. Cantone (Hg.), *Lecture Notes in Business Information Processing: Bd. 179, Agile processes in software engineering and extreme programming: 15th International Conference, XP 2014, Rome, Italy, May 26-30, 2014 ; proceedings* (S. 312–319). Springer.

Pugh, K. (2011). *Lean-agile acceptance test-driven development: Better software through collaboration*. *Net objectives lean-agile series*. Addison-Wesley.

Qattous, H., Gray, P. & Welland, R. (2010). An empirical study of specification by example in a software engineering tool. In G. Succi (Hg.), *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (S. 1). ACM. https://doi.org/10.1145/1852786.1852808

R. Kurnia, R. Ferdiana & S. Wibirama (2018). Software Metrics Classification for Agile Scrum Process: A Literature Review. In *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*.

Rademacher, F., Sorgalla, J. & Sachweh, S. (2018). Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. *IEEE Software*, *35*(3), 36–43. https://doi.org/10.1109/MS.2018.2141028

Ray Madachy. (15. März 2021). *COCOMO II - Constructive Cost Model.* Naval Postgraduate School. http://softwarecost.org/tools/COCOMO/

Sadowski, C. & Zimmermann, T. (2019). *Rethinking Productivity in Software Engineering*. Apress; Imprint,Apress.

Schwaber, K. & Sutherland, J. (2020). *Der Scrum Guide: Der gültige Leitfaden für Scrum: Die Spielregeln*. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-German.pdf

Shah, S. M. A., Papatheocharous, E. & Nyfjord, J. (2015). Measuring Productivity in Agile Software Development Process: A Scoping Study. In *ICSSP 2015, Proceedings of the 2015 International Conference on Software and System Process* (S. 102–106). Association for Computing Machinery. https://doi.org/10.1145/2785592.2785618

Shahin, M., Ali Babar, M. & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

ShuiYuan, H., LongZhen, D., Jun, X., JunCai, T. & GuiXiang, C. (2009). A Research and Practice of Agile Unified Requirement Modeling. In Q. Luo (Hg.), *2009 International Symposium*

on Intelligent Ubiquitous Computing and Education: IUCE 2009 ; Chengdu, China, 15 - 16 May 2009 (S. 180–184). IEEE. https://doi.org/10.1109/IUCE.2009.17

Sink, D. S. & Tuttle, T. C. (1995). *Planning and measurement in your organization of the future* (4. Dr). Industrial Engineering and Management Press.

Stray, V., Sjøberg, D. I. & Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software*, *114*, 101–124. https://doi.org/10.1016/j.jss.2016.01.004

Svensson, R. B., Gorschek, T., Bengtsson, P.-O. & Widerberg, J. (2019). BAM - Backlog Assessment Method. In P. Kruchten, S. Fraser & F. Coallier (Hg.), *Lecture Notes in Business Information Processing, Agile Processes in Software Engineering and Extreme Programming: 20th International Conference, XP 2019, Montréal, QC, Canada, May 21–25, 2019, Proceedings* (S. 53–68). Springer International Publishing; Imprint: Springer.

Tangen, S. (2005). Demystifying productivity and performance. *International Journal of Productivity and Performance Management*, *54*(1), 34–46. https://doi.org/10.1108/17410400510571437

Tezel, B. A., Koskela, L. J., Tzortzopoulos, P. & others (2010). Visual management in construction: Study report on Brazilian cases.

Tolfo, C. & Wazlawick, R. S. (2008). The influence of organizational culture on the adoption of extreme programming. *Journal of Systems and Software*, *81*(11), 1955–1967. https://doi.org/10.1016/j.jss.2008.01.014

Usman, M., Mendes, E. & Börstler, J. (2015). Effort Estimation in Agile Software Development: A Survey on the State of the Practice. In *EASE '15, Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering.* Association for Computing Machinery. https://doi.org/10.1145/2745802.2745813

Wautelet, Y., Heng, S., Kiv, S. & Kolp, M. (2017). User-story driven development of multi-agent systems: A process fragment for agile methods. *Computer Languages, Systems & Structures*, *50*, 159–176. https://doi.org/10.1016/j.cl.2017.06.007

Zheng, L., Baron, C., Esteban, P., Xue, R., Zhang, Q. & Gomez Sotelo, K. I. (2016). Pointing out the gap between academic research and supporting software tools in the domain of the performance measurement management of engineering projects. *IFAC-PapersOnLine*, *49*(12), 1561–1566. https://doi.org/10.1016/j.ifacol.2016.07.802

Zündorf, A. (2008). Story driven modeling. In B. S. Lee (Hg.), *Proceedings of the 2nd international workshop on Scalable stream processing system* (S. 714). ACM. https://doi.org/10.1145/1062455.1062632