Masterthesis

Bestimmung und Analyse von Kriterien zur optimalen Verteilung von Verantwortlichkeiten in der agilen Softwareentwicklung anhand einer Fallstudie am Beispiel von Collective Code Ownership

Erstellt von:

Joscha Nassenstein

Eingereicht am:

11. April 2022

Executive Summary

Einer der zentralen Aspekte agiler Methoden in der Softwareentwicklung ist die Selbstorganisation des Teams. Dabei ist es für einzelne Beteiligte jedoch meist nicht möglich, einen detaillierten Überblick über die gesamte Entwicklung zu haben. In diesem Zusammenhang wurden Ansätze entwickelt, welche auch die Verantwortung nicht an einzelne Entwickler:innen zuweisen, sondern als kollektive Aufgabe definieren. So entstand im Rahmen der Extreme Programming Methode das Prinzip des Collective Code Ownerships. Durch dessen Umsetzung soll erreicht werden, dass für alle Mitwirkenden die Möglichkeit besteht, ihre Ideen und Fähigkeiten in diversen Segmenten des Projekts einbringen zu können. Weitere Vorteile werden dadurch beschrieben, dass weniger Engpässe während der Entwicklung entstehen, der Austausch untereinander sowie ein einheitlicher Entwicklungsstil gefördert wird und ein gemeinsames Verständnis der geforderten Qualität existiert. Auf der anderen Seite werden hohe Anforderungen an die Entwickler:innen gestellt, da für die einzelnen Komponenten einer Software spezifische Fähigkeiten gefordert sind und die Umsetzung nur mit entsprechendem Wissen zu bewältigen ist. Die Anwendung des Prinzips birgt das Risiko, den Erfolg eines Projektes und die Qualität des Produkts negativ zu beeinflussen.

Im Rahmen einer quantitativen Fallstudie wurden solche Auswirkungen des Collective Code Ownerships untersucht. Dabei wurden Daten eines Entwicklungsprojekts in Bezug auf die Qualität der Software, Risiken in der Entwicklung und die Expertise der Entwickler:innen ausgewertet. Aus Sicht der Softwarequalität konnte festgestellt werden, dass in Bereichen mit vergleichsweise eindeutiger Verantwortung auch eine geringere Fehleranfälligkeit der Software vorlag. Hier konnten Ergebnisse einer anderen bereits abgeschlossenen Studie bestätigt werden. Gleichzeitig hat die Beteiligung weiterer Entwickler:innen nicht dazu geführt, dass vermehrt dort Fehler entstanden sind, wo diese etwas zum Quellcode beigesteuert haben. Stattdessen waren Fehler in den meisten Fällen in Bereichen verortet, welche unter Verantwortung der Person mit dem größten Anteil standen. Weiterführend konnte aus Sicht der Entwicklungsrisiken herausgestellt werden, dass Abhängigkeiten zu einzelnen Entwickler:innen in Bezug auf die verwendeten Technologien reduziert werden konnten, ohne dass es zu häufigeren Verzögerungen in der Entwicklung gekommen ist. Die Mehrheit der Entwickler:innen hat sich im Projektverlauf mit verschiedenen Themen auseinandergesetzt. Aus Sicht des Projektes kann daraus abgeleitet werden, dass Collective Code Ownership bei geringen Einbußen in der Softwarequalität dazu beitragen konnte, die fachliche Entwicklung der Beteiligten zu fördern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

ARM ist eine deklarative Konfigurationssprache mit dem Name Azure Resource Manager und die native Lösung zur Verwaltung von Ressourcen in der Microsoft Azure Cloud.[1]

AWS ist ein Tochterunternehmen von Amazon mit dem Namen Amazon Web Services, in der Branche des Cloud Computings tätig[2] und aktuell nach Umsatz der führende Anbieter in diesem Bereich.[3]

Azure von Microsoft ist wie auch AWS ein Anbieter im Cloud-Bereich[4] und erzielt nach letzterem aktuell den zweithöchsten Umsatz in diesem Bereich.[5]

Cloud oder Cloud Computing bezeichnet ein Modell, in welche Computerressourcen als Dienstleistung durch einen Drittanbieter zur Verfügung gestellt werden.[6] Die Konfiguration und der Zugriff erfolgt über das Internet. Aus Anwendersicht ist die Infrastruktur nicht statisch definiert, sondern dynamisch verteilt und kann softwareseitig verwaltet werden.[7]

CloudFormation ist eine deklarative Konfigurationssprache und die native Lösung zur Verwaltung von Ressourcen in der Cloud von AWS und somit das äquivalent zu ARM in der Azure Cloud.[8]

CloudWatch ist ein Service in der AWS-Cloud, welcher die Überwachung verschiedener Komponenten eines Softwareprodukts in der Cloud-Umgebung ermöglicht. Eine weitere wichtige Funktion ist die Erfassung von Protokolldaten und Metriken zu diesen Komponenten.[9]

Commit ist die Bezeichnung für den Schritt in einer Software zur verteilten Versionsverwaltung (wie beispielsweise Git), welcher die lokalen Änderungen und damit den aktuellen Stand der Dateien festhält. Im Hintergrund wird eine Momentaufnahme aller Daten abgespeichert.[10]

[1] vgl. Microsoft, 2022, online im Internet.
[2] vgl. Amazon Web Services, o. D., online im Internet.
[3] vgl. Statista Research Department, 2022, online im Internet.
[4] vgl. Microsoft, o. D., online im Internet.
[5] vgl. Statista Research Department, 2022, online im Internet.
[6] vgl. Mell und Grance, 2011, S. 2, online im Internet.
[7] vgl. Meyer, 2018, S. 101 f.
[8] vgl. Amazon Web Services, 2021a, online im Internet.
[9] vgl. Amazon Web Services, 2022, online im Internet.
[10] vgl. Chacon und Straub, 2014, S. 13.

Docker ist eine Plattform, welche verschiedene Lösungen für die Virtualisierung von Softwarekomponenten anbietet. Die Kernfunktion von Docker sind sogenannte Container, welche eine isolierte Ausführungsumgebung für diese Komponenten darstellen.[11]

DynamoDB ist eine dokumentenbasierte Datenbank in der AWS-Cloud.[12]

Git ist eine Software für die verteilte Versionsverwaltung. Sie ermöglicht es, Änderungen an Dateien sowie weitere Metadaten wie Zeitpunkte und Autor:innen der Änderungen zu erfassen und bei Bedarf zu einem späteren Zeitpunkt wiederherzustellen.[13] Durch die Erweiterung mit Lösungen wie beispielsweise GitLab ermöglicht sie zudem die Zusammenarbeit verschiedener Entwickler:innen an einem gemeinsamen Projekt.[14]

GitLab ist eine Plattform, welche grundlegend auf der Software Git aufbaut und diese um Funktionen zur Zusammenarbeit von Entwicklungsteams, der Bereitstellung von Software und dem Projektmanagement erweitert.[15]

IEEE ist eine technische Fachorganisation und beschreibt sich selbst als die Größte dieser Art. Sie hat über 400 000 Mitglieder und ist für die Publikation vieler Fachschriften und Standards sowie die Veranstaltung von Konferenzen und Events bekannt.[16]

ISO ist eine Nichtregierungsorganisation und bedeutet ausgeschrieben International Organization for Standardization. Sie ist in der Schweiz ansässig ist und beschäftigt sich mit dem Verfassen von internationalen Standards.[17]

Jira ist eine Software, welche vom Unternehmen Atlassian entwickelt und vertrieben wird. Sie bündelt verschiedene Funktionen im Bereich des agilen Projektmanagements. Für den hier vorgestellten Prozess ist das primäre Ziel des Einsatzes, den gesamten Entwicklungszyklus und dessen Planung zentral abzubilden.[18]

Kontinuierliche Integration bezeichnet die fortlaufende Bereitstellung von Weiterentwicklungen einer Software. Die Änderungen sollen dabei so früh wie möglich in ihre finalen Umgebung überführt werden.[19]

[11] vgl. Docker, 2022, online im Internet.
[12] vgl. Amazon Web Services, 2021b, online im Internet.
[13] vgl. Chacon und Straub, 2014, S. 12 f.
[14] vgl. GitLab, o. D., online im Internet.
[15] vgl. GitLab, o. D., online im Internet.
[16] vgl. IEEE, o. D., online im Internet.
[17] vgl. ISO, 2019, S. 3, online im Internet.
[18] vgl. Atlassian, o. D., online im Internet.
[19] vgl. Beck, 2005, S. 49 f.

Merge bezeichnet die Rückführung eines isolierten Arbeitszweigs in die gemeinsame Codebasis unter Git.[20] Dies kann durch in Softwareprodukten, welche auf Git aufbauen, beispielsweise mit Möglichkeiten zu Kommentaren und Code Reviews ergänzt werden.

Python ist eine objektorientierte Programmiersprache, welche den Fokus auf Lesbarkeit des Codes und Einfachheit der Nutzung legt, um eine schnelle Programmentwicklung zu ermöglichen.[21] Sie ist aktuell eine der beliebtesten und meistgenutzten Programmiersprachen.[22]

Terraform ist eine plattformübergreifende Konfigurationssprache zur Erstellung von Infrastruktur, welche Schnittstellen zu einzelnen Cloud-Anbietern wie AWS und Azure implementiert.[23]

TypeScript ist eine Programmiersprache, welche auf einer anderen Programmiersprache, JavaScript, aufsetzt und diese um Funktionen wie Typisierung erweitert und beispielsweise häufig bei der Webentwicklung eingesetzt wird.[24] Sie ist aktuell eine der beliebtesten und meistgenutzten Programmiersprachen.[25]

Virtualisierung bezeichnet eine Abstraktionsebene und die damit verbundenen Technologien, um eine physische Ausführungsumgebung in mehrere virtuelle Umgebungen aufzuteilen. Dadurch kann beispielsweise ein Betriebssystem innerhalb eines anderen ausgeführt werden.[26]

Vue.js ist ein Webframework, welches auf klassischen Webtechnologien aufbaut und die Erstellung von Benutzeroberflächen vereinfachen soll.[27]

Wissensarbeit bezeichnet Arbeit, welche nicht hauptsächlich durch körperliche Fähigkeiten, sondern durch den Einsatz von erworbenem Wissen bewältigt wird.[28]

[20] vgl. Chacon und Straub, 2014, S. 75.
[21] vgl. Python.org, o. D., online im Internet.
[22] vgl. Stackoverflow.com, 2021, online im Internet.
[23] vgl. HashiCorp, o. D., online im Internet.
[24] vgl. TypeScript, 2021, online im Internet.
[25] vgl. Stackoverflow.com, 2021, online im Internet.
[26] vgl. Meyer, 2018, S. 187.
[27] vgl. Vue.js, 2022, online im Internet.
[28] vgl. Drucker, 2008, S. XVII f.

1 Einleitung

Die Entwicklung moderner Software muss mit vielen Herausforderungen umgehen. Diese entstehen beispielsweise aus der globalen Verteilung der beteiligten Entwickler:innen, aus Anforderungen, welche sich erst im Verlauf eines Projektes entwickeln, oder schwierigen Entscheidungen, welche getroffen werden müssen. Der Entwurf und die Implementierung von Software ist nur mit entsprechendem technischen Wissen und Können zu bewältigen, welches sich stetig weiterentwickelt. Zusammengefasst lässt sich sagen, dass die Softwareentwicklung ein sehr komplexes Unterfangen ist.[29] Veranschaulichen lässt sich dies dadurch, dass in einer Auswertung zum Gesamtumfang der vom Unternehmen Google angebotenen Produkte 2015 geschätzt wurde, dass sich der dahinter stehende Quellcode auf einen Umfang von etwa 2 Milliarden Codezeilen beläuft.[30] Dieser Umfang liegt jedoch auch in den hohen Nutzerzahlen und der damit einhergehenden Diversifizierung der Anforderungen begründet.

Dementsprechend lässt sich ein Beispiel dieser Dimension nicht ohne weiteres auf andere Entwicklungen übertragen und die Zielsetzung eines Projektes kann sich in ganz verschiedenen Maßstäben bewegen. Trotzdem ist es für einzelne Entwickler:innen meist nicht mehr möglich, die Entwicklung einer Software in ihrer Gesamtheit zu überblicken.[31] Durch das Aufkommen agiler Softwaremethoden, welche den Übergang von einer plangetriebenen hin zu einer flexiblen Entwicklung als Ziel haben, hat sich auch der Fokus von den einzelnen Entwickler:innen hin zum Team als Ganzes gewandelt.[32] In diesem Kontext stellt sich daher die Frage, ob dieser Wandel auch in Bezug auf die Verantwortung über die Software und ihre Komponenten übertragen werden kann. Bereits im Jahr 1999 hat sich Kent Beck mit dieser Fragestellung beschäftigt und in seiner Veröffentlichung zur Methode Extreme Programming die Praktik der kollektiven Verantwortung vorgestellt, welcher er damals als „Collective Code Ownership“ betitelte. Diese sieht die Verantwortung des Quellcodes als Aufgabe des gesamten Teams und erlaubt den beteiligten Personen, zu jeder Zeit Änderungen am Quellcode vorzunehmen, wenn die Gelegenheit dazu gegeben ist und es in ihren Augen als sinnvoll erscheint.[33] Er grenzt dieses Prinzip von anderen Modellen ab, in denen einzelne Personen die Verantwortung tragen und auch bei Beiträgen anderer Entwickler:innen über

[29] vgl. Stober und Hansmann, 2010, S. 2.
[30] vgl. Metz, 2015, online im Internet.
[31] vgl. Booch, 1994, S. 3.
[32] vgl. Ahmed und Colomo-Palacios, 2021, S. 273.
[33] vlg. Beck, 1999, S. 71.

deren Integration bestimmen können.[34] Knapp 23 Jahre später werden die kollaborative Entwicklung und Verantwortung als wesentliche Fähigkeiten von Entwickler:innen im agilen Umfeld angesehen.[35]

Das Ziel dieser Thesis ist es, den Einfluss einer solchen Verteilung von Verantwortlichkeiten in agilen Softwareprojekten anhand einer Fallstudie zu untersuchen. Dabei geht es nicht um Verantwortlichkeiten im umschließenden Entwicklungsprozess, sondern um die technische Verantwortung in Bezug auf die Software selbst. Dies kann auch als die Verantwortung formuliert werden, die jede beteiligte Person für einzelne Teile des Quellcodes trägt. Ein Projekt kann durch klar zugewiesene, individuelle Verantwortlichkeiten organisiert, jedoch auch als eine kollektive Verantwortung definiert sein, bei der alle Entwickler:innen für die gesamte Codebasis zuständig sind.[36] Die Fallstudie wird dabei in einem Projektumfeld durchgeführt, in welchem das Prinzip des Collective Code Ownerships (kollektive Code-Verantwortung) angewandt wird. In diesem Zusammenhang soll untersucht werden, inwiefern sich die Anwendung des Prinzips auf verschiedene Aspekte der Softwareentwicklung auswirkt. Die entsprechende Forschungsfrage kann wie folgt formuliert werden:

> Welche Folgen hat das Anwenden des Collective Code Ownership Prinzips auf die Entwicklung von Software, insbesondere mit Blick auf Fehleranfälligkeit und Entwicklungsdauer sowie die Expertise des Entwicklungsteams, und welche Erkenntnisse lassen sich daraus für die Verteilung von Verantwortlichkeiten in agilen Softwareprojekten ableiten?

Neben dem Einfluss der kollektiven Verantwortung wird auch untersucht, ob sich das Prinzip ohne klar festgelegte Regeln selbst organisieren kann. Solche Regeln könnten beispielsweise die Aufgabenverteilung betreffen, um eine konstant gleichbleibende Beschäftigung der Entwickler:innen zu vermeiden. Ist es dem in der Fallstudie behandelten Projekt gelungen, die Entwickler:innen unabhängig von ihrer eigenen Expertise auf verschiedene Teilaspekte der Entwicklung aufzuteilen und so ihr Wissen zu erweitern? Oder lässt sich bei der Umsetzung trotz Anwendung des Prinzips eine Vielzahl individueller Verantwortungen feststellen? Diese Themen werden anhand einer Überprüfung von vorher aufgestellten Hypothesen erforscht.

Sowohl der Aufbau der Thesis, als auch die Methodik lassen sich entsprechend der Forschungsfrage in drei Bestandteile aufteilen. Zunächst wird anhand repräsentativer Literatur

[34] vgl. Beck, 2000, S. 59.
[35] vgl. Rimol, 2022, online im Internet.
[36] vgl. Beck, 2000, S. 59.

der aktuelle Stand zu den Themen der agilen Softwareentwicklung sowie der Zusammenarbeit und Verantwortung in Softwareentwicklungsprojekten dargestellt. Zusätzlich werden Kriterien aufgestellt, auf welche potenziell eine Einflussnahme durch die Verteilung von Verantwortlichkeiten möglich ist. Anschließend werden aus diesen Untersuchungen Hypothesen abgeleitet, anhand derer sich die Einflüsse der Kriterien bestätigen oder widerlegen lassen.

In den anschließenden Kapiteln werden diese Hypothesen in einer Fallstudie überprüft. Für Letztere wird eine quantitative Datenanalyse konzipiert und durchgeführt, um die aus der Theorie gewonnenen Erkenntnisse anhand der Daten eines realen Projektes weiter zu untersuchen. Bei dem Projekt handelt es sich um die Entwicklung einer Software bei der Bayer AG, welche in einen agilen Entwicklungsprozess organisiert ist und das Prinzip des Collective Code Ownerships verfolgt. Aus dem Bestätigen oder Widerlegen der Hypothesen können dann Erkenntnisse für die Verteilung von Verantwortlichkeiten in solchen Projekten abgeleitet werden. In diesem Teil erfolgt auch die Diskussion der Ergebnisse auf Grundlage der theoretischen Auseinandersetzung und der Analyse der Fallstudie. Ebenso werden Limitationen in der Datenanalyse aufgezeigt. Die Schlussbetrachtung enthält abschließend eine Zusammenfassung der wesentlichen Ergebnisse sowie ein Ausblick auf Möglichkeiten zur weiteren Vertiefung der Thematik.

2 Agile Softwareentwicklung

> „Nichts ist so nutzlos, wie mit großer Effizienz etwas zu tun, was gar nicht getan werden sollte."
>
> – Peter Drucker, 1963[37]

Zu Beginn einer Softwareentwicklung stehen die Entwickler:innen mitunter vor einer Herausforderung. Auf der einen Seite möchten sie einen umfassenden Blick auf das Projekt und seinen Abhängigkeiten haben, um ein Fundament für die anstehenden Planungsaktivitäten zu schaffen. Im Kontrast dazu besitzen diese Einflussfaktoren jedoch die Eigenschaft, teilweise erst während der Umsetzungsphase sichtbar zu werden. So kann es passieren, dass einzelne Einflüsse erst bekannt werden, wenn das Projekt kurz vor dem Abschluss steht.[38] Die im folgenden als „klassisch" bezeichneten Ansätze der Softwareentwicklung unterteilen den Entwicklungsprozess jedoch in solche Phasen, in denen erst nach Abschluss der Planung mit der Umsetzung begonnen wird.[39] Diese umfassen beispielsweise Modelle wie das Wasserfall- oder das V-Modell.[40]

Der Begriff der agilen Softwareentwicklung umfasst verschiedene Methoden, welche infolge solcher Unzulänglichkeiten im Einsatz klassischer Modelle entstanden sind. Sie gelten als eine Manifestation der wachsenden Bedeutung von Anpassungsfähigkeit in der Softwareentwicklung. Obwohl die mittlerweile weit verbreiteten Methoden sich voneinander unterscheiden, halten sie sich an gemeinsame Grundprinzipien. Neben der schnellen Anpassungsfähigkeit wird insbesondere Wert auf Einfachheit und Flexibilität im Entwurf und der Entwicklung von Software gelegt.[41] Bereits 1963 kritisierte Peter Drucker, der Urheber des am Anhang des Kapitels genannten Zitats, die Vorgehensweisen von Unternehmen, welche nach seiner Meinung den Fokus zu sehr auf Effizienz an Stelle von Effektivität legten.[42] Die agile Bewegung übernimmt diesen Grundsatz für die Entwicklung von Software. Dementsprechend werden die aus der Bewegung entstandenen Methoden nicht nur als Modelle, sondern eher als anpassbare Rahmenbedingungen angesehen.[43] Diese schaffen Voraussetzungen, um den Entwicklern und der Projektleitung ein pragmatisches Projektmanagement zu ermöglichen. Dazu werden

[37] übersetzt nach Drucker, 1963, S. 54: *„There is surely nothing quite so useless as doing with great efficiency what should not be done at all."*.
[38] vgl. Stober und Hansmann, 2010, S. 2.
[39] vgl. Sandhaus, Berg und Knott, 2014, S. 28.
[40] vgl. Sandhaus, Berg und Knott, 2014, S. 29–33.
[41] vgl. Maruping, Zhang und Venkatesh, 2009, S. 356.
[42] vgl. Drucker, 1963, S. 54 f.
[43] vgl. Sandhaus, Berg und Knott, 2014, S. 33.

in einer Art Minimalprinzip genau die Daten bereitgestellt, welche die Entscheidungsfindung unterstützen.[44]

Das Ziel der agilen Softwareentwicklung ist die Verbesserung der Produktqualität durch die Reduktion formaler Prozesse.[45] Dies ging aus der Erkenntnis hervor, dass sich extensive Planung in einer volatilen Projektumgebung mit veränderlichen Anforderungen zu einem Flaschenhals entwickelt, welcher den Fortschritt der Entwicklung ausbremst.[46] Solche Projekte sind dadurch charakterisiert, dass sie zwar „einen relativ klaren Auftrag haben, aber die spezifischen Anforderungen unbeständig sind und sich fortlaufend entwickeln können, da Kunden und Entwicklungsteams gleichermaßen das Unbekannte erforschen".[47] Ein zu Beginn präzise ausgearbeiteter Projektplan erweckt dabei lediglich den Eindruck, alles berücksichtigt zu haben, da sich grundlegende Annahmen jederzeit ändern können. Daher muss ein Projektplan in einer adaptiven Weise gestaltet sein, um auf nicht berücksichtigte Anforderungen reagieren zu können.[48]

Im folgenden Kapitel werden zunächst Grundlagen der agilen Softwareentwicklung dargestellt, um anschließend näher auf zwei Vertreter der agilen Methoden einzugehen und die dort definierten Grundsätze zu erläutern.

2.1 Grundlagen

Die Entstehung der Methoden und Prinzipien agiler Softwareentwicklung ging mit einem stetigen Wandel der Softwarebranche zu Beginn des 21. Jahrhunderts einher. Dieser kann durch den schnellen Aufstieg neuer Technologien und den konstanten Austausch zwischen Entwickler:innen begründet werden, welcher durch die globale Vernetzung im Internet angetrieben wurde. Die Erkenntnis über die neuen Potenziale führte zu hohen Investitionen in sowie Neugründungen und Zusammenschlüssen von Unternehmen. Als Konsequenz standen Projekte in diesen Bereichen oft vor umfassenden organisatorischen Veränderungen.[49]

[44] vgl. Martin, 2020, S. 33.
[45] vlg. Beck et al., 2001a, online im Internet.
[46] vgl. Stober und Hansmann, 2010, S. 5.
[47] übersetzt nach Highsmith, 2002, S. 4: *„Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown."*.
[48] vgl. Stober und Hansmann, 2010, S. 5.
[49] vgl. Schmidt, 2016, S. 14.

Aus Sicht der Software konnten jedoch ebenfalls einige Veränderungen festgestellt werden. Diese wurde vermehrt für den Endverbrauchermarkt bereitgestellt und stellte damit die Anforderungen an die Nutzerfreudlichkeit der Applikation in der Vordergrund. Die Rückmeldungen der Kunden sollten möglichst kurzfristig in den Entwicklungsprozess eingebunden werden, wodurch die Anforderungen an die Software nicht mehr vorhersehbar waren. Ein konstanter Strom an Änderungen wurde zur immanenten Charakteristik von Softwareentwicklungsprojekten und die Reaktion darauf ein Erfolgsfaktor in diesen Bereichen der Ungewissheit.[50] Man versuchte, mit kürzeren und flexibleren Produktlebenszyklen den neuen Herausforderungen entgegenzutreten. Daher bedurfte es neuer Methoden als Gegenstück zu den linearen und plangetriebenen Modellen, welche als zu statisch und daher ungeeignet für die Softwareentwicklung in solchen Projektumgebungen angesehen wurden.[51] Diese Methoden wurden ursprünglich als „leichtgewichtige Methoden“ bezeichnet.[52] Zu ihnen gehören beispielsweise die verbreiteten Methoden Scrum oder Extreme Programming (XP).[53] Dies bedeutet jedoch nicht, dass nicht bereits vorher ähnliche Herangehensweisen existierten. In der Anfangszeit der Softwareentwicklung können viele Beispiele für Verhaltensweisen gefunden werden, welche aus heutiger Sicht als „agil“ bezeichnet werden können. Als Beispiel dafür sind kurze Iterationen zu nennen, welche durch fortlaufende Tests der Software unterbrochen wurden.[54]

Um eine gemeinsame Basis als Gegenbewegung zu den phasenorientierten, dokumentationsgetriebenen und daher aus ihrer Sicht schwergewichtigen Softwareentwicklungsprozessen zu begründen, trafen verschiedene Vertreter der sich bereits verbreitenden agilen Methoden zusammen und verfassten das sogenannte „Agile Manifest“. Dieses fasst gemeinsame Grundsätze zusammen und gab der Bewegung letztlich ihren Namen.[55]

2.1.1 Agiles Manifest

Das Agile Manifest gilt als Grundstein für die agile Bewegung. Nachdem bereits einige auch heute noch verbreitete Methoden wie Scrum oder XP, welche im nächsten Abschnitt erläutert werden, entstanden waren, fand im Februar 2001 auf Einladung von Robert Martin ein Treffen

[50] vgl. Schmidt, 2016, S. 14 f.
[51] vgl. Schmidt, 2016, S. 14 f.
[52] vgl. Martin, 2020, S. 29, 31.
[53] vgl. Schmidt, 2016, S. 15.
[54] vgl. Martin, 2020, S. 23.
[55] vgl. Highsmith, 2001, online im Internet.

unter dem Titel „Lightweight Process Summit“[56] statt.[57] Dort versammelten sich viele Vertreter dieser Entwicklungen, um die bereits angewandten Prinzipien zu diskutieren. Als Ergebnis entstand das Agile Manifest, welches zentrale Werte der agilen Softwareentwicklung beschreibt.[58] Es definiert die folgenden Werte:[59]

> **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
> **Funktionierende Software** mehr als umfassende Dokumentation
> **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung
> **Reagieren auf Veränderung** mehr als das Befolgen eines Plans

Die Autoren betonen ausdrücklich, dass zwar beide Seiten der Aussagen wertgeschätzt werden sollen, die Erste jedoch generell über der Zweiten steht.[60] Ergänzt werden die Werte zusätzlich durch zwölf weitere Prinzipien, welche die Zusammenarbeit verschiedener beteiligter Parteien sowie Zielsetzungen in Bezug auf den Softwareentwicklungsprozess und die Organisation der Entwicklungsteams präzisieren.[61] Die Bezeichnung „agil“ wurde gewählt, da „leichtgewichtig“ nach Auffassung der Autoren Belanglosigkeit impliziert. Der Initiator des Gipfels selbst erklärte die Entscheidung zugunsten des Begriffs als „die beste Wahl aus einer Menge schlechter Alternativen“[62].

Die agile Bewegung steht nicht im Gegensatz zu den klassischen Modellen, sondern versucht sich an der Wiederherstellung der Balance zwischen Formalität und Produktivität. Beispielsweise soll Dokumentation keinesfalls vernachlässigt, sondern in einem wartungsfähigen und sinnvollen Rahmen zur Verfügung gestellt werden.[63] Agile Vorgehensweisen verzichten somit bewusst auf eine Formalisierung des Softwareentwicklungsprozesses und stellen die kontinuelle, informelle Weiterentwicklung des Produkts anhand von Kundenfeedback in den Vordergrund.[64]

[56] übersetzt etwa *„Gipfel für leichtgewichtige Prozesse“*.
[57] vgl. Martin, 2020, S. 29.
[58] vgl. Highsmith, 2001, online im Internet.
[59] Beck et al., 2001a, online im Internet.
[60] vgl. Beck et al., 2001a, online im Internet.
[61] vgl. Beck et al., 2001b, online im Internet.
[62] Martin, 2020, S. 31.
[63] vgl. Highsmith, 2001, online im Internet.
[64] vgl. Schmidt, 2016, S. 15.

2.1.2 Lean Software Development

Neben den Erfahrungen aus der bisherigen Softwareentwicklung und den Umgang mit den klassischen Modellen, welche zur Begründung der agilen Bewegung geführt haben, hat auch die industriellen Produktentwicklung Einfluss auf die Softwareentwicklungsprozessen gehabt. Eine Sammlung solcher Prinzipien, welche aus der Produktentwicklung abgeleitet und für die Softwareentwicklung adaptiert wurden, sind unter der Bezeichnung des „Lean Software Development“ zusammengefasst. Nach Vertretern der agilen Bewegung helfen diese dabei, die abstrakten Anforderungen der agilen Methoden wie der iterativen Entwicklung und der ständigen Überarbeitung der Aufgabenstellungen anhand konkreter Anforderungen an die Softwareentwicklung erklären zu können.[65]

Das Prinzip von Lean Software Development beschreibt eine Reihe von Leitlinien, welche aus der Lean Manufacturing abgeleitet sind und diese für die Softwareentwicklung adaptieren.[66] Lean Manufacturing entstammt aus dem Toyota Production System, welches Ende der 1940er Jahre entwickelt wurde, um die Produktpreise auch bei der Produktion von kleinen Stückzahlen niedrig halten zu können. Dies wird durch die Reduktion jeglicher Kosten beziehungsweise durch eine Steigerung der Produktivität durchgesetzt.[67] Der Begriff und die Prinzipien hinter Lean Software Development entstammen aus dem gleichnamigen Buch, welches im Jahr 2003 von Mary und Tom Poppendieck veröffentlicht wurde.[68] Sie wurden in einer zweiten Publikation von Mary und Tom Poppendieck nochmals überarbeitet, daher werden im Folgenden diese weiterentwickelten Werte referenziert.[69] Im Vergleich zur agilen Softwareentwicklung unterscheidet sich Lean Software Development durch seine Perspektive auf die Softwareentwicklung als Bestandteil der wertschöpfenden Prozesse des Unternehmens. Die Methoden der agilen Softwareentwicklung sind eine unterstützende Teilmenge von Lean Software Development, während umgekehrt die Prinzipien letzteres wiederum in der agilen Softwareentwicklung berücksichtigt werden.[70] Nach Poppendieck ist die Softwareentwicklung ein Teilprozess des gesamten Produktentwicklungsprozesses.[71]

Im Zentrum steht der Grundsatz der Vermeidung von Verschwendung („Eliminate Waste“) und damit das Identifizieren und Beseitigen von nicht-wertschöpfenden Tätigkeiten.

[65] vgl. Poppendieck und Poppendieck, 2003, S. XIII–XVI.
[66] vgl. Stober und Hansmann, 2010, S. 36 f.; Schmidt, 2016, S. 15 f.
[67] vgl. Monden, 2012, S. 3 f.
[68] vgl. Poppendieck und Poppendieck, 2003, S. 1–186.
[69] vgl. Poppendieck und Poppendieck, 2007, S. 23–41.
[70] vgl. Hibbs, Jewett und Sullivan, 2009, S. 16.
[71] vgl. Poppendieck und Poppendieck, 2007, S. 17.

Laut Poppendieck zeichnet sich ein guter Softwareentwicklungsprozess dadurch aus, dass die Kunden in die Entwicklung eingebunden werden und ein gemeinsames Verständnis für den Wert aufgebaut wird.[72] Dies wirkt sich sowohl auf Entwicklungstätigkeiten selbst, als auch auf Begleitfaktoren aus. Als Beispiel für Verschwendung bei der Entwicklung von Software wird der Fokus auf neue Technologien genannt, welche den Kundennutzen nicht direkt beeinflussen. Der Grundsatz beschreibt, dass Aufgaben ohne direkten Bezug zum Geschäftswert zu eliminieren sind.[73] Die sechs weiteren Grundsätze basieren auf diesem Prinzip.[74]

Diese umfassen zunächst die Priorisierung von Qualität bereits während des Entwicklungsprozesses („Build quality in“). Aus dem Lean Production System übernommen bedeutet dies für die Softwareentwicklung, dass die Tests im gleichen Schritt wie die eigentliche Entwicklung erfolgen müssen.[75] Durch das Beibehalten dieser Tests sollen zukünftige Fehler vermieden werden, welche bereits abgeschlossene Teilbestandteile der Software betreffen können.[76] Der Begriff der Qualität umfasst jedoch nicht nur den Code selbst, sondern auch das Verstehen und Erfüllen der Erwartungen des Kunden.[77] Der Grundsatz des Schaffens von Wissen („Create Knowledge“) beschreibt das Aneignen von Wissen als kontinuierlicher Begleiteffekt in der Softwareentwicklung.[78] Dadurch sollen Verluste in der Softwareentwicklung durch sich wiederholende, gleichartige Lernprozesse, vermieden werden.[79] Die Prinzipien des Aufschiebens von Verpflichtungen („Defer commitment“) und des schnellen Lieferns („Deliver fast“) tragen dazu bei, dass Entscheidungen möglichst lange herausgezögert werden können und durch die schnelle Auslieferung von entwickelten Teilen der Software vermieden wird, dass die Kunden ihre Meinung beziehungsweise ihre Anforderungen an diesen Teilbestandteil ändern.[80] Besonders in frühen Entwicklungsphasen kann so vermieden werden, dass eine kritische Entscheidung oder lange Entwicklungszeiten dazu führen, dass grundlegende Änderungen durchgeführt werden müssen, welche sonst hätten vermieden werden können.[81] Die letzten beiden Prinzipien umfassen zum einen die Wertschätzung des Könnens des Entwicklungsteams als elementarer Faktor des Projekterfolgs („Respect people“). Poppendieck beschreibt

[72] vgl. Poppendieck und Poppendieck, 2007, S. 23.
[73] vgl. Stober und Hansmann, 2010, S. 37.
[74] vgl. Poppendieck und Poppendieck, 2003, S. 2.
[75] vgl. Poppendieck und Poppendieck, 2007, S. 27.
[76] vgl. Hibbs, Jewett und Sullivan, 2009, S. 20.
[77] vgl. Stober und Hansmann, 2010, S. 37 f.
[78] vgl. Poppendieck und Poppendieck, 2007, S. 29.
[79] vgl. Hibbs, Jewett und Sullivan, 2009, S. 20 f.
[80] vgl. Poppendieck und Poppendieck, 2007, S. 34; Hibbs, Jewett und Sullivan, 2009, S. 21.
[81] vgl. Poppendieck und Poppendieck, 2007, S. 32.

„engagierte, mitdenkende Menschen als den nachhaltigsten Wettbewerbsvorteil“.[82] Entscheidungen können am Besten von der Person getroffen werden, welche in diesem bestimmten Bereich die meiste Expertise besitzt und möglichst aktuell über den Entwicklungsstand informiert ist. Dafür muss ein Teil der Verantwortlichkeiten an das Entwicklungsteam delegiert werden.[83] Zum anderen wird Wert auf das Optimieren des großen Ganzen („Optimize the whole“) gelegt, da die Optimierung eines Teilprozesses die Gefahr birgt, dass die Änderungen den umfassenden Prozess suboptimal beeinflussen.[84] Innerhalb des Softwareentwicklungsprozesses beziehen sich diese Optimierungen nicht nur auf das Produkt selbst, sondern auch auf die umgebenden Strukturen und Verantwortlichkeiten.[85]

Ähnliche Ansätze lassen sich auch in den zwölf Prinzipien hinter dem Agilen Manifest erkennen.[86] Lean Software Development versucht zusätzlich, neben der Softwareentwicklung auch die umliegenden Geschäftsprozesse zu berücksichtigen. Die Grundsätze sind generell so gestaltet, dass diese auch dort Anwendung finden können. Während die agilen Modelle formelle Methoden definieren, beschreibt Lean Software Development lediglich einzelne Grundsätze, welche für ein Umdenken genutzt werden können.[87]

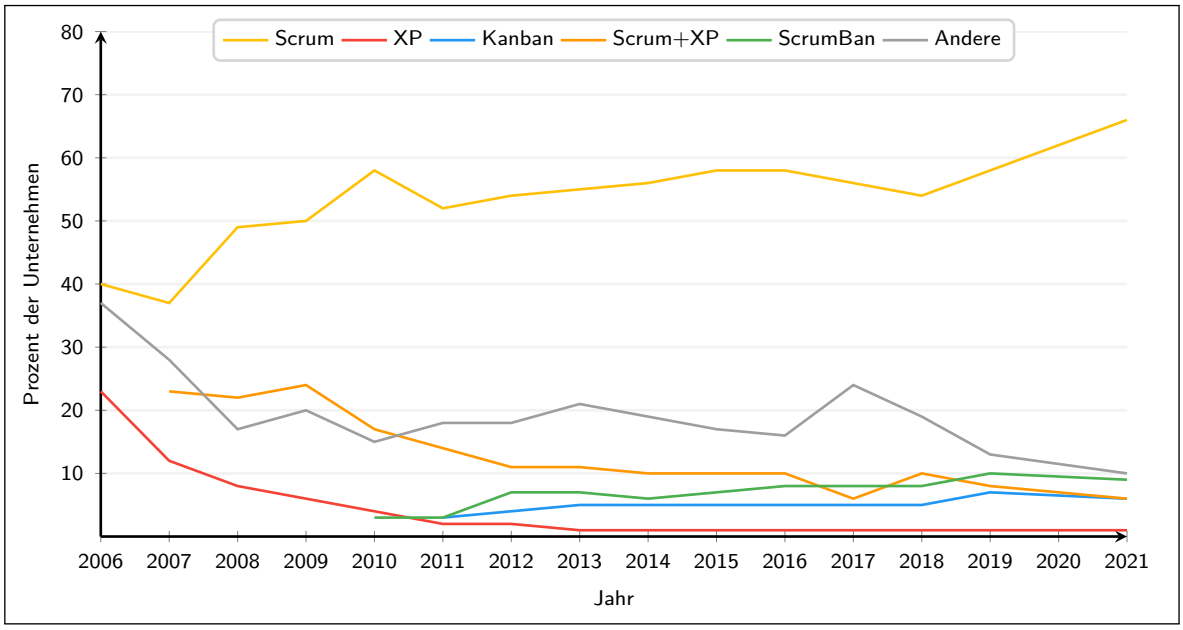
2.2 Methoden

Bereits zum Zeitpunkt der Erstellung des agilen Manifests existierten verschiedene Methoden der agilen Softwareentwicklung, deren Vertreter die Entstehung des Manifests begleiteten. Die größten Gruppierungen der dort Anwesenden bestanden aus Vertretern der Bewegung hinter XP, gefolgt von Vertretern von Scrum.[88] Die Verbreitung der Methoden wird seit 2007 jährlich in einem Bericht des Unternehmens Digital.ai untersucht. Das Unternehmen entwickelt Lösungen für die agile Planung und zählt in dieser Branche laut einer Marktanalyse von Gartner aus dem Jahr 2021 zu den führenden Anbietern.[89] Die jeweiligen Berichte basieren auf Umfragen, welche anhand von Unternehmen verschiedener Branchen und Größen durchgeführt wurden. Die Ergebnisse sind in Abbildung 1 dargestellt.

---

[82] übersetzt nach Poppendieck und Poppendieck, 2007, S. 279: *„Engaged, thinking people provide the most sustainable competitive advantage.“*.
[83] vgl. Stober und Hansmann, 2010, S. 38 f.
[84] vgl. Poppendieck und Poppendieck, 2007, S. 39.
[85] vgl. Poppendieck und Poppendieck, 2007, S. 39.
[86] vgl. Beck et al., 2001b, online im Internet.
[87] vgl. Hibbs, Jewett und Sullivan, 2009, S. 22 f.
[88] vgl. Martin, 2020, S. 29 f.
[89] vgl. Blosen et al., 2021, online im Internet.

Abbildung 1: Verbreitung agiler Methoden von 2006–2021



Quelle: Eigene Darstellung nach Werten von VersionOne (2007-2017), CollabNet-VersionOne (2018, 2019) und Digital.ai (2020, 2021).

Anhand der Umfrageergebnisse lassen sich verschiedene Trends ableiten. Scrum war bereits 2006 das Modell, an dem sich die meisten Unternehmen orientierten. Im Laufe der Jahre nahm dieser Wert recht konstant zu und liegt mittlerweile bei 66 %. Dieser Wert beschreibt den Anteil der Unternehmen, welche Scrum als den von ihnen am meisten beachteten Ansatz bezeichnen. Während Scrum an Beliebtheit hinzugewinnen konnte, verzeichneten XP und die Kombination aus Scrum und XP einen stetigen Abwärtstrend. Letztere wird aktuell noch von 6 % als primäre Methode verwendet, XP selbst jedoch seit 2013 jährlich nur noch von knapp 1 % der Unternehmen. Auf der anderen Seite nahm die Nutzung der Methode Kanban selbst sowie deren Kombination mit Scrum (bezeichnet durch das Kunstwort ScrumBan) stetig zu, befindet sich jedoch nach wie vor auf einem recht niedrigen Niveau. Auch bei der Diversifikation bei der Verwendung der Methoden ist eine Veränderung festzustellen. Während der Anteil alternativer Methoden in 2006 noch bei 37 % zu verzeichnen war, beschrieb er in folgenden Jahren ebenfalls einen Abwärtstrend und liegt im Jahr 2021 nur noch bei 10 %. Daraus kann die Hypothese abgeleitet werden, dass sich die Unternehmen zunehmend eher an den verbreiteteren Methoden orientiert haben.

Im Folgenden soll im Detail auf die Methoden XP und Scrum eingegangen werden. Diese sind für die Untersuchung im Rahmen der Fallstudie besonders relevant, da das im Titel dieser

Masterthesis genannte Prinzip des Collective Code Ownerships aus der Methode XP stammt und das Projekt der Fallstudie mithilfe einer an Scrum orientierten Methodik organisiert ist.

2.2.1 Scrum

Scrum ist ein Vorgehensmodell der agilen Softwareentwicklung, welches den Fokus primär auf den Entwicklungsprozess legt. Dabei werden Ideen zur Verbesserung von Flexibilität, Anpassungsfähigkeit und Produktivität aus der Theorie der industriellen Prozesssteuerung auf den Softwareentwicklungsprozess übertragen.[90] Nach dem Grundgedanken von Scrum beherbergt die Entwicklung einer umfangreichen Software eine Komplexität, von der sich keine Teilaspekte ausblenden lassen. Dies liegt zum einen daran, dass der Prozess eine ausschließlich intellektuelle Lösungsfindung repräsentiert, bei dem (Zwischen-) Produkte rein aus den Denkvorgängen der Beteiligten bestehen. Zum anderen ist das Fundament dieser Arbeit sehr volatil, darunter die Anforderungen der Benutzer, die Interaktion mit anderen Systemen und das eigene Team, welches die Implementierung vornimmt.[91] Das Vorgehensmodell basiert dahingehend auf der Grundannahme, dass die Entwicklung „sich weder in großen abgeschlossenen Phasen planen lässt, noch dass den einzelnen Mitarbeitern vorab detaillierte Arbeitspakete mit Aufwandsschätzungen in Stunden und Tagen zugeordnet werden können“[92].

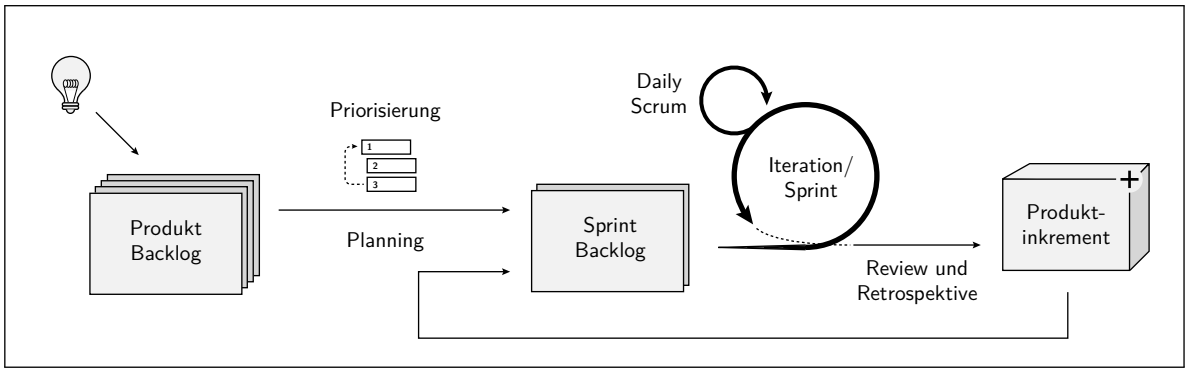
Die Methode entstammt ursprünglich aus Forschungen von Hirotaka Takeuchi und Ikujiro Nonaka aus dem Jahr 1986, welche Grundsätze in der Produktentwicklung damals führender Unternehmen untersuchten. Dabei stellten sie fest, dass sich sechs Charakteristiken feststellen ließen, welche den Unternehmen zu erhöhter Geschwindigkeit und Flexibilität verholfen haben. Zu diesen zählen das Zulassen von Instabilität, selbstorganisierte Projektteams, überlappende Entwicklungsphasen, vielfältiges Lernen über Grenzen hinaus, subtile Kontrolle und der Transfer des Gelernten in der Organisation.[93] Auch der Begriff Scrum (übersetzt „Gedränge“) entstammt dieser Veröffentlichung, wo die Interaktionen innerhalb eines Projektteams mit denen von Rugbyspielern verglichen werden. Als Gemeinsamkeit wird angeführt, dass die

[90] vgl. Paetsch, Eberlein und Maurer, 2003, S. 310.
[91] vgl. Schwaber, 2007, S. 101.
[92] Alpar et al., 2019, S. 371.
[93] vgl. Takeuchi und Nonaka, 1986, S. 138–144.

Prozesse im Zusammenspiel des Teams nicht vordefiniert, sondern aus der Situation heraus gebildet werden.[94]

Aus diesen Erkenntnissen leiteten Jeff Sutherland und Ken Schwaber das heutige Scrum-Modell ab, welches sie 1995 erstmalig auf einer Konferenz vorstellten. Dort beschreiben sie Scrum als Entwicklungsprozess, „welcher große Teile der Systementwicklung als kontrollierte Blackbox behandelt".[95] Dieser Ansatz soll die Flexibilität und Reaktionsfähigkeit auf initiale und später hinzugekommene Anforderungen verbessern.[96] Die stete Veränderung, Unvorhersehbarkeit und Komplexität werden als unausweichliche Konstante in der Produktentwicklung angesehen.[97] Das Vorgehen selbst zeichnet sich durch einen iterativen Charakter aus, durch den das Produkt bis zur finalen Version inkrementell weiterentwickelt wird. Dabei wird versucht, die Anforderungen der Prozesssteuerung in eine überschaubare Anzahl von Praktiken und Regeln zu überführen.[98] Der Prozess ist in Abbildung 2 dargestellt.

Abbildung 2: Scrum-Prozess



Quelle: Eigene Darstellung in Anlehnung an Schwaber, 2007, S. 108.

Das zentrale Element von Scrum ist ein iterativer Prozessrahmen, an welchen alle Praktiken angefügt werden. Er basiert auf einer priorisierten Aufgabenliste (genannt „Backlog"), in welchem neue Anforderungen und Ideen für das Produkt aufgenommen werden. Ein Teil dieses Backlogs wird zur Umsetzung in einen Sprint aufgenommen. Der untere Kreis repräsentiert diese wiederkehrende Iteration von Entwicklungsaktivitäten, dessen Zeitrahmen nach Scrum auf einen Monat festgelegt ist, jedoch variabel gestaltet werden kann. Das Ergebnis einer

[94] vgl. Takeuchi und Nonaka, 1986, S. 137 f.
[95] übersetzt nach Schwaber, 1995, S. 1, online im Internet: *„[...] that treats major portions of systems development as a controlled black box."*.
[96] vgl. Schwaber, 1995, S. 1, online im Internet.
[97] vgl. Schwaber, 2007, S. 4.
[98] vgl. Schwaber, 2007, S. 105.

solchen Iteration ist ein funktionsfähiges Produktinkrement. Der obere Kreis steht für den täglichen Teilprozess einer solchen Iteration, in welchem die Entwickler:innen ihre Aktivitäten gegenseitig besprechen und erforderliche Anpassungen vornehmen. Der Prozess wird solange fortgeführt, wie es die Finanzierung des Projekts erlaubt. Eine Iteration besteht dabei neben der Umsetzungsphase, welche sich durch die täglichen Iterationen äußert und in welchen das Entwicklungsteam auf sich alleine gestellt ist, aus einer Planungsphase („Planning“) zu Beginn und einem Bericht beziehungsweise Rückblick („Review“ und „Retrospective“) zum Ende eines Sprints. In der Planungsphase werden die Prioritäten für die nächste Iteration festgelegt und entschieden, welche Aufgaben in letzterer umgesetzt werden können. In der Berichtsphase werden die abgeschlossenen Aufgaben vorgestellt und gemeinsam mit allen beteiligten Personen entschieden, ob in Bezug auf die Implementierung oder den Projektrahmen noch Anpassungen notwendig sind. Zusätzlich werden Vorschläge gesammelt, wie die Zusammenarbeit nach den Erfahrungen des letzten Sprints weiter verbessert werden kann.[99]

Nach Schwaber spielt sich der wichtigste Teil von Scrum jedoch innerhalb der Iteration ab. Das Team der Entwickler:innen schaut gemeinsam auf die Anforderungen und die verwendeten Technologien sowie die Expertise und das Können jedes einzelnen Mitglieds. Anschließend wird versucht, auf kollektiver Basis zu definieren, welches der richtige Weg für die Umsetzung der Anforderungen ist. Im Verlauf eines Sprints wird dieses Vorgehen fortlaufend anhand von auftretenden Schwierigkeiten angepasst.[100] Scrum sieht sich selbst weniger als Prozess, sondern als ein Werkzeug, um passende Prozesse für das eigene Unternehmen zu entwickeln.[101] Es wird beschrieben als ein „leichtgewichtiges Framework, das Menschen, Teams und Organisationen hilft, durch adaptive Lösungen für komplexe Probleme Werte zu schaffen“[102] und basiert auf den Werten Engagement, Konzentration, Offenheit, Respekt und Mut.[103]

[99] vgl. Schwaber, 2007, S. 105–108.
[100] vgl. Schwaber, 2007, S. 106.
[101] vgl. Schwaber, 2007, S. 7.
[102] übersetzt nach Schwaber und Sutherland, 2020, S. 3, online im Internet: *„Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.“*.
[103] vgl. Schwaber und Sutherland, 2020, S. 4, online im Internet.

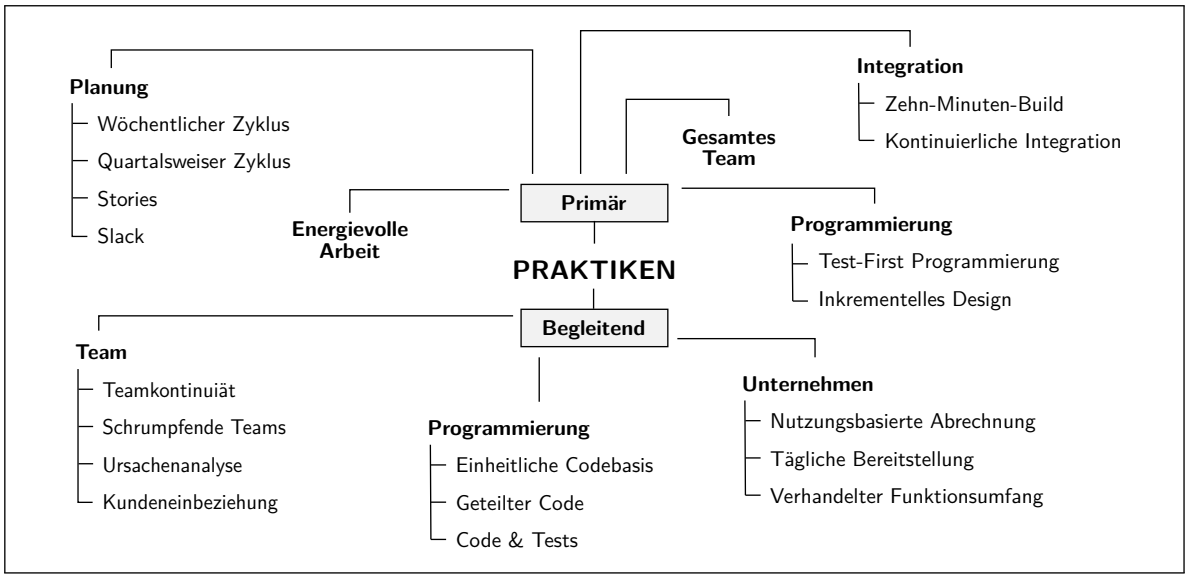
2.2.2 Extreme Programming

Die Methode XP wurde von Kent Beck, Ward Cunningham und Ron Jeffries innerhalb eines gemeinsames Softwareprojektes entwickelt und galt in den frühen 2000er Jahren als meistbeachteter agiler Ansatz.[104] Der Name entstand aus dem Anspruch, bereits etablierte Grundsätze wie die iterative Entwicklung in „das Extreme“ zu treiben.[105] Nach Aussage von Beck wurde die Methode als Antwort auf das von ihnen beobachtete Scheitern von Softwareprojekten entworfen, um den dadurch entstehenden wirtschaftlichen und menschlichen Auswirkungen entgegenzuwirken.[106] Das Scheitern lag aus seiner Sicht darin begründet, dass die Entwicklungszyklen zu lang geplant wurden und nicht auf Änderungen reagiert werden konnte. Die Motivation hinter der Nutzung kurzer Entwicklungszyklen war, dass anfangs nicht bedachte Änderungen, welche zu einem späteren Zeitpunkt umgesetzt werden müssen, zu sehr hohen Kosten führten.[107] Auf der anderen Seite wurde dieses Risiko jedoch auch von den Entwicklern erkannt und durch neue Technologien wie relationalen Datenbanken und verstärkter Modularisierung entgegengewirkt. Aus Becks Sicht musste man diese technologischen Bemühungen auch vonseiten der Entwicklungsmethode im Sinne von kürzeren Zyklen würdigen.[108]

Das Modell basiert auf den Werten Kommunikation, Feedback, Einfachheit, Mut und Respekt.[109] Neben diesen werden Prinzipien und Praktiken definiert. Letztere geben Vorgaben für die Entwicklung selbst, welche sich nach XP in der Vergangenheit beweisen konnten. Die Prinzipien sollen die Werte und Praktiken miteinander verbinden, insbesondere wenn für eine bestimmte Problemstellung keine konkrete Praktik in Frage kommt.[110] Beispiele dafür sind das Prinzip der Diversität, welches den Wert eines vielfältigen Teams betont, sowie das Prinzip Qualität, welches nach XP nicht als Variable der Projektkontrolle angesehen und daher bei Anpassungen in der Projektplanung nicht gekürzt werden darf.[111] Ähnlich wie bei Scrum wird auch in XP der Fokus auf das Team anstelle der einzelnen Individuen gelegt, was sich durch viele Praktiken in Bezug darauf äußert. Weiterführend werden Praktiken zu Arbeitsplatz, -pensum und Beziehung zum Kunden sowie zur Planung der Tätigkeiten angeführt. Eine Übersicht zu den Praktiken ist in Abbildung 3 dargestellt.

[104] vgl. Highsmith, 2002, S. 7.
[105] vgl. Sommerville, 2016, S. 77.
[106] vgl. Beck, 2000, S. 3.
[107] vgl. Beck, 1999, S. 70.
[108] vgl. Beck, 1999, S. 70.
[109] vgl. Beck, 2005, S. 2.
[110] vgl. Beck, 2005, S. 2.
[111] vgl. Beck, 2005, S. 24, 32 f.

Abbildung 3: Praktiken von Extreme Programming



Quelle: Eigene Darstellung in Anlehnung an Beck, 2005, S. 36.

Für die weiteren Ausführungen sind insbesondere die Haupt- und Begleitpraktiken mit Fokus auf die Programmierung und die Integration der daraus entstandenen Weiterentwicklungen relevant. Dazu gehören zum einen die primären Aspekte der kontinuierlichen Integration, des inkrementellen Designs und der Test-First Programmierung. Diese besagen, dass potenzielle Änderungen an der Software nicht als großes Paket, sondern in kleinen Einheiten inkrementell eingefügt werden sollen. Diese werden zunächst durch einen Test definiert, um darauf aufbauend die Implementierung anzugehen. Weiterführend bezeichnet die kontinuierliche Integration die Vorgehensweise, dass neue Entwicklungen sobald wie möglich in das Gesamtsystem zurückgeführt werden sollen.[112] Die begleitenden Praktiken erweitern diese Grundsätze durch das Pflegen einer einheitlichen, gemeinsamen Codebasis. Die darin enthaltenen Artefakte zu Code und Tests stellen nach Beck stets den aktuellen Stand der Entwicklung dar und sollten nach Möglichkeit den Bedarf für zusätzliche Dokumentation eliminieren. Das Prinzip des geteilten Codes bezeichnet die Möglichkeit, dass alle beteiligten Entwickler:innen die Möglichkeit haben sollen, zu allen Aspekten des Codes beitragen zu können.[113]

Hier ist zu erkennen, dass XP auch sehr konkrete Anforderungen an das fachliche Vorgehen stellt, was beispielsweise bei Scrum nicht der Fall ist. Die Unterteilung in Haupt- und

[112] vgl. Beck, 2005, S. 49–53.
[113] vgl. Beck, 2005, S. 66 ff.

Begleitpraktiken ist dadurch begründet, dass die Implementierung letzterer zu Implikationen führen kann, wenn die Hauptpraktiken noch keine Anwendung finden.[114] Ein Beispiel dafür ist die Test-First Programmierung, welche die Rahmenbedingungen für eine erfolgreiche Umsetzung des Prinzip des geteilten Codes schafft. Durch die Tests soll die Lauffähigkeit der Software auch bei Beteiligung vieler Personen sichergestellt werden.

Beck beschreibt die Ziele von XP damit, dass Projektrisiken reduziert, die Adaptionsfähigkeit auf veränderte Anforderungen verbessert, sowie die Produktivität und der Spaßfaktor bei der Softwareentwicklung erhöht werden sollen.[115] Nach ihm geht es bei XP im Kern nicht um die Vorgabe eines Modells, sondern um die soziale Komponente der Softwareentwicklung, der Offenheit gegenüber und der Arbeit mit allen Mitwirkenden sowie das Anwenden von bewährten Entwicklungspraktiken.[116] Beispielsweise wird im Gegensatz zu Scrum bei der Umsetzung der Anforderungen nicht nur auf Expertise und Können der Entwickler:innen geachtet, sondern explizit Wert auf den Austausch von Wissen untereinander gelegt.[117] Obwohl die Verbreitung von XP in den letzten Jahren rückläufig ist (siehe Abbildung 1), sind viele der Grundgedanken hinter der Methode aus anderen Modellen entliehen und dort nach wie vor präsent. Beispielsweise ist die Adaptionsfähigkeit auf Veränderungen auch ein zentraler Punkt von Scrum, welchen Beck von dort für seine XP-Methode abgeleitet hat.[118] Auch zum Lean Software Development sind Ähnlichkeiten zu erkennen und der Begriff der Verschwendung wird in XP aufgegriffen. Demnach liegt im Interesse des Kunden lediglich, was die Software heute leisten kann und was das Team in Zukunft zur Steigerung der Leistungsfähigkeit beitragen kann. Wenn ein Entwicklungsartefakt zu diesen beiden Aspekten beitragen kann, wird es als wertvoll betrachtet, andernfalls als Verschwendung.[119]

In der Praxis hat sich die Umsetzung der Praktiken aus XP im Zusammenspiel mit klassischen Managementmethoden als kompliziert herausgestellt, da es keine klaren Rahmenbedingungen für den Prozess, sondern eher für die Entwicklung vorgibt. Letztere geben sehr genaue Vorgaben dazu, wie die Entwicklung der Software gestaltet werden sollte, wodurch sich die Zusammenführung der Praktiken aus XP und den im Unternehmen bestehenden Prozessen als schwierig herausstellen kann. Daher haben viele Unternehmen sich entsprechend ihrer eigenen Anforderungen eine Teilmenge der Praktiken ausgewählt und in ihr Umfeld übernommen.

[114] vgl. Beck, 2005, S. 61.
[115] vgl. Beck, 2000, S. XVI.
[116] vgl. Beck, 2005, S. 1 f.
[117] vgl. Beck, 2005, S. 141.
[118] vgl. Beck, 1999, S. 73.
[119] vgl. Beck, 2005, S. 66.

Bewährt hat sich ebenfalls die Anwendung von Praktiken aus XP in Verbindung mit einer prozessorientierten Methode wie Scrum.[120]

Nachdem die für die Fallstudie relevanten Methoden der agilen Softwareentwicklung vorgestellt wurden, behandelt das nächste Kapitel die Organisation agiler Entwicklungsteams aus einer allgemeineren Perspektive. Dabei wird die Bedeutung des Teambegriffs im agilen Kontext nochmals herausgestellt und anschließend auf die Formalisierung der Entwicklung und Modelle für die Verteilung von Verantwortlichkeiten eingegangen.

[120] vgl. Sommerville, 2016, S. 78.

3 Organisation agiler Entwicklungsteams

> „Die Aufgabe des Softwareentwicklungsteams besteht darin, die Illusion von Einfachheit zu erzeugen."
>
> – Grady Booch, 1994[121]

Komplexität ist eine inhärente Charakteristik der Softwareentwicklung. Insbesondere Software im industriellen Umfeld zeichnet sich dadurch aus, dass es für einzelne Entwickler:innen schwierig bis unmöglich ist, alle Feinheiten ihres Aufbaus zu verstehen. Diese Komplexität ergibt sich aus den vier Faktoren der Definition des Problembereichs, der Steuerung des Entwicklungsprozesses, der durch die Software ermöglichten Flexibilität und der Charakterisierung des Verhaltens des Systems.[122] Die Organisation der Entwicklungsteams und ihrer Verantwortung kann einen Beitrag leisten, um mit dieser Komplexität umzugehen.

Um die Verteilung von Verantwortlichkeiten innerhalb eines Entwicklungsteams untersuchen zu können, muss zunächst die Frage gestellt werden, wie ein Solches überhaupt organisiert ist. Die Zusammenstellung eines Teams für ein Softwareprojekt erfolgt typischerweise durch eine Analyse der Anforderungen, sodass die geforderte Expertise abgedeckt ist.[123] Dabei wird das Team meist neu zusammengestellt, die Zuweisung eines bestehenden Projektteams auf ein neues Projekt kommt nur sehr selten vor.[124] Da Softwareentwicklung primär Wissensarbeit ist, gilt die Expertise als wichtigste Ressource innerhalb eines solchen Projekts. Die Effektivität der Entwicklung und die Qualität der Software wird jedoch nicht alleine durch das Vorhandensein von Expertise gesichert. Zusätzliche Faktoren umfassen die Koordination der Anforderungen und der Teamarbeit sowie das Verbreiten des Wissens innerhalb des Teams.[125] Nach Scrum ist das Zusammenarbeiten verschiedener Personen der hauptsächliche Treiber von Komplexität in der Entwicklung eines Produkts.[126]

Die Einführung von agilen Softwaremethoden hat auch dazu geführt, dass der Fokus nicht auf den individuellen Entwickler:innen selbst, sondern auf dem Team als Ganzes liegen.[127] Die Koordination eines solchen Teams benötigt einen Prozess, um die vorhandenen Fähigkeiten

[121] übersetzt nach Booch, 1994, S. 5: *„The task of the software development team is to engineer the illusion of simplicity."*.
[122] vgl. Booch, 1994, S. 3.
[123] vgl. Maruping, Zhang und Venkatesh, 2009, S. 356.
[124] vgl. Faraj und Sproull, 2000, S. 1554.
[125] vgl. Faraj und Sproull, 2000, S. 1555.
[126] vgl. Schwaber, 2007, S. 104 f.
[127] vgl. Ahmed und Colomo-Palacios, 2021, S. 273.

zur Lösung von definierten Anforderungen einzusetzen. Dafür wird vorausgesetzt, dass die Entwickler:innen ein gemeinsames Verständnis der Codebasis entwickeln, um in kollaborativer Arbeitsweise eine gemeinsame Lösung zu finden.[128]

Die im vorherigen Kapitel beschriebenen Methoden der agilen Softwareentwicklung definieren Grundsätze für die Zusammenarbeit in einem Projekt sowie für die konkreten Entwicklungstätigkeiten selbst. In den folgenden zwei Unterkapiteln wird untersucht, inwiefern sich die Methoden in der Hinsicht ähneln und welche Unterschiede zu verzeichnen sind. Anschließend werden verschiedene Modelle für die Verteilung von Verantwortlichkeiten bei der Entwicklung selbst dargestellt und die Methode des Collective Code Ownerships erläutert.

3.1 Bedeutung des Teams

Sowohl Scrum als auch XP stellen ein funktionierendes Team als zentralen Erfolgsfaktor dar und somit in den Vordergrund der Methode. Nach Scrum gelten für die Zusammenstellung eines solchen drei Variablen. Demnach sind Personen zu bevorzugen, welche bereits in der Vergangenheit erfolgreich miteinander gearbeitet haben, Personen mit Wissen in Bezug auf das Produkt oder die Geschäftsdomäne und Personen mit Wissen zu den ausgewählten Technologien. Weiterführend müssen die Mitglieder eines Teams nicht konstant sein, sondern können basierend auf den Anforderungen hinzugezogen werden.[129] Die Organisation, welche das Team selbst übernimmt, geschieht auf Grundlage der vorgegebenen Ziele, welche anhand der Ergebnisse einzelner Sprints fortlaufend angepasst werden können.[130] In den Praktiken von XP sind ähnliche Ansätze zu finden. Nach XP tendieren größere Organisationen dazu, Programmierer basierend auf den Anforderungen in Einheiten zusammenzufassen, obwohl „der Wert einer Software [..] nicht nur durch das [entsteht], was Menschen wissen und tun, sondern auch durch ihre Beziehungen und das, was sie gemeinsam erreichen“.[131] Daher sollen erfolgreiche Teams als solche fortgeführt werden[132] und alle fachlichen Anforderungen abdecken, ohne Personen auf mehrere Teams aufzuteilen.[133] Beck vertritt die Meinung, dass

[128] vgl. Maruping, Zhang und Venkatesh, 2009, S. 356.
[129] vgl. Schwaber, 2007, S. 73.
[130] vgl. Schwaber, 2007, S. 76.
[131] übersetzt nach Beck, 2005, S. 63: *„Value in software is created not just by what people know and do but also by their relationships and what they accomplish together.“*.
[132] vgl. Beck, 2005, S. 63 f.
[133] vgl. Beck, 2005, S. 38 f.

räumliche Nähe und der persönliche Austausch die Produktivität eines Teams erhöhen.[134] Eine Ausprägung in dieser Hinsicht ist die Paarprogrammierung, bei welcher zwei Entwickler:innen gemeinsam im Dialog an der Lösung einer Aufgabe arbeiten. Für den Austausch sollen diese Paare nach XP mehrmals am Tag gewechselt werden, um den Austausch auf das gesamte Team auszuweiten. Die Methode lässt jedoch auch die Möglichkeit offen, eigenständig an einer Problemstellung zu arbeiten, solange die Handlungen im Interesse des Teams stehen.[135] Der Nutzen von Paarprogrammierung hängt vom konkreten Einsatzgebiet ab, da die Produktivität im Vergleich zur individuellen Entwicklung geringer sein kann. Studien, welche sich mit dieser Fragestellung beschäftigen, kamen zu unterschiedlichen Ergebnissen. Teilweise ist die Produktivität beider Herangehensweisen vergleichbar,[136] teilweise wirkt sich die Paarprogrammierung jedoch insbesondere in Abhängigkeit zur Komplexität des Systems und der Expertise der Entwickler:innen negativ aus.[137] Dabei werden jedoch Begleiteffekte wie die Vermittlung von Wissen nicht berücksichtigt.[138]

Solche definierten Praktiken sind in den Ausführungen zu Scrum nicht zu finden. Für den Erhalt der Expertise wird dort definiert, dass die Entwickler:innen Zeit für die eigene Bildung zur Verfügung gestellt bekommen, nach Schwaber etwa zwanzig Prozent der Gesamtzeit.[139] Die anderweitigen Gemeinsamkeiten zwischen Scrum und XP lassen sich auf das Agile Manifest zurückführen. Bereits dort wurden die Prinzipien definiert, dass Projekte um motivierte Individuen aufgebaut werden sollen und die besten Ergebnisse aus selbstorganisierten Teams entspringen.[140]

3.2 Formalisierung der Entwicklung

Neben der Betrachtung der Softwareentwicklung aus Sicht der Methode können auch Grundsätze in der Vorgehensweise aus fachlicher Sicht festgelegt werden. Diese bestimmen die Zusammenarbeit im Team und können auch als Best Practices in der Entwicklung von Software bezeichnet werden. Von den beiden in Abschnitt 2.2 vorgestellten Methoden der agilen Softwareentwicklung stellt lediglich XP Grundsätze für die Vorgehensweise und die Zusammenarbeit während der Entwicklung auf.

[134] vgl. Beck, 2005, S. 39.
[135] vgl. Beck, 2005, S. 42 f.
[136] vgl. Williams et al., 2000, S. 23.
[137] vgl. Arisholm et al., 2007, S. 81 ff.
[138] vgl. Sommerville, 2016, S. 84.
[139] vgl. Schwaber, 2007, S. 80.
[140] vgl. Beck et al., 2001b, online im Internet.

Die im vorherigen Kapitel in Abbildung 3 gezeigten Praktiken von XP benennen in Zusammenhang mit Vorschriften zur eigentlichen Entwicklung des Quellcodes den Zehn-Minuten-Build, kontinuierliche Integration und Test-First Programmierung[141] sowie die Fehler-Ursachen-Analyse, geteilten Code, Codierung und Testen, die Etablierung einer gemeinsamen Code-Basis und das tägliche und inkrementelle Bereitstellen.[142] Diese Vorschriften lassen sich in die drei Bereiche der Strukturierung, Implementierung und Bereitstellung unterteilen.

Zur Strukturierung der Entwicklung tragen die Faktoren des geteilten Codes, der gemeinsamen Code-Basis sowie der Codierung und des Testens bei. Die ersten beiden Punkte beziehen sich zum einen auf die Verantwortung, als auch auf den Zugang zu den Quellcode-Ressourcen. Die zentrale Code-Basis sieht ebenfalls vor, dass zu einem Zeitpunkt lediglich eine aktuelle und gültige Version für den gesamten Quellcode existiert.[143] Das Codieren und Testen sieht vor, dass nur die Ergebnisse des Codierens und des Testens als permanente Artefakte bewahrt und andere notwendige Dokumente aus diesen abgeleitet werden sollen. Begründet wird dies damit, dass weiterführende Vorschriften den Wertefluss stören[144] und soziale Mechanismen zur Ausbreitung des Wissens genutzt werden können.[145]

Für die Implementierung sieht XP die Test-First Strategie und das Durchführen von Fehler-Ursachen-Analysen vor. Demnach soll noch vor der ersten Änderung des Quellcodes ein Testfall definiert und entsprechend implementiert werden. Die anschließende Entwicklung befasst sich damit, diesen Testfall zu lösen und durch weitere Testfälle zu präzisieren. Durch diese Definition soll erreicht werden, dass ein Rahmen für die Implementierung gesetzt wird, welcher nur den notwendigen Umfang der Aufgabe einschließt. Nach Beck sind Schwierigkeiten in der Definition von Testfällen ein Signal für ein Entwurfsproblem, nicht für eine Testproblematik.[146] Die Fehler-Ursachen-Analysen zielt darauf ab, dass Fehler in der entwickelten Software so früh wie möglich behoben werden. Dies ist mit dem Grundsatz „Build quality in“ aus dem Lean Software Development vergleichbar. Das Ziel ist, dass sowohl genau dieser Fehler, als auch ähnliche Fehler in Zukunft vermieden werden.[147]

Die meisten Grundsätze werden in Bezug auf die Bereitstellung von Änderungen aufgeführt. Darunter ist die erste Hauptpraktik der Zehn-Minuten-Build. Der Prozess von der Fertig-

[141] vgl. Beck, 2005, S. 37–53.
[142] vgl. Beck, 2005, S. 61–71.
[143] vgl. Beck, 2005, S. 66 f.
[144] vgl. Beck, 2005, S. 67.
[145] vgl. Beck, 2005, S. 66 f.
[146] vgl. Beck, 2005, S. 50 f.
[147] vgl. Beck, 2005, S. 64 ff.

stellung des Quellcodes bis zur lauffähigen Software inklusive aller Tests sollte nach Beck etwa zehn Minuten beanspruchen. Dieses Zeitlimit ist so gewählt, da der Prozess in diesem Rahmen viel öfter ausgeführt und so als direktes Feedback genutzt werden kann.[148] Die zweite Hauptpraktik bezeichnet die kontinuierliche Integration, bei der Änderungen möglichst frühzeitig wieder eingegliedert werden. Verzögerte Integrationen macht den Integrationsprozess komplizierter und weniger vorhersehbar.[149] Die Begleitpraktiken zur täglichen und inkrementellen Bereitstellung beziehen sich auf den Prozess, die gesammelten Änderung in die produktive Umgebung bereitzustellen. Nach Beck sind Änderungen, welche selbst nur kurzfristig unberührt bleiben, ein Risiko für die Entwicklung der Software.[150] Die inkrementelle Bereitstellung bezieht sich hingegen auf die Migration älterer Systeme, welche nur in Teilen und nicht in einem Schritt übernommen werden sollen, da größere Übernahmen nach seiner Meinung zum Scheitern verurteilt sind.[151]

Viele der in XP definierten Praktiken wie die Test-First Strategie und die Paarprogrammierung finden auch heute noch Anwendung oder haben sich weiterentwickelt. Ein Beispiel ist das Test-Driven Development (testgetriebene Entwicklung), welche ihren Ursprung in der Test-First Strategie hatte und diese mit dem Ziel umsetzt, dass jede einzelne Zeile des Quellcodes durch einen Test geprüft wird.[152] Auch der Software Engineering Body of Knowledge der IEEE Computer Society als Sammlung akzeptierten Wissens zum Software Engineering beinhaltet das Reduzieren von Komplexität, das Antizipieren von Veränderungen und das stetige Testen als bewährte Praktiken.[153]

Eine solche, welche in XP nicht berücksichtigt ist, ist das sogenannte Code Review. Nach IEEE Standard 730-2014 ist das Code Review definiert als „ein Prozess [...] in der Arbeitsprodukte einigen Interessengruppen zur Stellungnahme oder Genehmigung vorgelegt werden".[154] Die Anwendung von Code Reviews gilt primär der Vermeidung von Fehlern in der Software. Als Begleitfaktoren gelten das Aneignen von Wissen über die geänderten Abschnitte durch mehrere Personen, wodurch die Abwesenheit einzelner Entwickler:innen abgefangen werden kann. Zusätzlich führt das Review in den meisten Fällen zu einer höheren Qualität der Entwicklung, da sich die Verfasser:innen der Überprüfung durch andere bewusst sind. Durch die kollaborative Herangehensweise können über längere Zeiträume ebenfalls implizite Code-

[148] vgl. Beck, 2005, S. 49.
[149] vgl. Beck, 2005, S. 49 f.
[150] vgl. Beck, 2005, S. 68 f.
[151] vgl. Beck, 2005, S. 62 f.
[152] vgl. Jeffries und Melnik, 2007, S. 24 ff.
[153] vgl. IEEE Computer Society, 2014, S. 67 f., 76 f., online im Internet.
[154] IEEE, 2014, S. 7.

Standards für die Gruppe entwickelt werden.[155] Das Code Review ist ebenfalls im Software Engineering Body of Knowledge aufgeführt.[156] Die Methode XP betrachtet Code Reviews als überflüssig, solange die anderen Praktiken der Methode, insbesondere das Testen des Codes, das Programmieren in Paaren und das Prinzip des verteilten Codes befolgt werden. Das Testen stellt die Lauffähigkeit der Software sicher. Das Programmieren in Paaren trägt dazu bei, dass der Quellcode bereits während der Erstellung von zwei Personen überprüft wird und beide über die Änderungen informiert sind. Das Prinzip des verteilten Codes erweitert diesen Personenkreis auf mehrere Personen und stellt insbesondere sicher, dass sich durch die persönliche Verpflichtung gegenüber der Organisation an definierte Standards gehalten wird. Außerdem wird sichergestellt, dass nicht nur das initiale Paar der Entwickler:innen Wissen über bestimmte Abschnitte besitzt.[157]

3.3 Verantwortlichkeiten im Quellcode

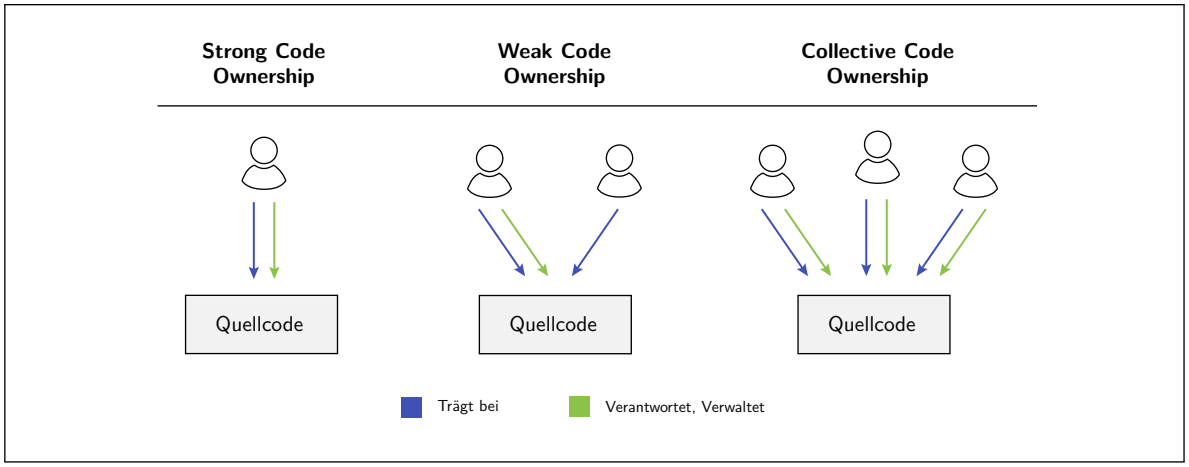
Die vorgestellten Methoden aus Abschnitt 2.2 definieren, wie bereits angedeutet, primär Rollen in Bezug auf den Entwicklungsprozess, jedoch nicht für die eigentliche Entwicklungstätigkeit selbst. Im Grunde genommen ist der Entwicklungsprozess von Software vergleichbar mit einer klassischen Produktentwicklung. Im Lean Software Development wird es als eine Form der Produktentwicklung bezeichnet, bei der die Software entweder Teil eines anderweitigen Produktes oder ein für sich stehendes Produkt ist.[158] Solch ein Produkt setzt sich aus verschiedenen Teilbestandteilen zusammen. Die Zuständigkeiten bei der (Weiter-) Entwicklung dieser Teile können nach verschiedenen Ansätzen definiert und verteilt werden. Dies geschieht mit den Zielsetzungen, die Softwarequalität bei einer möglichst hohen Entwicklungsgeschwindigkeit zu sichern und beiläufige Risiken zu minimieren.[159] Die Untersuchung der verschiedenen Modelle für die Verantwortlichkeiten im Quellcode ist auch für das Projektmanagement von Interesse, da der Umgang mit der Verantwortung direkt aus Sicht der Projektsteuerung beeinflusst werden kann.[160]

[155] vgl. McConnell, 2004, S. 663.
[156] vgl. IEEE Computer Society, 2014, S. 68, online im Internet.
[157] vgl. Jeffries und Beck, 2004, online im Internet.
[158] vgl. Poppendieck und Poppendieck, 2007, S. 17.
[159] vgl. Poppendieck und Poppendieck, 2007, S. 177.
[160] vgl. Bird et al., 2011, S. 4.

3.3.1 Modelle

In der Literatur sind verschiedene Ansätze für Verteilung von Verantwortung über den Quellcode (im Folgenden als „Code Ownership“ bezeichnet) präsent. Dabei wurde der Anstoß für eine solche Unterteilung von Kent Beck in XP gegeben, welcher dort zwischen der nicht vorhandenen, der individuellen und der kollektiven Verantwortung unterscheidet.[161] Letztere ist in der aktuellen Fassung im Prinzip des verteilten Codes inbegriffen.[162] Im weiteren Verlauf kam zu dieser Unterteilung eine weitere Zwischenebene hinzu, in welcher zwischen der eigentlichen Implementierung und der übergeordneten Verantwortung für die Software unterschieden wird.[163] Aus diesen Veröffentlichungen lassen sich drei Modelle ableiten, welche in Abbildung 4 dargestellt sind.

Abbildung 4: Modelle von Code Ownership



Quelle: Eigene Darstellung in Anlehnung an Nordberg III, 2003, S. 27.

Das erste Modell beschreibt die Zuweisung der Verantwortlichkeiten für eine Software oder einen Teilbestandteil dieser an eine einzelne Person. Diese wird als „Individual Ownership“[164] (Individuelle Verantwortung) oder „Strong Code Ownership“[165] (Starke Code-Verantwortung) bezeichnet. Teilweise wird dieses Modell dabei nochmals unterteilt. Falls es sich um eine Software mit kleinem Umfang handelt, welche in sich abgeschlossen ist, wird diese Rolle auch als „Product Specialist“ (Produktspezialist) bezeichnet. Werden die Verantwortlichkeiten anhand einzelner Subsysteme verteilt, handelt es sich hingegen um das sogenannte

[161] vgl. Beck, 2000, S. 59.
[162] vgl. Beck, 2005, S. 66.
[163] vgl. Smith, 2005, online im Internet.
[164] vgl. Beck, 2000, S. 59.
[165] vgl. Fowler, 2006, online im Internet.

„Subsystem Ownership“ (Verantwortung für ein Teilsystem).[166] Bei diesem Vorgehen wird ein System oder Teile eines Systems von einer Person verantwortet. Das bedeutet, dass nur diese Person Änderungen am Code durchführen darf. Bei neuen Anforderungen muss der Kontakt zu dieser Person hergestellt werden, um die Änderungen implementieren zu lassen. Die Mitwirkung anderer Personen beschränkt sich auf Vorschläge für die neuen Funktionen oder den neuen Quellcode.[167]

Beim „Weak Code Ownership“[168] (Schwache Code-Verantwortung) beziehungsweise „Code Stewardship“[169] (Code-Führung) wird dieses Prinzip aufgeweicht. Auch hier werden Verantwortlichkeiten einzelnen Personen zugewiesen, das Editieren des Codes durch andere Entwickler:innen ist jedoch ebenfalls erlaubt.[170] Die Codebasis liegt zunächst in der Verantwortung des gesamten Teams, einzelne Personen haben jedoch eine besondere Verantwortung für einzelne Teile des Codes und können bei Änderungen eine Art Vetorecht ausüben.[171] Die Rolle wird teils auch als die eines leitenden Architekts bezeichnet. Dies trifft jedoch eher dann zu, wenn eine Person die Verantwortung für ein gesamtes System übernimmt, die Entwicklungstätigkeit jedoch auch an andere Personen delegiert.[172]

Das dritte Modell wird als „Collective Code Ownership“[173] (Kollektive Code-Verantwortung) oder „Shared Code“[174] (geteilter Code) bezeichnet. Dabei wird keine Zuordnung von Teilen des Codes an einzelne Personen vorgenommen, sondern die Verantwortung für alle Teilbestandteile an das gesamte Team übertragen. Dadurch kann der Code auch von jedem Mitglied des Teams bearbeitet werden.[175] In der Literatur wird der Begriff auch mit einem Prinzip der fehlenden Verantwortung („No ownership“) verglichen.[176] Als entscheidender Unterschied zwischen den beiden Bezeichnungen wird das Sicherstellen der Integrität im Kontext des Gesamtsystems genannt, welches ein Teil von Collective Code Ownership ist.[177] Code Ownership impliziert dabei neben der Verantwortung selbst auch einen gewissen Stolz in Bezug auf die Software und das, was zur Erstellung dieser geleistet wurde.[178]

[166] vgl. Nordberg III, 2003, S. 27 f.
[167] vgl. Fowler, 2006, online im Internet.
[168] vgl. Fowler, 2006, online im Internet.
[169] vgl. Smith, 2005, online im Internet.
[170] vgl. Fowler, 2006, online im Internet.
[171] vgl. Smith, 2005, online im Internet.
[172] vgl. Nordberg III, 2003, S. 28 f.
[173] vgl. Wells, 1999, online im Internet; Fowler, 2006, online im Internet; Nordberg III, 2003, S. 29 f.
[174] vgl. Beck, 2005, S. 66.
[175] vgl. Nordberg III, 2003, S. 29.
[176] vgl. Fowler, 2006, online im Internet.
[177] vgl. Nordberg III, 2003, S. 29.
[178] vgl. Counsell, Eldh und Murphy, 2015, S. 18.

Die einzelnen Modelle bringen jeweils Vor- und Nachteile mit sich. Bei der individuellen Verantwortung über den Quellcode ist hervorzuheben, dass einzelne Entwickler:innen einen abgegrenzten Ausgabenbereich haben, in welchen sie effizient arbeiten und ihre eigene Vision verfolgen können. Des Weiteren sinkt der Bedarf an Kommunikation innerhalb des eigenen Bereiches und die externen Ansprechpartner sind klar definiert.[179] Die Nachteile der individuellen Verantwortung umfassen zunächst das Risiko, dass die Person nicht die jeweils beste Entscheidung zur Implementierung treffen kann und eigene Fehler unentdeckt bleiben.[180] Weiterhin besteht ein Ausfallrisiko, da nur eine einzelne Person die Änderungen des Teilbestandteils durchführt.[181] Durch das Warten auf Änderungen an anderen Teilen der Software, welche für die eigenen Anforderungen notwendig sind, kann die Entwicklung ausgebremst werden. Als Beispiel für eine einfache Änderung, welche durch das Prinzip des individuellen Code Ownerships erschwert wird, ist das Umbenennen einer (auch extern genutzten) Funktion. Die entsprechenden Änderungen an anderen Teilen der Software müssen dann dort durch den jeweils Verantwortlichen durchgeführt werden.[182]

Dieser Nachteil kann durch das Code Stewardship kompensiert werden. Während die Federführung bei den Verantwortlichen bleibt, werden Änderungen unter Vorbehalt auch für andere Entwickler:innen freigegeben.[183] Dies bringt jedoch einen erhöhten Kommunikationsaufwand mit sich. Außerdem können hier ebenfalls Engpässe entstehen, da zwar mehrere Personen den Quellcode editieren können, jedoch weiterhin nicht das gesamte Entwicklungsteam über die durchzuführenden Änderungen entscheiden kann.[184]

Dies kann wiederum durch das Modell von Collective Code Ownership umgangen werden, da dort jedes Teammitglied gleichwertige Berechtigungen zum Editieren des Codes erhält.[185] Da die Fallstudie vom Beispiel dieses Modells handelt, wird dieses Modell im Folgenden näher betrachtet und auch die damit verbundenen Implikationen gesondert dargestellt.

[179] vgl. Nordberg III, 2003, S. 28.
[180] vgl. Counsell, Eldh und Murphy, 2015, S. 18.
[181] vgl. Nordberg III, 2003, S. 28.
[182] vgl. Fowler, 2006, online im Internet.
[183] vgl. Smith, 2005, online im Internet.
[184] vgl. Nordberg III, 2003, S. 28.
[185] vgl. Wells, 1999, online im Internet.

3.3.2 Collective Code Ownership

Der Begriff Collective Code Ownership entstammt der ersten Veröffentlichung von Kent Becks XP-Methode, wo es als eine der zwölf Praktiken aufgeführt ist.[186] In der neueren Ausgabe aus dem Jahr 2005 wurde die Bezeichnung zu „Shared Code“ geändert, dies beschreibt jedoch nach wie vor das gleiche Grundprinzip.[187] Ein ähnlicher Ansatz ist bereits im IEEE Standard 610.12 von 1990 unter der Bezeichnung „egoless programming“ zu finden. Dieser beschreibt „eine Technik der Softwareentwicklung, die auf dem Konzept der Team- statt der Einzelverantwortung für die Programmentwicklung basiert. Sie soll verhindern, dass sich einzelne Programmierer so stark mit ihrer Arbeit identifizieren, dass eine objektive Bewertung beeinträchtigt wird“.[188]

Das Prinzip des Collective Code Ownerships besagt, dass alle am Softwareprodukt Mitwirkenden für die gesamte Codebasis verantwortlich sind. Im Umkehrschluss bedeutet dies auch, dass an jeder Stelle des Codes mitgewirkt werden kann. Um zu verhindern, dass Entwickler:innen ausschließlich an Bereichen mitarbeiten, in welchen sie bereits Erfahrung haben oder welche sie besonders interessieren, definiert XP Grundsätze zur regelmäßigen Rotation der Aufgabenzuweisungen.[189] Dadurch sollen die Mitwirkenden dazu angeregt werden, ihre Ideen und Fähigkeiten in allen Segmenten des Projektes einbringen zu können, um das Produkt zu verbessern.[190] Weiterführend sollen Engpässe während der Entwicklung nach Möglichkeit verhindert und schlechte Praktiken schneller aufgedeckt werden.[191] Die gemeinsame Arbeit an der gesamten Codebasis stärkt das Gemeinschaftsgefühl und fördert den Austausch untereinander sowie einen einheitlichen Entwicklungsstil.[192] Zusätzlich werden Ideen der Entwickler:innen gegenseitig wertgeschätzt und es entsteht ein gemeinsames Verständnis der geforderten Qualität.[193] Es steht damit im Gegensatz zur individuellen Zuständigkeit für bestimmte Teile einer Software. Dort müssen Anfragen und Änderungen für Teilbestandteile an die jeweiligen Verantwortlichen weitergeleitet und von ihnen abgenommen werden. Durch Collective Code Ownership entfällt dieses Nachvollziehen und direkte Prüfen von Änderungen

[186] vgl. Beck, 1999, S. 71.
[187] vgl. Beck, 2005, S. 66.
[188] übersetzt nach IEEE, 1990, S. 30: *„A software development technique based on the concept of team, rather than individual, responsibility for program development. Its purpose is to prevent individual programmers from identifying so closely with their work that objective evaluation is impaired.“*.
[189] vgl. Beck, 2005, S. 42, 47.
[190] vgl. Wells, 1999, online im Internet.
[191] vgl. Counsell, Eldh und Murphy, 2015, S. 19.
[192] vgl. Nordberg III, 2003, S. 29 f.; Lindstrom und Jeffries, 2004, S. 49.
[193] vgl. Beck, 2005, S. 66.

durch eine bestimmte Person.[194]

Mit dem Prinzip des Collective Code Ownerships gehen jedoch auch Risiken einher. Die Etablierung eines solchen Modells, welches sowohl organisatorisch, als auch für die Fähigkeiten der Entwickler:innen herausfordernd ist, kann sich als schwierig herausstellen, da für einzelne Softwarekomponenten bestimmte Fähigkeiten gefordert sind. Gleiches gilt bei der Verantwortung für Aufgaben und Fehler, welche trotz des kollektiven Ansatzes einer Zuweisung an Personen bedürfen, damit die technische Ursache des Fehlers gefunden und aus diesen gelernt werden kann.[195] Ein weiteres Hindernis kann Wissen sein, welches nicht explizit in Form von Dokumentation vorliegt und dadurch nur einem Teil der beteiligten Entwickler:innen bekannt ist, da sie dies schon angewandt haben. Dieses Wissen muss identifiziert und geteilt werden. Gegebenenfalls geht so jedoch auch der persönlichen Stolzes auf ein eigens entwickeltes Merkmal der Software verloren.[196] Untersuchungen haben außerdem ergeben, dass Entwickler:innen, welche bereits an der Entwicklung eines Codeausschnitts beteiligt waren, bei der Weiterentwicklung weniger Fehler produzierten. Auch die Fähigkeit zum Beantworten von Fragen zu einem Codeausschnitt wird stark davon beeinflusst, ob und wie lange sich Entwickler:innen mit diesem in der Vergangenheit beschäftigt haben.[197] Diese Punkte sprechen für die Definition klarer Verantwortlichkeiten. In der zweiten Auflage von XP wird ein gemeinschaftliches Verantwortungsgefühl als Voraussetzung genannt, welche für das erfolgreiche Einführen von Collective Code Ownership bestehen muss. Dadurch kann verhindert werden, dass sich durch unverantwortliches Handeln und nicht nachvollziehbare Änderungen ein negativer Einfluss auf die Qualität der Entwicklung einstellt.[198]

Neben der Definition in XP finden sich einige Aspekte von Collective Code Ownership auch in anderen Veröffentlichungen wieder. Die Prinzipien des Lean Software Developments definieren in Bezug auf die Entwickler:innen, dass deren Expertise alle Teilbereiche des Produktes abdecken muss und die Personen sich mit ihrem persönlichen Wissen in den entsprechenden Bereichen einbringen sollen. Andererseits wird jedoch auch das Team als solches hervorgehoben, welches gemeinsam an der Lösung von Problemen arbeitet. Dabei sollen alle vorhandenen Ressourcen genutzt und sich gegenseitig geholfen werden, um den Fortschritt zu maximieren und voneinander zu lernen.[199] Nach Poppendieck bekommt „keiner im Team [..]

[194] vgl. Maruping, Zhang und Venkatesh, 2009, S. 358.
[195] vgl. Nordberg III, 2003, S. 30.
[196] vgl. Sedano, Ralph und Péraire, 2016, S. 5.
[197] vgl. Bird et al., 2011, S. 5.
[198] vgl. Beck, 2005, S. 66.
[199] vgl. Poppendieck und Poppendieck, 2007, S. 130.

die Anerkennung für seine Arbeit, bis die gesamte Arbeit, zu der sich das Team verpflichtet hat, erledigt ist, so dass jeder nach seinen Möglichkeiten beiträgt“.[200]

Die Umsetzbarkeit von Collective Code Ownership hängt jedoch auch von verschiedenen Begleitfaktoren ab. Jede Gruppe von Softwareentwickler:innen unterscheidet sich voneinander, bestimmt durch ihre Erfahrung, Fähigkeiten, die Kultur des Unternehmens und ihre Vorgaben in Bezug auf den Quellcode selbst. Dadurch entscheidet sich ihr Umgang mit Verantwortung und der Erfolg ihres Entwicklungsmodells. Entsprechend kann das Modell der kollektiven Verantwortung für die eine Gruppe gut funktionieren, während es in einem anderen Kontext schnell an seine Grenzen stößt.[201] Collective Code Ownership ist „ein Gefühl, das erzeugt werden muss, und keine Richtlinie, die einfach verordnet werden kann“.[202] Standards in der Zusammenarbeit sind daher notwendig, da sonst die Gefahr von sich wiederholenden Änderungen des Quellcodes basierend auf verschiedenen Entwicklungsstilen bestehen.[203] Dies ist insbesondere dann der Fall, wenn an einer gemeinsamen Codebasis gearbeitet wird und wechselseitige Abhängigkeiten zwischen den Aufgaben verschiedener Personen bestehen. Collective Code Ownership ist ein Prozess, welcher die Rolle der Entwickler:innen so auslegt, dass sie ihre Arbeit selbst organisieren und deren Ergebnisse mit anderen teilen. Der Koordinationsaufwand ist dabei jedoch nicht zu vernachlässigen.[204]

3.3.3 Metriken

Für die Analyse einer Software in Bezug auf die Verteilung von Verantwortlichkeiten stellt sich die Frage, anhand welcher Metriken sich der Grad von Code Ownership feststellen lässt. Im Folgenden werden daher verschiedene Metriken vorgestellt, welche sich aus bei der Softwareentwicklung entstehenden Daten ableiten lassen. Diese entstehen durch die Verwendung einer Software für die Versionsverwaltung, was bei vielen Softwareentwicklungsprojekten der Fall ist.[205] Die zentrale Aufgabe eines Versionskontrollsystems ist es, Änderungen an Dateien zu Erfassen, um diese auch zu einem späteren Zeitpunkt nachvollziehen und bei Bedarf zu

[200] übersetzt nach Poppendieck und Poppendieck, 2007, S. 130: *„No one on the team gets credit for being done until all the work the team committed to is done, so everyone pitches in however they are able.“*.
[201] vgl. Counsell, Eldh und Murphy, 2015, S. 18.
[202] übersetzt nach Sedano, Ralph und Péraire, 2016, S. 6: *„Team code ownership is a feeling to be engendered not a policy to be decreed.“*.
[203] vgl. Nordberg III, 2003, S. 30.
[204] vgl. Maruping, Zhang und Venkatesh, 2009, S. 358.
[205] vgl. Stackoverflow.com, 2021, online im Internet.

einem älteren Stand zurückkehren zu können. Neben den Änderungen werden auch Metadaten wie der Zeitpunkt der Änderungen und die Namen der Autor:innen gespeichert.[206] Verteilte Versionskontrollsysteme vereinfachen zusätzlich die Zusammenarbeit zwischen mehreren Entwickler:innen und stellen sicher, dass die Daten auch im Falle eines Ausfalls nicht verloren gehen.[207] Ein populärer Vertreter der verteilten Versionsverwaltung ist die Software Git. Diese wird auch im Rahmen der Fallstudie und daher zur Extraktion der Daten verwendet und entsprechend in Abschnitt 5.1.1 näher betrachtet.

Zur Erfassung des Code Ownerships existieren verschiedene Ansätze, welche auf der Erfassung von Beiträgen zur Software aufbauen. Diese Beiträge können sich sowohl auf den Quellcode selbst, als auch die Überprüfung bereits existierenden Quellcodes oder dem Verfassen von Dokumentation beziehen. Anhand des entstandenen Beitrags lässt sich ableiten, dass diese Person Verantwortung für den Teil der Software trägt und Expertise für diesen aufgebaut hat.[208] Der erste Ansatz beschreibt die Erfassung von konkreten Beiträgen zum Quellcode selbst. Der Umfang des Beitrags wird dabei nicht beachtet, lediglich die Zahl der Entwickler:innen pro Teilkomponente wird erfasst.[209] Um dies zu präzisieren, kann auch der Anteil einer Person an der Gesamtzahl der Änderungen an einer Komponente betrachtet werden.[210] Noch detaillierter wird dies durch die Betrachtung der tatsächlich geänderten Codezeilen im Kontext einer Komponente. Dadurch wird nicht nur die Anzahl der Änderungen als solche, sondern auch der konkrete Umfang der einzelnen Änderungen mit einbezogen.[211]

Als zweiter Ansatz kann das Code Review hinzugezogen werden. Eine Untersuchung hat gezeigt, dass die Durchführung von Code Reviews einen positiven Effekt auf die Qualität einer Software haben kann.[212] Auch hier gilt der Grundsatz, dass Entwickler:innen durch die Beschäftigung mit den Änderungen am Quellcode und potenziellen Beiträgen Expertise hinzugewinnen und einen Teil des Code Ownerships für sich beanspruchen können. Durch die Auswahl eines oder die Kombination mehrerer Ansätze lässt sich somit ein Wert für jede Person berechnen, welche Beiträge zum Quellcode geleistet hat. Anhand dieser Werte lässt sich anschließend herausstellen, ob Korrelationen zwischen der Entwicklungsaktitivät auf der

[206] vgl. Chacon und Straub, 2014, S. 8.
[207] vgl. Chacon und Straub, 2014, S. 10 f.
[208] vgl. Mockus und Herbsleb, 2002, S. 505.
[209] vgl. Weyuker, Ostrand und Bell, 2008, S. 540.
[210] vgl. Bird et al., 2011, S. 6.
[211] vgl. Rahman und Devanbu, 2011, S. 492 f.
[212] vgl. Morales, McIntosh und Khomh, 2015, S. 7–10.

einen und anderer Kennzahlen wie die Fehleranfälligkeit der Software auf der anderen Seite bestehen.

In Abschnitt 5.2.3 erfolgt die Auswahl der Metrik für die Analyse im Rahmen der Fallstudie. Zunächst werden Kriterien für die Verteilung von Verantwortlichkeiten erarbeitet, welche im Rahmen letzterer anhand der Daten analysiert werden sollen.

4 Kriterien zur Verteilung von Verantwortlichkeiten

> „Ein Team, das zusammenarbeitet, kann mehr erreichen als die Summe der Bemühungen seiner einzelnen Mitglieder. Macht zu teilen ist pragmatisch, nicht idealistisch."
>
> – Kent Beck, 2005[213]

Die Organisation eines Entwicklungsteams bestimmt die Art und Weise, wie innerhalb eines Projektes Verantwortlichkeiten zugewiesen werden. Dabei muss jedoch auch die Frage gestellt werden, welchen Einfluss diese Verteilung auf die Entwicklung und den Betrieb der Software haben. Um den Einfluss eines Prinzips wie das des Collective Code Ownerships überprüfen zu können, werden daher im Folgenden Kriterien betrachtet, anhand derer man den Erfolg eines bestimmten Verteilungsmodells messen kann. Das Ziel ist dabei die termingerechte Erstellung zuverlässiger und nutzungsgerechter Software unter Einhaltung der vorgegebenen Rahmen- und Kostenbedingungen.[214] Die Verteilung von Verantwortlichkeiten während der Implementierung beeinflusst eine Teilmenge dieser Erfolgsfaktoren. Daher können die nachstehenden Kriterien auch als Erfolgsfaktoren für die Entwicklung von Software bezeichnet werden, auf die eine aktive Einflussnahme durch eine bestimmte Verteilung von Verantwortlichkeiten im konkreten Anwendungsfall möglich ist.

Die im Folgenden dargestellten Kriterien beschränken sich im wesentlichen auf die Eingrenzung der Themenbereiche, welche in der Forschungsfrage vorgenommen wurde. Diese bezieht sich auf die Fehleranfälligkeit der Software als Teil der Softwarequalität, die Dauer der Entwicklung und die Expertise des Entwicklungsteams. Die Themenbereiche werden aus theoretischer Sicht erläutert, um daraus in Abschnitt 5.3 Hypothesen abzuleiten und diese anschließend anhand der Fallstudie zu überprüfen.

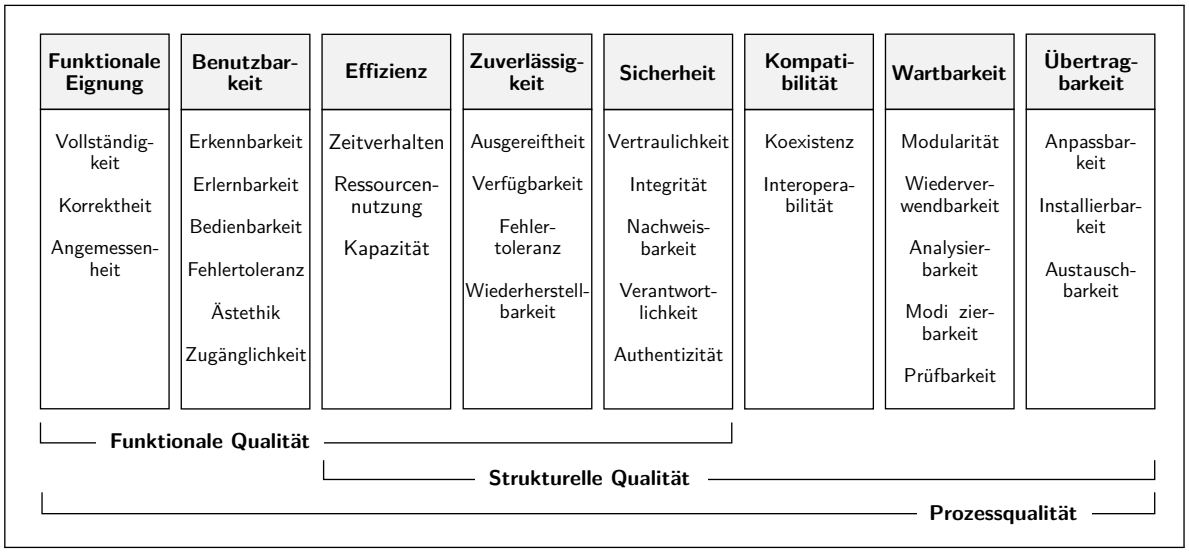
4.1 Qualität der Software

Ein zentraler Punkt in der Softwareentwicklung ist die Qualität der entstandenen Anwendung. Nach ISO Standard 25010 ist diese definiert als „[...] das Ausmaß, in dem das System die expliziten und implizierten Bedürfnisse der verschiedenen Beteiligten erfüllt und somit einen

[213] übersetzt nach Beck, 2005, S. 155: *„A team working together can accomplish more than the sum of it's members separate efforts. Sharing power is pragmatic, not idealistic."*.
[214] vgl. Broy und Kuhrmann, 2021, S. 12.

Wert für sie darstellt".[215] Erweitert werden kann diese durch „[...] ausdrücklich dokumentierte Entwicklungsstandards und implizite Merkmale, die von jeder professionell entwickelten Software erwartet werden".[216] Der Standard definiert acht Charakteristiken, welche als Eigenschaften einer Software Rückschlüsse auf die Qualität dieser geben. Sie sind in Abbildung 5 dargestellt.

Abbildung 5: Software-Produktqualität nach ISO 25010



Quelle: Eigene Darstellung in Anlehnung an ISO, 2011, S. 4.

Demnach stellt sich die Qualität einer Software anhand der Charakteristiken der funktionalen Eignung, Benutzbarkeit, Effizienz, Zuverlässigkeit, Sicherheit, Kompatibilität, Wartbarkeit und Übertragbarkeit zusammen. Zu diesen werden jeweils noch zugehörige Teilmerkmale definiert.[217] Aus ihnen können entsprechende Anforderungen abgeleitet werden, welchen die Software standhalten muss und auf die durch die Verteilung von Verantwortlichkeiten innerhalb der Entwicklung Einfluss genommen werden kann. Die Charakteristiken aus ISO 25010 können auch in Kriterien für funktionale und strukturelle Qualität sowie die Qualität des Entwicklungsprozesses eingeteilt werden. Funktionale Qualität beschreibt, dass die Anwendung ihre Aufgabe zuverlässig erfüllt. Das bedeutet, dass sie sowohl möglichst wenige Fehler aufweist, als auch performant und die Nutzung möglichst leicht erlernbar ist. Die strukturelle Qualität umfasst Anforderungen an den Quellcode selbst. Dazu gehören die Testbarkeit,

[215] übersetzt nach ISO, 2011, S. 2: *„The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value."*.

[216] übersetzt nach Pressman, 2001, S. 199: *„[...] explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software."*.

[217] vgl. ISO, 2011, S. 10.

Wartbarkeit, Verständlichkeit, Effizienz und Sicherheit. Im Kontrast zur funktionalen Qualität sind diese Kriterien nur bedingt überprüfbar. Zur Prozessqualität gehören abschließend die Einhaltung von Fristen, des Budgets und Qualitätsansprüchen in einer wiederholbaren Art und Weise.[218] In einem anderen Beispiel wird bei der Prozessqualität nochmals zwischen dem Prozess im eigentlichen Sinne und dessen Umsetzung im tatsächlichen Projekt differenziert. Bei letzterem wird dann von Projektqualität gesprochen. Ein bewährter Prozess begünstigt zwar eine hohe Projektqualität, ist jedoch kein Garant dafür.[219]

Aus der Perspektive agiler Methoden, in welcher die Entwicklung im Vordergrund steht, kann durch den Verzicht auf Dokumentation auch die explizite Festlegung von Standards in den Hintergrund geraten. Daher wird der Qualitätsbegriff durch ein Zusammenspiel aus der Qualität des Quellcodes selbst und den darum angeordneten Praktiken wie der testgetriebenen Entwicklung definiert.[220] Das „Qualitätsmanagement basiert auf gemeinsamen bewährten Verfahren und nicht auf einer formalen Dokumentation“.[221] Die Modelle selbst spezifizieren dabei keine Anforderungen aus Qualitätssicht an die Software selbst, sondern adressieren diese an den umgebenden Entwicklungsprozess. Beispiele dafür sind das Prinzip „Build quality in“ aus dem Lean Software Development oder verschiedene Praktiken aus XP, welche zur Qualität des Produkts beitragen sollen. Es wird jedoch nicht definiert, wie diese Qualität am Ende gemessen werden kann. Im Folgenden werden die funktionalen und strukturellen Qualitätsanforderungen näher untersucht.

4.1.1 Funktionale Qualität

Von den Charakteristiken der Softwarequalität aus ISO 25010 gehören die funktionale Eignung, Benutzbarkeit, Effizienz, Zuverlässigkeit und Sicherheit zur funktionalen Qualität der Anwendung. Die funktionale Eignung einer Software beschreibt den Grad, zu dem die Anwendung den Anforderungen entspricht, wenn sie unter bestimmten Bedingungen verwendet wird. Sie setzt sich dabei aus den Faktoren Vollständigkeit, Zweckmäßigkeit und Korrektheit der Funktionen zusammen.[222] Die Qualität einer Software wird oft auf dieses singuläre Kriterium reduziert, da funktionale Fehler in Software allgegenwärtig sind.[223] Die Benutzbarkeit und Effizienz bestimmen das Erlebnis der Benutzer:innen als maßgebliche Faktoren.

[218] vgl. Chappell, 2013, S. 3 f., online im Internet.
[219] vgl. Ludewig und Lichter, 2013, S. 66.
[220] vgl. Sommerville, 2016, S. 714 f.
[221] Sommerville, 2016, S. 714.
[222] vgl. ISO, 2011, S. 10 f.
[223] vgl. Hoffmann, 2013, S. 7.

Neben der Abdeckung von Anforderungen kommt auch der Zuverlässigkeit der Software eine hohe Bedeutung zu, da Software in vielen sicherheitskritischen Anwendungsbereichen im Einsatz ist.[224] Zur Zuverlässigkeit zählen unter anderem auch die Verfügbarkeit des Systems, die Fehlertoleranz und die Wiederherstellbarkeit nach einem Ausfall.[225] Die Nutzbarkeit der Software beruht im Grunde genommen auf den beiden vorherigen Charakteristiken, erweitert diese jedoch um Aspekte, welche sich teilweise auch auf persönliche Präferenzen und Fähigkeiten beziehen. Der Begriff subsumiert die Eigenschaften, welche sich mit den Interaktionen mit Benutzer:innen befassen.[226] Dazu gehören die Erkennbarkeit des Zwecks eines Systems oder einer Funktion, Erlernbarkeit, Einfachheit der Benutzung, Ästhetik der Benutzeroberfläche, Barrierefreiheit sowie Rückmeldungen zur Vermeidung von Fehlern durch Nutzer:innen.[227]

Das Testen von Software bezieht sich meist ausschließlich auf funktionale Qualität. Gleichzeitig wird die funktionale Qualität maßgeblich durch das Testen gesichert. Die strukturelle Qualität und die Qualität des Prozesses haben auch viele Verbindungen zur funktionalen Qualität, wodurch beispielsweise Verbesserungen im Entwicklungsprozess auch die funktionale Qualität fördern. Im umgekehrten Fall können jedoch auch Verbesserungen in einem Bereich die Qualität eines anderen Teilbereichs negativ beeinflussen.[228]

4.1.2 Strukturelle Qualität

Die strukturelle Qualität von Software ist von außen nicht direkt erkennbar. Sie bezieht sich auf die Strukturierung des Quellcodes selbst und ist damit hauptsächlich für die Entwickler:innen von Bedeutung.[229] Die Kriterien Effizienz, Zuverlässigkeit, Sicherheit, Kompatibilität, Wartbarkeit und Übertragbarkeit aus ISO 25010 lassen sich als Teil der strukturellen Qualität klassifizieren. Die Effizienz bestimmt, wie sich das System in Bezug auf die benötigte Zeit und genutzte Ressourcen für bestimmte Aufgaben verhält. Durch die Zuverlässigkeit ist beschrieben, wie tolerant das System gegenüber Fehlern ist und ob es einen ausgereiften Funktionsumfang besitzt. Zur Sicherheit gehört das Schützen von persönlichen Daten und Informationen sowie die Beschränkung des Zugangs zu solchen Ressourcen ausschließlich

[224] vgl. Hoffmann, 2013, S. 8.
[225] vgl. ISO, 2011, S. 13.
[226] vgl. Hoffmann, 2013, S. 8.
[227] vgl. ISO, 2011, S. 12 f.
[228] vgl. Chappell, 2013, S. 4 f., online im Internet.
[229] vgl. Chappell, 2013, S. 4, online im Internet.

für berechtigte Personen und Anwendungen. Dies wird durch die Wahrung von Vertraulichkeit, der Integrität der Informationen, Nachvollziehbarkeit und Rechenschaftspflicht bei Veränderungen umgesetzt. Die Kompatibilität bezieht sich auf die Möglichkeiten des Teilens von Informationen mit anderen Anwendungen. Die Wartbarkeit misst die Effektivität bei Modifikationen des Systems, sprich die Modularität, Wiederverwendbarkeit von Komponenten, Analysefähigkeit, Veränderbarkeit und Testfähigkeit. Unter Übertragbarkeit wird verstanden, wie aufwendig sich ein Transfer der Software in eine neue Umgebung gestaltet.[230]

Die Qualitätskriterien lassen sich dabei jedoch nicht als ganzheitliche Qualität vereinen, sondern müssen teilweise in ihrer Wichtigkeit für das konkrete Produkt abgewägt werden. Dies zeigt sich am Beispiel der Kriterien Leistungseffizienz und Portabilität. Während letztere von einer recht offenen und allgemeinen Implementierung profitiert, wird Leistungseffizienz durch eine hohe Spezialisierung auf bestimmte Technologien und die entsprechende Infrastruktur erreicht. Nach Weinberg lässt sich hier ein evolutionstheoretischer Ansatz übertragen, nach dem eine möglichst optimale Anpassung eines Systems an ein bestimmtes Umfeld die Anpassungsfähigkeit jenes an neue Umgebungen reduziert.[231] Die Merkmale der Portabilität und Leistungseffizienz stehen entsprechend in einem gewissen Konflikt zueinander.[232]

Eine mangelnde Qualität der Software exponiert beispielsweise Risiken in Bezug auf die Rentabilität der Software, da durch Fehler in der Anwendung hohe Kosten entstehen können. Dabei wird zwischen den Kosten zur Prävention, beispielsweise der Prüfung und Nachbesserung der Qualität, während der Entwicklung und den Reparatur- und Folgekosten im Falle von Fehlern im Einsatz unterschieden. Insbesondere letztere können im finalen Produkt dabei je nach Anwendungsfall die Herstellungskosten der Software übersteigen.[233] Neben Risiken in Bezug auf die Qualität der Software werden im nächsten Abschnitt weiterführend ausgewählte Risiken in der Entwicklung von Software dargestellt, welche in der Fallstudie näher betrachtet werden sollen.

[230] vgl. ISO, 2011, S. 11–16.
[231] vgl. Weinberg, 1971, S. 21.
[232] vgl. Hoffmann, 2013, S. 10.
[233] vgl. Ludewig und Lichter, 2013, S. 62 f.

4.2 Umgang mit Risiken in der Entwicklung

Neben Risiken in Bezug auf die Qualität des Produktes existieren in der Softwareentwicklung weitere Ausprägungen, welche den Verlauf und Erfolg eines Projektes beeinflussen können. Teilweise wird der Begriff des Risikos und des darauf bezogenen Managements mit dem Umgang mit Unsicherheiten verglichen. Bei letzterem geht es nicht nur um den Umgang mit Risiken im Sinne von wahrgenommenen Chancen und Bedrohungen, sondern primär darum, „die vielen Quellen der Unsicherheit zu identifizieren und zu verwalten, die unsere Wahrnehmung von Bedrohungen und Chancen hervorrufen und prägen“.[234] Diese Unsicherheiten sind eine unumgehbare Charakteristik der Softwareentwicklung, welche jedoch auch Chancen eröffnen kann. Sie kann sich beispielsweise auf die Aufgaben, die Anforderungen oder auch auf die Ressourcen in der Entwicklung beziehen.[235]

Die Unsicherheiten werden als potentielle Manifestation von Projektrisiken angesehen. Ein Beispiel dafür sind Unsicherheiten in Bezug auf Aufwände, wo eine falsche Abschätzung zu einer Verzögerung des Projektes führen und damit zu einem Projektrisiko werden kann. Der Erfolg eines Softwareentwicklungsprojektes steht in direkter Abhängigkeit zu den möglichen Risiken und damit auch den ihnen vorausgehenden Unsicherheiten. In einem kompetitiven Marktumfeld, in welcher in einer möglichst kurzen Zeit versucht wird, ein erfolgreiches Produkt auf den Markt zu bringen, sind besonders viele Risiken involviert.[236] In der Softwareentwicklung und insbesondere im agilen Umfeld wird daher versucht, den Unsicherheiten bewusst entgegenzutreten und sie für sich zu Nutzen. Das Lean Software Development schreibt vor, dass Entscheidungen zum spätmöglichsten Zeitpunkt getroffen werden sollen, um eine möglichst breite Entscheidungsgrundlage zur Verfügung zu haben. Dies bezieht sich insbesondere auf Entscheidungen, deren Umkehr große Risiken aufweist.[237] In Scrum wird der iterative Prozessrahmen als Möglichkeit angesehen, die Vorhersehbarkeit der Entwicklung zu verbessern und Risiken zu vermeiden.[238] Zur weiteren Mitigation soll eine kurze Sprintdauer beitragen, um auf Änderungen reagieren zu können.[239] Ähnliche Grundsätze sind auch in XP definiert, wo ebenfalls durch kurze Entwicklungszyklen und einem inkrementellen Planungsansatz eine Reaktion auf Unsicherheiten ermöglicht werden soll.[240]

[234] übersetzt nach Ward und Chapman, 2003, S. 98: *„It is about identifying and managing all the many sources of uncertainty which give rise to and shape our perceptions of threats and opportunities.“*.
[235] vgl. Dönmez und Grote, 2015, S. 193.
[236] vgl. Boban, Pozgaj und Sertić, 2003, S. 78.
[237] vgl. Poppendieck und Poppendieck, 2007, S. 32.
[238] vgl. Schwaber und Sutherland, 2020, S. 3, online im Internet.
[239] vgl. Schwaber und Sutherland, 2020, S. 7, online im Internet.
[240] vgl. Beck, 2005, S. 2.

Die Fallstudie bezieht sich dabei neben der Qualität der Software hauptsächlich auf Risiken in Bezug auf die Geschwindigkeit der Entwicklung und dem Erhalt von Expertise im Entwicklungsteam. Zur Vermeidung von Verzögerungen in der Entwicklung kommt der Aufwandsschätzung eine besondere Rolle zu, da auf deren Basis die Planung der nächsten Schritte erfolgt. Dabei wird versucht, die Größe der Aufgaben möglichst klein zu halten und bei Bedarf eine Teilung in kleinere Einheiten durchzuführen.[241] Nach XP ist eine Unterschätzung von Aufwänden unter allen Umständen zu vermeiden. Andererseits sollen sich verändernde Anforderungen und Rahmenbedingungen nach Aussage von Beck kein Problem für ein Projekt unter Anwendung der XP-Methode darstellen, da es darauf ausgelegt ist, ständig auf Veränderungen reagieren zu können.[242] In der agilen Softwareentwicklung profitiert man allgemein von einem vergleichsweise kurzen Planungshorizont. Die Geschwindigkeit der Entwicklung hängt maßgeblich vom Prozess ab. Im Lean Software Development wird die Zykluszeit als zentrale Metrik der Performanz eines Entwicklungsprozesses definiert. Zuverlässige, wiederholbare Abläufe sind nach dieser Definition ein Indiz für dessen Leistungsfähigkeit. Die Zykluszeit ist ein Indikator für Fehlerrate, Produktivität, Komplexität, Änderungsintoleranz und die allgemeine Qualität der Software.[243] Ähnlich wie in linearen Modellen führt eine lange Aufgabenliste (Backlog) dazu, dass bereits definierte Aufgaben nach einiger Zeit obsolet werden.[244] Collective Code Ownership kann dabei beispielsweise dazu beitragen, dass die einzelnen Aufgaben an mehrere Entwickler:innen verteilt werden können und so auch die Verwaltung und Priorisierung von Aufgaben anhand eines Backlogs vereinfacht wird.

Ein weiteres Risiko in der Softwareentwicklung ist das Abwandern von Expertise aus einem Projekt. Wird eine Person beispielsweise in einem anderen Projekt mit höherer Priorität benötigt und daher dorthin transferiert, muss sie durch eine Neue ersetzt werden. Dabei kann es dazu kommen, dass erfahrene Entwickler:innen, insbesondere mit Blick auf projektspezifisches Wissen, mit Unerfahrenen ersetzt werden. Die Problematik dabei ist, dass der größte Vermögenswert in der Softwareentwicklung die Expertise der einzelnen Entwickler:innen ist und so Verzögerungen eintreten können.[245] Nach XP ist eine gewisse Fluktuation innerhalb des Projektteams als positiver Faktor anzusehen. Die Methode hebt dabei hervor, dass sich neue Perspektiven auf die Software eröffnen.[246] Trotzdem ist zu erwarten, dass mit einem Wechsel

[241] vgl. Dönmez und Grote, 2015, S. 195 f.
[242] vlg. Beck, 1999, S. 75 f.
[243] vgl. Poppendieck und Poppendieck, 2007, S. 98.
[244] vgl. Poppendieck und Poppendieck, 2003, S. 2.
[245] vgl. Rus und Lindvall, 2002, S. 26.
[246] vgl. Beck, 1999, S. 76.

von Personen auch ein Teil der Expertise verloren geht. Darum besteht die Anforderung, dem Verlust des Wissens mit geplanten Maßnahmen entgegenzuwirken.

4.3 Erhalt und Ausbau von Expertise

Die Softwareentwicklung ist eine wissensintensive Aktivität in einer Umgebung, welche oft durch Ungewissheit geprägt ist. Durch die wachsende Komplexität und Qualitätsanforderungen von Software beruht der Erfolg eines Entwicklungsprojektes auf dem Wissen und der Erfahrung der Personen, welche innerhalb des Projekts vorhanden sind. Diese werden als hauptsächlicher Wettbewerbsvorteil einer Organisation angesehen.[247] Bereits im Lean Software Development wird der Grundsatz des Schaffens von Wissen definiert, durch welches der Entwicklungsprozess optimiert werden soll.[248] Eine zentrale Herausforderung ist, dieses intellektuelle Kapital zu halten und zu erweitern, um das vorhandene Kompetenzniveau abzusichern. Nur so kann die Organisation ihren Anforderungen gerecht werden. Das zentrale Problem hierbei ist, dass das Wissen an Individuen gebunden ist und mit dem Verlust einer erfahrenen Person das Wissen ebenfalls zumindest teilweise verloren geht. In diesem Zusammenhang ist es notwendig, dass ein konstanter Wissensaustausch zwischen erfahrenen und unerfahrenen Entwickler:innen etabliert ist.[249]

Dabei wird zwischen implizitem und explizitem Wissen unterschieden. Während expliziertes Wissen in Form von Dokumentation vorliegt, beruht das implizite Wissen auf der Erfahrung und den Fähigkeiten der einzelnen Personen. Das Wissensmanagement versucht, implizites Wissen in explizites Wissen zu verwandeln und individuelles Wissen auf Gruppen zu transferieren.[250] In Folge dessen hat das Wissensmanagement in der Softwareentwicklung eine wachsende Bedeutung erfahren. Es kann „als Teil der Strategie zur Risikoprävention und -minderung gesehen werden, da es explizit Risiken adressiert, welche oft ignoriert werden“.[251] Das Vorhandensein von Expertise in einem Projekt führt jedoch nicht direkt zum Erfolg, da dieses Wissen auch koordiniert werden muss. Eine Studie hat dabei gezeigt, dass zwar das Vorhandeinsein von Expertise in einem Projektteam bereits positive Auswirkungen auf die

[247] vgl. Levy und Hazzan, 2009, S. 60.
[248] vgl. Poppendieck und Poppendieck, 2007, S. 29.
[249] vgl. Rus und Lindvall, 2002, S. 26.
[250] vgl. Levy und Hazzan, 2009, S. 61 f.
[251] übersetzt nach Rus und Lindvall, 2002, S. 29: *„Organizations can view knowledge management as a risk prevention and mitigation strategy, because it explicitly addresses risks that are too often ignored.“*.

Leistungsfähigkeit hat, etablierte Koordinationsprozesse diesen Effekt jedoch deutlich verstärken. Zu diesen Prozessen gehören primär das Erfassen des Bedarfs und der Lokalisierung des Wissens sowie dessen Nutzung.[252]

Eine zentrale Anforderung an das Wissensmanagement ist somit die Zusammenarbeit und das damit verbundene Teilen von Wissen mit anderen. Dabei besteht die Herausforderung, dass zum einen Expert:innen bei Bedarf identifiziert werden müssen und das Wissen dieser auch bei Verlassen der Organisation möglichst nicht verloren gehen darf. Diese Herausforderung besteht bereits zu Beginn eines Projektes.[253] Agile Ansätze unterstützen diese Prozesse dadurch, dass der Fokus von festgelegten Prozessen hin zu den beteiligten Personen und ihren Interaktionen gerichtet wird.[254] Die Softwareentwicklung ist eine Gruppenaktivität, deren Mitglieder sich über viele Orte verteilen können und dabei trotzdem in einem ständigen Austausch miteinander stehen. Dieser Austausch ist meist an einen Wissenstransfer geknüpft, weshalb innerhalb eines Projektes ein Weg gefunden werden muss, wie die Zusammenarbeit und die Verteilung von Wissen erreicht werden kann.[255] Der Austausch mit anderen ist somit eine zentrale Säule des Wissensmanagements, gleichzeitig jedoch zeit- und arbeitsintensiv.[256]

Aus Sicht der Softwareentwicklung gibt es verschiedene Ansätze, durch die eine Distribution des Wissens erreicht werden soll. Ein Beispiel dafür sind die festgelegten Ereignisse der Scrum-Methode, in denen ein Austausch stattfindet. Weitere Beispiele umfassen die Paarprogrammierung, Rotation der Zuständigkeiten, einen regelmäßigen Austausch mit Fachexperten auf der Seite des Kunden sowie die Zusammenstellung crossfunktionaler Teams.[257] Auch durch das Anwenden des Collective Code Ownerships Prinzips kann eine Verteilung des Wissens innerhalb des Teams gefördert werden.[258] Es wird jedoch neben den Praktiken explizit Wert darauf gelegt, dass die Entwickler:innen ihr Wissen untereinander austauschen und auch den zeitlichen Rahmen dafür erhalten.[259] Allgemein setzen die Methoden der agilen Softwareentwicklung viel Wert auf den Wissensaustausch der Entwickler:innen untereinander. Auch das Lean Software Development formuliert den Grundsatz des „Create knowledge“, welcher neben dem Wissen zu technischen Aspekten beispielsweise auch die Dokumentati-

[252] vgl. Faraj und Sproull, 2000, S. 1554, 1558–1564.
[253] vgl. Rus und Lindvall, 2002, S. 28.
[254] vgl. Levy und Hazzan, 2009, S. 62.
[255] vgl. Rus und Lindvall, 2002, S. 28.
[256] vgl. Levy und Hazzan, 2009, S. 60.
[257] vgl. Dorairaj, Noble und Malik, 2012, S. 69 f.
[258] vgl. Lindstrom und Jeffries, 2004, S. 49.
[259] vgl. Beck, 2005, S. 141.

on über getroffene Entscheidungen als wertvoll ansieht, um gleichartige Lernprozessen in Zukunft zu vermeiden.[260]

4.4 Ableitung von Kriterien für die Fallstudie

Die in der Forschungsfrage vorgenommenen Eingrenzung der Bereiche, welche in Bezug auf die Verteilung von Verantwortlichkeiten hin untersucht werden sollen, wurde in diesem Kapitel weiter konkretisiert. Im Rahmen der Fallstudie wird eine Einschränkung auf drei Kriterien vorgenommen, welche sich potenziell anhand der Daten überprüfen lassen, welche für die Analyse zur Verfügung stehen. Diese orientieren sich an der Eingrenzung zur Fehleranfälligkeit, der Entwicklungsdauer und der Expertise des Entwicklungsteams aus der Forschungsfrage.

Der erste Teilbereich bezieht sich auf die Fehleranfälligkeit als Teil der Softwarequalität. Diese ist nach ISO 25010 in den Bereich der Zuverlässigkeit einzuordnen und betrifft dementsprechend die strukturelle Qualität der Software. Die Ausgereiftheit als Kriterium der Zuverlässigkeit bezieht sich auf den „Grad, zu dem ein System, Produkt oder eine Komponente die Anforderungen an die Zuverlässigkeit bei normalem Betrieb erfüllt“.[261] Dadurch wird es von der Fehlertoleranz abgegrenzt, welche eine Erfüllung dieser Anforderungen auch bei irregulärem Betrieb bezeichnet, beispielsweise bei der Fehlfunktion einer Hardwarekomponente.[262] In Bezug auf die Verteilung von Verantwortlichkeiten ist zu erwarten, dass die einzelnen Teilkomponenten der Software unterschiedliche Beteiligungen der Entwickler:innen aufweisen und somit das Code Ownership unterschiedlich hoch ausfällt. Dementsprechend zielen die Untersuchungen darauf ab, inwiefern der aktuelle Code Ownership Wert einer Teilkomponente einen Einfluss auf deren Fehleranfälligkeit hat. Erweitert werden kann dies durch die Untersuchung persönlicher Code Ownership Werte der Entwickler:innen und dessen Einfluss auf Fehler in den Teilen des Quellcodes, welche in der Verantwortung der jeweiligen Person stehen.

Das zweite Kriterium bezieht sich auf die Dauer der Entwicklung als Teil der Risiken einer Softwareentwicklung. Das Ziel einer solchen ist es, die vorgegebenen Rahmenbedingungen in Bezug auf das Projekt zu erfüllen.[263] Durch Verzögerungen in der Entwicklung kann es

[260] vgl. Hibbs, Jewett und Sullivan, 2009, S. 20 f.
[261] übersetzt nach ISO, 2011, S. 13: *„Degree to which a system, product or component meets needs for reliability under normal operation.“*.
[262] vgl. ISO, 2011, S. 13.
[263] vgl. Broy und Kuhrmann, 2021, S. 12.

dazu kommen, dass zeitliche Vorgaben nicht eingehalten werden können und so auch der Kostenrahmen überschritten wird. In diesem Zusammenhang wurde in Abschnitt 4.2 die Rolle der genauen Aufwandsschätzung herausgestellt, durch welche Verzögerungen vermieden werden sollen. Mit Blick auf die Fallstudie soll dementsprechend untersucht werden, ob es bei bestimmten Code Ownership Werten einer Teilkomponente vermehrt zu Verzögerungen oder Fehleinschätzungen von Aufwänden kommt.

Das letztes Kriterium bezieht sich auf die Auswirkungen von Collective Code Ownership auf die Expertise der Entwickler:innen. Wie auch das zweite Kriterium kann dieses zusätzlich als Teil der Risiken während der Entwicklung angesehen werden. Falls die Expertise innerhalb eines Teams ungleich verteilt ist und einzelne Entwickler:innen das Team verlassen, kann dies einen Risikofaktor für das Projekt darstellen, da das notwendige Wissen nicht mehr vorhanden ist und neu aufgebaut werden muss. Dadurch ist es notwendig, diesen Risiken mit einem Austausch von Wissen zu begegnen. Das kann sowohl die Weitergabe von implizitem Wissen, als auch die Umwandlung von implizitem zu explizitem Wissen einbeziehen. Hier stellt sich die Frage, ob das Prinzip der kollektiven Verantwortung dazu beitragen kann, den Austausch untereinander und damit die Weitergabe von Wissen zu fördern. Eine weitere Untersuchung kann sich darauf beziehen, inwiefern dieses Prinzip zur Selbstorganisation fähig ist oder die Beschäftigung mit neuartigen Themen explizit gefördert werden muss.

Um diese Kriterien anhand von Daten eines realen Entwicklungsprojektes zu untersuchen, erfolgt im Anschluss die Konzeption der Datenanalyse im Rahmen der Fallstudie.

5 Konzeption der Fallstudie

Die aus der bisherigen Darstellung gewonnenen Erkenntnisse werden in den nachfolgenden Kapiteln anhand eines realen Fallbeispiels und einer daran durchgeführten Fallstudie weiterführend untersucht. Diese ist an einem Softwareentwicklungsprojekt der Bayer AG durchgeführt worden und fundiert auf verschiedenen Daten, welche während den Entwicklungstätigkeiten entstanden sind. Die Fallstudie selbst umfasst dementsprechend zum einen die Extraktion und Transformation der Daten, sodass diese für die Auswertung nutzbar sind, sowie darauffolgenden Analysen in Bezug auf die Hypothesen. Dazu werden zunächst das Projektumfeld und die verschiedenen Datenquellen und -arten erläutert, um anschließend den implementierten Extraktionsprozess darzustellen und die Konzeption mit der Aufstellung der Hypothesen abzuschließen.

5.1 Projektumfeld

Bei dem genannten Projekt handelt es sich um ein reines IT-Projekt der Bayer AG. Eine genaue Vorstellung der Themen und Zielsetzungen des Projektes selbst wird hier nicht vorgenommen, da dies für die Untersuchungen nicht von Relevanz ist. Trotzdem werden im Folgenden einige Randbedingungen zur Organisation, der Zusammenarbeit im Projekt und dort genutzten Technologien dargestellt, welche zum Verständnis und der Konzeption der Fallstudie beitragen. Zum Zeitpunkt letzterer ist das Projekt nicht abgeschlossen, sondern befindet sich weiterhin in der Umsetzungsphase. Daher bildet die Datenanalyse den Stand eines laufenden Entwicklungsprozesses ab und bezieht sich auf Daten, welche während der Implementierung der Software vom Startzeitpunkt bis dorthin entstanden sind.

Das Projektumfeld ist für die Konzeption der Datenanalyse von Bedeutung, da sich diese sehr nah am eigentlichen Entwicklungsprozess orientiert und die dort verwendeten Werkzeuge zur Gewinnung der Daten nutzt. Eine zusätzliche Einordnung der in der entwickelten Software und ihren Teilkomponenten verwendeten Technologien in Abschnitt 5.1.2 bildet weiterführend die Grundlage für die Untersuchung technologiebezogener Abhängigkeiten.

5.1.1 Entwicklungsprozess

Die Entwicklung der Software orientiert sich aus Sicht des Prozesses an der Methode Scrum. Es werden zwar die Grundbestandteile des Modells entliehen, einige weitere jedoch nicht angewandt. Die Entwicklung ist in Sprints von typischerweise jeweils zwei Wochen organisiert, mit einer vorausgehenden Planungs- und einer anschließenden Vorstellungs- und Bewertungsphase. Die darauf bezogene operative Projektverwaltung wird innerhalb der Planungssoftware Jira vorgenommen.

Die Zuweisung von Aufgaben erfolgt in diesem Zusammenhang stets an einzelne Personen. Der Umfang einer Aufgabe wird durch das Team geschätzt und die Planung erfolgt pro Sprint, wobei der Aufwand einer Aufgabe die Dauer eines Sprints nicht überschreiten sollte. Ist dies der Fall, wird die Aufgabe in kleinere Bestandteile aufgeteilt. Diese Vorgehensweise schließt nicht aus, dass mehrere Entwickler:innen gemeinsam an einer Aufgabe arbeiten können, die Verantwortung für die Aufgabe selbst verbleibt jedoch bei einer einzelnen Person.

Aus Sicht der Entwicklung werden viele Werte der XP-Methode und weitere Prinzipien befolgt. Für die Datenanalyse von Relevanz sind dabei insbesondere die Prinzipien des Collective Code Ownerships, der Code Reviews und die Fehler-Ursachen-Analyse. Im Falle des Collective Ownerships ist jedoch entgegen den Vorgaben von XP keine festgelegte Aufgabenrotation vorgesehen, diese unterliegt der freien Auswahl. Die Orientierung an Werten von XP erfolgte dabei jedoch weitgehend unabhängig von der Methode und ist darauf zurückzuführen, dass viele dieser Prinzipien sich seit der Entstehung der Methode als de facto Standards in der Softwareentwicklung etabliert haben.[264] Dies gilt ebenfalls für Werte des Lean Software Developments, welche sich auch sich im Projekt wiedererkennen lassen. Beispiele dafür sind „Build quality in“ oder „Optimize the whole“. Weitere Prinzipien innerhalb des Projekts umfassen die kontinuierliche Integration, eine gemeinsame Codebasis, das Prinzip des Codierens und Testens sowie die inkrementelle Bereitstellung und die vollständige Testabdeckung.

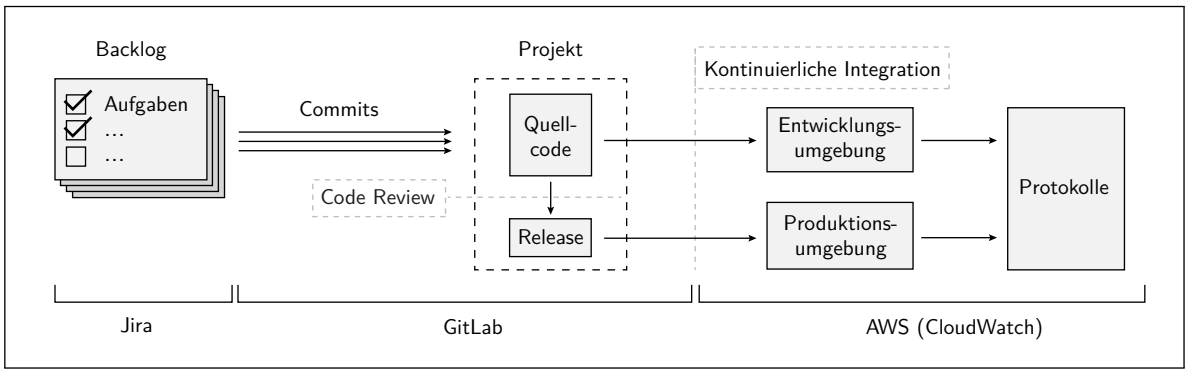
Zur Zusammenarbeit innerhalb des Entwicklungsteam wird die Versionsverwaltungssoftware Git eingesetzt. Sie ist die in diesem Bereich am weitesten verbreitete Lösung.[265] Der Begriff der Versionskontrolle beschreibt ein System, welches jegliche Änderungen an Dateien erfasst. Zusätzlich zu den Versionsdaten werden weitere Metadaten wie die bearbeitende

[264] vgl. IEEE Computer Society, 2014, S. 68–73, online im Internet.
[265] vgl. Stackoverflow.com, 2021, online im Internet.

Person oder der Zeitpunkt erfasst. Dadurch lässt sich die genaue Historie der Entwicklung nachvollziehen.[266] Git selbst bezeichnet sich als ein verteiltes System, da stets die gesamte Historie der Daten zur Verfügung steht und diese im Falle eines Ausfalls aus jeder beteiligten Instanz wiederhergestellt werden kann. Zusätzlich kann es dazu genutzt werden, die Zusammenarbeit in verteilten Teams zu organisieren. Git bildet dabei die Grundlage für verschiedene Softwareangebote, welche die Funktionalitäten von Git ergänzen.[267] Diese Lösungen umfassen typischerweise zentrale Ablagemöglichkeiten des Codes auf einem Server sowie Issue-Tracking-Systeme und Funktionen für den sozialen Austausch. Innerhalb des in der Fallstudie behandelten Projektes wird für diese Aufgaben die Software GitLab eingesetzt. Weitere Funktionen umfassen die kontinuierliche Integration neuer Entwicklungen und die Dokumentation von Code Reviews.[268]

Die dritte Instanz innerhalb des Entwicklungsprozesses stellt neben der Planung und der Entwicklung die Bereitstellung der Software dar. Im behandelten Projekt erfolgt diese in der Cloud, also nicht in einer unternehmenseigenen Umgebung, sondern derer eines Drittanbieters, welcher verschiedene Dienste anbietet und nutzungsabhängig abrechnet.[269] Innerhalb der Cloud-Umgebung werden zusätzlich Überwachungs- und Protokollierfunktionen bereitgestellt, welche neben Jira und GitLab die dritte Datenquelle für die Analyse darstellen. In dieser Fallstudie handelt es sich dabei um den CloudWatch-Service des Anbieters AWS. Dort wurden seit Beginn des Projektes dauerhaft alle Fehlermeldungen in Bezug auf die Dienste, welche im Hintergrund laufen, abgespeichert. Der gesamte Prozess ist in Abbildung 6 dargestellt.

Abbildung 6: Entwicklungsprozess des Projekts



Quelle: Eigene Darstellung.

[266] vgl. Chacon und Straub, 2014, S. 8 ff.
[267] vgl. Chacon und Straub, 2014, S. 10 f.
[268] vgl. GitLab, o. D., online im Internet.
[269] vgl. Sadiku, Musa und Momoh, 2014, S. 34 f.

Die Entwicklung lässt sich entsprechend in drei Teile unterteilen, deren Prozesse jeweils von einer Software gestützt werden. Neben der prozessbegleitenden Software werden innerhalb der Fallstudie jedoch noch weitere Technologien verwendet. Diese beziehen sich auf die eigentliche Entwicklungstätigkeit, sprich die technische Umsetzung der Anforderungen. Diese werden nicht von Grund auf neu entwickelt, sondern setzen auf bereits existierenden Technologien auf. Dazu gehören sowohl Programmiersprachen wie auch Lösungen für die Visualisierung der Benutzeroberfläche und der Speicherung von Daten. Die Architektur, welche die einzelnen Komponenten beschreibt, wird im Folgenden erläutert.

5.1.2 Softwarearchitektur

Die Architektur einer Software bezeichnet eine „Sammlung von Hardware- und Softwarekomponenten und ihrer Schnittstellen, um den Rahmen für die Entwicklung eines Computersystems zu schaffen“.[270] Die Softwarearchitektur beschreibt dabei sowohl den Prozess, als auch das Ergebnis der Spezifizierung dieser Komponenten. Dies umfasst die Struktur, die einzelnen genutzten Technologien und ihre Verbindung miteinander.[271]

Tabelle 1: Zentrale Komponenten der Softwarearchitektur

Bezeichnung	Typ	Anwendungsbereich im Projekt
ARM	Konfigurationssprache	Verwaltung der Infrastruktur
CloudFormation	Konfigurationssprache	Verwaltung der Infrastruktur
Terraform	Konfigurationssprache	Verwaltung der Infrastruktur
Python	Programmiersprache	Hintergrundprozesse
TypeScript	Programmiersprache	Authentifizierung und Benutzeroberfläche
Vue.js	Webframework	Benutzeroberfläche
Docker	Virtualisierung	Kontinuierliche Integration
DynamoDB	Datenbank	Speicherung der Daten

Quelle: Eigene Tabelle.

Auch die in der Fallstudie entwickelte Software besteht aus der Kombination verschiedener bereits etablierter Technologien, welche die einzelnen Teilbereiche der Softwarearchitektur abdecken. Im Rahmen der Analyse wurde eine Beschränkung auf acht Kerntechnologien vorgenommen, welche als Überblick in Tabelle 1 dargestellt sind. In der Realität lassen sich noch weitere Bestandteile feststellen, da die hier dargestellten Technologien teilweise

[270] übersetzt nach IEEE, 1990, S. 10: *„The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.“*.
[271] vgl. Maier, Emery und Hilliard, 2001, S. 107.

wiederum auf Bestehenden aufbauen. Eine ausführlichere Beschreibung zu den einzelnen Technologien ist im Glossar zu finden.

Zur Anwendung der einzelnen Technologien ist Fachwissen vonnöten. Die Nutzung neuer Technologien kann einerseits einen Erfolgsfaktor für ein Projekt darstellen, dieses gleichzeitig jedoch durch die noch nicht vorhandene Expertise einem gewissen Risiko aussetzen.[272] Für ein Projekt bedeutet dies, dass für die Auswahl der Mitarbeitenden neben der Funktion beziehungsweise Rolle auch das Know-How als Kriterium beachtet werden muss. Hierbei lässt sich zwischen einer horizontalen und einer vertikalen Organisation unterscheiden. Die horizontale Organisation betitelt die Aufteilung nach Technologien anhand der Zuweisung von Experten. Die Alternative dazu ist eine technologieunabhängige Zuteilung anhand von von Zielen des Projekts.[273] Die Koordination der Expertise steht in direkter Assoziation mit der Effektivität des Teams.[274]

Übertragen auf die Fallstudie bedeutet dies, dass das Prinzip des Collective Code Ownerships auch mit Blick auf die Zuteilung der Aufgaben nach Technologien hin untersucht werden soll. In diesem Bereich stellt sich insbesondere die Frage, inwiefern sich Expert:innen für einzelne Themen herausstellen und ob das Prinzip dabei hilft, das Fachwissen innerhalb des Projektteams zu verbreiten und die Zuweisung von technologiebezogenen Aufgaben über den Projektzeitraum hinweg heterogener zu gestalten.

5.2 Erfassung der Daten

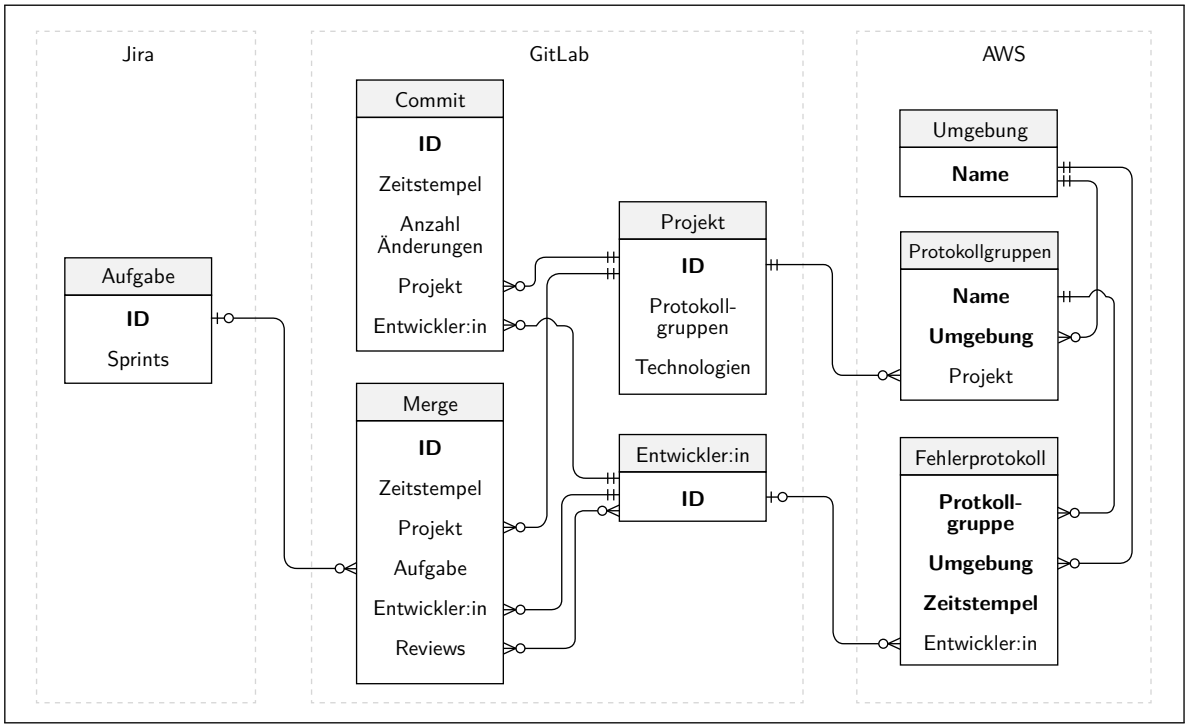
Die Basis der Fallstudie bildet die Erfassung von Daten zur Softwareentwicklung im behandelten Projekt. Bei den Daten handelt es sich um Artefakte, welche während des Zeitraumes vom 01. März 2020 bis zum 31. Dezember 2021 in verschiedenen Systemen generiert wurden. Der Startpunkt des Zeitraums spiegelt dabei auch den Zeitpunkt des Projektbeginns beziehungsweise des Beginns der Implementierung wieder. Der Endzeitpunkt ist durch den Zeitpunkt der Auswertung limitiert, das Projekt lief unabhängig davon weiter.

[272] vgl. Rus und Lindvall, 2002, S. 27.
[273] vgl. Meyer, 2018, S. 13 f.
[274] vgl. Faraj und Sproull, 2000, S. 1563.

5.2.1 Datenquellen und -arten

Die Fallstudie ist anhand von Daten konzipiert, die aus den Programmen extrahiert werden können, welche den Entwicklungsprozess über den gesamten Zeitraum begleitet haben und in Abschnitt 5.1.1 vorgestellt wurden. In der Entwurfsphase wurden aus den drei Quellen verschiedene Datentypen identifiziert, welche zur Untersuchung der Forschungsfrage relevant sind. Diese Datentypen und ihre Beziehung zueinander sind in vereinfachter Form in Abbildung 7 dargestellt.

Abbildung 7: Entity-Relationship-Modell der Daten



Quelle: Eigene Darstellung.

Die Projektplanung anhand der Scrum-Methode wird über die Daten zu Aufgaben aus Jira abgebildet. Die Verbindung zur tatsächlichen Entwicklung wird über den sogenannten Merge hergestellt, durch welchen Implementierungen in die produktive Umgebung überführt werden. Innerhalb von GitLab bestehen zusätzliche Daten zu den Beteiligten an der Entwicklung und ihren Beiträgen zum Quellcode, den Commits. Letztere beziehen sich auf die einzelnen Projekte in GitLab. Ein Projekt repräsentiert dabei eine Teilkomponente der Software. Über das Projekt wird die Verbindung zur Ausführungsumgebung hergestellt. Dort sind sowohl Daten aus der Entwicklungs-, als auch Produktionsumgebung vorgesehen. Die Fehlerprotokolle sind in Gruppen zusammengefasst, welche sich auf die entsprechenden Projekte in GitLab

zuordnen lassen. Zusätzlich wird, falls möglich, bei den einzelnen Fehlerprotokollen eine Assoziation mit den verantwortlichen Entwickler:innen hergestellt.

In diesem Zusammenhang ist anzumerken, dass alle Merkmale, welche Rückschlüsse auf unternehmensinterne Daten oder beteiligte Personen zulassen würden, für die Analyse in pseudonymisierter Form vorgesehen sind. Dadurch bleibt die Beziehung der Daten zueinander erhalten, eine Rekonstruktion der ursprünglichen Informationen ist jedoch nicht möglich. Im Folgenden wird dargestellt, wie die Extraktion der Daten implementiert wurde.

5.2.2 Extraktion und Aufbereitung der Daten

Das Auslesen der Daten wurde in einer Konsolenanwendung gekapselt. Diese ist in Python implementiert und nutzt Schnittstellen zu AWS und GitLab. Aufgrund von Barrieren in der Schnittstellennutzung zu der unternehmensinternen Jira-Instanz wurden die Daten zur Projektplanung anhand eines manuell durchgeführten Exports importiert. Die Assoziation der Daten wurde durch die Anwendung vollkommen automatisiert durchgeführt. Der daraus resultierende Gesamtaufbau der Anwendung ist in Anhang 4 enthalten.

Für die nachgelagerte Analyse wurden die Daten in einer Datenbank zusammengeführt. Zusätzlich wurden auf Basis der extrahierten Daten teilweise weitere Schritte durchgeführt, um zusätzliche Informationen zu gewinnen. Als Beispiel dafür sind die Fehlerprotokolle zu nennen, welche auf Basis der Entwicklungsdaten falls möglich mit der zu diesem Zeitpunkt für den entsprechenden Abschnitt zuständigen Person ergänzt wurden. Gleichermaßen wurde bei den erfolgten Code-Reviews verfahren, welche anhand ihrer Aufgabennummer mit der zugehörigen Aufgabe in der Sprintplanung in Verbindung gebracht werden konnten, falls diese im entsprechenden Merge dokumentiert worden ist. Diese Extraktionsprozesse wurden durch einige manuell gepflegte Metadaten ergänzt. Dazu gehören die Assoziation von Entwicklungsdaten zu den verwendeten Technologien und die Beziehung zwischen Protokollgruppen und Entwicklungsprojekten. Zusätzlich sind in der dafür genutzten Konfigurationsdatei die Rahmenbedingungen für die automatisierten Prozesse festgelegt, darunter der Zeitraum der Analyse und die Einstiegspunkte für die einzelnen Datenquellen. Ein Überblick über die Gesamtheit der dadurch gewonnenen Daten ist in Tabelle 2 dargestellt.

Anhand der Fehlerprotokolle, Aufgaben und Reviews lässt sich erkennen, dass die Assoziation einzelner Datenobjekte nicht immer erfolgen konnte. Dies ist beispielsweise bei den Code-Review auf eine fehlende Zuweisung der Aufgabennummer oder im Falle der Fehlermeldungen

Tabelle 2: Übersicht der extrahierten Daten

Plattform	Metrik	Wert
GitLab	Entwickler:innen	16
	Projekte	83
	Commits	4 129
	Merges	800
	Merges mit Review	332
	Merges mit zugewiesener Aufgabe	589
Jira	Aufgaben	583
AWS CloudWatch	Umgebungen	2
	Protokollgruppen pro Umgebung	56
	Entwicklung - alle Fehlerprotokolle	43 819
	Entwicklung - zugewiesene Fehlerprotokolle	33 256
	Produktion - alle Fehlerprotokolle	2 155
	Produktion - zugewiesene Fehlerprotokolle	1 575

Quelle: Eigene Tabelle.

auf Quellcode-Experimente ohne Dokumentation der Entwicklungstätigkeit zurückzuführen, wodurch die verantwortliche Person nicht identifiziert werden konnte.

5.2.3 Berechnung des Code Ownerships

Auf Basis einiger der im vorherigen Abschnitt vorgestellten Daten werden in der Fallstudie zur Auswertung weitere Berechnungen zugrundegelegt, um diese auf das Code Ownership hin zu untersuchen. Diese Berechnungen finden zwar entkoppelt vom Extraktionsprozess und streng genommen erst während der Auswertung der Daten statt, die Festlegung auf stringente Berechnungsformeln ist jedoch als Teil der Konzeption anzusehen. Die Berechnung des Code Ownerships bezieht sich auf einzelne Personen oder ein gesamtes Projekt im Sinne des Datenmodells. Als Basis der Auswertung des Code Ownerships können, wie bereits in Abschnitt 3.3.3 beschrieben, Werte zur Anzahl von Commits, geänderten Codezeilen oder Code Reviews zur Berechnung hinzugezogen werden. Da letztere während der Auswertung gesondert betrachtet werden, werden sie hier nicht berücksichtigt. Die Berechnung des Code Ownerships bezieht sich entsprechend nur auf die ersten beiden Metriken.

Die Kombination beider Werte begründet sich dadurch, dass diese sich, beeinflusst durch die betreffende Person, den Prozess und die Rahmenbedingungen der Entwicklung, sehr unterschiedlich verhalten können. Im Falle der Commits sind die Unterschiede primär durch

den Entwicklungsstil der jeweiligen Person charakterisiert. So kann der gleiche Umfang an Änderungen in Form mehrerer Commits zur Codebasis hinzugefügt werden, gleichermaßen können diese innerhalb eines einzigen Commits zusammengefasst sein. Anhand der Anzahl von Commits lassen sich dementsprechend keine ebenmäßige Rückschlüsse auf die tatsächlichen Änderungen innerhalb des Quellcodes ziehen. Ein ähnliches Verhalten findet sich in der Metrik der geänderten Codezeilen wieder, die dortigen Ungenauigkeiten sind jedoch nur geringfügig auf den Entwicklungsstil zurückzuführen, sondern auf die genutzte Technologie und Rahmenbedingungen. Die genutzte Technologie kann durch die Notwendigkeit verschiedener, automatisch angelegter Fragmente wie Konfigurationsdateien die Angabe der Anzahl geänderter Codezeilen verfälschen. Im Git-Prozess erscheinen diese unter dem Namen der Person, welche den Erstellungsprozess durchlaufen hat. Bei den in der Auswertung berücksichtigten Technologien ist dies insbesondere bei Projekten, welche Vue.js oder TypeScript verwenden, relevant. Dort werden Dateien angelegt, welche beispielsweise alle Abhängigkeiten zu externen Paketen aufzeichnen. Diese können mitunter mehrere Tausend Zeilen umfassen, ohne dass eine Person dort selbst Änderungen durchgeführt hätte. Ein Beispiel für Einflüsse des Prozesses auf der anderen Seite ist das Verfassen von Dokumentation zum entsprechenden Projekt beziehungsweise der Komponente. Dies ist im Kontrast zu den technologischen Besonderheiten zwar als Tätigkeit anzusehen, welche einen höheren Anteil der Person am Code Ownership rechtfertigt, diese ist im Vergleich zu Änderungen am Quellcode jedoch generell im Umfang deutlich größer und verfälscht somit ebenfalls die Angabe der geänderten Codezeilen. Das Ziel der Kombination beider Metriken der Commits und der geänderten Codezeilen ist es, die Genauigkeit der Berechnung des Code Ownerships zu verbessern.

Nach Bird et al. wird der „Anteil des Code Ownerships [...] eines Mitwirkenden an einer bestimmten Komponente [durch] das Verhältnis der Anzahl der Commits, die der Mitwirkende gemacht hat, im Verhältnis zur Gesamtzahl der Commits für diese Komponente“[275] definiert. Durch die Ergänzung dieser Berechnung mit der Metrik der geänderten Codezeilen kann die folgende Formel für die Berechnung des Code Ownerships einer Person P zum Zeitpunkt t zugrundegelegt werden:

$$\text{Code Ownership}(P)_t = \frac{\frac{\text{Commits}(P)_t}{\text{Commits}_t} + \frac{\text{Änderungen}(P)_t}{\text{Änderungen}_t}}{2}$$

[275] übersetzt nach Bird et al., 2011, S. 6: *„The proportion of ownership [...] of a contributor for a particular component is the ratio of number of commits that the contributor has made relative to the total number of commits for that component.“*.

Das Code Ownership einer Person ist somit durch den Mittelwert der Code Ownership Berechnung anhand der Metriken der Commits und der geänderten Codezeilen definiert.

Weiterführend wird das Code Ownership für eine Komponente der Software nach Bird et al. durch das Maximum der individuellen Werte des Code Ownerships ausgedrückt, also das Code Ownership der Person mit den insgesamt meisten Beiträgen zum Quellcode.[276] Die Formel für das Code Ownership einer Komponente K (im folgenden auch als Projekt bezeichnet) zum Zeitpunkt t kann entsprechend aus den mit der vorherigen Formeln berechneten individuellen Anteilen des Code Ownerships (hier abgekürzt mit CO) der Personen 1 bis n gebildet werden:

$$\text{Code Ownership}(K)_t = \max\Big(\text{CO}(P_1)_t \,, \text{CO}(P_2)_t \,, \ldots \,, \text{CO}(P_n)_t \Big)$$

Das Ziel dieser Schritte war es, eine Basis zur Untersuchung der Auswirkungen von Collective Code Ownership auf die agile Softwareentwicklung und die Verteilung von Verantwortlichkeiten zu schaffen. Für diese werden im nächsten Abschnitt Hypothesen in verschiedenen Kategorien, welche sich anhand der gewonnenen Daten überprüfen lassen, beschrieben.

5.3 Hypothesen

Anhand der in Abschnitt 1 definierten Forschungsfrage lassen sich die drei Kernbereiche Fehleranfälligkeit, Entwicklungsrisiken und Expertise des Entwicklungsteams ableiten, welche aus Sicht der Verantwortlichkeiten und des Collective Code Ownerships untersucht werden. Daher werden im Folgenden Hypothesen formuliert, welche im weiteren Verlauf bestätigt oder widerlegt werden können. Die Auswahl der Hypothesen erfolgte zum einen aus den in XP beschriebenen Merkmalen zur verteilten Verantwortung und zum anderen aus den vorhandenen Daten, durch welche die Möglichkeiten der Analyse beschränkt werden. Falls möglich, wurde die Orientierung der Thesen entsprechend bereits in der Literatur durchgeführten Studien und deren Ergebnissen oder anhand der in den Modellen der agilen Softwareentwicklung beschrieben Folgen solcher Maßnahmen festgelegt.

[276] vgl. Bird et al., 2011, S. 7.

H1 Fehleranfälligkeit der Software

Die ersten beiden Hypothesen zielen auf die Überprüfung der Auswirkungen von Collective Code Ownership auf die Fehleranfälligkeit der Software ab. In einer Untersuchung von Bird et al. am Beispiel der Entwicklung von Windows Vista wurde festgestellt, dass ein höheres Code Ownership eine marginale Reduzierung der Fehlerzahl zur Folge hatte.[277] Zur Überprüfung des Effekts innerhalb der hier vorliegenden Fallstudie wird die erste Hypothese im Sinne dieses Ergebnisses formuliert.

> **H1a** Niedriges Code Ownership führt dazu, dass in einer Softwarekomponente häufiger Fehler auftreten.

Durch die vorliegenden Daten sind jedoch im Unterschied zu der vorausgegangenen Untersuchung bei einem Großteil der Fehlerprotokolle (siehe Tabelle 2) Rückschlüsse auf das Code Ownership der jeweiligen Autor:innen möglich. Dadurch kann eine Analyse der tatsächlich bei einem Fehler zugrundeliegenden Verantwortung der Person im Gesamtkontext durchgeführt werden. Entsprechend der Orientierung von H1a wird in der zweiten Hypothese ebenfalls von einem negativen Effekt eines niedrigen Code Ownerships ausgegangen.

> **H1b** Fehler in einer Softwarekomponente werden eher durch Entwickler:innen verursacht, welche nicht den größten Teil des Code Ownerships für sich beanspruchen.

Dies bedeutet, dass bei einer Bestätigung der Hypothese im Fehlerfall das Code Ownership des Projektes von dem der für den Fehler verantwortlichen Person abweicht. Die beiden hier formulierten Hypothesen beschreiben die Rahmenbedingungen für die Untersuchungen zur Fehleranfälligkeit. Die Modelle Scrum und XP beschreiben beide das Team als zentralen Erfolgsfaktor der Softwareentwicklung.[278] Das Prinzip der verteilten Verantwortung bezieht auch die Verantwortung für Fehler mit ein, welche in diesem Fall durch das Team getragen wird. Eine Rückverfolgung der Fehlerursachen ist, wie in Abschnitt 3.3.2 beschrieben, jedoch für den Lerneffekt der Organisation und der einzelnen Personen von Bedeutung. Zur Untersuchung weiterer Auswirkungen von Collective Code Ownership werden als nächstes die Auswirkungen des Prinzips auf Risiken in der Entwicklung betrachtet.

[277] vgl. Bird et al., 2011, S. 9 f.
[278] siehe Abschnitte 2.2.1 und 2.2.2.

H2 Risiken in der Entwicklung

Anhand der in Abschnitt 4 beschriebenen Kriterien zur Verteilung von Verantwortlichkeiten und der gewonnenen Daten lassen sich in Bezug auf Entwicklungsrisiken zwei Ausprägungen von Risiken untersuchen. Zunächst wird vor dem Hintergrund des Entwicklungsrisikos durch Ausfälle einzelner Entwickler:innen und der dadurch vorliegenden Notwendigkeit von Wissensaustausch zwischen diesen[279] untersucht, ob Collective Code Ownership ein Treiber dafür sein kann. Die positive Auslegung der Hypothese ist dadurch begründet, dass XP beispielsweise auch das Beheben von Fehlern als eine kollektive Aufgabe sieht, die nach Möglichkeit von jedem Mitglied zu erledigen sein sollte.[280] Dies würde implizit zu einer Reduktion von Abhängigkeiten führen. Auch die Bedeutung der Lokalisation von Expert:innen,[281] welche mutmaßlich durch die Einblicke in alle Komponenten eines Softwareprojekts gefördert wird, kann hier einen positiven Effekt haben.

H2a Das Prinzip Collective Code Ownership reduziert mittelfristig technologiebezogene Abhängigkeiten zur Expertise einzelner Personen.

Ein weiteres Entwicklungsrisiko stellen Verzögerungen in der Entwicklung dar. Im Falle von Collective Code Ownership ist, abhängig von der Expertise der Entwickler:innen, eine Zuweisung von Aufgaben nicht nur an eine vordefinierte Person, sondern an jedes verfügbare Teammitglied möglich. Dadurch kann potenziell eine schnelle Antwort auf inkrementelle oder sich verändernde Anforderungen gegeben werden.[282] Auf diese Auswirkungen zielt die Formulierung der vierten Hypothese ab.

H2b Niedriges Code Ownership führt zu weniger Verzögerungen in der Entwicklung.

Die zu analysierenden Daten lassen vor dem Hintergrund von Hypothese H2a noch tiefere Einblicke in das Wissensmanagement von Projekten zu, welche gängige Entwicklungspraktiken wie das Collective Code Ownership oder das Code Review anwenden. Daher werden die vier bereits formulierten Hypothesen im Folgenden durch zwei weitere komplettiert.

[279] siehe Abschnitt 4.3.
[280] vgl. Beck, 2000, S. 66.
[281] siehe Abschnitt 4.3.
[282] vgl. Nordberg III, 2003, S. 30.

H3 Expertise der Entwickler:innen

Als Unterschied der Anwendung von Collective Code Ownership im Projekt der Fallstudie im Vergleich zur Referenz in XP wurde bereits in Abschnitt 5.1.1 dargestellt, dass dort keine feste Aufgabenrotation definiert ist. In XP soll durch diese festgelegte Rotation eine stets gleichartige Beschäftigung vermieden und Lernprozesse gefördert werden. Dies ist explizit als eine der Anforderungen des Pair Programmings formuliert, bei welchem in regelmäßigen Abständen die Entwicklungspartner:innen getauscht werden sollen.[283] Die Entwickler:innen dieses Projektes haben dementsprechend, begrenzt durch die Priorisierung der Aufgaben, einen gewissen Entscheidungsspielraum bei der Auswahl ihrer Tätigkeit. Die explizite Anforderung dieser Rotation in XP lässt vermuten, dass das Fehlen eines solchen Mechanismus zu gleichartiger Beschäftigung führen kann.

H3a Die Einführung von Collective Code Ownership ohne definierte Aufgabenrotation führt dazu, dass die Entwickler:innen sich weiterhin bevorzugt mit ihnen bereits bekannten Technologien beschäftigen.

Im Gegensatz zu den Praktiken von XP, wo das Durchführen von Code Reviews aufgrund der Anwendung von Pair Programming als nicht notwendig angesehen wird,[284] wird dies innerhalb des hier analysierten Projektes regelmäßig durchgeführt. Hierbei gilt der Grundsatz der verteilten Verantwortung, dass grundsätzlich jede beteiligte Person auch Rückmeldungen zu neuen Entwicklungen geben kann. Dementsprechend soll mithilfe der folgenden Hypothese untersucht werden, ob die Durchführung von Code Reviews von in diesem Bereich unerfahrenen Entwickler:innen zu einer anschließenden Auseinandersetzung mit der Technologie führt. Insbesondere soll somit geprüft werden, ob es eine Möglichkeit darstellt, eventuellen negativen Folgen einer fehlenden Aufgabenrotation entgegenzuwirken.

H3b Beteiligungen in Code Reviews tragen dazu bei, dass Entwickler:innen sich in Zukunft, falls noch nicht geschehen, eher mit den dort verwendeten Technologien beschäftigen.

Auch Hypothese H2a kann zusätzlich zur Betrachtung der Auswirkungen auf die Expertise der Entwickler:innen hinzugezogen werden.

[283] vgl. Beck, 2000, S. 42 f.
[284] siehe Abschnitt 3.2.

Damit ist die Aufstellung der sechs Hypothesen in Bezug auf die Fehleranfälligkeit, Entwicklungsrisiken und Expertise des Entwicklungsteams abgeschlossen. Das nächste Kapitel beschäftigt sich mit der Fragestellung, inwiefern sich die hier formulierten Hypothesen anhand der Analyse der gewonnenen Daten bestätigen oder widerlegen lassen. Anschließend wird anhand der Erkenntnisse dieser Auswertungen Bezug auf die Kriterien für die Verteilung von Verantwortlichkeiten genommen.

6 Auswertung der Daten

Anhand der Grundlagen, welche durch die im vorherigen Kapitel beschriebene Vorgehensweise aus den entsprechenden Quellen extrahiert wurden, kann dieser Prozess im Folgenden mit der Kombination und Auswertung der verschiedenen Datenarten fortgeführt werden. Entsprechend der Daten sind die Analysen ebenfalls auf den Zeitraum von März 2020 bis Ende des Jahres 2021 angesetzt. Teilweise variieren diese Grenzen, falls für den Gegenstand der Analyse keine Daten für bestimmte Abschnitte vorliegen.

Die Auswertung der Daten ist, wie auch der Extraktionsprozess, dessen Daten hier genutzt werden, Teil des Quellcodes der Masterthesis.[285] Als Ergebnis dieser Analysen sind jeweils Abbildungen entstanden, welche die Daten als Grundlage für die Analysen visualisieren. Zur Nachvollziehbarkeit der Werte sind zusätzlich Wertetabellen für die Abbildungen der folgenden Abschnitte beigefügt.[286] Um eine fortlaufende Analyse zu ermöglichen, sind die Entwickler:innen und Projekte jeweils mit einer gleichbleibenden Identifikationsnummer und Farbgebung versehen. Teilweise können durch die Limitation in der Größe der Abbildungen und die Fülle an Informationen und Farben Details verloren gehen. In solchen Fällen sind die betroffenen Abbildungen ebenfalls in einem größeren Maßstab im Anhang zu finden.[287]

6.1 Grundlegende Analysen

Die grundlegenden Analysen haben das Ziel, die Entwicklungen innerhalb der Fallstudie über den gesamten Verlauf hinweg darzustellen. Insbesondere soll dadurch ermöglicht werden, diese Ursachen während der Überprüfung der Hypothesen auszuschließen und gegebenenfalls anhand von Randbedingungen erklären zu können. Dazu werden im Folgenden die Entwicklungsaktivität im gewählten Zeitraum, das Auftreten von Fehlern in Bezug dazu sowie der Verlauf des Code Ownerships in den einzelnen Teilkomponenten dargestellt. Auf diese kann während der darauffolgenden Analysen zu den jeweiligen Hypothesen zurückgegriffen werden.

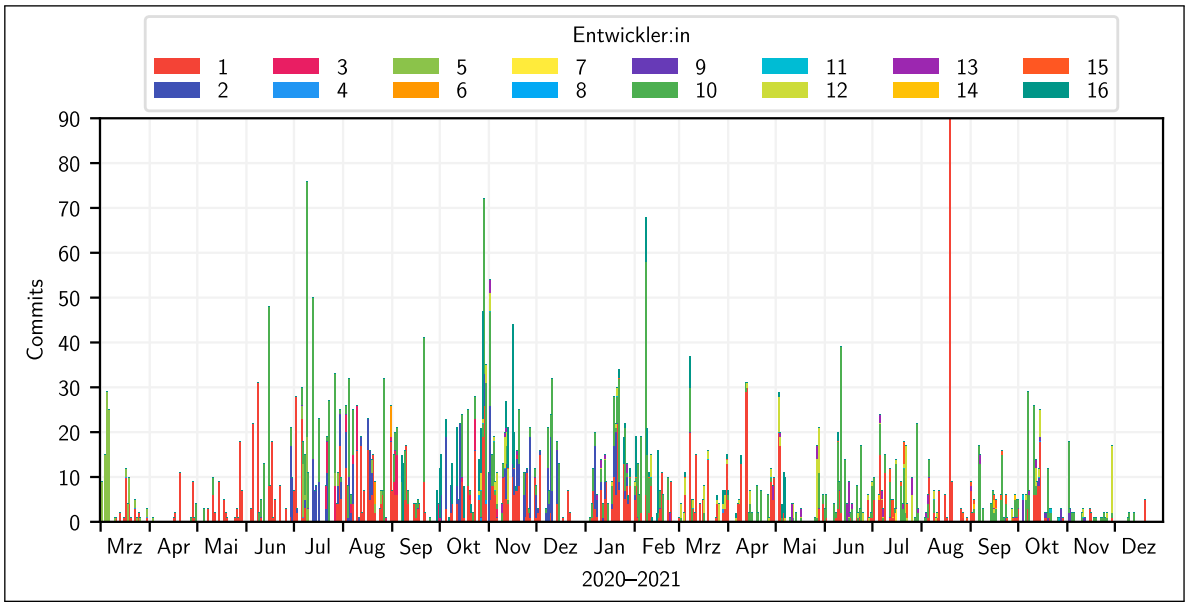
[285] siehe Anhang 4.
[286] siehe Anhang 3.
[287] siehe Anhang 2.

6.1.1 Entwicklungsaktivität

Die Komponenten der Software wurden über den gesamten Zeitraum hinweg recht konstant weiterentwickelt. Insgesamt waren 16 Entwickler:innen an der Entwicklung beteiligt, deren Anteile jedoch sehr unterschiedlich ausfallen. Etwa 70 % der Gesamtzahl an Commits sind auf zwei Personen zurückzuführen, während der Anteil von acht Personen weniger als 1 % beträgt.[288] Insgesamt wurden über den Untersuchungszeitraum 4 129 Commits mit 600 666 geänderten Codezeilen durchgeführt.[289]

In Abbildung 8 sind die Entwicklungsaktivitäten nach Personen aufgeschlüsselt und pro Tag kumuliert dargestellt.[290] Der Großteil der Commits wurde durch die Entwickler:innen 1 und 10 beigetragen. Weiterführend ist zu erkennen, dass die Aktivität sich zwar konstant verhält, in einigen Zeitabschnitten jedoch eine höherer beziehungsweise niedrigerer Aktivität vorliegt. Insbesondere zu Beginn und zum Ende des Betrachtungszeitraums fanden weniger Entwicklungen statt.

Abbildung 8: Entwicklungsaktivität nach Entwickler:in



Quelle: Eigene Darstellung.

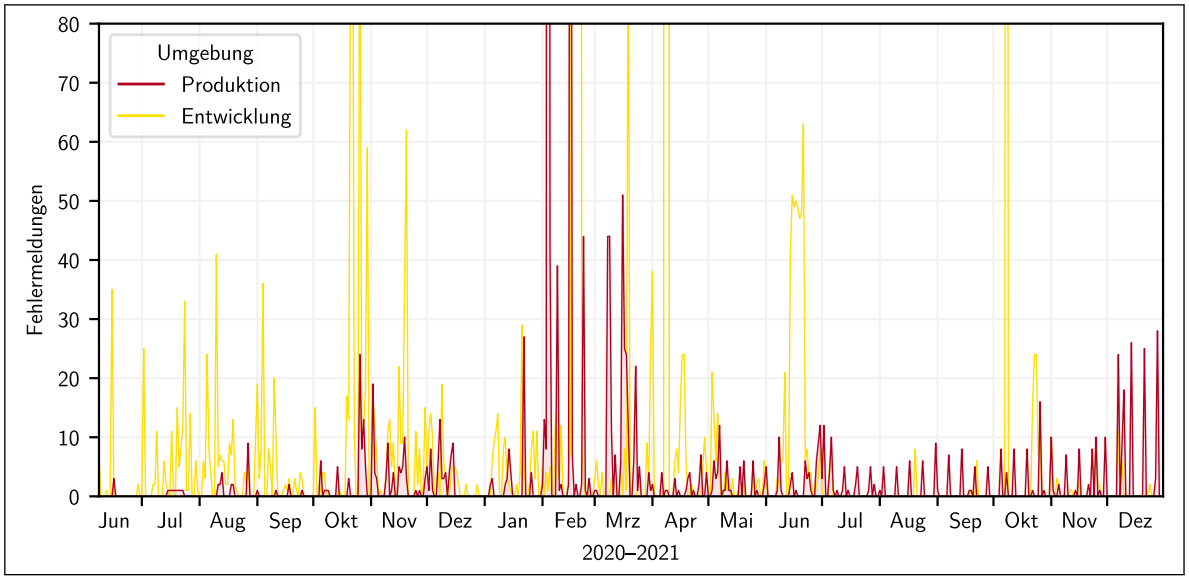
Der Zeitpunkt, zu dem der jeweilige Commit erfasst wird, entspricht dem Datum, wann die entsprechenden Änderungen im (GitLab-) Projekt hinzugefügt wurden. Bei der Über-

[288] siehe Anhang 3.2.
[289] siehe Tabelle 2.
[290] siehe auch Anhang 2.1 in größerem Maßstab.

führung von Commits aus anderen Quellen, beispielsweise experimentellen Projekten, wird dieser Zeitpunkt für alle dort enthaltenen Commits übernommen, was zu einer großen Zahl an Commits einer einzelnen Person an einem Tag, wie etwa Mitte August 2021, führen kann.

Zur Betrachtung der Entwicklungsaktivität kann auch das Auftreten von Fehlern im Projekt der Fallstudie hinzugezogen werden. Die Aufzeichnung solcher Protokolle wurde erst im Juni 2020 begonnen. Die Analyse zu den Fehlerprotokollen bezieht sich lediglich auf eine Teilmenge der Projekte, in welchen die Fehler langfristig protokolliert wurden. Dies ist allgemein bei allen Teilkomponenten erfolgt, welche Prozesse darstellen, die im Hintergrund ablaufen und in Python programmiert wurden. Das Auftreten der Fehler ist, ebenfalls pro Tag kumuliert, in Abbildung 9 aufgeführt.[291] Zur Übersichtlichkeit wurde die Höhe des Diagramms begrenzt, an einigen Tagen lag die Fehleranzahl jedoch bei deutlich über 80.[292]

Abbildung 9: Fehlerprotokolle



Quelle: Eigene Darstellung.

Bei der Betrachtung der beiden Abbildungen lassen sich an einigen Stellen Korrelationen zwischen der Häufigkeit des Auftretens von Fehlern und der Entwicklungsaktivität feststellen. Dies ist insbesondere bei Fehlern, welche innerhalb der Entwicklungsumgebung auftraten, der Fall, am Stärksten ist dies im Juli und August 2020 ausgeprägt. Auf der anderen Seite gibt es Abschnitte im Frühjahr 2021, in welchen in beiden Umgebungen überproportional

[291] siehe auch Anhang 2.2 in größerem Maßstab.
[292] siehe Anhang 3.2.

viele Fehler entstanden sind, obwohl nicht zu jedem Zeitpunkt eine hohe Entwicklungsaktivität festzustellen ist. Die Ursachen dafür lassen sich zu diesem Zeitpunkt nicht mehr genau nachvollziehen und können dementsprechend auch unterschiedlicher Natur sein. Insgesamt lassen sich etwaige Abhängigkeiten zwischen beiden Ausprägungen besonders in der kombinierten Darstellung von Entwicklungsaktivitäten und Fehlerprotokollen in Anhang 2.2 nachvollziehen.

Abschließend lässt sich festhalten, dass viele Änderungen an den Softwarekomponenten durchaus ein Grund für das Auftreten von Fehlern sein können, dies jedoch nicht der einzige Grund für ein solches ist. In der Überprüfung der Hypothesen zur Fehleranfälligkeit der Software soll weiterführend analysiert werden, ob auch das Code Ownership einen Einfluss auf diese hat.

6.1.2 Code Ownership

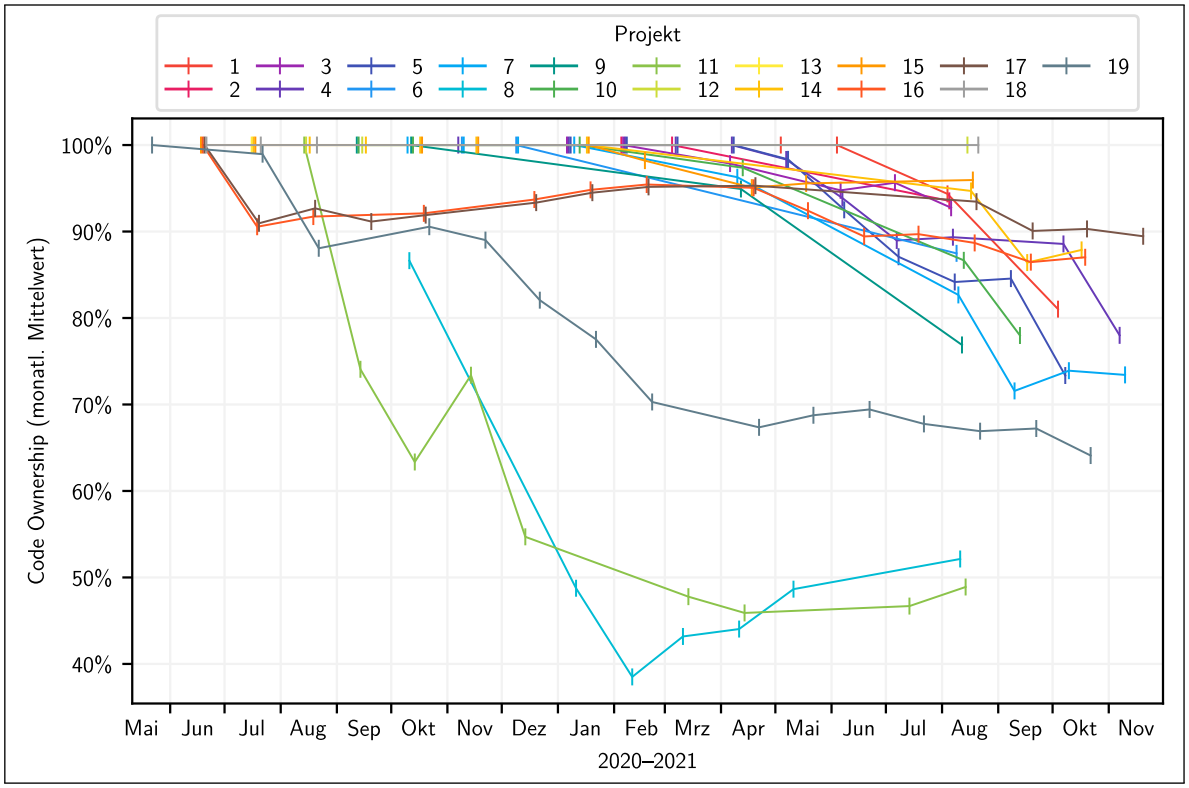
Als weitere grundlegende Analyse wird in diesem Abschnitt das Code Ownership der einzelnen Projekte dargestellt. Auch hier bezieht es sich dabei auf jene, welche als Teil der Fehleranalyse betrachtet werden können, in denen also Fehlerprotokolle abgespeichert wurden. Für die Analyse des Code Ownerships wurde für jedes dieser Projekte jeweils das monatliche arithmetische Mittel dieser Metrik berechnet. Diese erfolgte mithilfe der in Abschnitt 5.2.3 genannten Berechnungsformel anhand der Commits und geänderten Codezeilen. Der Mittelwert wurde gewählt, um ähnliche Werte des Code Ownerships mehrerer Projekte innerhalb eines Monats darstellen zu können. Die Projekte sind innerhalb eines Monats horizontal leicht versetzt angeordnet, diese Reihenfolge hat aus Sicht der Daten keine Bedeutung. Die Datenpunkte in Form der vertikalen Markierung sind nur vorhanden, falls in diesem Monat Entwicklungsaktivitäten in dem entsprechenden Projekt verzeichnet wurden. Der Verlauf des Code Ownerships ist in Abbildung 10 dargestellt.[293]

Erwartungsgemäß entwickelt sich das Code Ownership der einzelnen Projekte in vielen Fällen von der Beteiligung lediglich einem/einer hin zu mehreren Entwickler:innen, was sich in Form eines niedrigeren Wertes äußert. Während der Durchschnittswert über alle Projekte hinweg zum Ende des Jahres 2020 noch bei 94 % liegt, ist er Ende 2021 bereits auf 82 % gesunken. Weiterführend lässt sich erkennen, dass der Wert nahezu immer über 40 %, meist sogar deutlich höher liegt. Im Falle der Projekte 12, 13 und 18 bleibt der Wert über den gesamten Zeitraum bei 100 %. Gleichzeitig ist bei diesen Projekten auch erkennbar, dass nur in ein bis

[293] siehe auch Anhang 2.3 in größerem Maßstab.

vier Monaten neue Entwicklungen stattgefunden haben, was im Vergleich zu den anderen Projekten einen sehr niedrigen Wert darstellt.[294] Der Großteil der Projekte wurde zum Ende des Untersuchungszeitraums hin von mindestens zwei Entwickler:innen verwaltet, wobei im Dezember 2021 innerhalb der dargestellten Projekte keine Entwicklungsaktivität verzeichnet wurde. Durch die Wahl des Mittelwertes lässt sich ebenfalls erklären, dass Projekt 8 direkt zu Beginn ein Code Ownership von nur 87 % besitzt.

Abbildung 10: Code Ownership der Teilprojekte



Quelle: Eigene Darstellung.

Die Darstellung dieser drei grundlegenden Analysen lassen gewisse Interpretationen zu, welche sich nicht anhand der Daten eingrenzen lassen. Dies ist insbesondere bei der Betrachtung der Fehlerprotokolle in Zusammenhang mit der Entwicklungsaktivität der Fall. Um die Auswirkungen von Collective Code Ownership zu überprüfen und gleichzeitig den Spielraum für Interpretationen zu verkleinern, werden im Folgenden tiefgreifende Analysen zu den konkreten Hypothesen durchgeführt.

[294] siehe Anhang 3.3.

6.2 Betrachtung der Hypothesen

Auf Basis der vorangegangenen Basisanalysen werden die Daten im Folgenden dafür genutzt, um die Hypothesen zu überprüfen. Dafür wurde für jede der formulierten Hypothesen eine Auswertung der Daten vorgenommen. Das Ziel ist es, diese Annahmen aus Sicht der Fallstudie bestätigen oder widerlegen zu können.

Bei der Betrachtung eines solchen komplexen Entwicklungsprozesses können nicht alle Randbedingungen in der Analyse berücksichtigt werden. Daher werden einige allgemeine Aspekte zur Betrachtung der Hypothesen in den folgenden Abschnitten zunächst ausgeblendet. Eine gesonderte Ausführung zu allgemeinen Faktoren, durch welche sich beispielsweise Ungenauigkeiten in mehreren Analysen begründen können, ist in Abschnitt 6.3 zu finden, der sich mit den Limitationen der Datenanalyse befasst.

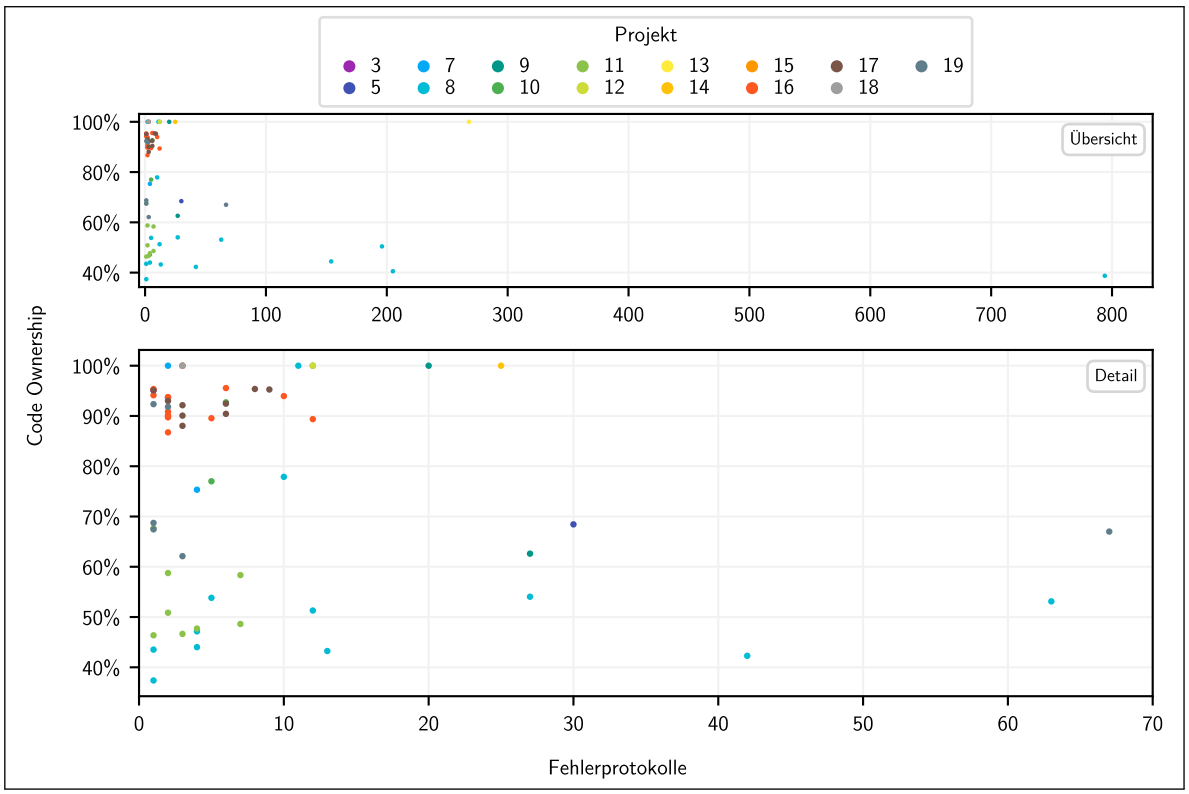
6.2.1 Fehleranfälligkeit der Software

Die Fehleranfälligkeit stellt einen Teil der strukturellen Qualität von Software dar. Falls die Voraussetzungen dafür geschaffen ist, lässt sich diese anhand von Fehlerprotokollen überprüfen. In der Analyse werden dabei die Fehler während der Ausführung der Software betrachtet, da beispielsweise Ergebnisse der Softwaretests zu einem späteren Zeitpunkt nicht mehr verfügbar waren. Somit definiert sich ein Fehler hier als ein unerwartetes Verhalten oder Ergebnis der Software, welches als solches protokolliert wurde und in den meisten Fällen im weiteren Verlauf zu einem Abbruch des Prozesses führt. Teilweise wurden bekannte Fehler auch als solche protokolliert, obwohl diese nicht zwingend zu einer Beendigung des Prozesses geführt haben. Obwohl in der Extraktion auch Fehler in der Entwicklungsumgebung erfasst wurden, werden hier nur Fehler in der produktiven Umgebung des Projektes in die Analyse einbezogen, um eine möglichst hohe Datenqualität zu gewährleisten, da in der Entwicklungsumgebung auch Tests der Software durchgeführt und teilweise absichtlich Fehler provoziert wurden.

Im Rahmen der ersten Hypothese (H1a) werden die Fehler zunächst im Zusammenhang mit dem entsprechenden Code Ownership des Projekts betrachtet. Diesbezüglich wurde die These aufgestellt, dass ein niedrigeres Code Ownership zu einem vermehrten Auftreten von Fehlern führt. Für die Auswertung wurden dabei die tatsächlich entstandenen Fehler anhand ihres Zeitstempels mit dem aktuellen Code Ownership des Projekts assoziiert. Die Darstellung fasst Fehler eines Projektes, welche denselben Wert des Code Ownerships referenzieren, zu

einem Datenpunkt zusammen und stellt sie in einem Streudiagramm dar. Dieses Ergebnis der Datenanalyse ist in Abbildung 11 zu finden.[295]

Abbildung 11: Fehlerprotokolle nach Code Ownership



Quelle: Eigene Darstellung.

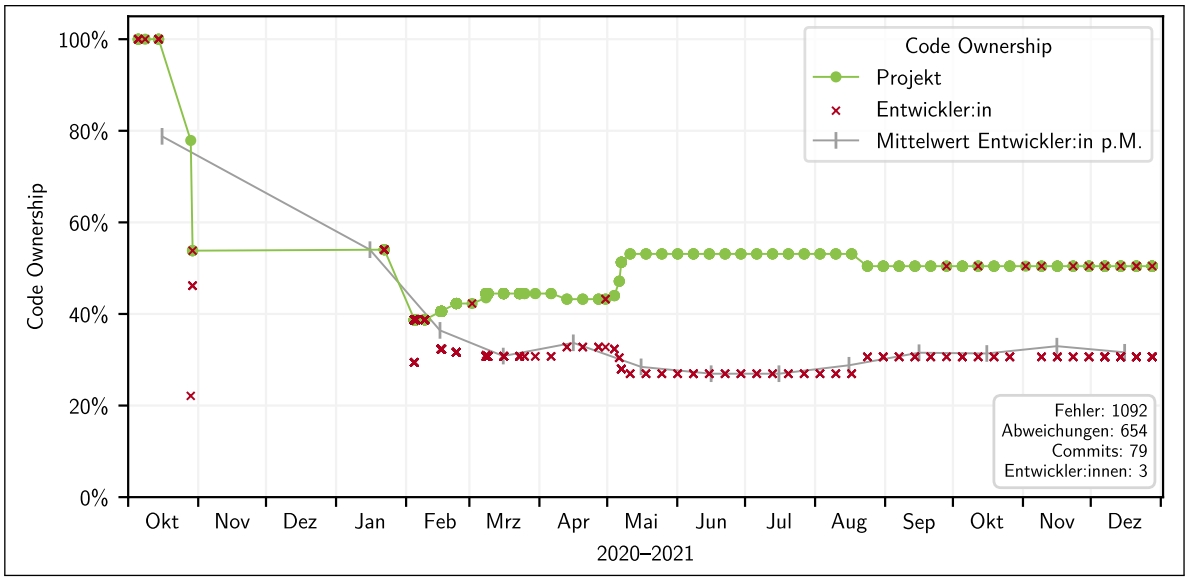
Allgemein lässt sich anhand des Ergebnisses feststellen, dass in der gesamten Spannweite der im vorherigen Abschnitt festgestellten Code Ownership Werte Fehlerprotokolle aufgenommen wurden. Die meisten Fehler (795) wurden dem Bereich zwischen 31 und 40 % zugeordnet, während im Bereich zwischen 91 und 100 % lediglich 433 Fehler verordnet wurden. Insgesamt wurde bei 57 % der aufgetretenen Fehlern ein Code Ownership zwischen 41 und 60 % festgestellt. Dies lässt zunächst vermuten, dass ein niedriger Code Ownership Wert potenziell die Fehlerquote einer Komponente erhöht.[296] Auf der anderen Seite lassen sich jedoch alle Fehlerprotokolle in Assoziation mit einem Code Ownership Wert von unter 60 % auf die Projekte 8 und 11 zurückführen. Diese Projekte sind nach Abbildung 10 allgemein die Einzigen, welche über den gesamten Analysezeitraum einen Code Ownership Wert in diesem Bereich erreicht haben. In insgesamt 10 der 15 betrachteten Projekte mit vorhandenen

[295] siehe auch Anhang 2.4 in größerem Maßstab.
[296] siehe Anhang 3.4.

Fehlerprotokollen wurden kein Fehler mit einem assoziierten Code Ownership Wert von unter 75 % festgestellt. Abschließend lässt sich festhalten, dass die Hypothese zwar bestätigt werden konnte, jedoch weiterhin die Frage gestellt werden muss, ob dies mit Blick auf die Datenbasis tatsächlich ausreichend bewiesen werden konnte.

Zur detaillierteren Untersuchung dieses Sachverhaltes werden anhand von Hypothese H1b die Fehlerprotokolle mit den persönlichen Code Ownership Werten der Entwickler:innen assoziiert. Dadurch kann festgestellt werden, welchen Anteil die Person, welche den fehlerverursachenden Abschnitt des Quellcodes als letztes bearbeitet hatte, am gesamten Quellcode der Komponente zum Zeitpunkt des Fehlers hat. In der Hypothese wird angenommen, dass solche eher durch Entwickler:innen verursacht werden, deren Code Ownership Wert unter dem des Projektes liegt. Zu diesem Zweck wurde für jedes der bereits in der vorherigen Abbildung dargestellten Projekte eine individuelle Auswertung vorgenommen. Dabei wurde für jedes Fehlerprotokoll die Werte des Code Ownerships für das Projekt (der höchste Einzelwert) und der jeweilige Einzelwert der verantwortlichen Person berechnet und dargestellt. Bei Übereinstimmung beider Werte liegen diese in der Abbildung aufeinander. Falls für das entsprechende Projekt abweichende Werte des Code Ownerships existieren, wurde zusätzlich der Mittelwert der Code Ownership Werte der verantwortlichen Entwickler:innen pro Monat dargestellt, um den Trend darzustellen. In Abbildung 12 ist ein Beispiel für ein Projekt dargestellt, bei welchem letzteres der Fall ist.

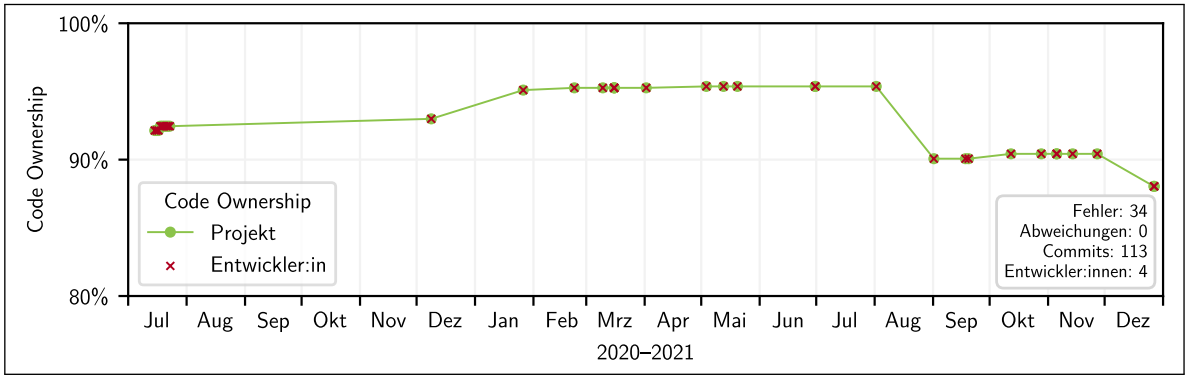
Abbildung 12: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 8)



Quelle: Eigene Darstellung.

Ein Datenpunkt repräsentiert dabei das Auftreten von einem Fehler zu einem bestimmten Zeitpunkt. Aufgrund der Art der Darstellung können mehrere Datenpunkte übereinanderliegen. Von den insgesamt 14 Projekten, in welchen Fehler in der produktiven Umgebung auf einzelne Entwickler:innen zurückverfolgt werden konnten, waren lediglich in dreien Unterschiede zwischen Code Ownership des Projektes und der verantwortlichen Person festzustellen. In den restlichen 11 Projekten entsprach das Code Ownership der Person stets dem des Projektes, was im Umkehrschluss auch bedeutet, dass die Person zum Zeitpunkt des jeweiligen Fehlers den größten Teil des Quellcodes zu verantworten hatte.[297] Ein Beispiel für letzteres ist in Abbildung 13 dargestellt.

Abbildung 13: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 17)



Quelle: Eigene Darstellung.

Auch hier zeigt sich, dass Projekte, deren Code Ownership einen für die Fallstudie unterdurchschnittlichen Wert aufweisen, bei Fehlern eher Abweichungen des Wertes zwischen Projekt und verantwortlicher Person zeigen. Dies ist insbesondere bei Projekt 8 und 11 der Fall. Bei einem niedrigen Code Ownership kann generell davon ausgegangen werden, dass insgesamt eine größere Zahl an Entwickler:innen am Projekt beteiligt war und so auch eine Abweichung des betroffenen Code Ownership Wertes wahrscheinlicher ist. Die individuellen Fehlerzahlen der Projekte unterscheiden sich jedoch teilweise sehr stark voneinander, wodurch auch die Basis der Analyse eine unterschiedliche Größe aufweist. Die Hypothese H1b lässt sich anhand dieser Daten nicht eindeutig beantworten. Im Großteil der Projekte waren die Fehler jeweils auf die Entwickler:innen mit dem höchsten Code Ownership zurückzuführen. Gleichzeitig konnte jedoch auch herausgestellt werden, dass ein allgemein niedriges Code Ownership des Projektes das Auftreten von Fehlern, welche durch Entwickler:innen mit einem niedrigeren Code Ownership zurückzuführen waren, wahrscheinlicher macht.

[297] siehe Anhang 1.1.

Insgesamt zeigen die hier vorliegenden Daten, dass in Projekten mit vergleichbar niedrigem Code Ownership ein vermehrtes Auftreten von Fehlern beobachtet werden konnte. Dieser Effekt ist jedoch von Projekt zu Projekt sehr unterschiedlich ausgeprägt, da nur in wenigen Projekten ein niedriges Code Ownership vorlag. Das Ergebnis entspricht insgesamt den Beobachtungen, welche bereits im Fallbeispiel von Bird et al. festgestellt wurden.[298]

6.2.2 Entwicklungsrisiken

Neben den Auswirkungen auf die Fehleranfälligkeit der Softwarekomponenten im Betrieb kann weiterführend untersucht werden, ob das Prinzip Collective Code Ownership Auswirkungen auf Risiken während der Entwicklung hat. Anhand der gewonnenen Daten wurden hier zwei Bereiche identifiziert, welche sich überprüfen lassen. Zum einen können anhand der Entwicklungsaktivität im zeitlichen Verlauf Abhängigkeiten zu einzelnen Entwickler:innen in Bezug auf die genutzten Technologien[299] untersucht werden. Auf der anderen Seite lassen sich anhand der Daten zur Sprintplanung nach Scrum[300] und den damit verbundenen Merges Verzögerungen in der Projektplanung herausstellen und mit dem Code Ownership in Verbindung bringen.

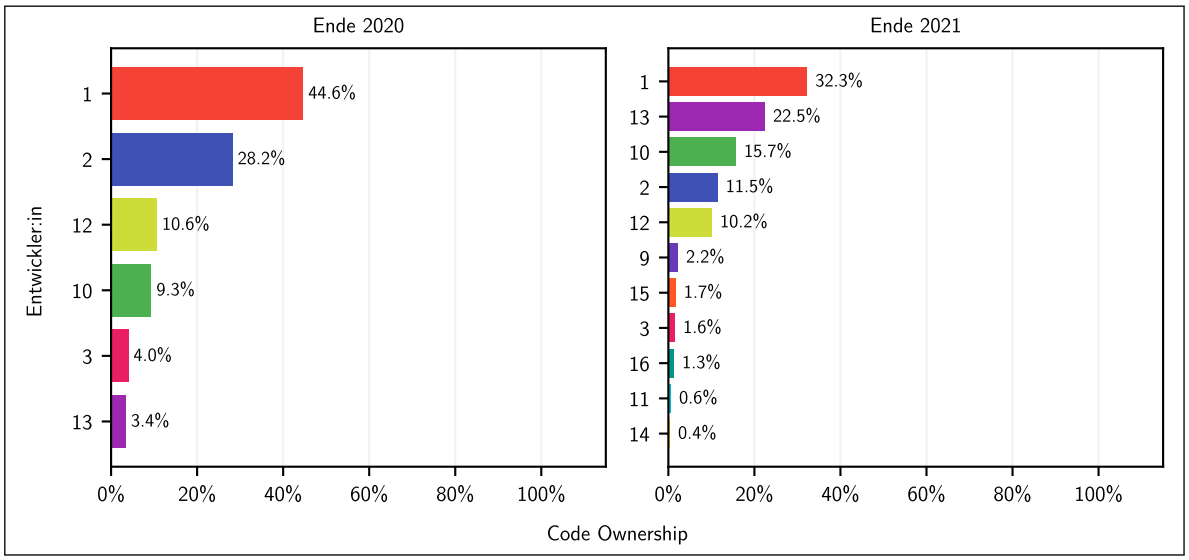
In Bezug auf Hypothese H2a wird zunächst der erste Sachverhalt untersucht, um zu überprüfen, ob Collective Code Ownership tatsächlich dazu beitragen kann, mittelfristig technologische Abhängigkeiten zur Expertise einzelner Personen zu reduzieren. Ein Faktor dafür ist auch, dass das fach- und projektspezifische Wissen im Team weitergetragen wird. Für die Analyse wurden die einzelnen Projekte jeweils in Bezug auf die verwendeten Technologien und ihrer Schnittstellen in der Verwendung klassifiziert. Einem Projekt können dabei mehrere Technologien zugewiesen werden. Anhand dieser Klassifizierung kann anschließend ausgewertet werden, welche Entwickler:innen sich in Projekten eingebracht haben, welche eine bestimmte Technologie verwenden, und ob und wie sich das Verhältnis der Beiträge über den Betrachtungszeitraum verändert hat.

Um dies zu erreichen, wurden für jede der Technologien zwei Auswertungen durchgeführt, jeweils zum Ende des Jahres 2020 und 2021. Die Daten bis Ende 2020 beziehen somit alle Änderungen vom Start der Analyse bis dorthin mit ein, die zweite Auswertung bezieht sich auf die gesamten Daten der Analyse. Dabei konnte festgestellt werden, dass das Wissen und die

[298] vgl. Bird et al., 2011, S. 9 f.
[299] siehe Abschnitt 5.1.2.
[300] siehe Abschnitte 2.2.1 und 5.1.1.

Beiträge zu einigen Technologien sich über mehrere Entwickler:innen verteilen und auch eine Zunahme dieses Phänomens beobachtet werden kann, auf der anderen Seite jedoch teilweise nur ein marginaler Effekt zu erkennen ist oder sich die Abhängigkeit sogar gefestigt hat. Diese Effekte sind im Folgenden anhand von drei Beispieltechnologien dargestellt. Zunächst sind in Abbildung 14 die Daten zur Technologie Vue.js wiedergegeben.

Abbildung 14: Technologiebezogene Abhängigkeiten (Vue.js)



Quelle: Eigene Darstellung.

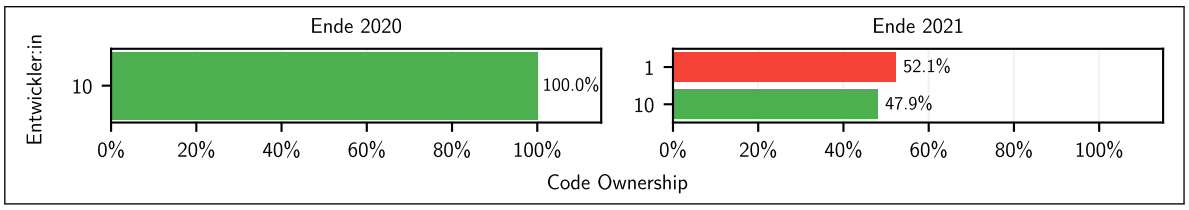
In diesem ersten Beispiel ist erkennbar, dass sich zum einen die Zahl der beteiligten Entwickler:innen fast verdoppelt hat, zum anderen sich aber auch die Verhältnisse deutlich verschoben haben. Während Entwickler:in 1 zu beiden Zeitpunkten den größten Anteil der Änderungen für sich behauptet, hat sich beispielsweise der Anteil von Entwickler:in 13 um ein vielfaches erhöht, während der Anteil von Entwickler:in 2 zurückgegangen ist. Auch sind mehrere Entwickler:innen mit kleineren Änderungen hinzugekommen. Eine ähnliche Entwicklung ist bei anderen Technologien zu erkennen, bei welchen bereits Ende 2020 überdurchschnittlich viele Entwickler:innen beteiligt waren. Dazu gehören neben Vue.js noch Python, Terraform, TypeScript und DynamoDB.[301]

In anderen Fällen waren zum Zeitpunkt Ende 2020 nur wenige Entwickler:innen an den Projekten zu den Technologien beteiligt. In diesen Fällen waren auch im Folgejahr nur wenige Personen an der Entwicklung beteiligt. Die Verhältnisse zwischen den Entwickler:innen haben sich dort jedoch teilweise stark verändert. Ein Beispiel dafür ist in Abbildung 15 dargestellt.

[301] siehe Anhang 1.2.

Die Abhängigkeit konnte dort durch die Beteiligung einer weiteren Person soweit verändert werden, dass diese mittlerweile den größten Anteil besitzt.

Abbildung 15: Technologiebezogene Abhängigkeiten (ARM)

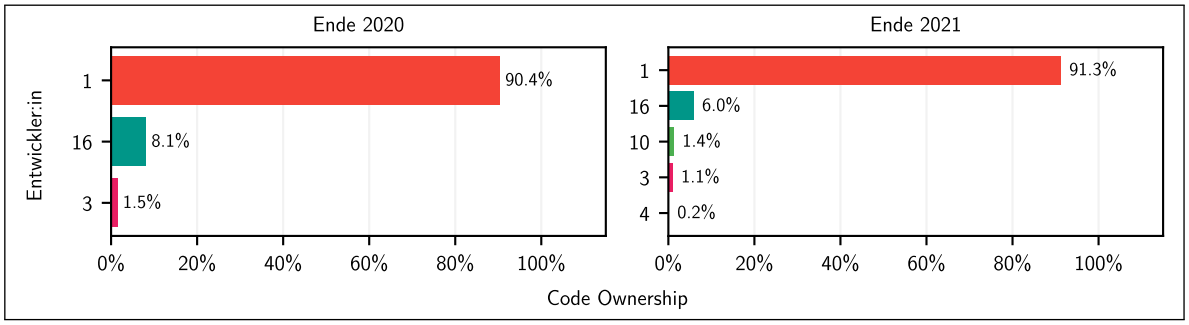


Quelle: Eigene Darstellung.

Diese Ausprägung ist neben ARM auch bei der Docker-Technologie zu beobachten, wo sich das Verhältnis ebenfalls verschoben hat.[302] Dort waren im Kontrast zu ARM zu jedem Zeitpunkt der Analyse zwei Entwickler:innen beteiligt und das Verhältnis hat sich im Vergleich deutlich weniger verändert.

Nur in einem einzigen Fall hat sich die Abhängigkeit zu einer Person vergrößert. Bei der Betrachtung von CloudFormation ist zu erkennen, dass die Zahl der Entwickler:innen zwar von drei auf fünf angestiegen ist, diese jedoch mit Ausnahme der hauptsächlich verantwortlichen Person nur einen sehr geringen Anteil an den Änderungen durchgeführt haben. Da Entwickler:in 1 auch im Jahr 2021 viele Beiträge zur Technologie geleistet hat, haben sich teilweise Anteile anderer Entwickler:innen etwas verkleinert. So hat sich die Abhängigkeit zu Entwickler:in 1 trotz der Beteiligung mehrerer Personen vergrößert.

Abbildung 16: Technologiebezogene Abhängigkeiten (CloudFormation)



Quelle: Eigene Darstellung.

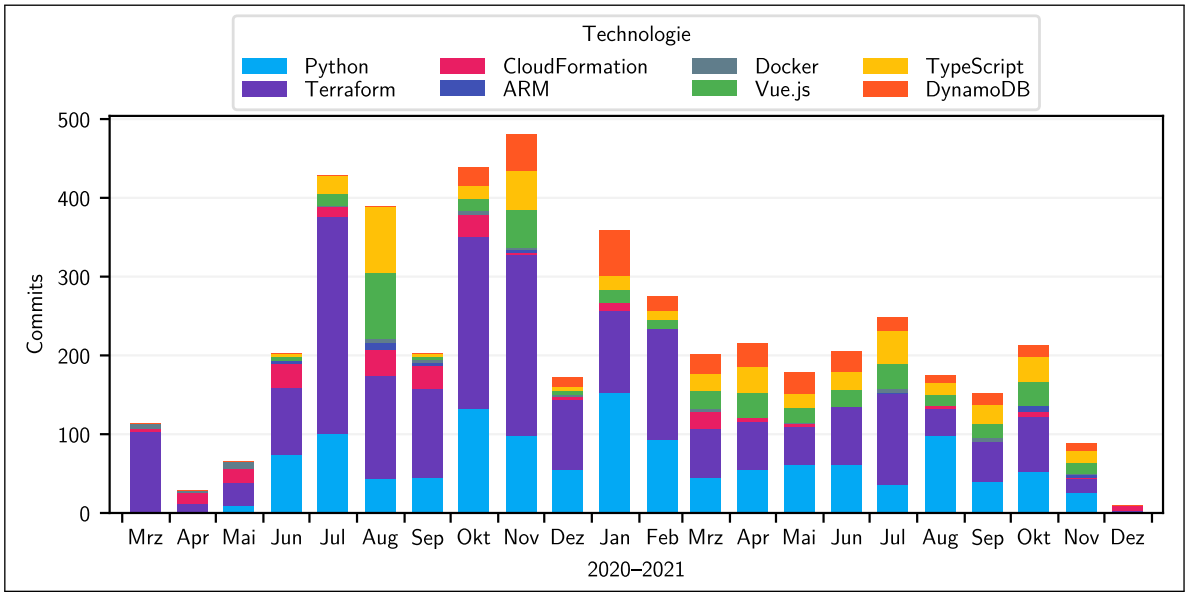
Insgesamt ist bei einer Mehrzahl der Projekte eine Verringerung der Abhängigkeiten festzustellen. Lediglich bei zwei von acht stagniert die Entwicklung, während sich das Verhältnis

[302] siehe Anhang 1.2.

aus Sicht der Abhängigkeit bei einem Projekt verschlechtert hat. Die meisten Abhängigkeiten bestehen erwartungsgemäß zu den Entwickler:innen, welche nach der Entwicklungsaktivität aus Abschnitt 6.1.1 den größten Anteil beigesteuert haben.

Zur Erforschung der Ursache für diese Unterschiede zwischen den Projekten kann zusätzlich die Entwicklungsaktivität nach Technologien aufgeschlüsselt werden. Zieht man diese zu den Beobachtungen hinzu, ist zu erkennen, dass die Technologien, in welchen sich die Verhältnisse nur geringfügig verändert haben, eine niedrige Entwicklungsaktivität im Jahr 2021 vorzuweisen haben. Der Verlauf ist in Abbildung 17 dargestellt. Ein Commit kann sich dabei auf mehrere Technologien beziehen und ist dann in der Statistik mehrerer Technologien inbegriffen.

Abbildung 17: Entwicklungsaktivität nach Technologie

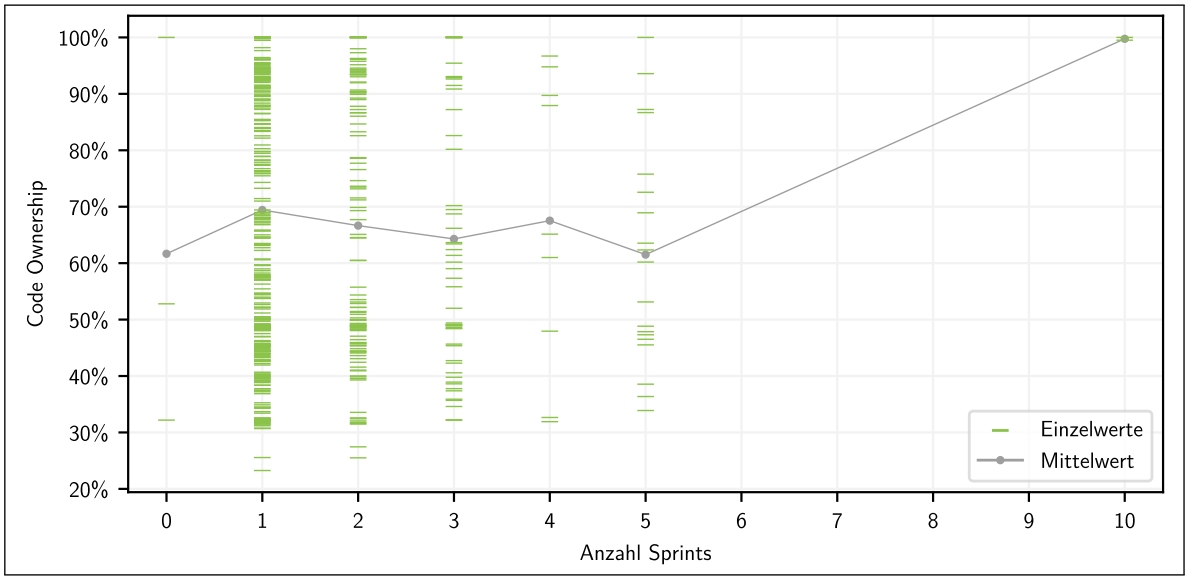


Quelle: Eigene Darstellung.

Die Technologien ARM, Docker und CloudFormation haben im zweiten Jahr im Vergleich zu den anderen Technologien wenige Änderungen erfahren. Bei Technologien, an welchen konstant Entwicklungen vorgenommen wurden, konnten die Abhängigkeiten im Vergleich zum Vorjahr reduziert werden. Zur Auswirkung von Collective Code Ownership auf technologische Abhängigkeiten lässt sich abschließend festhalten, dass ein positiver Effekt festgestellt werden kann. Das Prinzip unterstützt die Erweiterung des Personenkreises, welcher sich um die verschiedenen Komponenten der Software kümmert, und dies unabhängig von der verwendeten Technologie. Der Effekt tritt dabei vor allem dann auf, wenn für jene eine konstante Entwicklungsaktivität vorliegt.

In einem weiteren Schritt wurden die Daten zur Entwicklungsdauer betrachtet. Dabei wird die Anzahl der Sprints, welche für eine Aufgabe bis zum Abschluss benötigt wurden, dem jeweiligen Code Ownership Wert gegenübergestellt. Nach Scrum wird die Aufgabengröße in der Sprintplanung geschätzt und gegebenenfalls auf in handhabbare Teile aufgespalten, sodass diese innerhalb der Dauer eines Sprints abgeschlossen werden können.[303] Eine Verzögerung in der Entwicklung tritt somit ein, wenn sich eine Aufgabe über mehrere Sprints ausdehnt. Dies kann sowohl auf eine falsche Einschätzung des Aufwands, als auch auf unerwartete Herausforderungen während der Entwicklung zurückzuführen sein. Die Hypothese H2b stellt dabei die Vermutung auf, dass ein niedriges Code Ownership zu einer Reduktion von Verzögerungen in der Entwicklung führt. In der in Abbildung 18 dargestellten Auswertung wird dementsprechend der Fokus auf das Erkennen von Korrelationen zwischen diesen Verzögerungen und dem Code Ownership innerhalb der Projekte gelegt.[304]

Abbildung 18: Verzögerungen in der Entwicklung



Quelle: Eigene Darstellung.

Verzögerungen in der Entwicklung sind innerhalb der Fallstudie in jedem der beobachteten Bereiche des Code Ownerships aufgetreten. Obwohl etwa 64 % der Aufgaben innerhalb eines Sprints abgeschlossen werden konnten, konnten Aufgaben mit einer Dauer von zwei bis fünf Sprints identifiziert werden. Der Mittelwert für das Code Ownership bei spezifischer Sprintdauer liegt in allen Fällen zwischen 60 und 70 % mit Ausnahme von zwei Aufgaben, deren Sprintdauer bei 10 liegt. Diese können hier jedoch als Ausreißer gewertet werden. Für

[303] vgl. Sutherland, 2021, S. 20 f., online im Internet.
[304] siehe auch Anhang 2.5 in größerem Maßstab.

Hypothese H2b kann somit geschlussfolgert werden, dass sich das Code Ownership in der Fallstudie nicht auf Verzögerungen in der Entwicklung ausgewirkt hat und die These somit widerlegt ist.

Für die hier untersuchten Risiken während der Entwicklungen konnten lediglich positive Auswirkungen von Collective Code Ownership festgestellt werden. In Bereichen mit einer hohen Entwicklungsaktivität konnten über den Beobachtungszeitraum Abhängigkeiten zu den technologiebezogenen Fähigkeiten einzelner Personen reduziert werden. Auf der anderen Seite konnten bei der Betrachtung von Verzögerungen in der Entwicklung keine negativen Auswirkungen von niedrigem Code Ownership festgestellt werden.

6.2.3 Expertise der Entwickler:innen

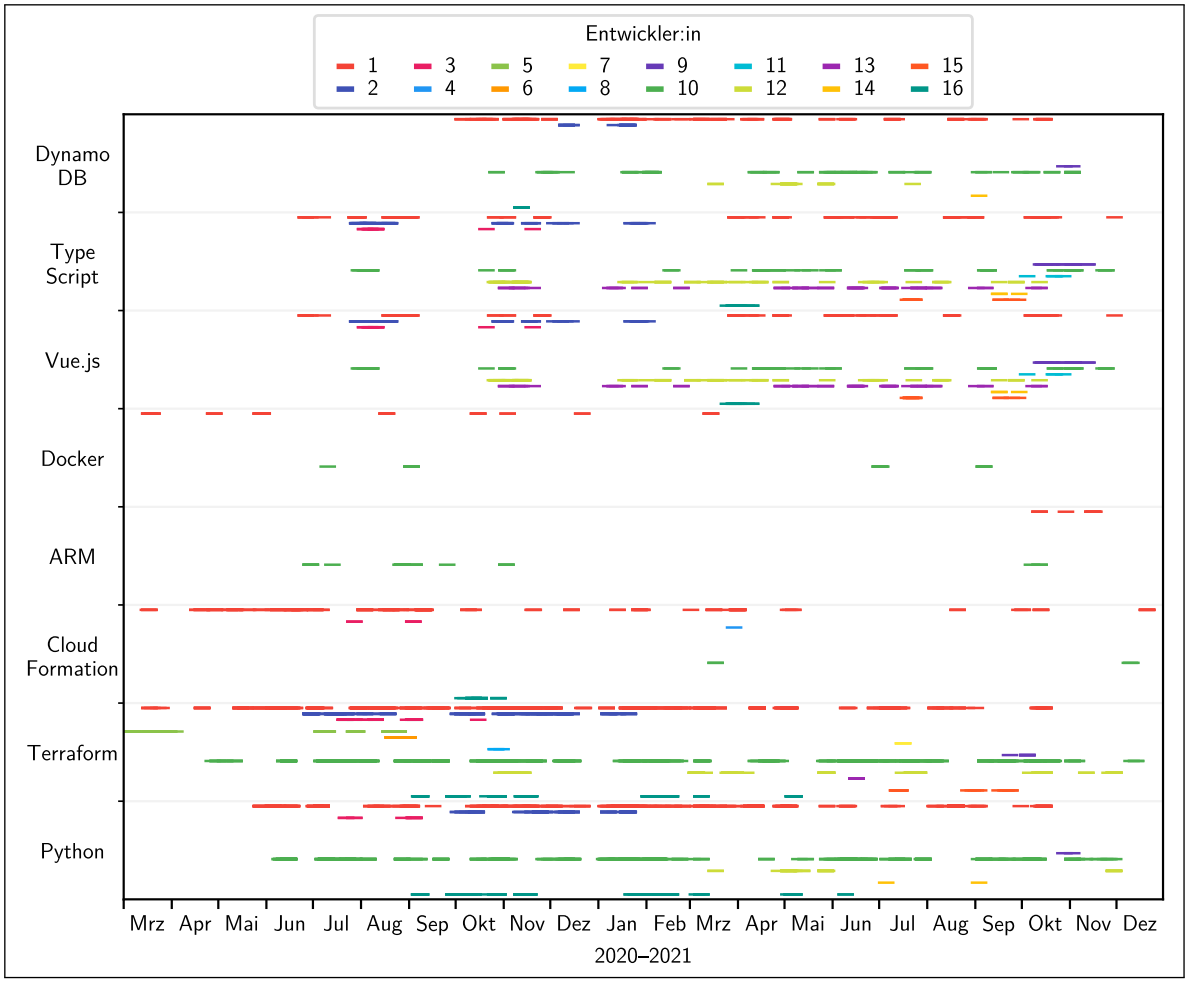
Als letztes wurden die Auswirkungen von Collective Code Ownership auf die Expertise der Entwickler:innen untersucht. Bereits in den Ausführungen zu technologischen Abhängigkeiten in Abschnitt 6.2.2 konnte festgestellt werden, dass die Anteile an der Entwicklung sehr ungleich verteilt sind. Um eine abwechslungsreiche Beschäftigung der Beteiligten zu gewährleisten, ist in der XP-Methode eine regelmäßige Rotation der Aufgaben vorgesehen.[305] Innerhalb der Fallstudie ist diese Anforderung nicht umgesetzt worden. Basierend auf den Daten zur Entwicklungsaktivität der Entwickler:innen in Bezug auf die verwendeten Technologien wurde im Folgenden überprüft, ob sich das Verhalten der Beteiligten über den Analysezeitraum diesbezüglich verändert hat oder eine gleichbleibende Zuweisung beobachtet werden kann. Nach Hypothese H3a wird erwartet, dass letzteres der Fall ist. Zu diesem Zweck wurden die Aktivitäten der Entwickler:innen anhand der vorliegenden Technologien und des zeitlichen Verlaufs ausgewertet. In Abbildung 19 ist das Ergebnis dieser Auswertung gezeigt.[306] Dazu wurde jedem der 16 Entwickler:innen ein horizontaler Bereich innerhalb jeder Technologie zugewiesen, in dem Aktivitäten mit der jeweiligen Farbe gekennzeichnet sind. So lässt sich nachvollziehen, welche Person zu welchem Zeitpunkt an welcher Technologie mitgewirkt hat.

Die Gruppe der 16 Entwickler:innen lässt sich in vier Bereiche klassifizieren. Zum einen besteht eine Teilmenge der Entwickler:innen, welche sich einerseits mit überdurchschnittlich vielen Beiträgen in der Entwicklung beteiligt und diese heterogen über mehrere Technologien hinweg aufgeteilt haben. Zu dieser Gruppe zählen die Entwickler:innen 1, 2, 10, 12 und 16. In

[305] vgl. Beck, 2005, S. 42, 47.
[306] siehe auch Anhang 2.6 in größerem Maßstab.

dieser Personengruppe lässt sich feststellen, dass bei den meisten eine stetige Abwechslung der Technologien vorherrscht. Nur bei 2 und 16 wurde die Entwicklung mit den Technologien DynamoDB respektive TypeScript und Vue.js vergleichbar spät angestoßen. Eine weitere Gruppe hat zwar viele Beiträge erstellt, sich dabei jedoch mit einer kleinen Zahl an Technologien beschäftigt. Dazu zählen Entwickler:innen 5 und 13. Während Entwickler:in 5 sich ausschließlich mit Terraform beschäftigt hat, bezieht sich dies bei 13 auf die Entwicklung der Benutzeroberfläche mit Vue.js und TypeScript.

Abbildung 19: Entwicklungstätigkeit nach Entwickler:in und Technologie



Quelle: Eigene Darstellung.

Die beiden weiteren Gruppen beinhalten Personen, welchen einen vergleichbar geringen Beitrag zu den Softwarekomponenten geleistet haben. Auch dort ist zu beobachten, dass teilweise eine aus technologischer Sicht heterogene Beschäftigung vorliegt, wie bei Entwickler:innen 3 und 9. Bei der Mehrzahl der Entwickler:innen ist jedoch das Gegenteil der Fall.

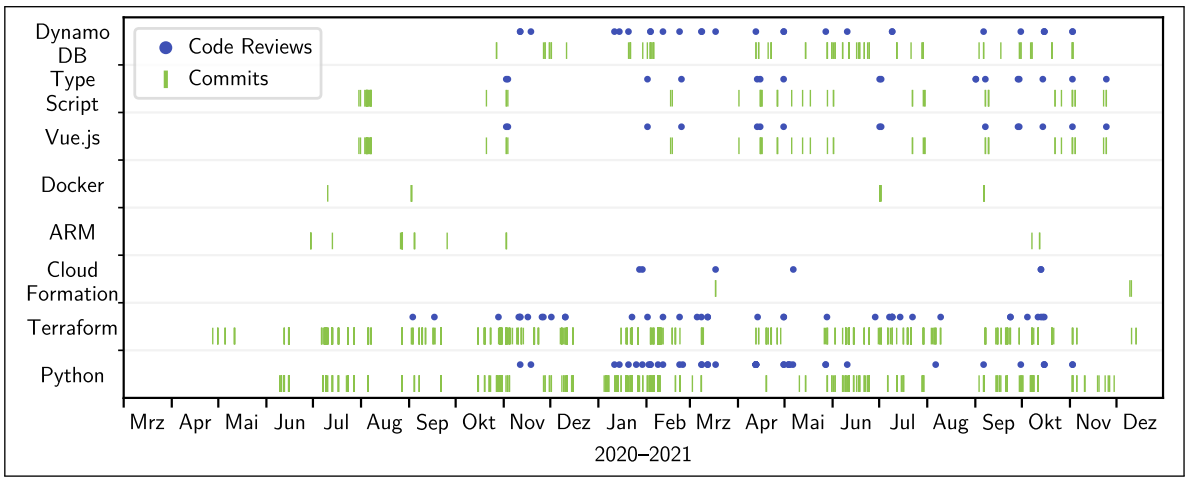
Letzteres trifft hier auf 4, 6, 7, 8, 11 und 14 zu. Alle Personen dieser beiden Gruppen waren nur für einen kurzen Zeitraum an der Entwicklung beteiligt. Eine weitere Gemeinsamkeit der Entwickler:innen mit geringem Beitrag, jedoch hoher Technologievielfalt ist, dass sie ein ähnliches Beschäftigungsprofil ausweisen. Entwickler:innen 3 und 9 haben sich beide primär mit Python, Terraform, Vue.js und TypeScript beschäftigt. Die Entwickler:innen mit geringem Beitrag und auch geringer Vielfalt haben sich hingegen mit jeweils unterschiedlichen Themen beschäftigt.

Das Ergebnis aus diesen Beobachtungen stellt sich differenziert dar. Bei Entwickler:innen, welche sich in hohem Maße an der Implementierung der Komponenten beteiligt haben, ist in der Mehrzahl eine große technologische Vielfalt festzustellen. Diese ist meist von Beginn an ausgeprägt und nur in einzelnen Fällen über einen längeren Zeitraum entstanden. Bei Entwickler:innen mit einer geringen Beteiligung im Projekt der Fallstudie ist hingegen festzustellen, dass sich jene auf einzelne Technologien beschränken. Bezogen auf die Hypothese H3a bedeutet dies, dass sich die Frage nach einer möglichen gleichartigen Beschäftigung anhand dieser Ergebnisse nicht eindeutig beantworten lässt. Unter der Zunahme der Ergebnisse aus Abschnitt 6.2.2 zu technologischen Abhängigkeiten lässt sich nur im Falle von ARM ein deutlicher Unterschied in der Beschäftigung einer Person zu einem späteren Zeitpunkt feststellen. Die meisten Entwickler:innen mit einem vergleichbar hohen Code Ownership kamen in der Entwicklung bereits früh mit mehreren Technologien in Kontakt. Offen bleibt jedoch, ob die Entwickler:innen bereits mit Vorkenntnissen zu den untersuchten Technologien in das Projekt eingestiegen sind oder sie die Fähigkeiten während des Projektverlaufs erlernt haben.

Während die Aufgabenrotation in der Fallstudie nicht festgeschrieben war, wurden im Kontrast zu den in XP festgelegten Praktiken auf regelmäßiger Basis Code Reviews vorgenommen. In einer zweiten Auswertung soll entsprechend untersucht werden, ob das Durchführen von Code Reviews dazu beitragen kann, dass sich die Entwickler:innen häufiger mit für sie neuen Technologien beschäftigen. Dazu wurde die Hypothese H3b aufgestellt, nach der die Beteiligung in solchen Reviews eine Beschäftigung mit neuen Technologien fördert. Dies basiert auf der in Abschnitt 3.3.3 formulierten Annahme, dass die Beteiligung in Code Reviews den Aufbau von Wissen zur behandelten Technologie unterstützt. Ergänzend zu den Ausführungen des ersten Teils dieses Abschnittes werden dementsprechend neben der Entwicklungsaktivität zusätzlich die Code Reviews betrachtet. Dazu wurde die kombinierte Übersicht aus Abbildung 19 in Darstellungen für jede beteiligte Person aufgeteilt und um

Daten zu den durchgeführten Code Reviews ergänzt.[307] Ein Beispiel für Entwickler:in 10 ist in Abbildung 20 gezeigt.

Abbildung 20: Entwicklungsaktivität und Code Reviews (Entwickler:in 10)



Quelle: Eigene Darstellung.

Dort ist zu erkennen, dass die Code Reviews den Softwareentwicklungsprozess stetig begleitet haben. In diesem Fall war bei den meisten Technologien zunächst eine Beteiligung in der Entwicklung zu verzeichnen, bevor das erste Code Review durchgeführt wurde. Die einzige Ausnahme bildet CloudFormation, wo im Januar 2021 zunächst zwei Code Reviews abgehalten wurden, bevor im März die erste Beteiligung im Quellcode stattfand. Anschließend sind nur wenige Aktivitäten zu dieser Technologie zu beobachten. Generell wurden Code Reviews nach den Daten erst ab Oktober 2020 regelmäßig durchgeführt. Auch bei den restlichen Entwickler:innen ist ein ähnliches Verhalten festzustellen. Die Code Reviews entstehen als Folge von der Beschäftigung mit einer Technologie und sind nicht der Auslöser dafür. Weiterführend wurde herausgestellt, dass der überwiegende Teil der Reviews durch die Entwickler:innen 1, 10, 12 und 13 umgesetzt wurden, welche an sich schon einen maßgeblichen Teil an der Entwicklung zu den einzelnen Technologien für sich beanspruchen können. Die Aussage von Hypothese H3b lässt sich somit anhand der Daten der Fallstudie widerlegen, da anhand der Durchführung von Code Reviews nur ein marginaler Effekt auf die Beschäftigung mit neuen Themen beobachtet werden konnte.

[307] siehe Anhang 1.3.

6.3 Limitationen der Datenanalyse

Die Durchführung einer solchen Datenanalyse birgt die Gefahr, dass bedeutende Randbedingungen nicht einbezogen wurden oder die Daten gewisse Ungenauigkeiten aufweisen. Der letzte Abschnitt zur Auswertung der Daten widmet sich solchen Faktoren, um ein Bewusstsein für diese Einflüsse zu schaffen.

In den Ausführungen zur Fehleranfälligkeit der Software wurden die Fehlerprotokolle lediglich als solche betrachtet. Die Auswertung erfolgte aufgrund der hohen Fehlerzahl automatisiert, wodurch es nicht möglich war, eine Fehler-Ursachen-Analyse durchzuführen. Dadurch wurde nicht betrachtet, wie es zur Entstehung des jeweiligen Fehlers gekommen ist. Innerhalb der untersuchten Softwarekomponenten bestehen in unterschiedlichem Ausmaß Abhängigkeiten zu externe Ressourcen. Eine hohe Fehleranfälligkeit kann dementsprechend ebenso in der Unzuverlässigkeit einer solchen Schnittstelle begründet sein. Auch der Funktionsumfang der jeweiligen Komponente wurde außen vor gelassen, genauso wie die Intensität der Nutzung. Falls letztere überdurchschnittlich hoch ausfällt, sind mehr Fehler zu erwarten. Nachdem in einem ersten Schritt die Fehleranfälligkeit in Bezug auf das allgemeine Code Ownership des Projektes untersucht worden ist, brach die zweite Analyse in dem Bereich diesen Wert auf die jeweilige Person herunter, welche für den zu dem Fehler führenden Codeabschnitt verantwortlich war. Diese Verantwortlichkeit wurde über die Nutzung der sogenannten „Blame“-Funktion von Git hergestellt, welche anhand der dort gespeicherten Metadaten eine Zuweisung zur Person ermöglicht.[308] Diese zeigt auf diejenige Person, welche die entsprechende Zeile im Quellcode zuletzt geändert hat. Zu diesen Änderungen können jedoch auch Ereignisse wie Restrukturierung oder Formatierung des Quellcodes gehören, bei denen kein Einfluss auf die Logik genommen wird. Diese Charakteristik kann ebenfalls zu Ungenauigkeiten in den analysierten Daten führen. In Tabelle 2 ist zusätzlich zu erkennen, dass für etwa 27 % der Fehlerprotokolle in der Produktionsumgebung keine Zuweisung erfolgen konnte. Zudem steigt mit einer Erhöhung des Code Ownerships einer Person auch die Wahrscheinlichkeit für Fehler, welche ihren Ursprung in einem von ihr verantworteten Bereich haben.

Auch bei der Analyse des Einflusses auf Risiken in der Entwicklung können solche Faktoren beschrieben werden. Die Abhängigkeiten zu einzelnen Entwickler:innen in Bezug auf die Technologien wurde anhand ihres Beitrags zu den jeweiligen Komponenten gemessen.

[308] vgl. Chacon und Straub, 2014, S. 300.

Dabei wird nicht näher betrachtet, in welchem Umfang sie sich im Allgemeinen im Projekt der Fallstudie eingebracht haben. Da nach den Untersuchungen aus Abschnitt 6.1.1 acht Personen jeweils weniger als 1 % zu den gesamten Implementierungen beigetragen haben, konnten sie nicht zu einer maßgeblichen Veränderung der Abhängigkeiten beitragen. Weiterführend war zu erkennen, dass nach Abbildung 19 viele Entwickler:innen nicht über den gesamten Analysezeitraum an der Entwicklung beteiligt waren, sondern nur für einen begrenzten Zeitraum. Durch die Einbeziehung neuer Entwickler:innen ist auch ohne das Prinzip des Collective Code Ownerships bereits eine Reduktion der Abhängigkeiten zu erwarten, welche jedoch nicht genau beziffert werden kann. Bei der Betrachtung der Verzögerungen während der Entwicklung gilt wie bei der Untersuchung der Fehlermeldungen, dass keine Ursachenanalyse für die Verzögerung durchgeführt wurde. Auch hier besteht die Möglichkeit, dass die Ursachen in externen Abhängigkeiten sowie weitere Faktoren wie der Verfügbarkeit der verantwortlichen Person oder einer falschen Einschätzung von Aufwänden begründet sind.

Abschließend wurden die Auswirkungen auf die Expertise der Entwickler:innen betrachtet. Dabei ist nicht berücksichtigt worden, welche Vorkenntnisse die einzelnen Personen bereits mit in das Projekt gebracht haben und wo gänzlich neues Wissen aufgebaut werden musste. Diese Daten sind dort aus technischer Sicht nicht hinterlegt. Daher konnte in der Analyse nur auf Veränderungen innerhalb des Projektes eingegangen werden. Bei der anschließenden Betrachtung der Code Reviews wurde festgestellt, dass diese erst mit einer Verzögerung von einigen Monaten nach Entwicklungsstart regelmäßig durchgeführt wurden. Die Analyse stützt sich dementsprechend nur auf die Daten zu Reviews ab diesem Zeitpunkt und konnte Einflüsse zu Beginn der Entwicklung nicht herausstellen.

Die erfolgte qualitative Datenanalyse wurde unter Einbeziehung aller aus Sicht des Projektes verfügbaren Daten zu den beschriebenen Hypothesen durchgeführt. Sie erfolgte mit dem Grundsatz, dass die Datenextraktion und -analyse, mit Ausnahme einzelner Metainformationen, ohne manuelle Interaktion erfolgen kann, um die großen Datenmengen zu bewältigen. Eine Aufhebung der genannten Limitationen könnte nur unter Einbeziehung von Daten aus weiteren Quellen erfolgen. Dafür wären beispielsweise qualitative Methoden wie eine Befragung denkbar. Um die daraus gewonnenen Daten mit den technischen Daten zu verknüpfen, ist jedoch eine Assoziation von personalisierten Daten notwendig. Da die Analyse bewusst gänzlich Rückschlüsse zu einzelnen Personen vermeidet, ist eine Ergänzung der Datenbasis auf diesem Wege nicht möglich gewesen.

7 Schlussbetrachtung

Die Masterthesis beschäftigte sich mit der Verteilung von Verantwortlichkeiten in der agilen Softwareentwicklung. Dazu erfolgte im ersten Teil neben der Darstellung von Methoden der agilen Softwareentwicklung eine Bestimmung von Modellen für eine solche Verteilung. Anschließend wurden in Bezug darauf Kriterien entwickelt, anhand derer sich ein Verteilungsmodell auf dessen Erfolg hin überprüfen lässt. Das Kapitel wurde mit der Definition von drei Kriterien abgeschlossen, welche sich in die Bereiche der Qualität der Software, Risiken in der Entwicklung und der Expertise des Entwicklungsteams aufteilen.

Auf Basis der theoretischen Auseinandersetzung wurde anschließend eine Fallstudie konzipiert, welche sich mit den Auswirkungen des Collective Code Ownership Prinzips befasste. Durch die Analyse von Daten eines IT-Projektes der Bayer AG sollte untersucht werden, welche Auswirkungen dieses spezifische Verantwortungsmodell auf dessen Erfolg hatte. Zur Durchführung dieser Analyse wurden die internen Projektabläufe dargestellt, um daraus Daten für eine quantitative Analyse abzuleiten. Darauf folgte der Entwurf und die Implementierung einer Anwendung, welche die Daten aus den identifizierten Systemen extrahiert und aufbereitet. Den Abschluss der Konzeption bildete die Formulierung von Hypothesen, welche die Auswertung der Daten einleitete und deren Ziele beschrieben. Diese orientierten sich an den Kriterien, welche zum Abschluss des Theorieteils gebildet wurden. Die daraus gewonnenen Erkenntnisse werden im Folgenden dargestellt.

7.1 Zusammenfassung der Ergebnisse

Im Anschluss an die Konzeption der Fallstudie und die Extraktion der dafür relevanten Informationen aus dem Entwicklungsprozess konnte mit der Analyse der Daten fortgefahren werden. Diese ist in zwei Schritte unterteilt. Zunächst wurden allgemeine Aspekte in Bezug auf das Projekt untersucht, darunter die Entwicklungsaktivität, die Häufigkeit von Fehlern in Bezug darauf sowie der Verlauf des Code Ownerships der einzelnen Teilkomponenten der Software. Anschließend erfolgte die Auswertung der Daten in direktem Bezug zu den Hypothesen.

Die grundlegenden Analysen erfolgten mit dem Ziel, das Fundament für die tiefgreifenden Analysen zu legen und Randbedingungen in diese Analysen mit einbeziehen zu können. Dazu wurde zunächst die Entwicklungsaktivität der einzelnen Entwickler:innen im Gesamtkontext

des Projektes dargestellt. Über den gesamten Analysezeitraum von März 2020 bis Dezember 2021 konnte eine recht konstante Aktivität festgestellt werden, welche sich nur in einzelnen Phasen unterschied. Mithilfe der anschließenden Darstellung zum Auftreten von Fehlern sollte untersucht werden, ob schon die Varianz der Entwicklungsaktivität zu zeitlichen Unterschieden im Auftreten von Fehlern führt, damit solche Auswirkungen in späteren Analysen berücksichtigt werden können. Es konnte herausgestellt werden, dass die Entwicklungsaktivität zwar durchaus einen Einfluss auf das Auftreten von Fehlern während der Laufzeit der einzelnen Teilkomponenten hatte, dies jedoch nur in einigen Bereichen und primär im Entwicklungssystem der Software beobachtet werden konnte. Die Analyse in Bezug auf die Hypothesen zur Fehleranfälligkeit im späteren Verlauf sollten sich im Kontrast dazu lediglich auf Daten der produktiven Umgebung beziehen.

In einem zweiten grundlegenden Schritt wurde eine Analyse zum Verlauf des Code Ownerships aus Sicht der Teilprojekte durchgeführt. Der Begriff des Projekts bezieht sich hierbei nicht auf das Gesamtprojekt, sondern auf einzelne Teilprojekte der Software, welche dort entwickelt wurden. Der Begriff kann somit synonym zu den Teilkomponenten verwendet werden. Für die einzelnen Teilprojekte wurde festgestellt, dass deren Code Ownership Werte sich durchgehend auf einer Skala zwischen knapp unter 40 % und 100 % bewegen. Dieser Prozentwert bezeichnet den Anteil der Person, welche den größten Teil der gesamten Änderungen der Teilkomponente für sich beansprucht. Über den Verlauf des Analysezeitraums hat sich dieser Wert bei den meisten Teilprojekten verringert, was bedeutet, dass eine Diversifizierung bei den beteiligten Entwickler:innen beziehungsweise deren Beiträgen stattgefunden hat. Zusätzlich konnte beobachtet werden, dass einige Teilkomponenten nur von einer einzelnen Person entwickelt wurden.

Auf Basis dieser gewonnenen Erkenntnisse zu den Rahmenbedingungen des Projekts in Bezug auf die Entwicklungsaktivität, dem Auftreten von Fehlern und dem Verlauf des Code Ownerships konnte anschließend in die Betrachtung der Hypothesen eingestiegen werden. Diese orientierten sich neben der Fehleranfälligkeit der Software zusätzlich an den Auswirkungen auf Risiken in der Entwicklung und die Expertise des Entwicklungsteams. Diese Eingrenzung wurde bereits in der Forschungsfrage und bei der Bestimmung der Kriterien zum Ende des Theorieteils vorgenommen.

Die ersten beiden Hypothesen beschäftigten sich mit den Auswirkungen des Collective Code Ownership Prinzips auf die Häufigkeit des Auftretens von Fehlern. In einer ersten Betrachtung wurde dabei untersucht, ob das Code Ownership eines Teilprojekts einen Einfluss darauf hat, wie häufig dort über den Analysezeitraum hinweg Fehler aufgezeichnet wurden. Anhand

der im vorherigen erfolgten Auseinandersetzung mit diesbezüglichen Studien wurde in der Hypothese die Vermutung aufgestellt, dass ein niedriges Code Ownership das Auftreten von Fehlern begünstigt. Dies bedeutet, dass in Teilprojekten, an denen mehrere Entwickler:innen beteiligt sind und auch entsprechende Anteile an dortigen Änderungen haben, häufiger Fehler auftreten müssten. Die Hypothese konnte dabei insofern bestätigt werden, dass im Bereich des Code Ownerships zwischen 31 und 60 % mehr als die Hälfte aller Fehler verordnet werden konnten. Diese waren jedoch primär auf zwei der fünfzehn Teilprojekte zurückzuführen, wodurch die Frage aufgekommen ist, ob die Datenbasis für eine Bestätigung der Hypothese dementsprechend groß genug war. Um diesem Phänomen weiter auf den Grund zu gehen, wurde in einer zweiten Hypothese formuliert, dass die Ursache für Fehler eher in solchen Abschnitten des Quellcodes vermutet werden, welche von Entwickler:innen mit niedrigen persönlichen Code Ownership Werten verantwortet wurden. Dabei wurde festgestellt, dass die Fehler bei den meisten Teilprojekten auf die Entwickler:innen mit den jeweils höchsten Anteilen am Quellcode zurückgeführt werden konnten. Gleichzeitig wurde jedoch herausgestellt, dass ein insgesamt niedriges Code Ownership in einer Teilkomponente allgemein zu einem häufigeren Auftreten von Fehlern geführt hat und zum anderen auch Fehler wahrscheinlicher macht, welche aus Codeabschnitten entstammen, deren Urheber:in einen geringen Anteil am gesamten Quellcode der Komponente hat. Die Sachverhalte der zweiten Hypothese ließen sich dementsprechend nicht abschließend beantworten.

Im zweiten Abschnitt wurden Hypothesen untersucht, welche sich auf Risiken in der Entwicklung bezogen. Die erste Analyse bezog sich dabei auf Abhängigkeiten zu einzelnen Entwickler:innen in Bezug auf die im Projekt eingesetzten Technologien, welche dafür genutzt wurden, das Produkt beziehungsweise die Software zu entwickeln. In der Hypothese wurde diesbezüglich formuliert, dass durch das Prinzip des Collective Code Ownerships eine Reduzierung solcher Abhängigkeiten vermutet wird, da die Entwickler:innen sich in allen Teilbereichen der Software einbringen können. Anhand der Auswertungen konnte festgestellt werden, dass sich bei fünf von den acht verwendeten Technologien von der ersten Untersuchung zum Ende des Jahres 2020 bis zum Ende des Jahres 2021 eine Reduzierung der Abhängigkeit eingestellt hat. Bei zwei Technologien konnte währenddessen eine Stagnation der Abhängigkeit zu einzelnen Entwickler:innen festgestellt werden, während sich diese bei einer Technologie erhöht hat. Als Einflussfaktor für eine Reduzierung konnte zusätzlich herausgestellt werden, dass eine vergleichbar hohe und konstante Entwicklungsaktivität in den Bereichen, wo eine bestimmte Technologie angewandt wurde, dazu geführt hat, dass sich dort die Abhängigkeiten reduziert haben. Dementsprechend konnte die Hypothese insofern bestätigt werden, dass das Prinzip der kollektiven Verantwortung in der Fallstudie einen

positiven Effekt im Sinne der Reduktion von Abhängigkeiten hatte. Als nächstes wurde das Projektrisiko der Verzögerungen in der Entwicklung betrachtet. Dafür wurde anhand der nach der Methode Scrum erfolgten Projektplanung untersucht, wann es zu Verzögerungen bei der Umsetzung von Weiterentwicklungen gekommen ist und welches Code Ownership zu diesem Zeitpunkt im betroffenen Teilprojekt vorlag. Dabei wurde anhand der analysierten Daten geschlussfolgert, dass sich die Werte des Code Ownerships bei Aufgaben, welche innerhalb der vorgesehenen Zeit abgeschlossen wurden und bei Verzögerungen nicht maßgeblich voneinander unterscheiden und somit kein Einfluss festgestellt werden konnte.

Der letzte Teil der Datenanalyse in Bezug auf die Hypothesen setzte sich mit der Expertise des Entwicklungsteams auseinander. Neben der bereits im zweiten Abschnitt erfolgten Betrachtung der Technologien sollte hier zusätzlich untersucht werden, ob sich die Beschäftigung der Entwickler:innen über den Zeitraum der Analyse diversifiziert und ob die Durchführung von Code Reviews einen Einfluss auf diese Verteilung hatte. In der XP-Methode ist eine definierte Aufgabenrotation vorgesehen, welche eine gleichbleibende Beschäftigung der Entwickler:innen verhindern soll. Eine solche Maßnahme wurde im Projekt der Fallstudie nicht umgesetzt. Für die Analyse wurden in einem ersten Schritt die Aktivitäten der Entwickler:innen in Bezug auf die Technologien dargestellt. Aus dieser konnten verschiedene Beobachtungen abgeleitet werden. Zum einen gab es eine relativ große Gruppe der Entwickler:innen, welche sich einerseits mit vergleichsweise vielen Beiträgen in die Entwicklung eingebracht haben und dabei diverse Technologien verwendeten. Bei der Gruppe der Entwickler:innen, welche einen geringen Beitrag geleistet haben, war hingegen auch nur eine geringe Technologievielfalt festzustellen. Durch deren geringen Beitrag konnte jedoch nicht abschließend beantwortet werden, ob hier gegebenenfalls eine fortlaufend gleichartige Beschäftigung festgestellt werden könnte, da dafür die Datenmenge nicht ausreichte. Daher ließ sich die fünfte Hypothese weder bestätigen, noch widerlegen. Um diese Thematik zu vertiefen, wurde anschließend der Einfluss von Code Reviews auf eine solche Diversifizierung untersucht. In der Hypothese wurde dabei vermutet, dass das Code Review durch die Auseinandersetzung mit Entwicklungen anderer Personen eine Beschäftigung mit den dort verwendeten Technologien fördern kann. Aufgrund der Tatsache, dass Code Reviews im Projekt fast ausschließlich nach bereits erfolgter Beschäftigung mit einer Technologie beobachtet wurden, konnte kein Einfluss dieser Praktik festgestellt und die Hypothese im Kontext der Fallstudie widerlegt werden.

Abschließend kann mit Blick auf die Forschungsfrage zusammengefasst werden, dass in der Analyse zu den drei untersuchten Teilbereichen unterschiedliche Folgen des Collective Co-

de Ownership Prinzips festgestellt werden konnten. In Bezug auf die Fehleranfälligkeit der Software wurde herausgestellt, dass eine kollektive Verantwortung in Bezug auf einzelne Teilkomponenten durchaus dazu geführt hat, dass dort vermehrt Fehler aufgetreten sind. Gleichzeitig konnten nur vergleichsweise wenige Fälle festgestellt werden, bei denen diese auf Entwickler:innen mit einem niedrigen Anteil an den Entwicklungen zurückzuführen waren. Auf der anderen Seite konnte das Prinzip mit Blick auf Entwicklungsrisiken jedoch dazu beitragen, dass Abhängigkeiten zu einzelnen Entwickler:innen in Bezug auf die verwendeten Technologien verringert werden konnten, ohne dass es zu häufigeren Verzögerungen in der Entwicklung gekommen ist. Auch mit Blick auf die Expertise der Entwickler:innen ist hervorzuheben, dass sich viele der beteiligten Personen mit mehreren Technologien auseinandergesetzt haben und nur bei wenigen eine gleichbleibende Beschäftigung festgestellt werden konnte.

Für Projekte der agilen Softwareentwicklung im Allgemeinen lässt sich daraus ableiten, dass das Prinzip der kollektiven Verantwortung dort Sinn macht, wo die Priorität auf dem Ausbau der Expertise der beteiligten Entwickler:innen liegt und dafür geringfügige Einbußen in der Softwarequalität hingenommen werden können. Ist letzteres nicht der Fall, kann eine klare Zuweisung von Verantwortlichen dazu beitragen, die Fehleranfälligkeit der Komponenten einer Software zu reduzieren. Collective Code Ownership kann ebenfalls dabei helfen, durch das Fehlen von Rollenzuweisungen solche Risiken zu reduzieren, welche in Bezug zu äußeren Einflüssen wie Ausfällen von Entscheidungsträger:innen stehen. Durch den Aspekt des Aufbaus von Expertise kann das Prinzip außerdem dazu genutzt werden, durch die Ausbildung von Fachpersonal dem Fachkräftemangel im Unternehmen entgegenzuwirken. Um solche Auswirkungen weiter zu untersuchen, gibt es verschiedene Möglichkeiten zu einer Vertiefung der Thematik. Daher wird im Anschluss ein Ausblick auf weitere Möglichkeiten der Analyse gegeben.

7.2 Ausblick

Während der Analyse zur Fallstudie wurde an mehreren Stellen festgestellt, dass der Umfang der Daten für die Untersuchungen zu bestimmten Hypothesen nicht ausreichend groß ist. Zusätzlich konnte bei einer Teilmenge der beteiligten Entwickler:innen beobachtet werden, dass sie nur für einen kurzen Zeitraum am Projekt beteiligt waren und daher nur einen geringen Einfluss auf die Ergebnisse hatten. Dadurch ergeben sich verschiedene Möglichkeiten, wie die untersuchten Themen weiter vertieft werden können. Zum einen kann die

Analyse über einen größeren Zeitraum ausgedehnt werden. Hier war dies nicht möglich, da das Projekt erst im Frühjahr 2020 begonnen wurde und somit der Analysezeitraum durch dieses Startdatum begrenzt war. Eine weitere Möglichkeit ist, die Fallstudie auf ein größeres Projekt oder sogar mehrere Projekte auszuweiten. Auch eine Analyse unter Einbeziehung personenbezogener Daten wäre denkbar, falls die organisatorischen Gegebenheiten dies ermöglichen.

Dabei könnten Beobachtungen dazu angestellt werden, ob sich die in der Datenanalyse gewonnenen Erkenntnisse langfristig so fortsetzen oder sich in eine andere Richtung entwickeln. Dies betrifft insbesondere die Korrelation von Fehlern und Code Ownership, die Entwicklung der Expertise der beteiligten Personen und die Entwicklung des Code Ownerships im Allgemeinen. Eine Vermutung wäre beispielsweise, dass sich nach einer bestimmten Zeit ein stagnierender Wert des Code Ownerships in den Teilprojekten einstellt, oder sogar eine umgekehrte Entwicklung beobachtet werden kann. Dies könnte dadurch begründet sein, dass durch den Übergang von der Entwicklung der Software hin zum fortlaufenden Betrieb wieder klare Verantwortlichkeiten zugewiesen werden, da nur noch Änderungen in geringerem Umfang durchgeführt werden müssen. Weiterführend könnte untersucht werden, inwiefern sich die Expertise der einzelnen Entwickler:innen unter Einbeziehung der bereits vorhandenen Kenntnisse weiterentwickelt haben.

Zu letzterem Punkt könnte, wie im Kapitel zu Limitationen in der Datenanalyse erwähnt, eine Ergänzung der quantitativen Daten mit einer qualitativen Analyse sinnvoll sein, um die offen gebliebenen Fragen zu den Auswirkungen auf das Projekt zu beantworten. Zu diesen zählen beispielsweise die tatsächliche Entwicklung der Expertise einzelner Personen unter Einbeziehung der Vorkenntnisse, die persönliche Wahrnehmung der Umsetzung des Prinzips und dessen Erfolg sowie der Nutzen von Praktiken wie Paarprogrammierung oder Code Reviews. Zusätzlich könnten Erfahrungen in der Umsetzung und aufgetretene Herausforderungen aus persönlicher Sicht dargestellt werden.

Bei einer projektübergreifenden Analyse stellt sich die Frage, inwiefern diese aufgrund verschiedener Gegebenheiten und Anforderungen der Projekte überhaupt vergleichbar sind. Davon abgesehen könnte man anhand der dadurch gewonnenen Perspektiven eine differenziertere Antwort darauf geben, für welche Projekte beispielsweise ein bestimmtes Verantwortungsmodell besser funktioniert hat und welche Rahmenbedingungen dafür ausschlaggebend waren. Weitere Untersuchungen könnten sich auch damit befassen, ob es Möglichkeiten zur Variation des Prinzips gibt, welches die positiven Auswirkungen erhält und die negativen Auswirkungen, beispielsweise in Bezug auf die Qualität, aufhebt.

Quellenverzeichnis

Monografien

Beck, K. (2000). *Extreme Programming Explained*. Addison-Wesley.

Beck, K. (2005). *Extreme Programming Explained* (2. Aufl.). Addison-Wesley.

Booch, G. (1994). *Object-oriented analysis and design with applications* (2. Aufl.). Addison-Wesley.

Broy, M., & Kuhrmann, M. (2021). *Einführung in die Softwaretechnik*. Springer.

Chacon, S., & Straub, B. (2014). *Pro Git* (2. Aufl.). Apress.

Drucker, P. F. (2008). *Concept of the Corporation*. Transaction Publishers.

Hibbs, C., Jewett, S., & Sullivan, M. (2009). *The Art of Lean Software Development*. O'Reilly.

Hoffmann, D. W. (2013). *Software-Qualität*. Springer Vieweg.

Ludewig, J., & Lichter, H. (2013). *Software Engineering* (3. Aufl.). dpunkt.

Martin, R. C. (2020). *Clean Agile: Die Essenz der agilen Softwareentwicklung*. mitp.

McConnell, S. C. (2004). *Code Complete* (2. Aufl.). Microsoft Press.

Meyer, A. (2018). *Softwareentwicklung*. De Gruyter.

Monden, Y. (2012). *Toyota Production System: An Integrated Approach to Just-In-Time* (4. Aufl.). Taylor & Francis.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development*. Addison-Wesley.

Poppendieck, M., & Poppendieck, T. (2007). *Implementing Lean Software Development*. Addison-Wesley.

Pressman, D. R. S. (2001). *Software Engineering*. McGraw-Hill.

Sandhaus, G., Berg, B., & Knott, P. (2014). *Hybride Softwareentwicklung*. Springer Vieweg.

Schmidt, C. (2016). *Agile Software Development Teams*. Springer.

Schwaber, K. (2007). *The Enterprise and Scrum*. Microsoft Press.

Sommerville, I. (2016). *Software Engineering* (10. Aufl.). Pearson.

Stober, T., & Hansmann, U. (2010). *Agile Software Development*. Springer.

Weinberg, G. M. (1971). *The Psychology of Computer Programming*. Litton Educational Publishing.

Sammelwerke

Alpar, P., Alt, R., Bensberg, F., & Weimann, P. (2019). Phasenmodelle in der Softwareentwicklung. In *Anwendungsorientierte Wirtschaftsinformatik* (S. 347–402). Springer Vieweg.

Fachartikel

Arisholm, E., Gallis, H., Dybå, T., & Sjøberg, D. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions on Software Engineering*, *33*, 65–86.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77.

Boban, M., Pozgaj, Z., & Sertić, H. (2003). Strategies for successful software development risk management. *Management*, *8*(2), 77–91.

Counsell, S., Eldh, S., & Murphy, B. (2015). Code Ownership Perspectives. *IEEE Software*, *32*(6), 18–19.

Drucker, P. F. (1963). Managing for Business Effectiveness. *Harvard Business Review*, *Mai 1963*, 53–60.

Faraj, S., & Sproull, L. (2000). Coordinating Expertise in Software Development Teams. *Management Science*, *46*(12), 1554–1568.

Highsmith, J. (2002). What Is Agile Software Development? *Crosstalk: Journal of Defense Software Engineering*, *Oktober 2002*, 4–9.

Jeffries, R., & Melnik, G. (2007). Guest Editors' Introduction: TDD - The Art of Fearless Programming. *IEEE Software*, *24*, 24–30.

Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management*, *21*, 41–52.

Maier, M., Emery, D., & Hilliard, R. (2001). Software architecture: introducing IEEE Standard 1471. *Computer*, *34*(4), 107–109.

Maruping, L., Zhang, X., & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, *18*, 355–371.

Nordberg III, M. E. (2003). Managing Code Ownership. *IEEE Software*, *20*(2), 26–33.

Rus, I., & Lindvall, M. (2002). Knowledge Management in Software Engineering. *IEEE Software*, *19*(3), 26–38.

Sadiku, M. N., Musa, S. M., & Momoh, O. D. (2014). Cloud Computing: Opportunities and Challenges. *IEEE Potentials*, *33*(1), 34–36.

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard Business Review*, *Januar-Februar 1986*, 137–146.

Ward, S., & Chapman, C. (2003). Transforming project risk management into project uncertainty management. *International Journal of Project Management*, *21*, 97–105.

Weyuker, E., Ostrand, T., & Bell, R. (2008). Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, *13*, 539–559.

Williams, L., R. Kessler, R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair-Programming. *IEEE Software*, (July/August 2000), 19–25.

Standards

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*.

IEEE. (2014). IEEE Standard for Software Quality Assurance Processes. *IEEE Std 730-2014 (Revision of IEEE Std 730-2002)*.

ISO. (2011). Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. *ISO 25010:2011*.

Konferenzbeiträge

Ahmed, W., & Colomo-Palacios, R. (2021). Team Topologies in Software Teams: A Multivocal Literature Review. *Computational Science and Its Applications – ICCSA 2021*, 272–282.

Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2011). Don't Touch My Code! Examining the Effects of Ownership on Software Quality. *Proceedings of the the eighth joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 4–14.

Dönmez, D., & Grote, G. (2015). The Two Faces of Uncertainty: Threat vs Opportunity Management in Agile Software Development. *Agile Processes in Software Engineering and Extreme Programming*, 4–14.

Dorairaj, S., Noble, J., & Malik, P. (2012). Knowledge Management in Distributed Agile Software Development. *2012 Agile Conference*, 64–73.

Levy, M., & Hazzan, O. (2009). Knowledge management in practice: The case of agile software development. *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 60–65.

Mockus, A., & Herbsleb, J. (2002). Expertise Browser: A Quantitative Approach to Identifying Expertise. *Proceedings of the 24th International Conference on Software Engineering*, 503–512.

Morales, R., McIntosh, S., & Khomh, F. (2015). Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings*, 171–180.

Paetsch, F., Eberlein, D. A., & Maurer, D. F. (2003). Requirements Engineering and Agile Software Development. *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*, 308–313.

Rahman, F., & Devanbu, P. (2011). Ownership, Experience and Defects: a fine-grained study of Authorship. *Proceedings - International Conference on Software Engineering*, 491–500.

Sedano, T., Ralph, P., & Péraire, C. (2016). Practice and Perception of Team Code Ownership. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, (36), 1–6.

Internetquellen

Amazon Web Services. (2021a). *AWS CloudFormation*. Abgerufen am 6. März 2022 unter https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html.

Amazon Web Services. (2021b). *AWS DynamoDB*. Abgerufen am 3. April 2022 unter https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html.

Amazon Web Services. (2022). *AWS CloudWatch*. Abgerufen am 3. April 2022 unter https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

Amazon Web Services. (o. D.). *Cloud Computing mit AWS*. Abgerufen am 5. März 2022 unter https://aws.amazon.com/de/what-is-aws/.

Atlassian. (o. D.). *Jira Software*. Abgerufen am 4. März 2022 unter https://www.atlassian.com/de/software/jira.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001a). *Manifesto for Agile Software Development*. Abgerufen am 27. November 2021 unter https://agilemanifesto.org/iso/de/manifesto.html.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001b). *Principles behind the Agile Manifesto*. Abgerufen am 27. November 2021 unter https://agilemanifesto.org/principles.html.

Blosen, B., West, M., Wan, D. D., Sklavounakis, A., Mann, K., Chan, W. F., & Ennaciri, H. (2021). *Magic Quadrant for Enterprise Agile Planning Tools*. Abgerufen am 5. Dezember 2021 unter https://www.gartner.com/doc/reprints?id=1-25SRWZ04&ct=210414.

Chappell, D. (2013). *The three Aspects of Software Quality*. Abgerufen am 4. Januar 2022 unter http://www.davidchappell.com/writing/white_papers/The_Three_Aspects_of_Software_Quality_v1.0-Chappell.pdf.

CollabNet-VersionOne. (2018). *12th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/versionone-12th-annual-state-of-agile-report.pdf.

CollabNet-VersionOne. (2019). *13th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/13th-annual-state-of-agile-report.pdf.

Digital.ai. (2020). *14th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/14th-annual-state-of-agile-report.pdf.

Digital.ai. (2021). *15th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://digital.ai/resource-center/analyst-reports/state-of-agile-report.

Docker. (2022). *Docker overview*. Abgerufen am 3. April 2022 unter https://docs.docker.com/get-started/overview/.

Fowler, M. (2006). *CodeOwnership*. Abgerufen am 23. Dezember 2021 unter https://martinfowler.com/bliki/CodeOwnership.html.

GitLab. (o. D.). *GitLab DevOps Plattform*. Abgerufen am 4. März 2022 unter https://about.gitlab.com/de-de/.

HashiCorp. (o. D.). *Terraform*. Abgerufen am 6. März 2022 unter https://www.terraform.io/.

Highsmith, J. (2001). *History: The Agile Manifesto*. Abgerufen am 27. November 2021 unter https://agilemanifesto.org/history.html.

IEEE. (o. D.). *IEEE at a Glance*. Abgerufen am 9. März 2022 unter https://www.ieee.org/about/at-a-glance.html.

IEEE Computer Society. (2014). *Guide to the Software Engineering Body of Knowledge, Version 3.0*, abgerufen am 10. März 2022 unter https://www.computer.org/education/bodies-of-knowledge/software-engineering.

ISO. (2019). *ISO in brief*. Abgerufen am 3. April 2022 unter https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100007.pdf.

Jeffries, R., & Beck, K. (2004). *Extreme Programming Code Reviews*. Abgerufen am 22. Dezember 2021 unter http://wiki.c2.com/?ExtremeProgrammingCodeReviews.

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. Abgerufen am 5. März 2022 unter https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

Metz, C. (2015). *Google Is 2 Billion Lines of Code – And It's All in One Place*. Abgerufen am 29. März 2022 unter https://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/.

Microsoft. (2022). *Was ist Azure Resource Manager?* Abgerufen am 6. März 2022 unter https://docs.microsoft.com/de-de/azure/azure-resource-manager/management/overview.

Microsoft. (o. D.). *Was ist Azure?* Abgerufen am 5. März 2022 unter https://azure.microsoft.com/de-de/overview/what-is-azure/.

Python.org. (o. D.). *What is Python? Executive Summary*. Abgerufen am 3. April 2022 unter https://www.python.org/doc/essays/blurb/.

Rimol, M. (2022). *12 Essential Skills for Agile Developers*. Abgerufen am 30. März 2022 unter https://www.gartner.com/en/articles/12-essential-skills-for-agile-developers.

Schwaber, K. (1995). *SCRUM Development Process*. Abgerufen am 16. Dezember 2021 unter http://www.jeffsutherland.org/oopsla/schwapub.pdf.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Abgerufen am 16. Dezember 2021 unter https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

Smith, D. (2005). *Code Stewardship*. Abgerufen am 23. Dezember 2021 unter https://wiki.c2.com/?CodeStewardship.

Stackoverflow.com. (2021). *2021 Developer Survey*. Abgerufen am 28. Dezember 2021 unter https://insights.stackoverflow.com/survey/2021.

Statista Research Department. (2022). *Marktanteile der führenden Unternehmen am Umsatz im Bereich Cloud Computing weltweit im 4. Quartal 2021*. Abgerufen am 5. März 2022 unter https://de.statista.com/statistik/daten/studie/150979/umfrage/marktanteile-der-fuehrenden-unternehmen-im-bereich-cloud-computing/.

Sutherland, J. (2021). *The Scrum Papers*. Abgerufen am 12. Dezember 2021 unter http://jeffsutherland.org/scrum/scrumpapers.pdf.

TypeScript. (2021). *TypeScript*. Abgerufen am 3. April 2022 unter https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

VersionOne. (2007a). *1st Survey: The State of Agile Development*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/1st-annual-state-of-agile-report.pdf.

VersionOne. (2007b). *2nd Annual Survey: The State of Agile Development*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/2nd-annual-state-of-agile-report.pdf.

VersionOne. (2008). *3rd Annual Survey 2008: The State of Agile Development*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/3rd-annual-state-of-agile-report.pdf.

VersionOne. (2009). *State of Agile Survey: 2009*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/4th-annual-state-of-agile-report.pdf.

VersionOne. (2010). *State of Agile Survey: 2010*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/5th-annual-state-of-agile-report.pdf.

VersionOne. (2011). *State of Agile Survey: 2011*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/6th-annual-state-of-agile-report.pdf.

VersionOne. (2013). *7th Annual State of Agile Development Survey*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/7th-Annual-State-of-Agile-Development-Survey.pdf.

VersionOne. (2014). *8th Annual State of Agile Survey*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/8th-annual-state-of-agile-report.pdf.

VersionOne. (2015a). *10th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/versionone-10th-annual-state-of-agile-report.pdf.

VersionOne. (2015b). *9th Annual State of Agile Survey*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/state-of-agile-development-survey-ninth.pdf.

VersionOne. (2017). *11th Annual State of Agile Report*. Abgerufen am 5. Dezember 2021 unter https://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/versionone-11th-annual-state-of-agile-report.pdf.

Vue.js. (2022). *Introduction*. Abgerufen am 3. April 2022 unter https://vuejs.org/guide/introduction.html.

Wells, D. (1999). *Collective Ownership*. Abgerufen am 16. Dezember 2021 unter http://www.extremeprogramming.org/rules/collective.html.

Anhang

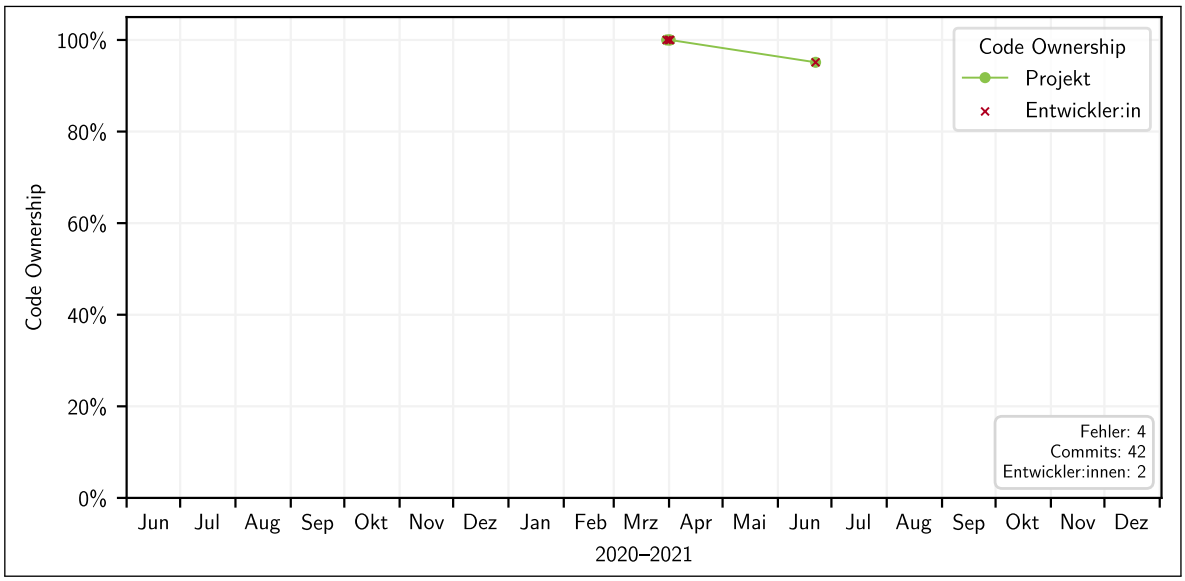
Anhangsverzeichnis

Anhang 1 Datenanalysen im Detail

Im Folgenden sind die als Ausschnitte in Abschnitt 6.2 enthaltenen, mehrteiligen Datenanalysen in ihrer Gesamtheit dargestellt. Um die Vergleichbarkeit zu gewährleisten, wurden die im Haupttext enthaltenen Darstellungen nochmals aufgegriffen.

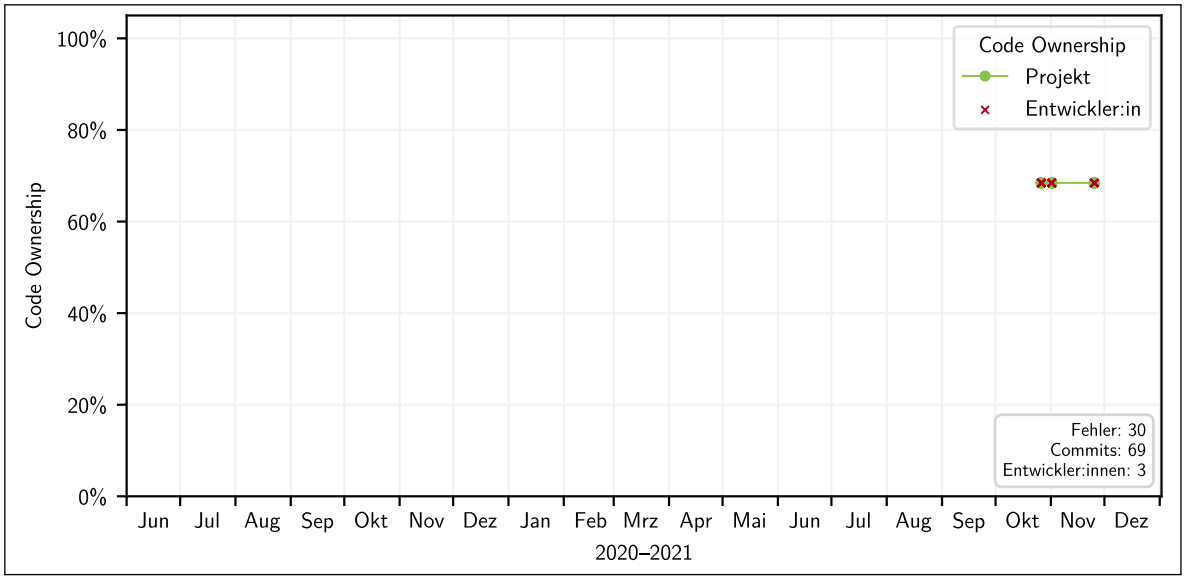
Anhang 1.1 Zu Abschnitt 6.2.1: Fehleranfälligkeit der Software

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 3)



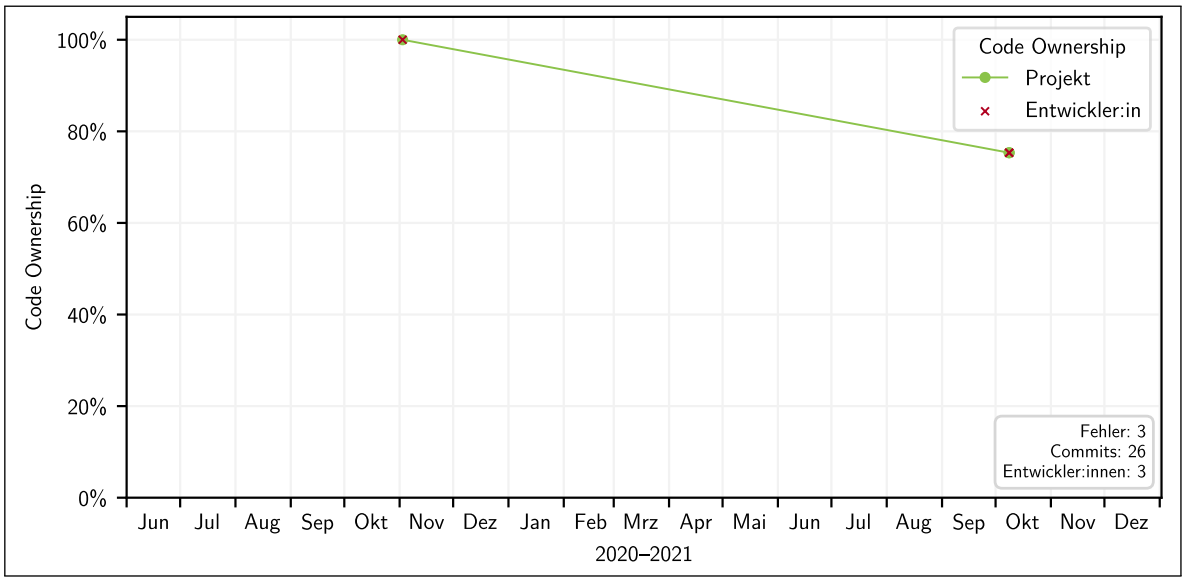
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 5)



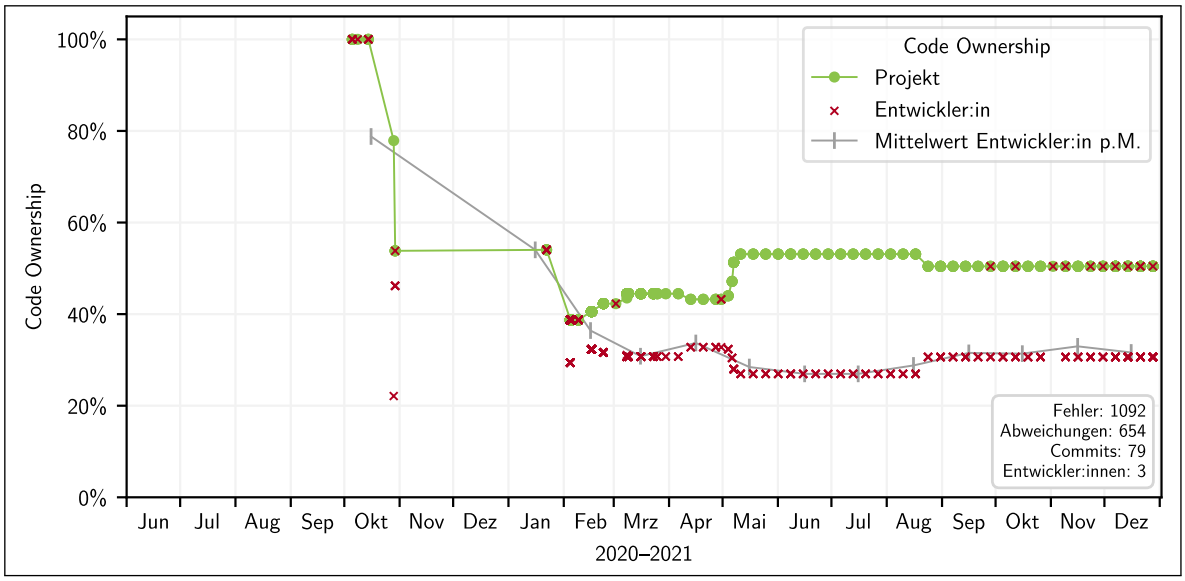
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 7)



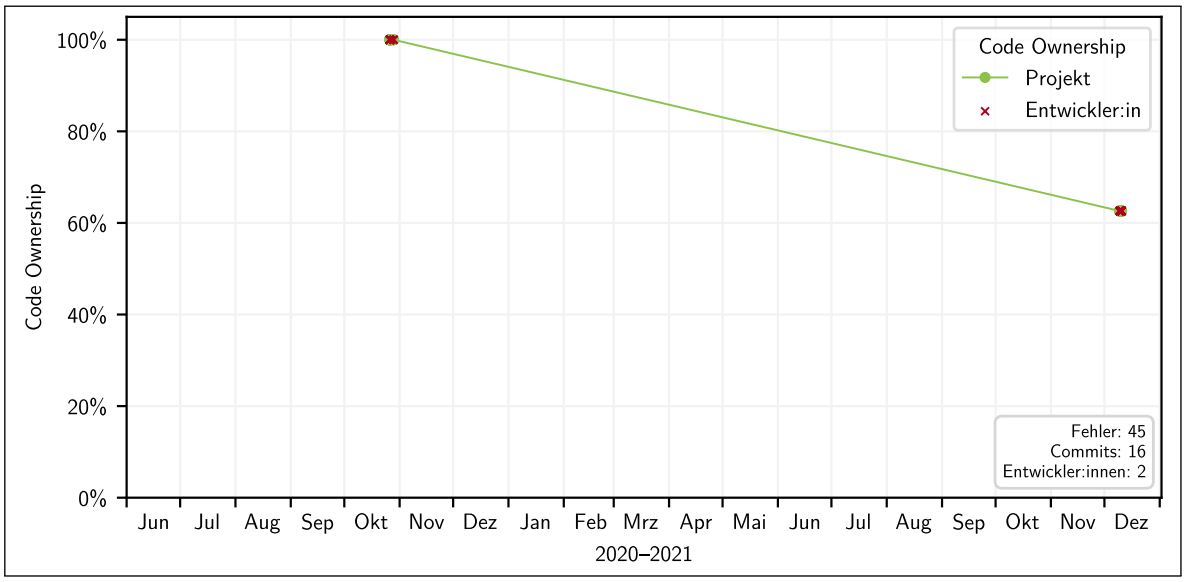
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 8)



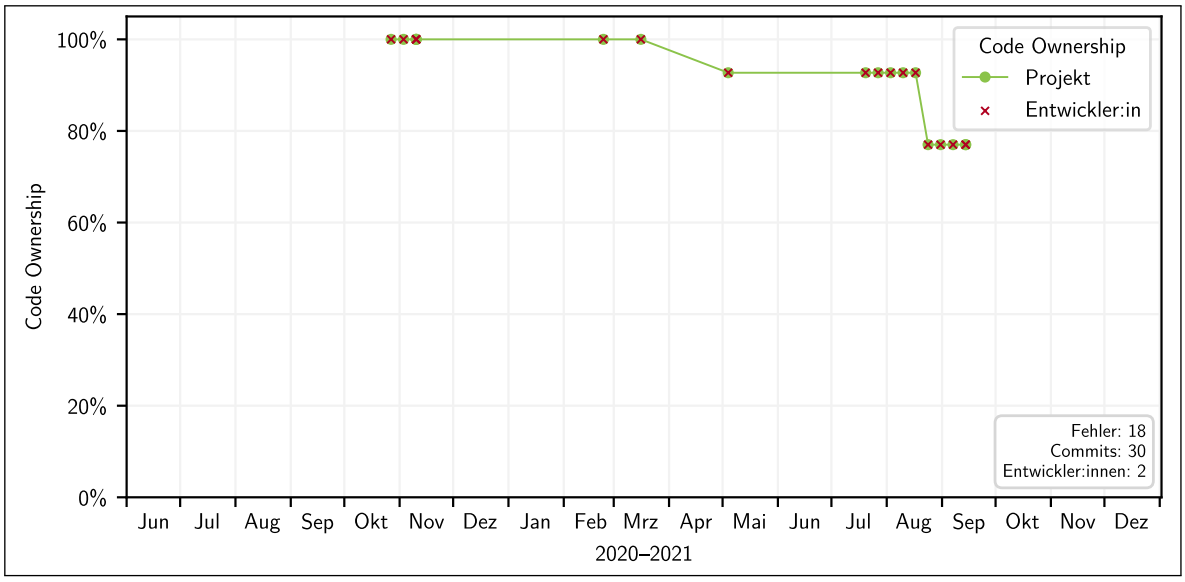
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 9)



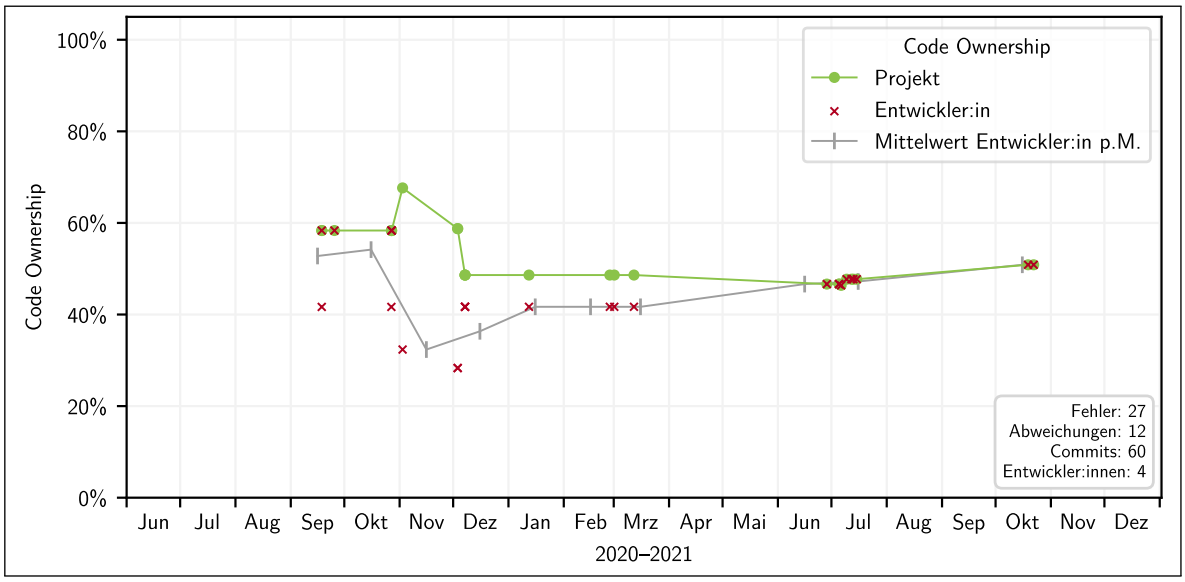
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 10)



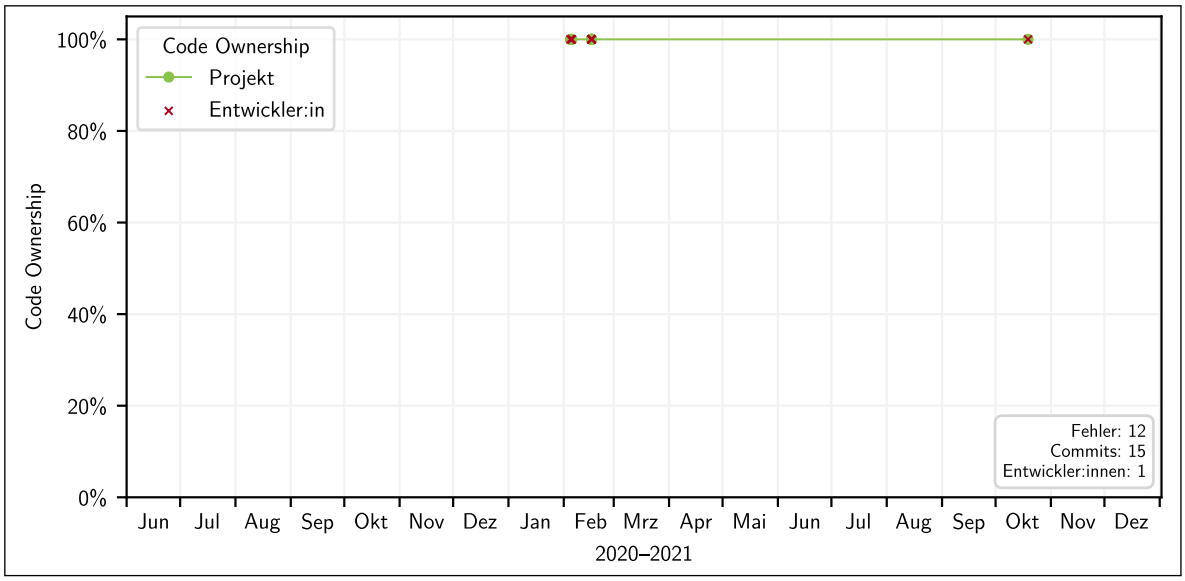
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 11)



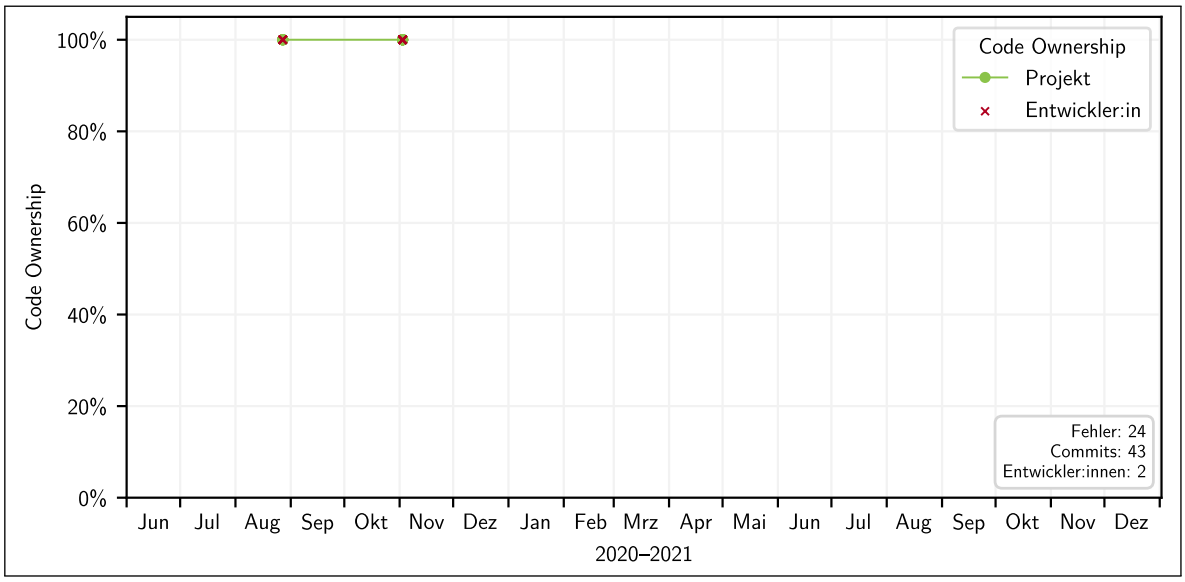
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 12)



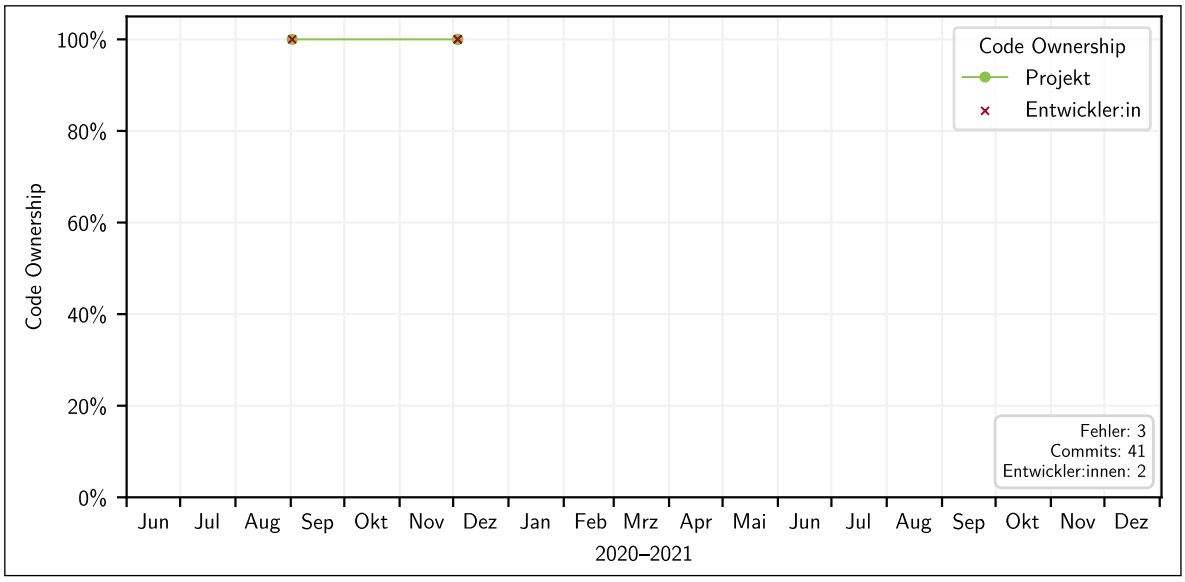
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 14)



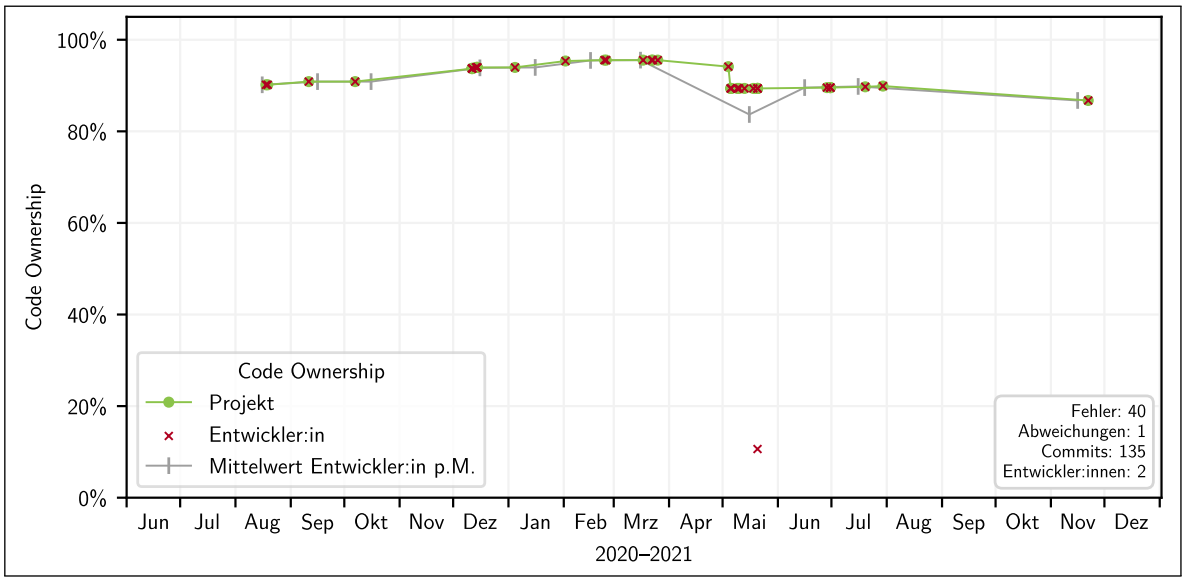
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 15)



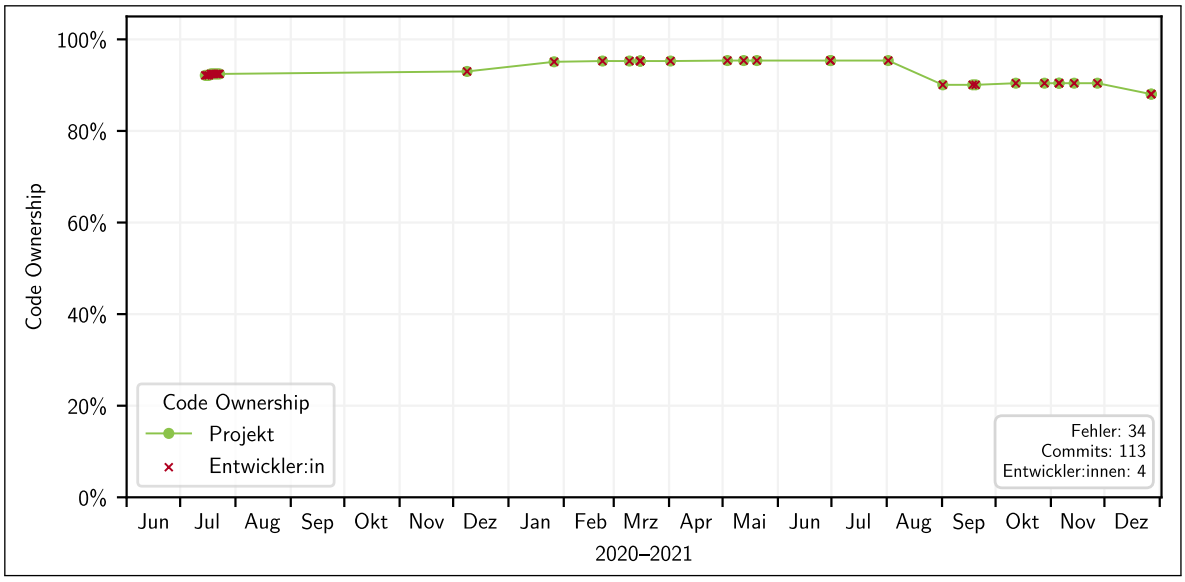
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 16)



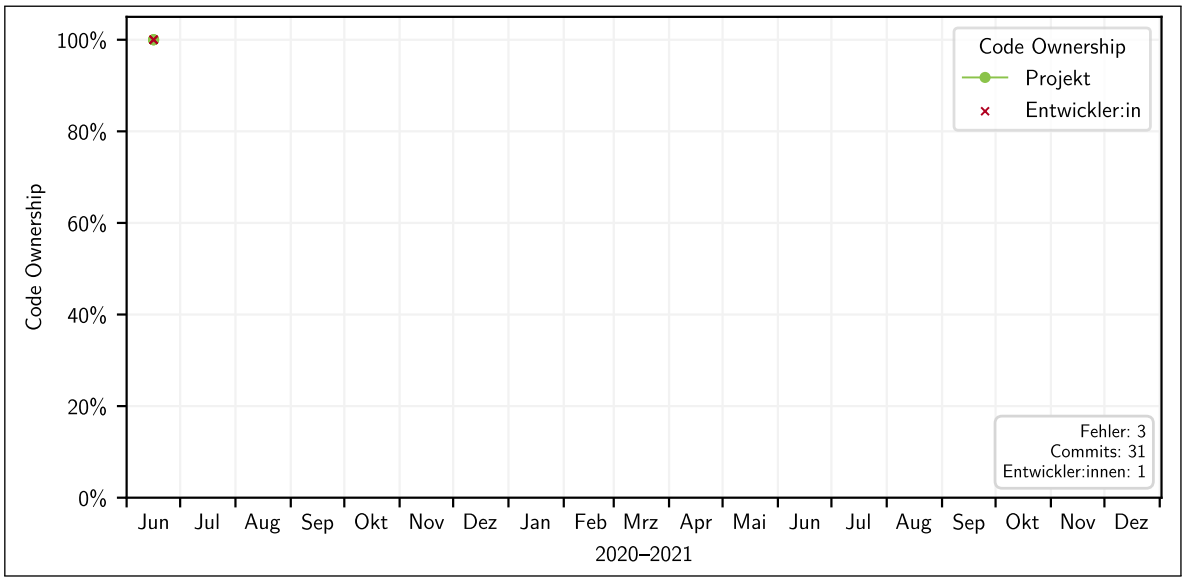
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 17)



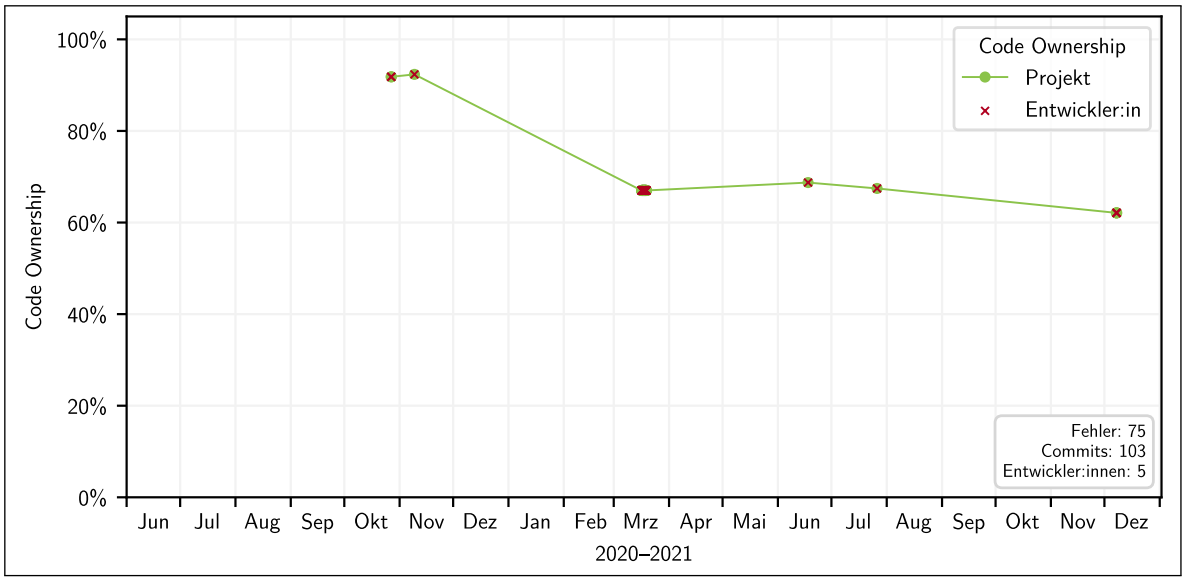
Quelle: Eigene Darstellung.

Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 18)



Quelle: Eigene Darstellung.

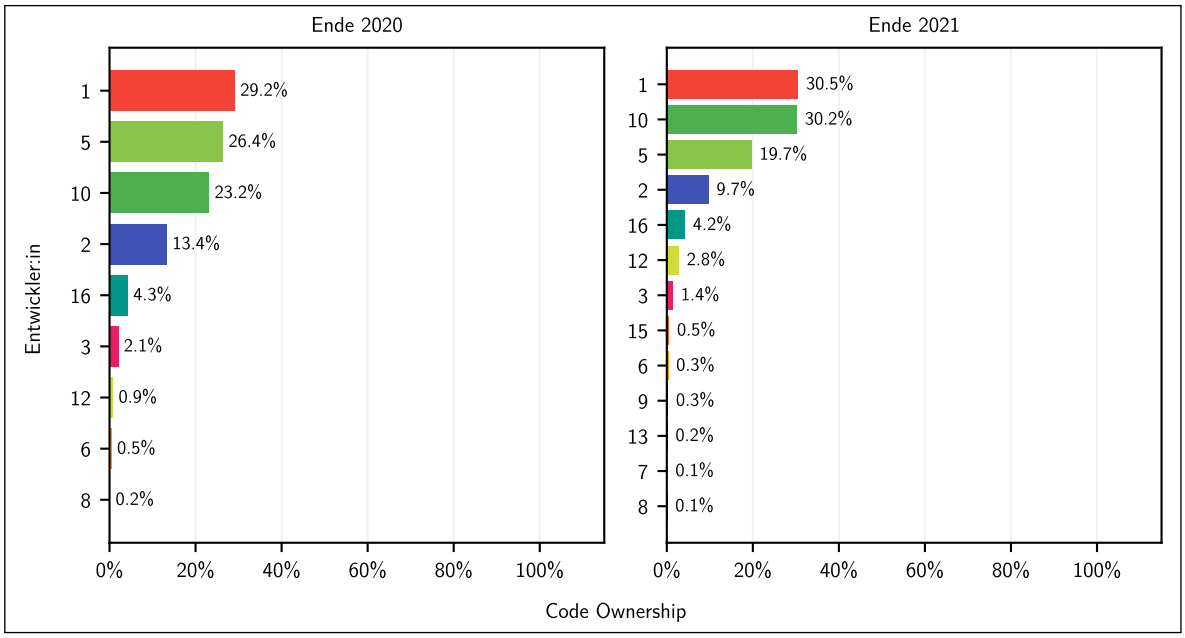
Abbildung: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 19)



Quelle: Eigene Darstellung.

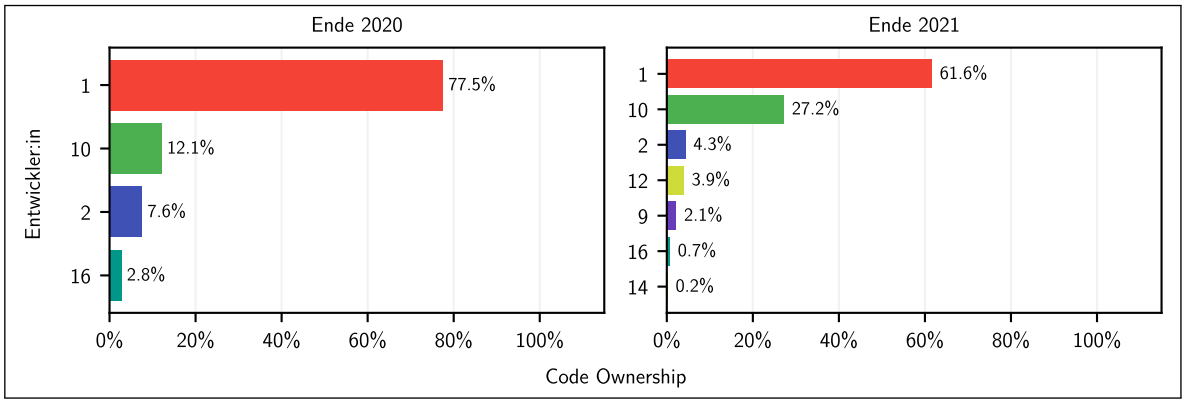
Anhang 1.2 Zu Abschnitt 6.2.2: Entwicklungsrisiken

Abbildung: Technologiebezogene Abhängigkeiten (Terraform)



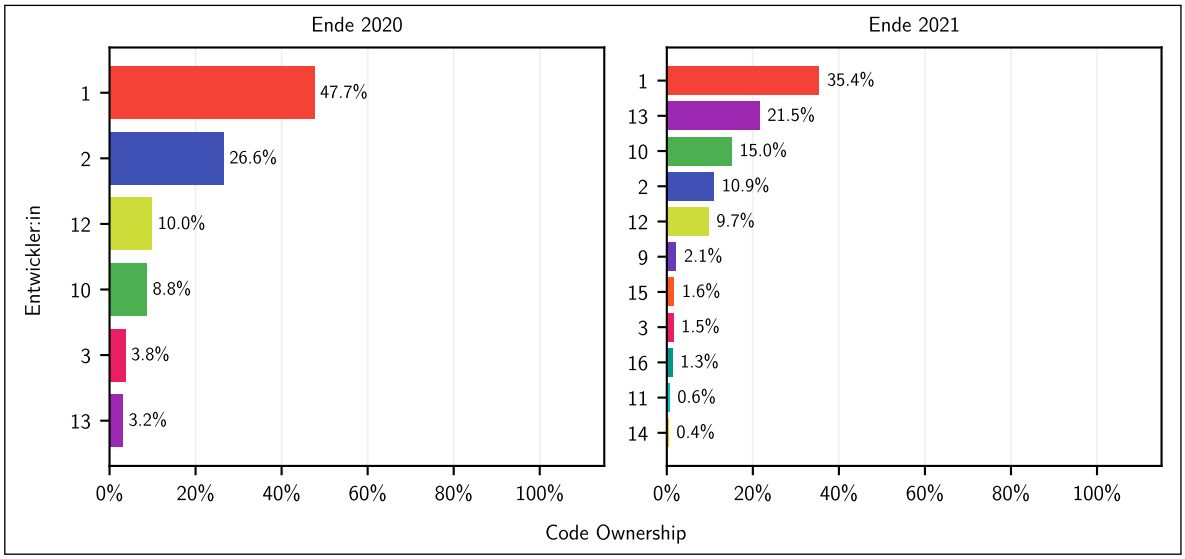
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (DynamoDB)



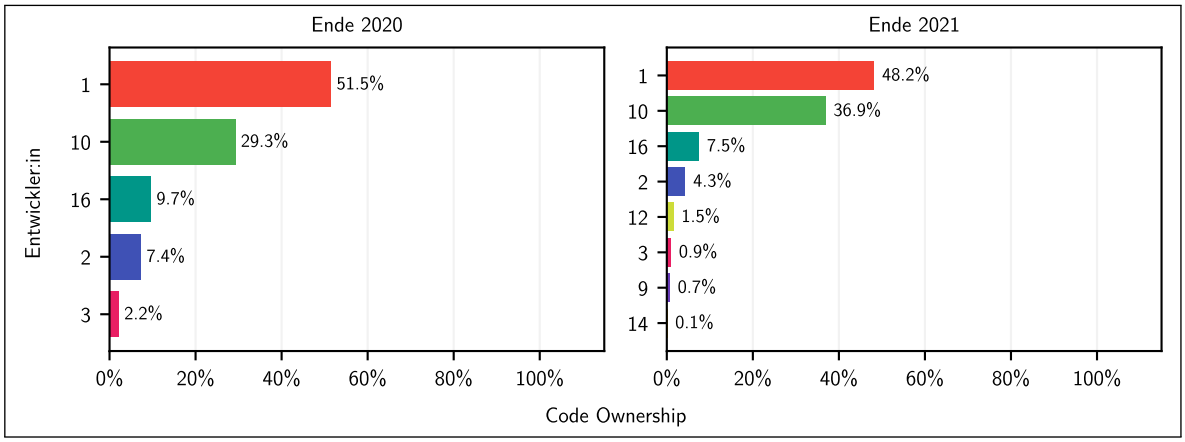
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (TypeScript)



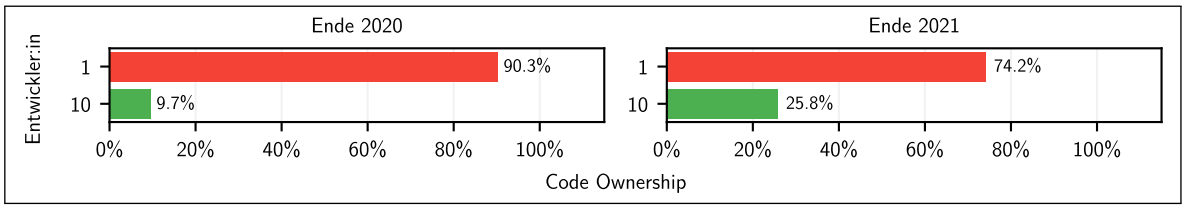
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (Python)



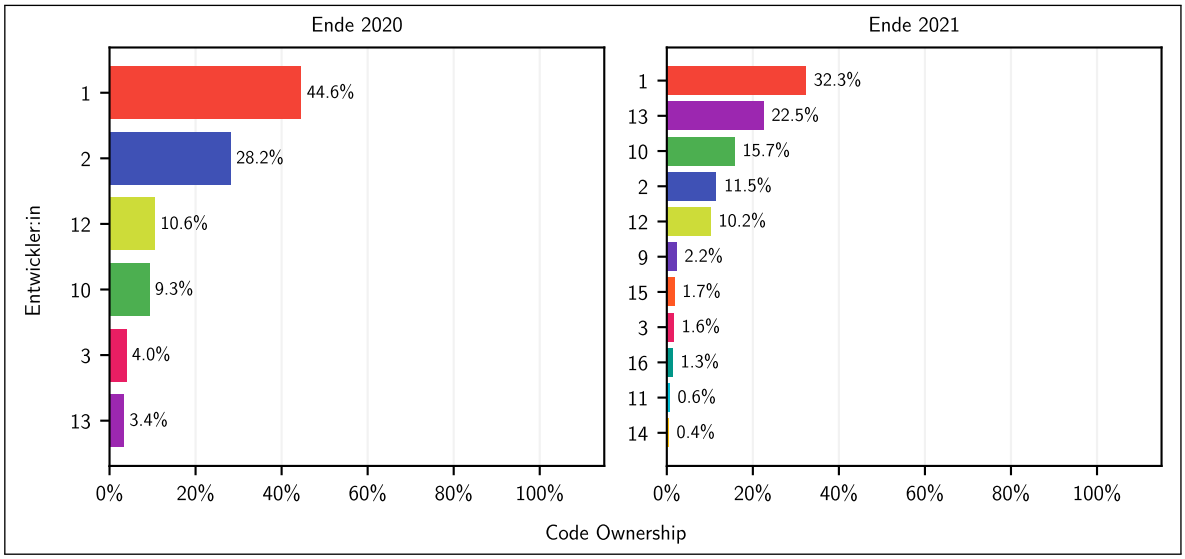
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (Docker)



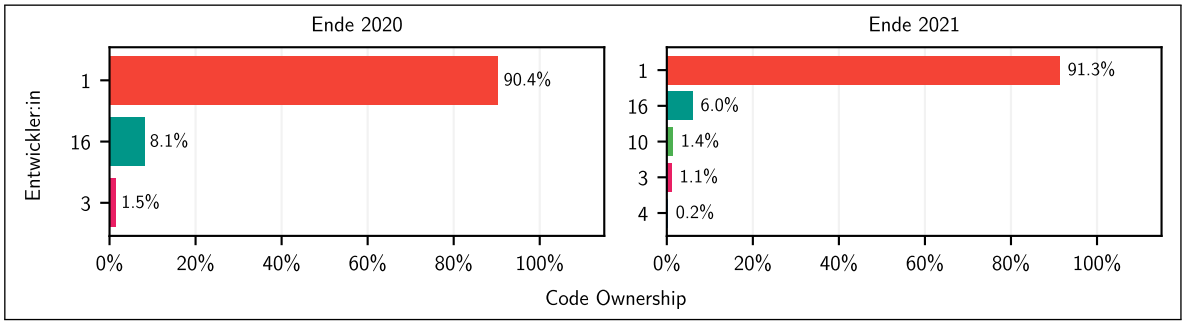
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (Vue.js)



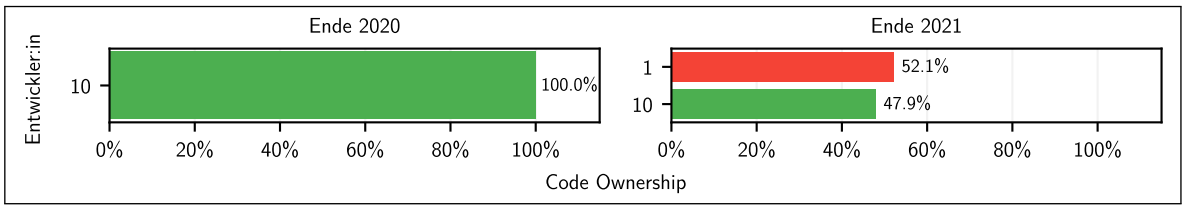
Quelle: Eigene Darstellung.

Abbildung: Technologiebezogene Abhängigkeiten (CloudFormation)



Quelle: Eigene Darstellung.

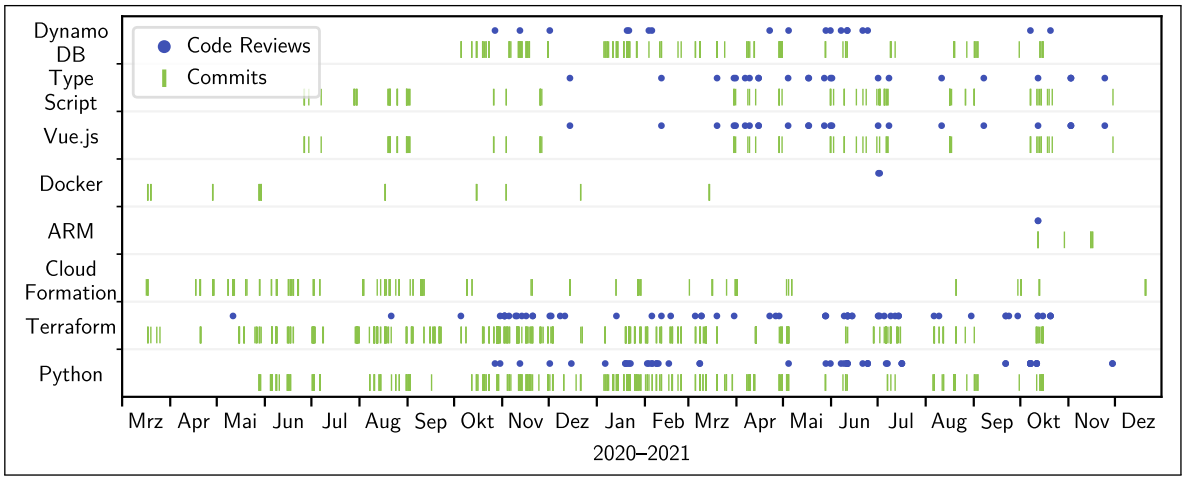
Abbildung: Technologiebezogene Abhängigkeiten (ARM)



Quelle: Eigene Darstellung.

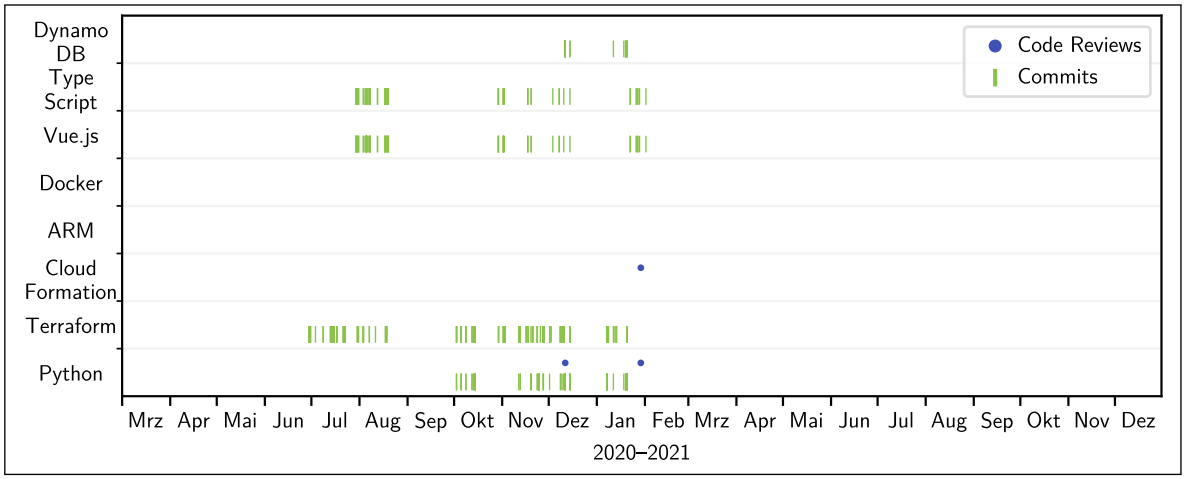
Anhang 1.3 Zu Abschnitt 6.2.3: Expertise der Entwickler:innen

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 1)



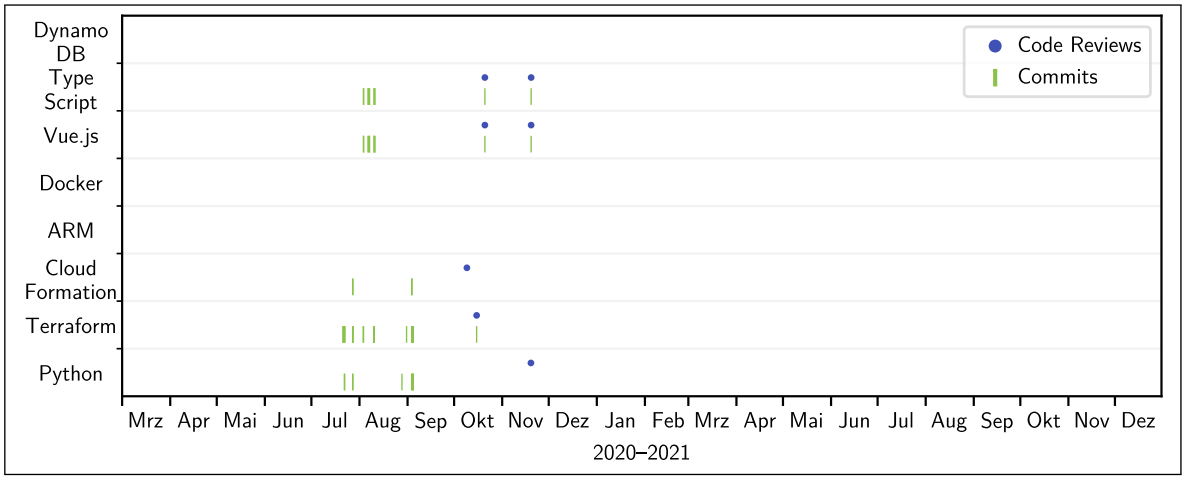
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 2)



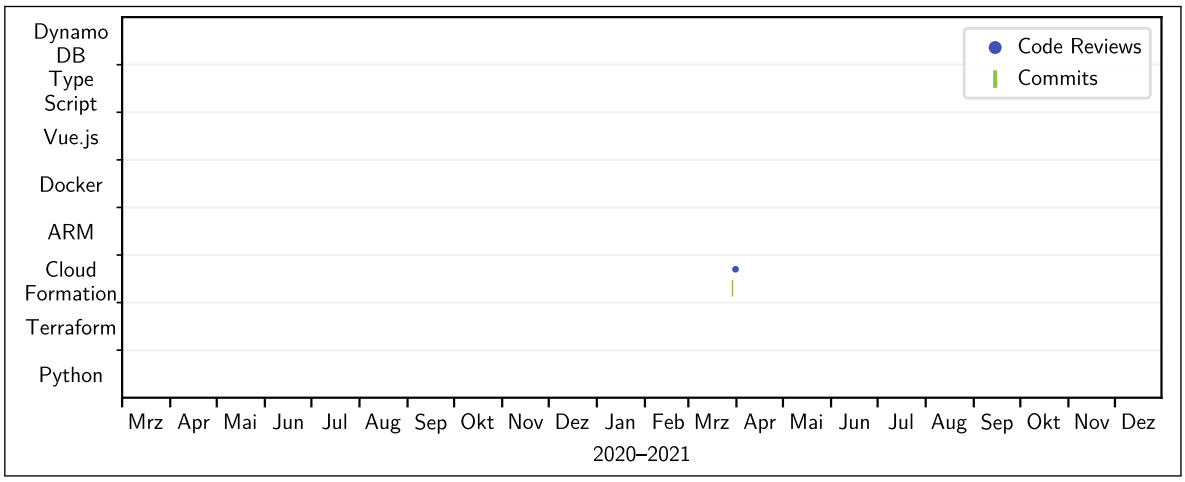
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 3)



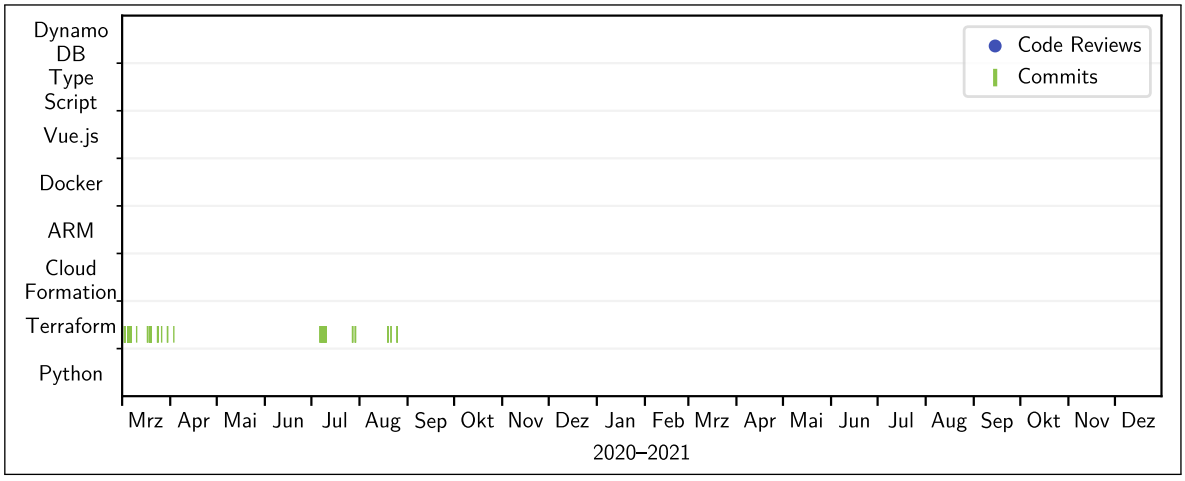
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 4)



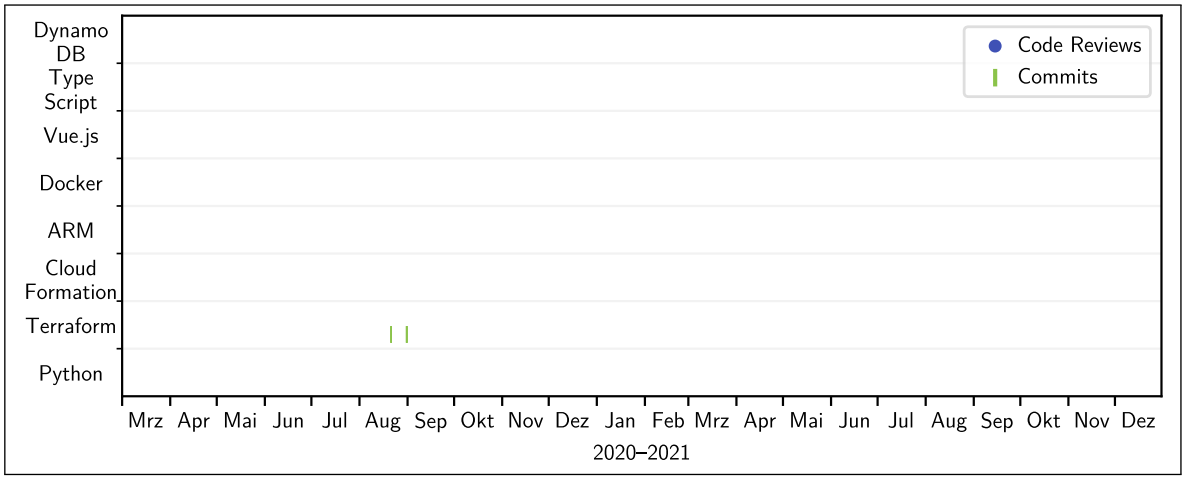
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 5)



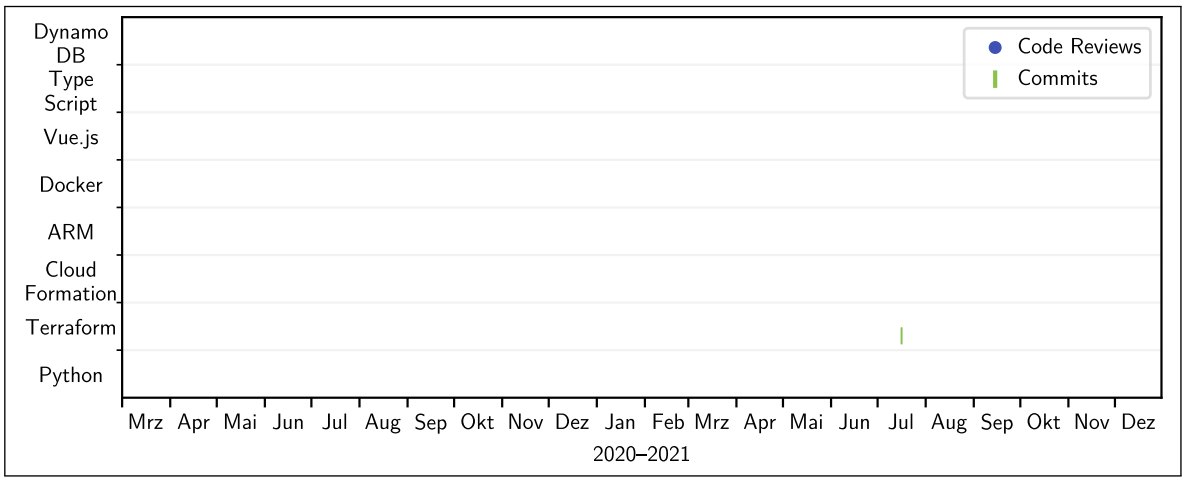
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 6)



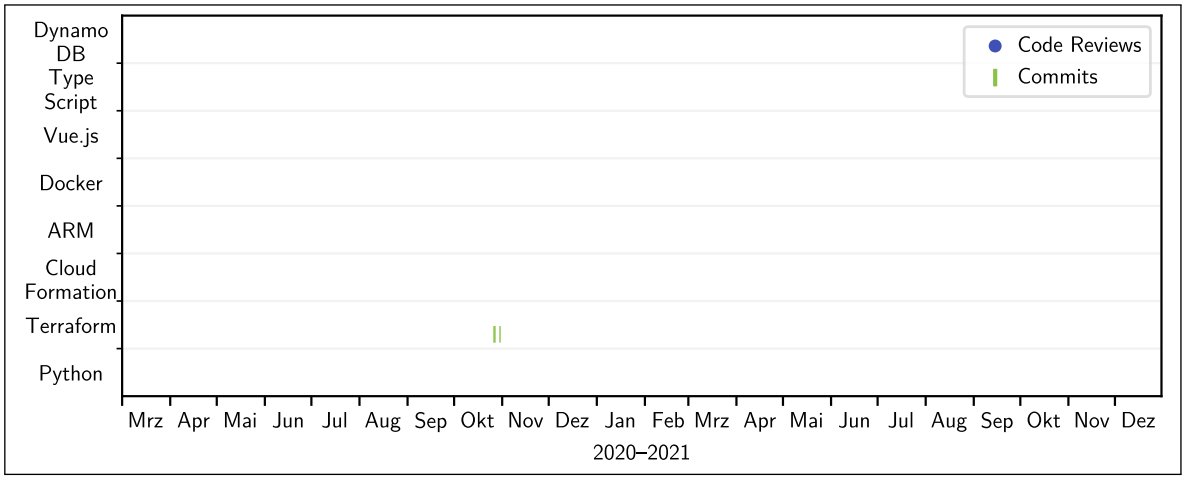
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 7)



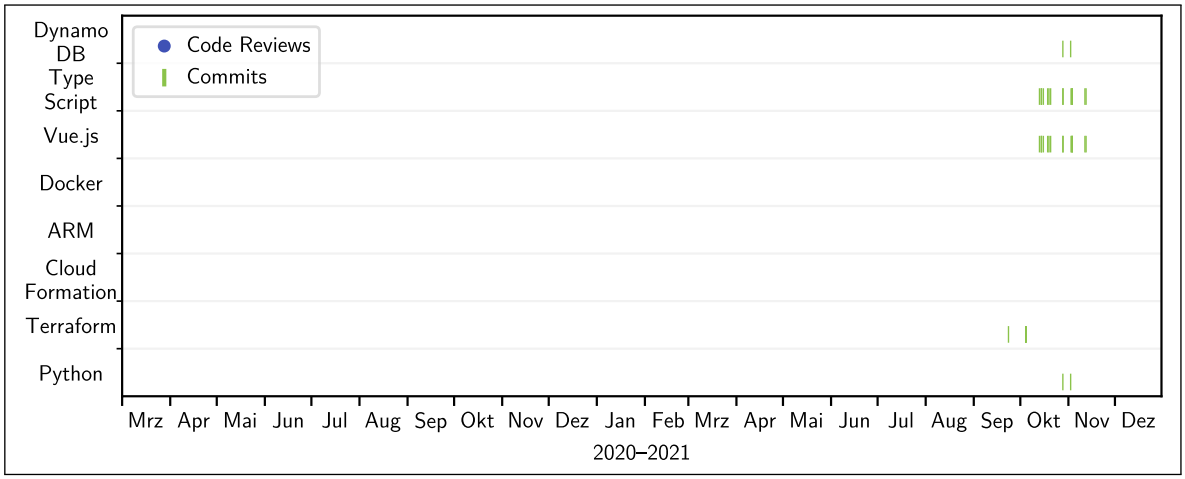
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 8)



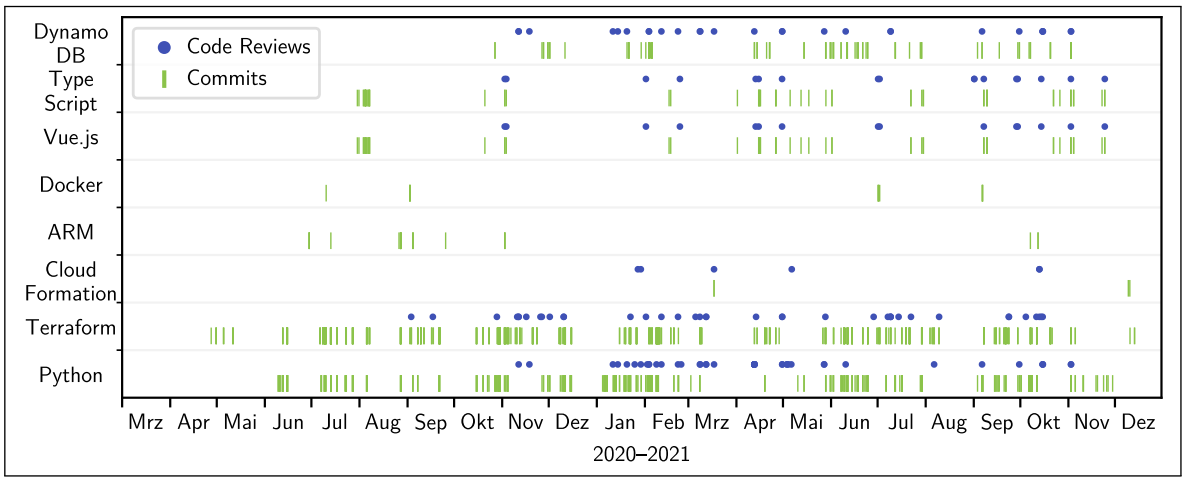
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 9)



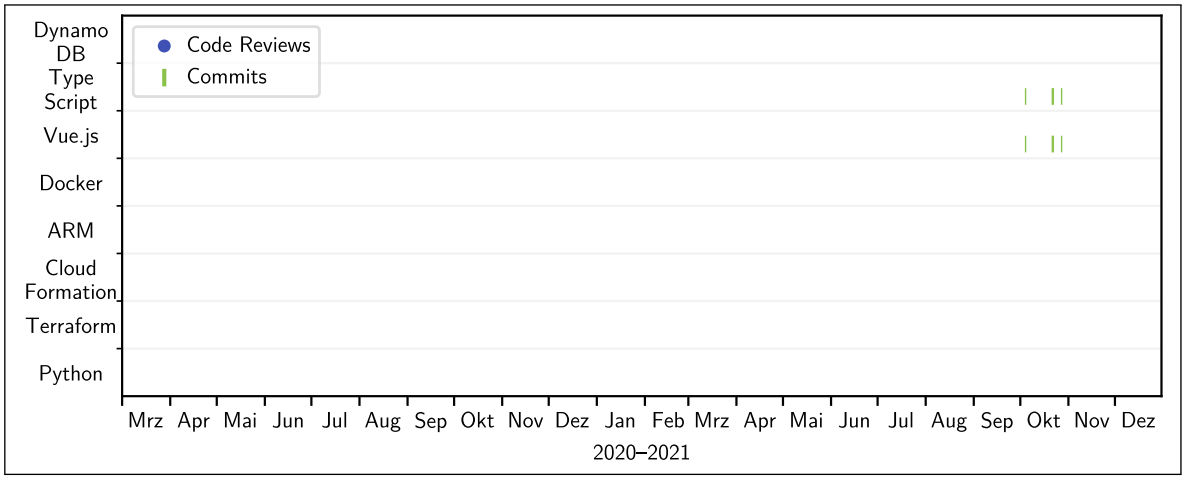
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 10)



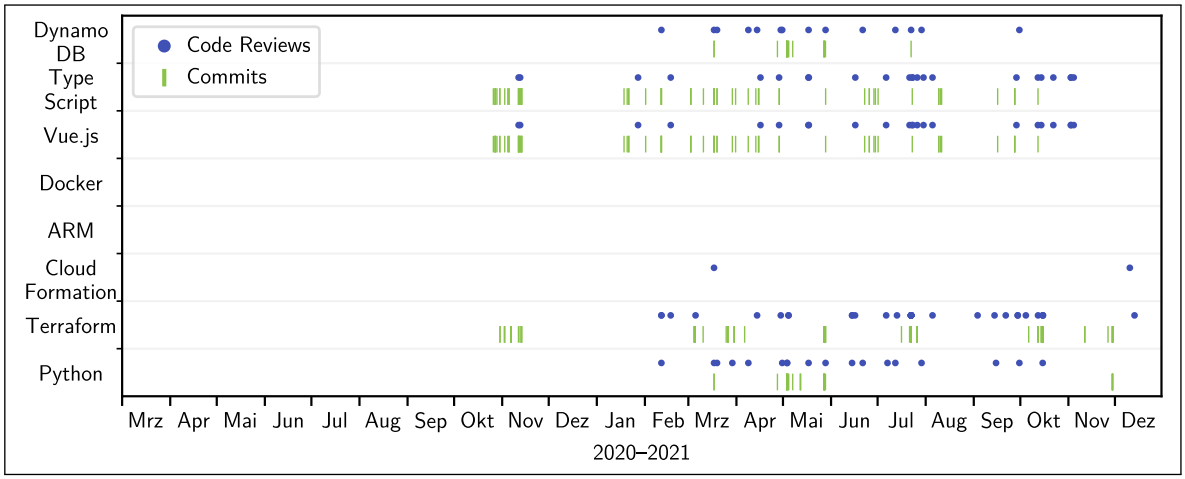
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 11)



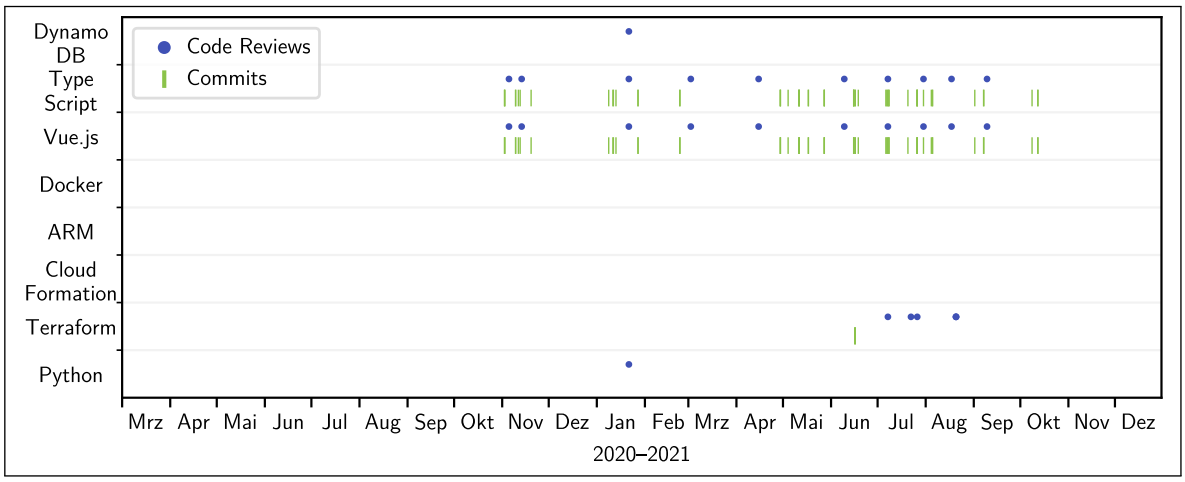
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 12)



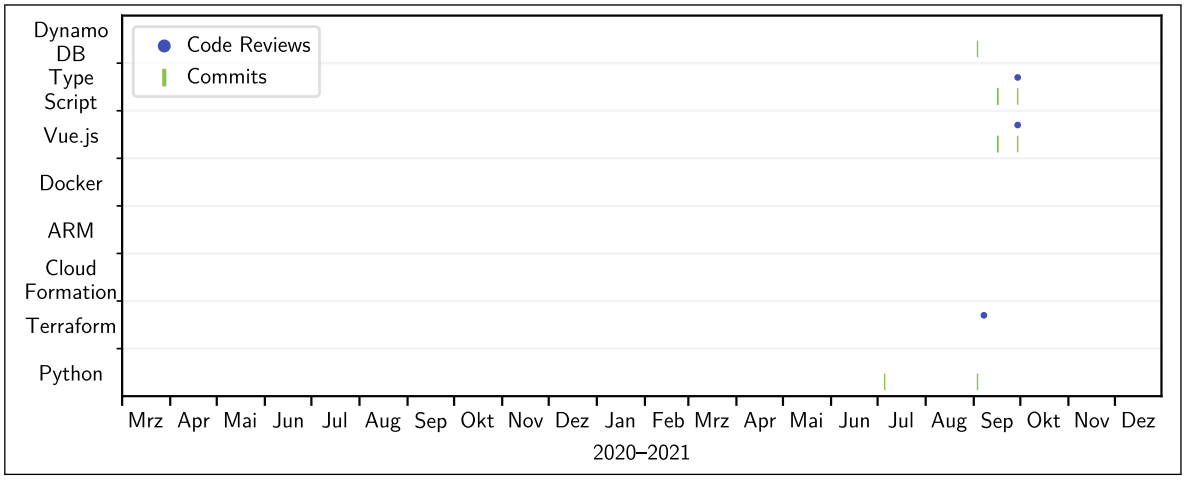
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 13)



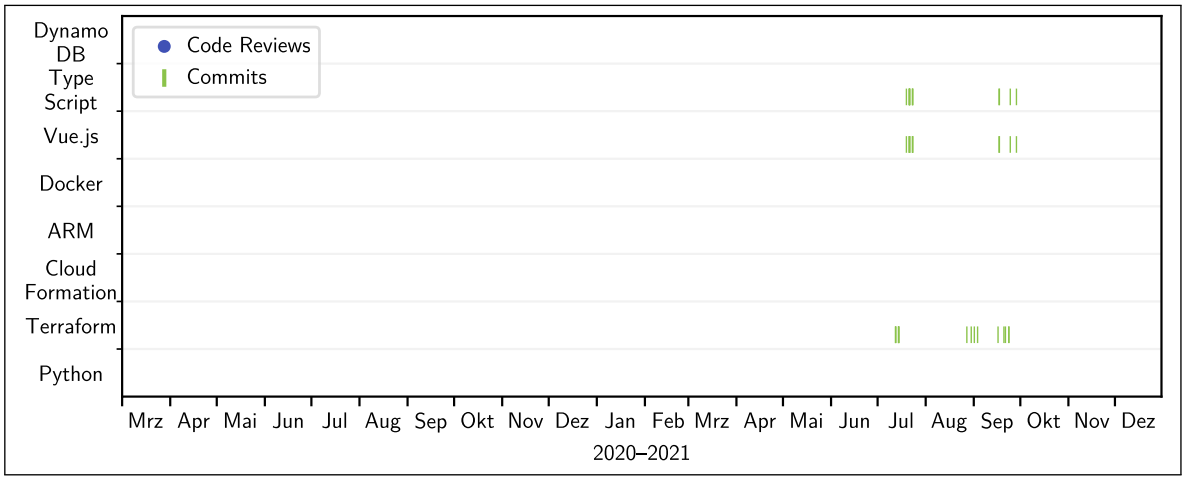
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 14)



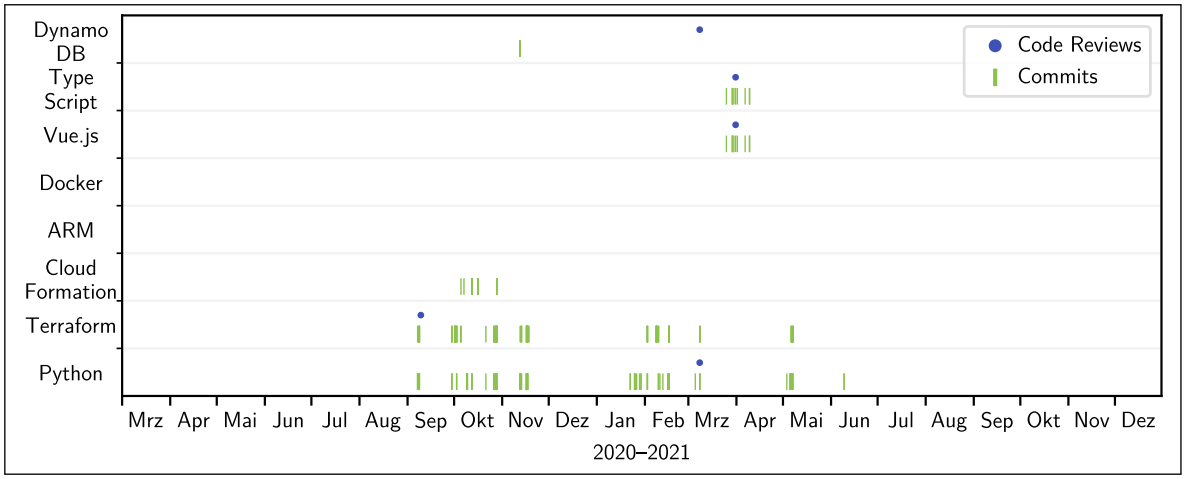
Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 15)



Quelle: Eigene Darstellung.

Abbildung: Entwicklungsaktivität und Code Reviews (Entwickler:in 16)



Quelle: Eigene Darstellung.

Anhang 2 Abbildungen in größerem Maßstab

Anhang 2.1 Zu Abbildung 8: Entwicklungsaktivität

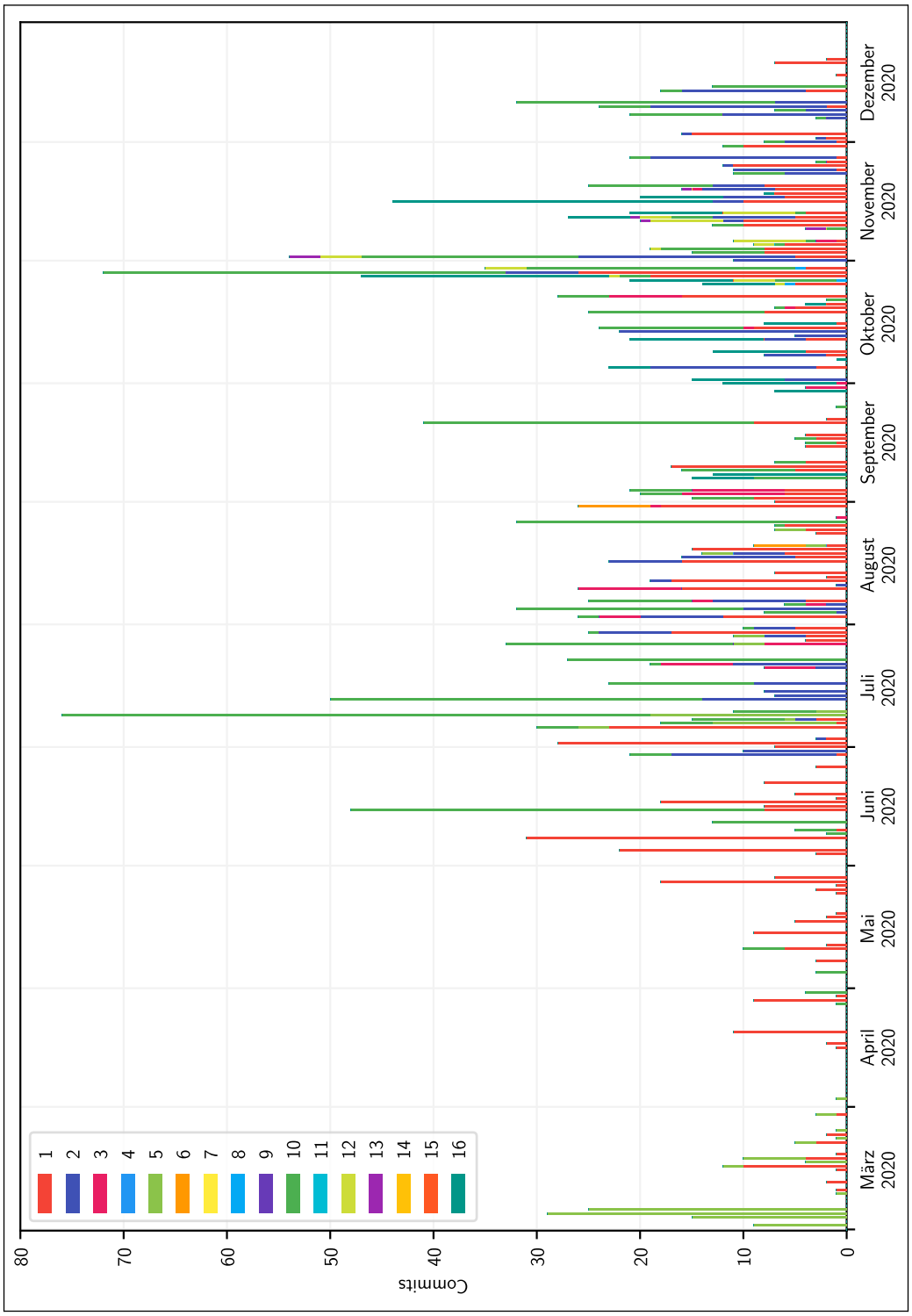


Abbildung: Entwicklungsaktivität (2020)

Quelle: Eigene Darstellung.

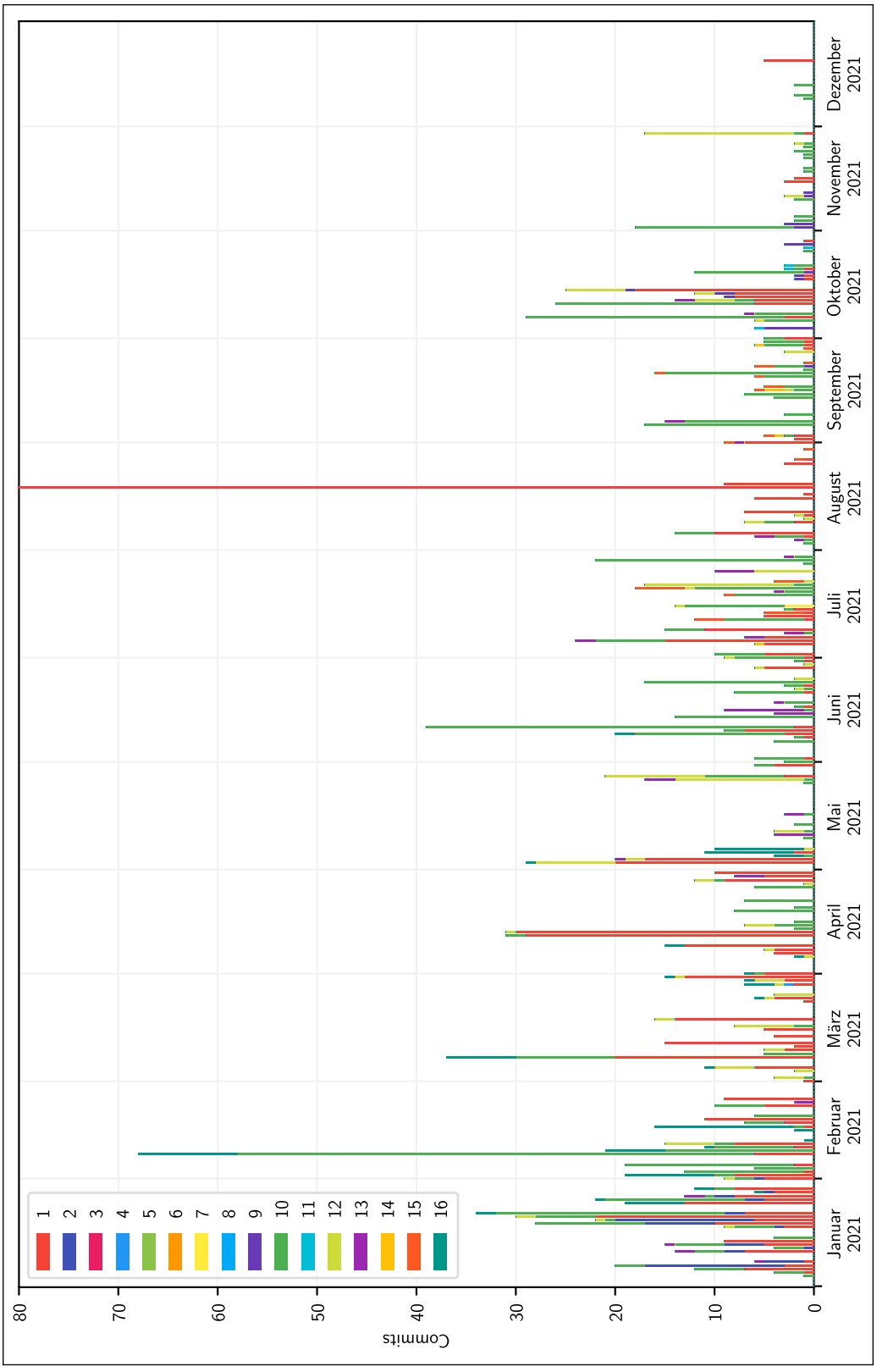


Abbildung: Entwicklungsaktivität (2021)

Quelle: Eigene Darstellung.

Anhang 2.2 Zu Abbildung 9: Fehlerprotokolle

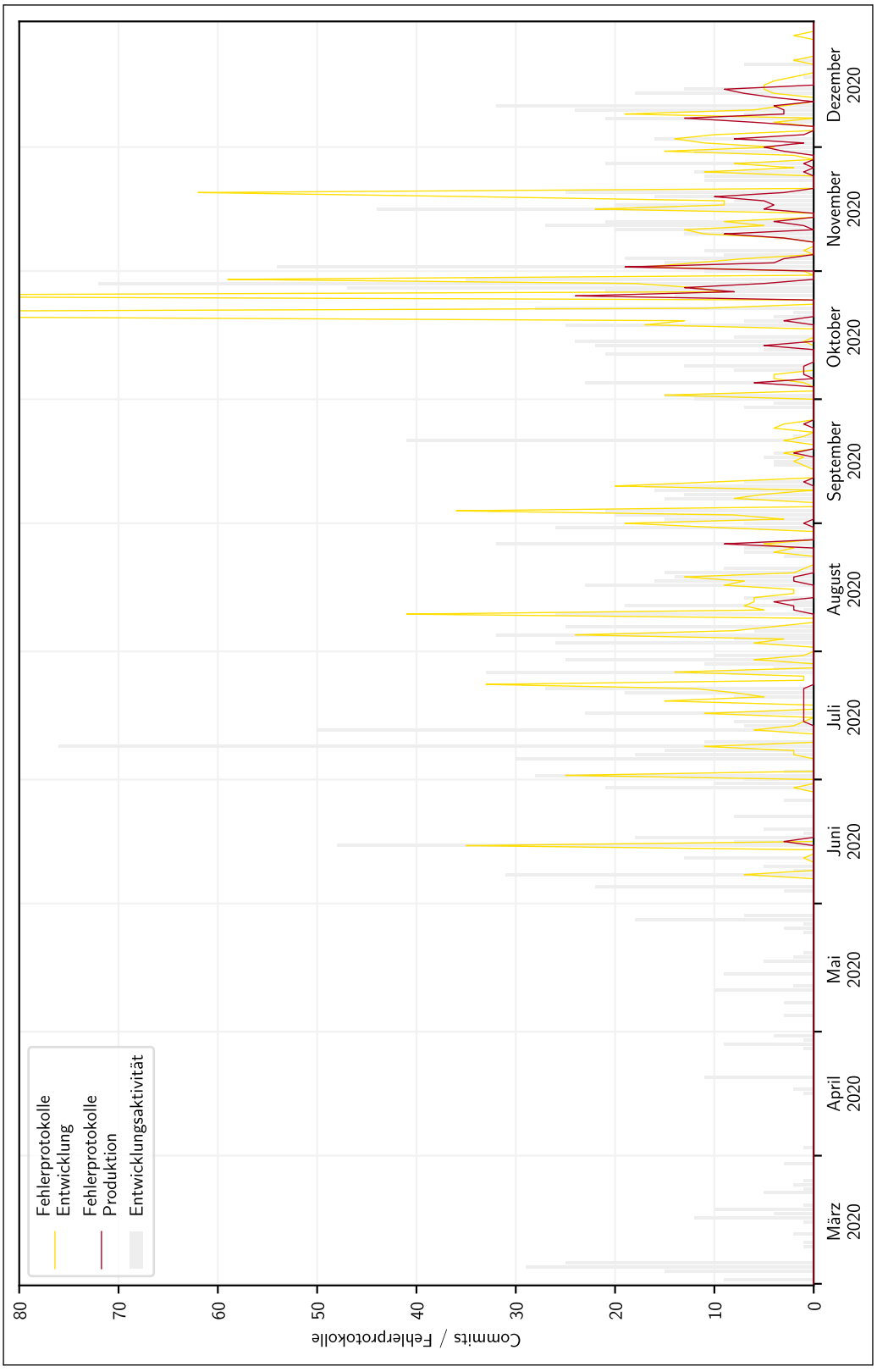


Abbildung: Entwicklungsaktivität und Fehlerprotokolle (2020)

Quelle: Eigene Darstellung.

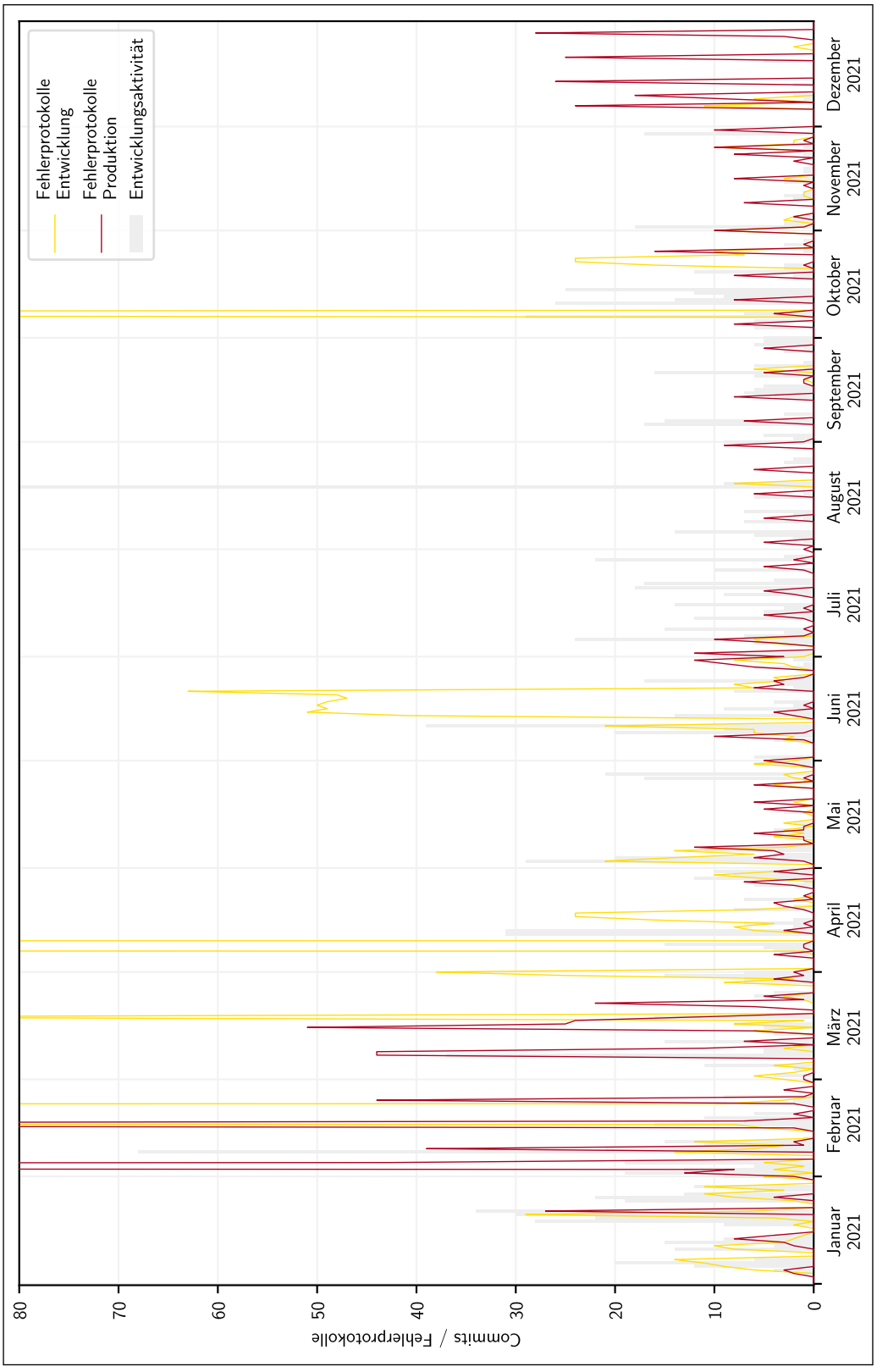


Abbildung: Entwicklungsaktivität und Fehlerprotokolle (2021)

Quelle: Eigene Darstellung.

Anhang 2.3 Zu Abbildung 10: Code Ownership der Teilprojekte

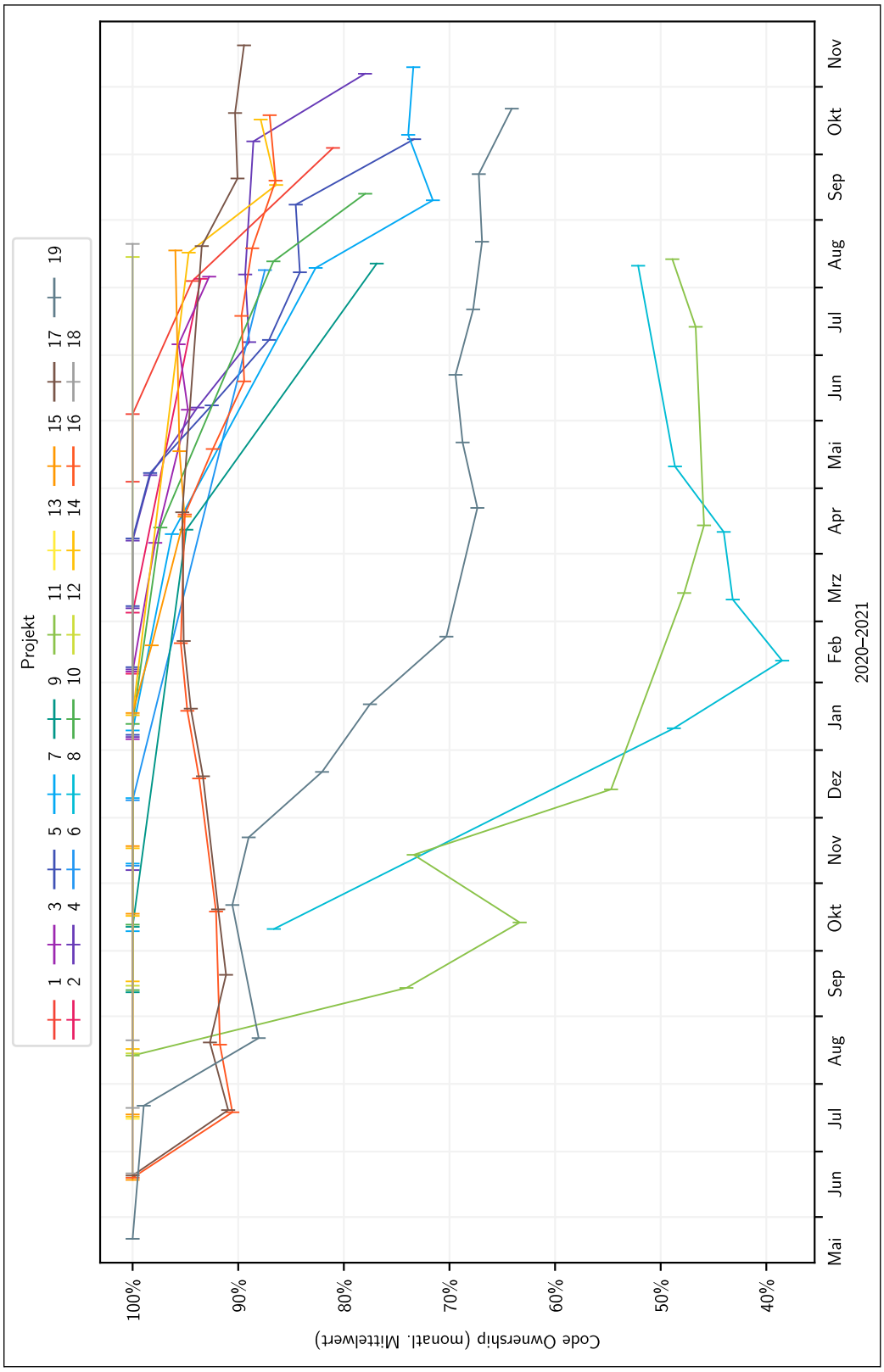
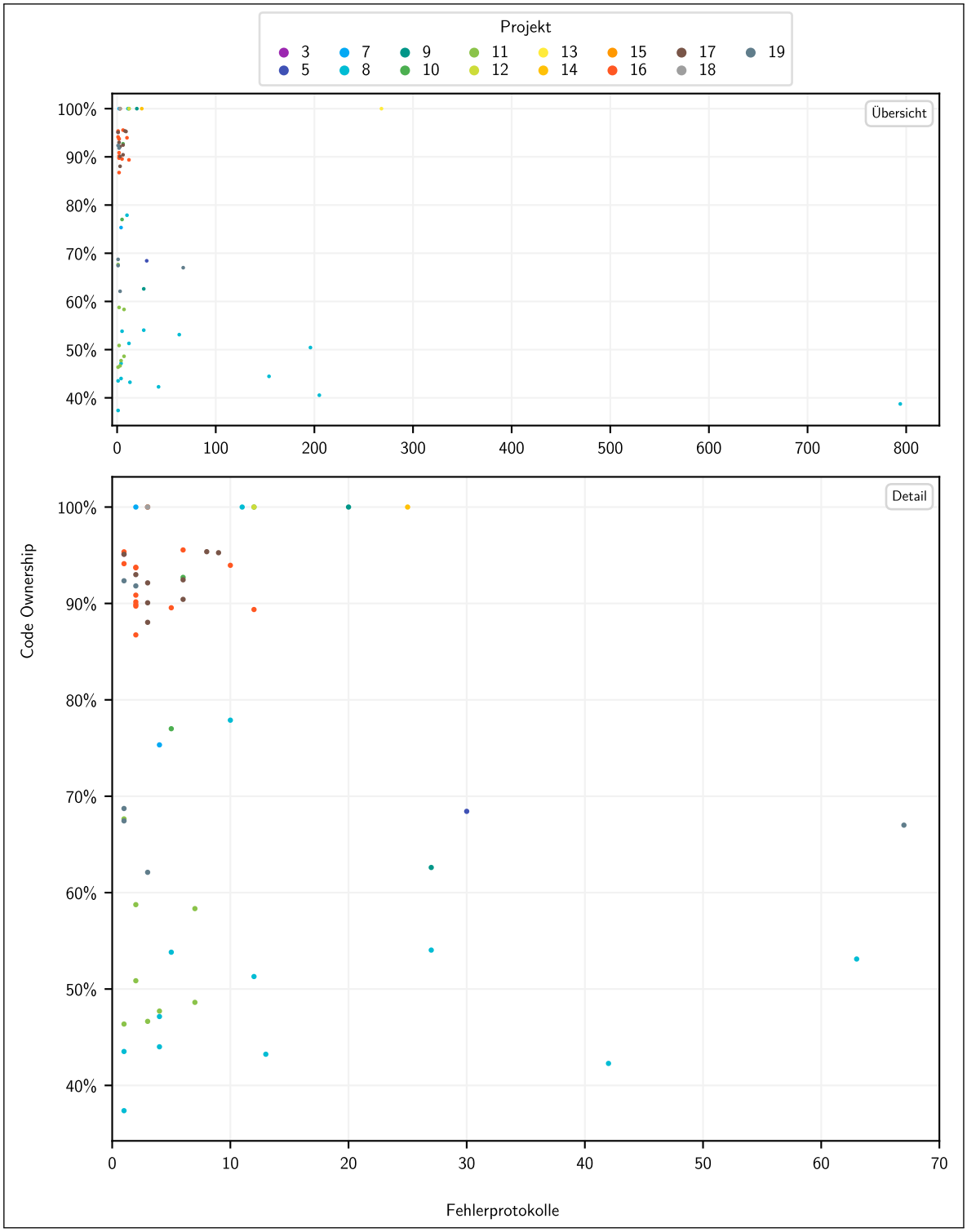


Abbildung: Code Ownership nach Teilprojekt

Quelle: Eigene Darstellung.

Anhang 2.4 Zu Abbildung 11: Fehlerprotokolle und Code Ownership

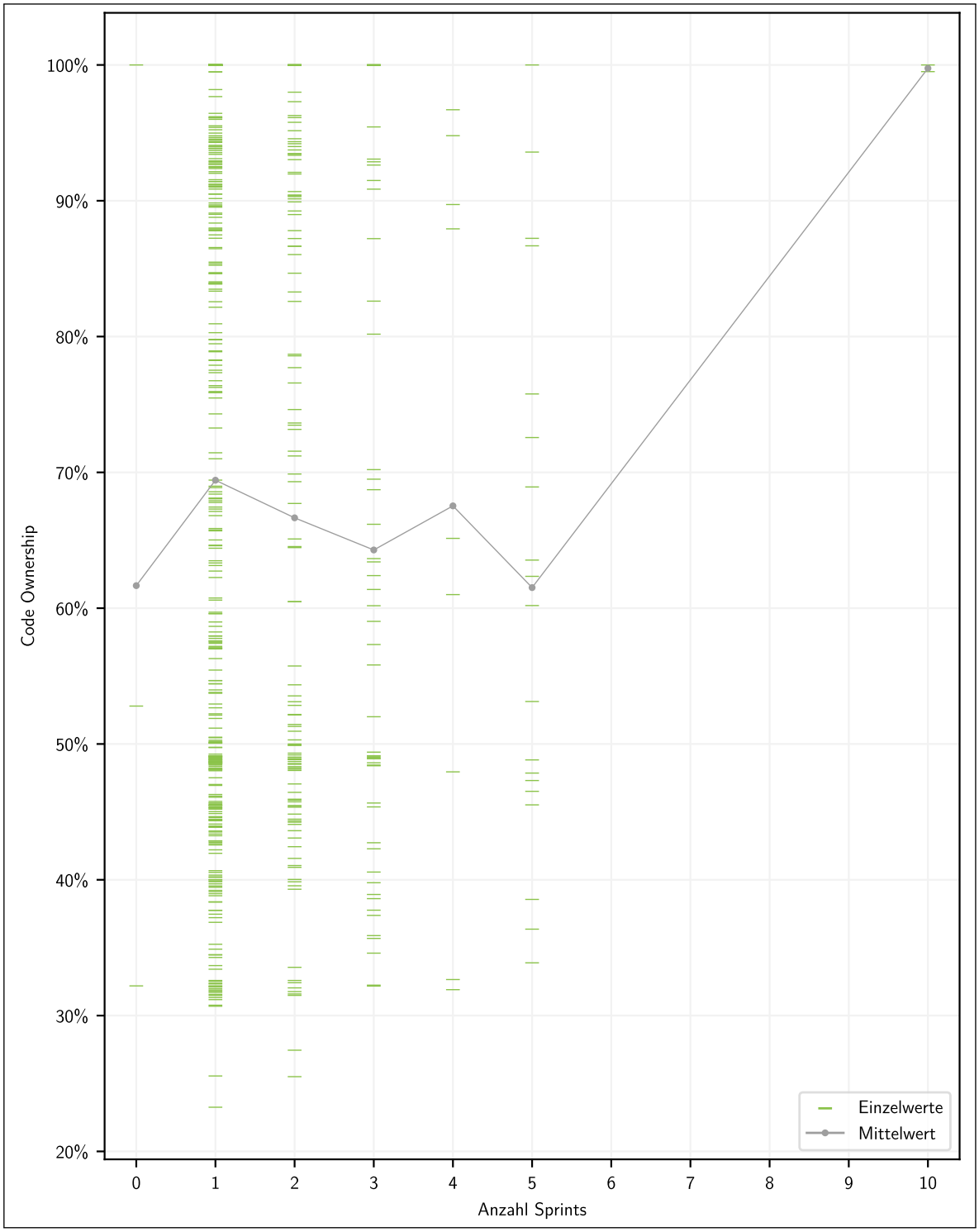
Abbildung: Fehlerprotokolle nach Code Ownership



Quelle: Eigene Darstellung.

Anhang 2.5 Zu Abbildung 18: Verzögerungen in der Entwicklung

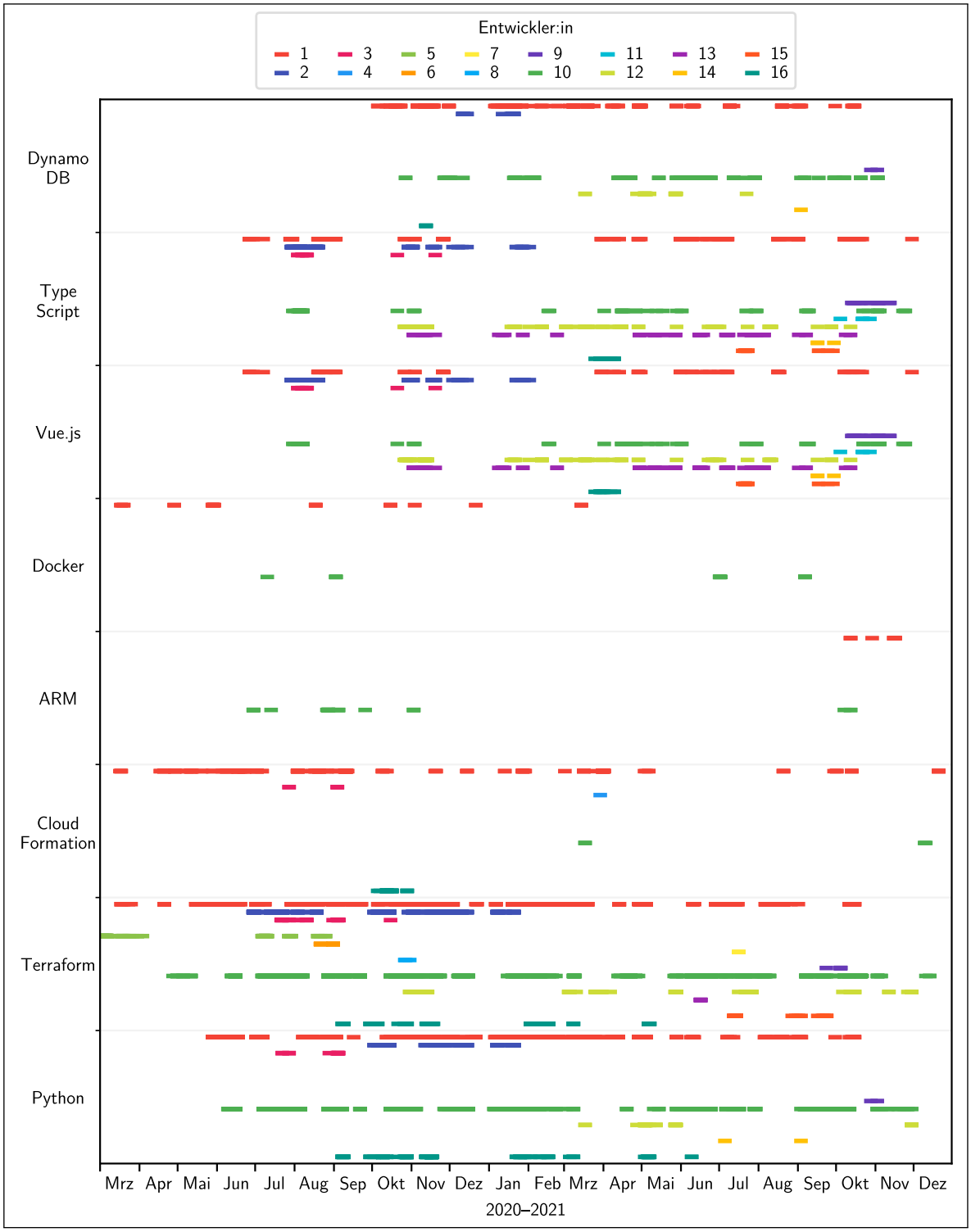
Abbildung: Verzögerungen in der Entwicklung



Quelle: Eigene Darstellung.

Anhang 2.6 Zu Abbildung 19: Expertise der Entwickler:innen

Abbildung: Entwicklungstätigkeit nach Entwickler:in und Technologie



Quelle: Eigene Darstellung.

Anhang 3 Wertetabellen

Im folgenden sind die Werte zu den Abbildungen aus dem Textteil und den vorangegangenen Anhängen in Tabellenform aufgeführt. Innerhalb eines Unterkapitels ist die Reihenfolge nicht immer chronologisch aufgebaut, um die Seiten möglichst auszufüllen. Die Werte des Code Ownerships werden als Dezimalzahl angegeben.

Anhang 3.1 Zu Abbildung 1

Tabelle: Umfrage zur Verbreitung agiler Methoden von 2007-2021

Jahr	**Einsatz generell**[1]	**Unternehmen nutzt vorwiegend**[2]					
		Scrum	XP	Kanban	Scrum+XP	ScrumBan[3]	Andere
2006	84 %	40 %	23 %	-	-	-	37 %
2007	-	37 %	12 %	-	23 %	-	28 %
2008	95 %	49 %	8 %	-	22 %	-	17 %
2009	84 %	50 %	6 %	-	24 %	-	20 %
2010	90 %	58 %	4 %	-	17 %	3 %	15 %
2011	80 %	52 %	2 %	3 %	14 %	3 %	18 %
2012	84 %	54 %	2 %	4 %	11 %	7 %	18 %
2013	88 %	55 %	1 %	5 %	11 %	7 %	21 %
2014	94 %	56 %	1 %	5 %	10 %	6 %	19 %
2015	95 %	58 %	1 %	5 %	10 %	7 %	17 %
2016	94 %	58 %	1 %	5 %	10 %	8 %	16 %
2017	97 %	56 %	1 %	5 %	6 %	8 %	24 %
2018	97 %	54 %	1 %	5 %	10 %	8 %	19 %
2019	95 %	58 %	1 %	7 %	8 %	10 %	13 %
2021	97 %	66 %	1 %	6 %	6 %	9 %	10 %

Quelle: Eigene Tabelle nach Werten von VersionOne, 2007b; VersionOne, 2007a; VersionOne, 2008; VersionOne, 2009; VersionOne, 2010; VersionOne, 2011; VersionOne, 2013; VersionOne, 2014; VersionOne, 2015a; VersionOne, 2015b; VersionOne, 2017; CollabNet-VersionOne, 2018; CollabNet-VersionOne, 2019; Digital.ai, 2020 und Digital.ai, 2021, online im Internet.

[1] Die Prozentangabe bedeutet in diesem Kontext, dass agile Methoden im Unternehmen eingesetzt werden und spiegelt kein Verhältnis zwischen agilen und anderen Methoden wider.

[2] Angabe in Prozent der befragten Unternehmen.

[3] Das Kunstwort ScrumBan beschreibt eine Kombination der Methoden Scrum und Kanban.

Anhang 3.2 Zu Abbildungen 8 und 9 und Anhängen 2.1 und 2.2

Im Folgenden sind links die Werte zur Aktivität der Entwickler:innen als Anzahl Commits pro Tag aufgelistet. Auf der rechten Seite befindet sich die Fehleranzahl pro Tag in der Produktions- (P) und Entwicklungsumgebung (E). Falls an einem Tag kein Wert verzeichnet wurde, ist er in der Tabelle nicht aufgeführt.

Tabelle: Entwicklungsaktivität und Fehlerprotokolle

Datum	Entwickler:in																Umgebung	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
02.03.2020	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
04.03.2020	-	-	-	-	15	-	-	-	-	-	-	-	-	-	-	-	-	-
05.03.2020	-	-	-	-	29	-	-	-	-	-	-	-	-	-	-	-	-	-
06.03.2020	-	-	-	-	25	-	-	-	-	-	-	-	-	-	-	-	-	-
10.03.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
11.03.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13.03.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.03.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.03.2020	10	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
18.03.2020	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
19.03.2020	4	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
20.03.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23.03.2020	3	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
24.03.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
25.03.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.03.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.03.2020	1	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
03.04.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
16.04.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.04.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20.04.2020	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
27.04.2020	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
28.04.2020	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.04.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30.04.2020	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
05.05.2020	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
08.05.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11.05.2020	6	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
12.05.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15.05.2020	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
18.05.2020	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19.05.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20.05.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.05.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.05.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
27.05.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
28.05.2020	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.05.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
04.06.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
05.06.2020	22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08.06.2020	31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7
09.06.2020	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
10.06.2020	1	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
12.06.2020	-	-	-	-	-	-	-	-	-	13	-	-	-	-	-	-	-	1
15.06.2020	8	-	-	-	-	-	-	-	-	40	-	-	-	-	-	-	-	35
16.06.2020	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
17.06.2020	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
18.06.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19.06.2020	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
22.06.2020	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.06.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.06.2020	1	16	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	2
30.06.2020	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
01.07.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
02.07.2020	28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	25
03.07.2020	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
06.07.2020	23	-	-	-	3	-	-	-	-	4	-	-	-	-	-	-	-	-
07.07.2020	1	-	-	-	12	-	-	-	-	5	-	-	-	-	-	-	-	2
08.07.2020	3	2	-	-	1	-	-	-	-	9	-	-	-	-	-	-	-	2
09.07.2020	-	-	-	-	19	-	-	-	-	57	-	-	-	-	-	-	-	11
10.07.2020	-	-	-	-	3	-	-	-	-	8	-	-	-	-	-	-	-	-
13.07.2020	-	14	-	-	-	-	-	-	-	36	-	-	-	-	-	-	-	6
14.07.2020	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
15.07.2020	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1
16.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
17.07.2020	-	9	-	-	-	-	-	-	-	14	-	-	-	-	-	-	1	11
18.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
19.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
20.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	15
21.07.2020	-	3	5	-	-	-	-	-	-	-	-	-	-	-	-	-	1	5
22.07.2020	-	11	7	-	-	-	-	-	-	1	-	-	-	-	-	-	1	8
23.07.2020	-	-	-	-	-	-	-	-	-	27	-	-	-	-	-	-	1	12
24.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33
25.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
26.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
27.07.2020	-	-	8	-	3	-	-	-	-	22	-	-	-	-	-	-	-	14
28.07.2020	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.07.2020	4	4	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
30.07.2020	17	7	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	6
31.07.2020	5	4	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	1
03.08.2020	12	8	4	-	-	-	-	-	-	2	-	-	-	-	-	-	-	6
04.08.2020	-	1	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	3
05.08.2020	-	10	-	-	-	-	-	-	-	22	-	-	-	-	-	-	-	24
06.08.2020	-	2	2	-	-	-	-	-	-	2	-	-	-	-	-	-	-	8
07.08.2020	4	9	2	-	-	-	-	-	-	10	-	-	-	-	-	-	-	4
10.08.2020	16	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	41
11.08.2020	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	5
12.08.2020	17	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	7
13.08.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	6
14.08.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6
15.08.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
16.08.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
17.08.2020	16	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9
18.08.2020	5	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	7
19.08.2020	6	5	-	-	3	-	-	-	-	-	-	-	-	-	-	-	2	13

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
20.08.2020	15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
21.08.2020	2	-	-	-	2	5	-	-	-	-	-	-	-	-	-	-	-	1
24.08.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.08.2020	4	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	4
26.08.2020	6	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	2
27.08.2020	-	-	-	-	-	-	-	-	-	32	-	-	-	-	-	-	9	5
28.08.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
31.08.2020	18	-	1	-	-	7	-	-	-	-	-	-	-	-	-	-	-	10
01.09.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	19
02.09.2020	9	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	-	3
03.09.2020	6	-	10	-	-	-	-	-	-	4	-	-	-	-	-	-	-	8
04.09.2020	6	-	9	-	-	-	-	-	-	6	-	-	-	-	-	-	-	36
07.09.2020	-	-	-	-	-	-	-	-	-	9	-	-	-	-	-	6	-	8
08.09.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13	-	5
09.09.2020	5	-	-	-	-	-	-	-	-	11	-	-	-	-	-	-	-	-
10.09.2020	17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20
11.09.2020	4	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	1	10
15.09.2020	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
16.09.2020	1	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	2
17.09.2020	3	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	1
18.09.2020	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	3
21.09.2020	9	-	-	-	-	-	-	-	-	32	-	-	-	-	-	-	-	3
22.09.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
24.09.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4
25.09.2020	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	3
29.09.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-
30.09.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
01.10.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	11	-	-
02.10.2020	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-	9	-	15
05.10.2020	3	16	-	-	-	-	-	-	-	-	-	-	-	-	-	4	6	1
06.10.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4
07.10.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	4
08.10.2020	2	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
09.10.2020	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	1	-
12.10.2020	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	13	-	-
13.10.2020	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14.10.2020	-	22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
15.10.2020	9	-	1	-	-	-	-	-	-	14	-	-	-	-	-	-	-	1
16.10.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-
19.10.2020	8	-	-	-	-	-	-	-	-	17	-	-	-	-	-	-	-	17
20.10.2020	5	-	1	-	-	-	-	-	-	1	-	-	-	-	-	-	3	13
21.10.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	96
22.10.2020	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	123
23.10.2020	16	-	7	-	-	-	-	-	-	5	-	-	-	-	-	-	-	11
26.10.2020	5	-	-	-	-	-	-	1	-	-	-	1	-	-	-	7	24	118
27.10.2020	-	-	-	-	-	-	-	1	-	6	-	4	-	-	-	10	8	9
28.10.2020	19	-	-	-	-	-	-	-	-	3	-	1	-	-	-	24	13	12
29.10.2020	26	7	-	-	-	-	-	-	-	39	-	-	-	-	-	-	5	18
30.10.2020	4	-	-	-	-	-	-	1	-	26	-	4	-	-	-	-	-	59
01.11.2020	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
02.11.2020	5	21	-	-	-	-	-	-	-	21	-	4	3	-	-	-	19	17
03.11.2020	8	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	4	12
04.11.2020	8	-	-	-	-	-	-	-	-	10	-	1	-	-	-	-	3	7
05.11.2020	6	-	-	-	-	-	-	-	-	1	-	2	-	-	-	-	-	-
06.11.2020	1	-	2	-	-	-	-	-	-	1	-	7	-	-	-	-	-	1
09.11.2020	-	-	-	-	-	-	-	-	-	2	-	-	2	-	-	-	3	3

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
10.11.2020	10	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	9	11
11.11.2020	10	2	-	-	-	-	-	-	-	-	-	7	1	-	-	-	-	13
12.11.2020	5	8	-	-	-	-	-	-	-	4	-	3	1	-	-	6	1	5
13.11.2020	4	-	-	-	-	-	-	-	-	1	-	7	-	-	-	9	4	9
16.11.2020	10	3	-	-	-	-	-	-	-	-	-	-	-	-	-	31	5	22
17.11.2020	6	6	-	-	-	-	-	-	-	-	-	-	-	-	-	8	4	9
18.11.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	5	9
19.11.2020	7	7	1	-	-	-	-	-	-	-	-	-	1	-	-	-	10	34
20.11.2020	8	5	-	-	-	-	-	-	-	12	-	-	-	-	-	-	3	62
23.11.2020	-	6	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-
24.11.2020	1	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.11.2020	11	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	11
26.11.2020	2	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	2
27.11.2020	1	18	-	-	-	-	-	-	-	2	-	-	-	-	-	-	1	8
29.11.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
30.11.2020	10	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	3	15
01.12.2020	1	5	-	-	-	-	-	-	-	2	-	-	-	-	-	-	5	4
02.12.2020	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	11
03.12.2020	15	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	14
04.12.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	10
07.12.2020	-	2	-	-	-	-	-	-	-	1	-	-	-	-	-	-	6	4
08.12.2020	-	12	-	-	-	-	-	-	-	9	-	-	-	-	-	-	13	-
09.12.2020	-	4	-	-	-	-	-	-	-	3	-	-	-	-	-	-	3	19
10.12.2020	2	17	-	-	-	-	-	-	-	5	-	-	-	-	-	-	3	6
11.12.2020	-	7	-	-	-	-	-	-	-	25	-	-	-	-	-	-	4	3
13.12.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-
14.12.2020	4	12	-	-	-	-	-	-	-	2	-	-	-	-	-	-	7	4
15.12.2020	-	-	-	-	-	-	-	-	-	13	-	-	-	-	-	-	9	5
16.12.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5
17.12.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4
18.12.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
21.12.2020	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
22.12.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
28.12.2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
04.01.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	2	-
05.01.2021	1	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	3	6
06.01.2021	7	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	9
07.01.2021	3	14	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	11
08.01.2021	1	4	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	14
11.01.2021	7	2	-	-	-	-	-	-	-	3	-	-	2	-	-	-	-	8
12.01.2021	-	1	-	-	-	-	-	-	-	3	-	-	-	-	-	-	2	10
13.01.2021	5	4	-	-	-	-	-	-	-	5	-	-	1	-	-	-	3	3
14.01.2021	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	2
15.01.2021	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	4	1
18.01.2021	3	1	-	-	-	-	-	-	-	4	-	1	-	-	-	-	-	2
19.01.2021	10	7	-	-	-	-	-	-	-	11	-	-	-	-	-	-	-	-
20.01.2021	6	14	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	4
21.01.2021	22	-	-	-	-	-	-	-	-	6	-	2	-	-	-	-	-	29
22.01.2021	7	2	-	-	-	-	-	-	-	23	-	-	-	-	-	2	27	6
25.01.2021	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-	2
26.01.2021	5	2	-	-	-	-	-	-	-	14	-	-	-	-	-	1	4	8
27.01.2021	8	2	-	-	-	-	-	-	-	1	-	-	2	-	-	-	-	11
28.01.2021	4	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	3
29.01.2021	8	-	-	-	-	-	-	-	-	2	-	-	-	-	-	2	-	11
01.02.2021	5	1	-	-	-	-	-	-	-	2	-	1	-	-	-	-	2	5
02.02.2021	8	-	-	-	-	-	-	-	-	2	-	-	-	-	-	9	13	-

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
03.02.2021	1	-	-	-	-	-	-	-	-	12	-	-	-	-	-	-	8	4
04.02.2021	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	704	1
05.02.2021	2	-	-	-	-	-	-	-	-	17	-	-	-	-	-	-	43	5
08.02.2021	6	-	-	-	-	-	-	-	-	52	-	-	-	-	-	10	-	14
09.02.2021	-	-	-	-	-	-	-	-	-	15	-	-	-	-	-	6	39	6
10.02.2021	2	-	-	-	-	-	-	-	-	8	-	-	-	-	-	1	1	3
11.02.2021	8	-	-	-	-	-	-	-	-	2	-	5	-	-	-	-	2	12
12.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-
15.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	5
16.02.2021	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	14	213	8
17.02.2021	3	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	7	3468
18.02.2021	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5677
19.02.2021	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	2	4686
20.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4744
21.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1841
22.02.2021	5	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	2	9
23.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	44	3
24.02.2021	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
26.02.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
01.03.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	3
02.03.2021	-	-	-	-	-	-	-	-	-	1	-	3	-	-	-	-	1	6
03.03.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
04.03.2021	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-
05.03.2021	6	-	-	-	-	-	-	-	-	-	-	4	-	-	-	1	-	4
08.03.2021	20	-	-	-	-	-	-	-	-	10	-	-	-	-	-	7	44	-
09.03.2021	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	44	-
10.03.2021	3	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	11	3
11.03.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.03.2021	15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-
14.03.2021	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15.03.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	6
16.03.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	51	-
17.03.2021	-	-	-	-	-	-	-	-	-	2	-	6	-	-	-	-	25	8
18.03.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24	1
19.03.2021	14	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	12	104
22.03.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-
23.03.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	22	-
24.03.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1
25.03.2021	4	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	5	3
26.03.2021	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-
29.03.2021	2	-	-	1	-	-	-	-	-	-	-	1	-	-	-	3	-	9
30.03.2021	3	-	-	-	-	-	-	-	-	-	-	3	-	-	-	1	4	2
31.03.2021	13	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	1	25
01.04.2021	5	-	-	-	-	-	-	-	-	1	-	-	-	-	-	1	2	38
06.04.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	4	-
07.04.2021	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
08.04.2021	4	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1	1644
09.04.2021	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	1	18780
12.04.2021	29	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
13.04.2021	30	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	3	6
14.04.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	8
15.04.2021	-	-	-	-	-	-	-	-	-	4	-	3	-	-	-	-	1	4
16.04.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	16
17.04.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24
18.04.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24
19.04.2021	-	-	-	-	-	-	-	-	-	8	-	-	-	-	-	-	1	6

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
20.04.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	3	-
21.04.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-
22.04.2021	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	2
23.04.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
26.04.2021	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	2	-
27.04.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	7	-
28.04.2021	9	-	-	-	-	-	-	-	-	1	-	2	-	-	-	-	-	5
29.04.2021	5	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	10
30.04.2021	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4
03.05.2021	20	-	-	-	-	-	-	-	-	-	-	8	-	-	-	1	1	21
04.05.2021	17	-	-	-	-	-	-	-	-	-	-	2	1	-	-	-	6	14
05.05.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	3	3	6
06.05.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	4	14
07.05.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	9	12	3
09.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
10.05.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	4
11.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	6	1
12.05.2021	-	-	-	-	-	-	-	-	-	1	-	3	-	-	-	-	1	3
13.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
14.05.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	3
17.05.2021	-	-	-	-	-	-	-	-	-	1	-	-	2	-	-	-	-	1
18.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
20.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	2
25.05.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	4
26.05.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
27.05.2021	-	-	-	-	-	-	-	-	-	1	-	13	3	-	-	-	1	2
28.05.2021	3	-	-	-	-	-	-	-	-	8	-	10	-	-	-	-	-	3
31.05.2021	4	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	2	6
01.06.2021	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	5	2
02.06.2021	1	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-
07.06.2021	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	1	3
08.06.2021	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	10	2
09.06.2021	3	-	-	-	-	-	-	-	-	15	-	-	-	-	-	2	1	6
10.06.2021	7	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	6
11.06.2021	2	-	-	-	-	-	-	-	-	37	-	-	-	-	-	-	-	21
14.06.2021	-	-	-	-	-	-	-	-	-	14	-	-	-	-	-	-	2	41
15.06.2021	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	4	51
16.06.2021	-	-	-	-	-	-	-	-	-	1	-	-	8	-	-	-	-	49
17.06.2021	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	50
18.06.2021	-	-	-	-	-	-	-	-	-	3	-	-	1	-	-	-	-	49
19.06.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	47
20.06.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	48
21.06.2021	1	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	63
22.06.2021	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	6	6
23.06.2021	1	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	3	8
24.06.2021	-	-	-	-	-	-	-	-	-	17	-	-	-	-	-	-	4	3
25.06.2021	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	1	4
28.06.2021	5	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	6	2
29.06.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	9	3
30.06.2021	1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	12	8
01.07.2021	1	-	-	-	-	-	-	-	-	7	-	1	-	-	-	-	3	1
02.07.2021	5	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	12	-
05.07.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	4	5
06.07.2021	15	-	-	-	-	-	-	-	-	7	-	-	2	-	-	-	10	6
07.07.2021	5	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	1	-
08.07.2021	-	-	-	-	-	-	-	-	-	1	-	-	2	-	-	-	-	-

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
09.07.2021	11	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	1	-
12.07.2021	1	-	-	-	-	-	-	-	-	8	-	-	-	-	3	-	1	-
13.07.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
14.07.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-
15.07.2021	2	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	-
16.07.2021	-	-	-	-	-	-	3	-	-	10	-	1	-	-	-	-	-	-
19.07.2021	-	-	-	-	-	-	-	-	-	8	-	-	-	-	1	-	2	-
20.07.2021	-	-	-	-	-	-	-	-	-	3	-	-	1	-	-	-	5	-
21.07.2021	-	-	-	-	-	-	-	-	-	12	-	1	-	-	5	-	-	-
22.07.2021	-	-	-	-	-	-	-	-	-	2	-	15	-	-	-	-	-	-
23.07.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	3	-	-	-
26.07.2021	-	-	-	-	-	-	-	-	-	-	-	6	4	-	-	-	1	-
27.07.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
28.07.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
29.07.2021	-	-	-	-	-	-	-	-	-	22	-	-	-	-	-	-	2	-
30.07.2021	-	-	-	-	-	-	-	-	-	2	-	-	1	-	-	-	-	-
01.08.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
03.08.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	5	-
04.08.2021	-	-	-	-	-	-	-	-	-	1	-	-	1	-	-	-	-	-
05.08.2021	1	-	-	-	-	-	-	-	-	3	-	-	2	-	-	-	-	-
06.08.2021	10	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
09.08.2021	2	-	-	-	-	-	-	-	-	3	-	2	-	-	-	-	-	-
10.08.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	5	-
11.08.2021	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-
12.08.2021	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.08.2021	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.08.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-
19.08.2021	90	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
20.08.2021	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8
24.08.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-
26.08.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
27.08.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
30.08.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
31.08.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	-
01.09.2021	7	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	1	-
02.09.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
03.09.2021	2	-	-	-	-	-	-	-	-	1	-	-	-	1	1	-	-	-
06.09.2021	-	-	-	-	-	-	-	-	-	17	-	-	-	-	-	-	-	-
07.09.2021	-	-	-	-	-	-	-	-	-	13	-	-	2	-	-	-	7	-
09.09.2021	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
14.09.2021	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	8	-
15.09.2021	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-
16.09.2021	-	-	-	-	-	-	-	-	-	2	-	1	-	2	1	-	-	-
17.09.2021	-	-	-	-	-	-	-	-	-	3	-	-	-	-	2	-	-	-
18.09.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
19.09.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1
20.09.2021	-	-	-	-	-	-	-	-	-	5	-	-	-	-	1	-	-	-
21.09.2021	-	-	-	-	-	-	-	-	-	15	-	-	-	-	1	-	5	-
22.09.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	6
23.09.2021	-	-	-	-	-	-	-	-	1	3	-	-	-	-	2	-	-	-
24.09.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
27.09.2021	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-
28.09.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	5	-
29.09.2021	1	-	-	-	-	-	-	-	-	4	-	-	-	1	-	-	-	-
30.09.2021	1	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
01.10.2021	3	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-

Datum	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	P	E
04.10.2021	-	-	-	-	-	-	-	-	5	-	1	-	-	-	-	-	-	-
05.10.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	-
06.10.2021	-	-	-	-	-	-	-	-	-	5	-	1	-	-	-	-	-	-
07.10.2021	3	-	-	-	-	-	-	-	-	26	-	-	-	-	-	-	-	-
08.10.2021	-	-	-	-	-	-	-	-	-	6	-	-	1	-	-	-	4	373
11.10.2021	6	-	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-	-
12.10.2021	6	-	-	-	-	-	-	-	-	2	-	4	2	-	-	-	8	-
13.10.2021	8	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
14.10.2021	8	-	-	-	-	-	-	-	2	-	-	2	-	-	-	-	-	-
15.10.2021	18	-	-	-	-	-	-	-	1	-	-	6	-	-	-	-	-	-
18.10.2021	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
19.10.2021	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	8	-
20.10.2021	-	-	-	-	-	-	-	-	1	11	-	-	-	-	-	-	-	-
21.10.2021	1	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-
22.10.2021	-	-	-	-	-	-	-	-	-	2	1	-	-	-	-	-	1	16
23.10.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24
24.10.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24
25.10.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7
26.10.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	16	10
27.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-
28.10.2021	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	1	1
29.10.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
01.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	10
02.11.2021	-	-	-	-	-	-	-	-	2	16	-	-	-	-	-	-	1	1
03.11.2021	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-
04.11.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	3
05.11.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	2	2
09.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-
10.11.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
11.11.2021	-	-	-	-	-	-	-	-	1	-	-	2	-	-	-	-	-	1
12.11.2021	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1
14.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
15.11.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.11.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	3
18.11.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
19.11.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
21.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
22.11.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
23.11.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	8	-
24.11.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
25.11.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	10	10
26.11.2021	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-	-	-	2
27.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	2
29.11.2021	1	-	-	-	-	-	-	-	-	1	-	15	-	-	-	-	-	-
30.11.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	-
07.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	24	11
09.12.2021	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	9	6
10.12.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	18	-
13.12.2021	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
14.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26	-
20.12.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	25	-
24.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
27.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
28.12.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	28	-

Quelle: Eigene Tabelle.

Anhang 3.3 Zu Abbildung 10 und Anhang 2.3

Tabelle: Code Ownership der Teilprojekte

Monat	Projekt																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Mai 2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1,00
Jun 2020	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1,00	1,00	1,00	1,00	-
Jul 2020	-	-	-	-	-	-	-	-	-	-	-	-	1,00	1,00	1,00	0,91	0,91	1,00	0,99
Aug 2020	-	-	-	-	-	-	-	-	-	-	1,00	1,00	-	1,00	-	0,92	0,93	1,00	0,88
Sep 2020	-	-	-	-	-	-	-	-	1,00	1,00	0,74	1,00	-	1,00	-	-	0,91	-	-
Okt 2020	-	-	-	-	-	-	1,00	0,87	1,00	1,00	0,63	-	-	1,00	1,00	0,92	0,92	-	0,91
Nov 2020	-	-	-	1,00	-	1,00	1,00	-	-	-	0,73	-	-	1,00	1,00	-	-	-	0,89
Dez 2020	-	-	-	-	-	1,00	1,00	-	-	-	0,55	-	-	-	-	0,94	0,93	-	0,82
Jan 2021	-	-	1,00	1,00	1,00	-	1,00	0,49	-	1,00	-	-	-	1,00	1,00	0,95	0,94	-	0,78
Feb 2021	-	1,00	1,00	1,00	1,00	-	-	0,39	-	-	-	-	-	-	0,98	0,95	0,95	-	0,70
Mrz 2021	-	1,00	-	1,00	1,00	-	-	0,43	-	-	0,48	-	-	-	-	-	-	-	-
Apr 2021	-	-	0,98	1,00	1,00	-	0,96	0,44	0,95	0,97	0,46	-	-	-	0,95	0,95	0,95	-	0,67
Mai 2021	1,00	-	-	0,98	0,98	-	-	0,49	-	-	-	-	-	-	0,96	0,92	-	-	0,69
Jun 2021	1,00	-	0,95	0,94	0,93	-	-	-	-	-	-	-	-	-	-	0,89	-	-	0,69
Jul 2021	-	-	0,96	0,89	0,87	-	-	-	-	-	0,47	-	-	-	-	0,90	-	-	0,68
Aug 2021	0,94	0,94	0,93	0,89	0,84	0,87	0,83	0,52	0,77	0,87	0,49	1,00	-	0,95	0,96	0,89	0,93	1,00	0,67
Sep 2021	-	-	-	-	0,85	-	0,72	-	-	0,78	-	-	-	0,86	-	0,86	0,90	-	0,67
Okt 2021	0,81	-	-	0,89	0,73	-	0,74	-	-	-	-	-	-	0,88	-	0,87	0,90	-	0,64
Nov 2021	-	-	-	0,78	-	-	0,73	-	-	-	-	-	-	-	-	-	0,89	-	-

Quelle: Eigene Tabelle.

Anhang 3.4 Zu Abbildung 11 und Anhang 2.4

Tabelle: Fehlerprotokolle nach Code Ownership (Übersicht)

Bereich (%)	0-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
Fehlerprotokolle	0	0	0	795	438	314	130	19	26	433

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership (Detail)

Code Ownership	Projekt														
	3	5	7	8	9	10	11	12	13	14	15	16	17	18	19
1	3	-	2	11	20	12	-	12	268	25	3	-	-	3	-
0.956	-	-	-	-	-	-	-	-	-	-	-	6	-	-	-
0.954	-	-	-	-	-	-	-	-	-	-	-	1	8	-	-
0.953	-	-	-	-	-	-	-	-	-	-	-	-	9	-	-
0.952	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0.951	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-
0.942	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
0.94	-	-	-	-	-	-	-	-	-	-	-	10	-	-	-
0.938	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.937	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.93	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-
0.928	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-
0.925	-	-	-	-	-	-	-	-	-	-	-	-	6	-	-
0.924	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
0.922	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-
0.919	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2
0.909	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.905	-	-	-	-	-	-	-	-	-	-	-	-	6	-	-
0.902	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.901	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-
0.899	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.898	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.896	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-
0.894	-	-	-	-	-	-	-	-	-	-	-	12	-	-	-
0.881	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-
0.868	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-
0.779	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
0.771	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-
0.754	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-
0.688	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
0.685	-	30	-	-	-	-	-	-	-	-	-	-	-	-	-
0.677	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
0.675	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
0.671	-	-	-	-	-	-	-	-	-	-	-	-	-	-	67
0.627	-	-	-	-	27	-	-	-	-	-	-	-	-	-	-
0.622	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3
0.588	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
0.584	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-
0.541	-	-	-	27	-	-	-	-	-	-	-	-	-	-	-
0.539	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-

Code Ownership	3	5	7	8	9	10	11	12	13	14	15	16	17	18	19
0.532	-	-	-	63	-	-	-	-	-	-	-	-	-	-	-
0.513	-	-	-	12	-	-	-	-	-	-	-	-	-	-	-
0.509	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-
0.505	-	-	-	196	-	-	-	-	-	-	-	-	-	-	-
0.487	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-
0.478	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
0.472	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-
0.467	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
0.464	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
0.445	-	-	-	154	-	-	-	-	-	-	-	-	-	-	-
0.441	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-
0.436	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
0.433	-	-	-	13	-	-	-	-	-	-	-	-	-	-	-
0.423	-	-	-	42	-	-	-	-	-	-	-	-	-	-	-
0.406	-	-	-	205	-	-	-	-	-	-	-	-	-	-	-
0.388	-	-	-	794	-	-	-	-	-	-	-	-	-	-	-
0.374	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Anhang 3.5 Zu Abbildungen 12 und 13 und Anhang 1.1

Die folgenden Wertetabellen stellen bei Auftreten eines Fehlers potenzielle Unterschiede zwischen dem Code Ownership von Projekten und verantwortlichen Entwickler:innen dar. Die Anzahl des Auftretens eines bestimmten Wertes ist in Klammern dahinter notiert.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 3)

Datum	**Code Ownership**	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
30.03.2021	1,00 (1)	1,00 (1)
31.03.2021	1,00 (1)	1,00 (1)
01.04.2021	1,00 (1)	1,00 (1)
22.06.2021	0,95 (1)	0,95 (1)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 5)

Datum	**Code Ownership**	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
26.10.2021	0,68 (10)	0,68 (10)
01.11.2021	0,68 (10)	0,68 (10)
25.11.2021	0,68 (10)	0,68 (10)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 7)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
02.11.2020	1,00 (1)	1,00 (1)
08.10.2021	0,75 (2)	0,75 (2)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 8)

Datum	Code Ownership		
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)	Mittelwert Entwickler:in p.M.
05.10.2020	1,00 (5)	1,00 (5)	0,79
08.10.2020	1,00 (1)	1,00 (1)	0,79
14.10.2020	1,00 (4)	1,00 (4)	0,79
28.10.2020	0,78 (1)	0,22 (1)	0,79
29.10.2020	0,54 (5)	0,46 (4); 0,54 (1)	0,79
22.01.2021	0,54 (7)	0,54 (7)	0,54
04.02.2021	0,39 (361)	0,29 (24); 0,39 (337)	0,36
05.02.2021	0,39 (34)	0,39 (34)	0,36
09.02.2021	0,39 (35)	0,39 (35)	0,36
16.02.2021	0,41 (184)	0,32 (184)	0,36
23.02.2021	0,42 (35)	0,32 (35)	0,36
02.03.2021	0,42 (1)	0,42 (1)	0,31
08.03.2021	0,44 (1); 0,44 (37)	0,31 (1); 0,31 (37)	0,31
09.03.2021	0,44 (36)	0,31 (36)	0,31
16.03.2021	0,44 (33)	0,31 (33)	0,31
23.03.2021	0,44 (21)	0,31 (21)	0,31
25.03.2021	0,44 (4)	0,31 (4)	0,31
30.03.2021	0,44 (2)	0,31 (2)	0,31
06.04.2021	0,44 (3)	0,31 (3)	0,34
13.04.2021	0,43 (3)	0,33 (3)	0,34
20.04.2021	0,43 (3)	0,33 (3)	0,34
27.04.2021	0,43 (4)	0,33 (4)	0,34
30.04.2021	0,43 (3)	0,43 (2); 0,33 (1)	0,34
04.05.2021	0,44 (4)	0,32 (4)	0,28
06.05.2021	0,47 (4)	0,30 (4)	0,28
07.05.2021	0,51 (12)	0,28 (12)	0,28
11.05.2021	0,53 (4)	0,27 (4)	0,28
18.05.2021	0,53 (4)	0,27 (4)	0,28
25.05.2021	0,53 (4)	0,27 (4)	0,28
01.06.2021	0,53 (4)	0,27 (4)	0,27
08.06.2021	0,53 (6)	0,27 (6)	0,27
15.06.2021	0,53 (4)	0,27 (4)	0,27
22.06.2021	0,53 (4)	0,27 (4)	0,27
29.06.2021	0,53 (4)	0,27 (4)	0,27
06.07.2021	0,53 (4)	0,27 (4)	0,27
13.07.2021	0,53 (4)	0,27 (4)	0,27
20.07.2021	0,53 (4)	0,27 (4)	0,27
27.07.2021	0,53 (4)	0,27 (4)	0,27
03.08.2021	0,53 (4)	0,27 (4)	0,29
10.08.2021	0,53 (4)	0,27 (4)	0,29
17.08.2021	0,53 (5)	0,27 (5)	0,29

Datum	Projekt (+Anzahl)	Entwickler:in (+Anzahl)	Mittelwert Entwickler:in p.M.
24.08.2021	0,50 (5)	0,31 (5)	0,29
31.08.2021	0,50 (8)	0,31 (8)	0,29
07.09.2021	0,50 (6)	0,31 (6)	0,32
14.09.2021	0,50 (6)	0,31 (6)	0,32
21.09.2021	0,50 (5)	0,31 (5)	0,32
28.09.2021	0,50 (4)	0,31 (3); 0,50 (1)	0,32
05.10.2021	0,50 (8)	0,31 (8)	0,31
12.10.2021	0,50 (6)	0,31 (5); 0,50 (1)	0,31
19.10.2021	0,50 (6)	0,31 (6)	0,31
26.10.2021	0,50 (6)	0,31 (6)	0,31
02.11.2021	0,50 (1)	0,50 (1)	0,33
09.11.2021	0,50 (6)	0,31 (5); 0,50 (1)	0,33
16.11.2021	0,50 (8)	0,31 (8)	0,33
23.11.2021	0,50 (7)	0,31 (6); 0,50 (1)	0,33
30.11.2021	0,50 (9)	0,31 (8); 0,50 (1)	0,33
07.12.2021	0,50 (20)	0,31 (19); 0,50 (1)	0,32
14.12.2021	0,50 (25)	0,31 (24); 0,50 (1)	0,32
21.12.2021	0,50 (23)	0,31 (21); 0,50 (2)	0,32
28.12.2021	0,50 (27)	0,31 (26); 0,50 (1)	0,32

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 9)

Datum	**Code Ownership**	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
26.10.2020	1,00 (15)	1,00 (15)
28.10.2020	1,00 (3)	1,00 (3)
09.12.2021	0,63 (9)	0,63 (9)
10.12.2021	0,63 (18)	0,63 (18)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 10)

Datum	**Code Ownership**	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
27.10.2020	1,00 (1)	1,00 (1)
03.11.2020	1,00 (1)	1,00 (1)
10.11.2020	1,00 (3)	1,00 (3)
23.02.2021	1,00 (1)	1,00 (1)
16.03.2021	1,00 (1)	1,00 (1)
04.05.2021	0,93 (1)	0,93 (1)
20.07.2021	0,93 (1)	0,93 (1)
27.07.2021	0,93 (1)	0,93 (1)
03.08.2021	0,93 (1)	0,93 (1)
10.08.2021	0,93 (1)	0,93 (1)
17.08.2021	0,93 (1)	0,93 (1)
24.08.2021	0,77 (1)	0,77 (1)
31.08.2021	0,77 (1)	0,77 (1)
07.09.2021	0,77 (1)	0,77 (1)
14.09.2021	0,77 (2)	0,77 (2)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 11)

Datum	Code Ownership		
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)	Mittelwert Entwickler:in p.M.
18.09.2020	0,58 (2)	0,58 (1); 0,42 (1)	0,53
25.09.2020	0,58 (1)	0,58 (1)	0,53
27.10.2020	0,58 (4)	0,58 (3); 0,42 (1)	0,54
02.11.2020	0,68 (1)	0,32 (1)	0,32
03.12.2020	0,59 (2)	0,28 (2)	0,36
07.12.2020	0,49 (3)	0,42 (3)	0,36
12.01.2021	0,49 (1)	0,42 (1)	0,42
26.02.2021	0,49 (1)	0,42 (1)	0,42
01.03.2021	0,49 (1)	0,42 (1)	0,42
12.03.2021	0,49 (1)	0,42 (1)	0,42
28.06.2021	0,47 (1)	0,47 (1)	0,47
05.07.2021	0,47 (2)	0,47 (2)	0,47
06.07.2021	0,46 (1)	0,46 (1)	0,47
09.07.2021	0,48 (1)	0,48 (1)	0,47
12.07.2021	0,48 (1)	0,48 (1)	0,47
13.07.2021	0,48 (1)	0,48 (1)	0,47
15.07.2021	0,48 (1)	0,48 (1)	0,47
19.10.2021	0,51 (1)	0,51 (1)	0,51
22.10.2021	0,51 (1)	0,51 (1)	0,51

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 12)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
04.02.2021	1,00 (1)	1,00 (1)
05.02.2021	1,00 (3)	1,00 (3)
16.02.2021	1,00 (7)	1,00 (7)
19.10.2021	1,00 (1)	1,00 (1)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 14)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
27.08.2020	1,00 (8)	1,00 (8)
02.11.2020	1,00 (16)	1,00 (16)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 15)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
01.09.2020	1,00 (1)	1,00 (1)
03.12.2020	1,00 (2)	1,00 (2)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 16)

Datum	Code Ownership		
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)	Mittelwert Entwickler:in p.M.
18.08.2020	0,90 (1)	0,90 (1)	0,9
19.08.2020	0,90 (1)	0,90 (1)	0,9
11.09.2020	0,91 (1)	0,91 (1)	0,91
07.10.2020	0,91 (1)	0,91 (1)	0,91
11.12.2020	0,94 (2); 0,94 (2)	0,94 (2); 0,94 (2)	0,94
13.12.2020	0,94 (1)	0,94 (1)	0,94
14.12.2020	0,94 (4)	0,94 (4)	0,94
04.01.2021	0,94 (1)	0,94 (1)	0,94
01.02.2021	0,95 (1)	0,95 (1)	0,95
23.02.2021	0,96 (1)	0,96 (1)	0,95
24.02.2021	0,96 (1)	0,96 (1)	0,95
17.03.2021	0,96 (1)	0,96 (1)	0,96
22.03.2021	0,96 (2)	0,96 (2)	0,96
25.03.2021	0,96 (1)	0,96 (1)	0,96
04.05.2021	0,94 (1)	0,94 (1)	0,84
05.05.2021	0,89 (3)	0,89 (3)	0,84
09.05.2021	0,89 (1)	0,89 (1)	0,84
10.05.2021	0,89 (1)	0,89 (1)	0,84
13.05.2021	0,89 (1)	0,89 (1)	0,84
18.05.2021	0,89 (1)	0,89 (1)	0,84
20.05.2021	0,89 (4)	0,11 (1); 0,89 (3)	0,84
28.06.2021	0,90 (1)	0,90 (1)	0,9
30.06.2021	0,90 (3)	0,90 (3)	0,9
19.07.2021	0,90 (1)	0,90 (1)	0,9
29.07.2021	0,90 (1)	0,90 (1)	0,9
21.11.2021	0,87 (1)	0,87 (1)	0,87

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 18)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
16.06.2020	1,00 (3)	1,00 (3)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 19)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
27.10.2020	0,92 (2)	0,92 (2)
09.11.2020	0,92 (1)	0,92 (1)
16.03.2021	0,67 (10)	0,67 (10)
17.03.2021	0,67 (24)	0,67 (24)
18.03.2021	0,67 (24)	0,67 (24)
19.03.2021	0,67 (9)	0,67 (9)
17.06.2021	0,69 (1)	0,69 (1)
26.07.2021	0,67 (1)	0,67 (1)
07.12.2021	0,62 (3)	0,62 (3)

Quelle: Eigene Tabelle.

Tabelle: Fehlerprotokolle nach Code Ownership, personalisiert (Projekt 17)

Datum	Code Ownership	
	Projekt (+Anzahl)	Entwickler:in (+Anzahl)
15.07.2020	0,92 (1)	0,92 (1)
16.07.2020	0,92 (1)	0,92 (1)
17.07.2020	0,92 (1)	0,92 (1)
18.07.2020	0,92 (1)	0,92 (1)
19.07.2020	0,92 (1)	0,92 (1)
20.07.2020	0,92 (1)	0,92 (1)
21.07.2020	0,92 (1)	0,92 (1)
22.07.2020	0,92 (1)	0,92 (1)
23.07.2020	0,92 (1)	0,92 (1)
08.12.2020	0,93 (1)	0,93 (1)
26.01.2021	0,95 (1)	0,95 (1)
22.02.2021	0,95 (1)	0,95 (1)
09.03.2021	0,95 (1)	0,95 (1)
15.03.2021	0,95 (3)	0,95 (3)
01.04.2021	0,95 (1)	0,95 (1)
03.05.2021	0,95 (1)	0,95 (1)
12.05.2021	0,95 (1)	0,95 (1)
20.05.2021	0,95 (1)	0,95 (1)
30.06.2021	0,95 (2)	0,95 (2)
01.08.2021	0,95 (1)	0,95 (1)
01.09.2021	0,90 (1)	0,90 (1)
18.09.2021	0,90 (1)	0,90 (1)
19.09.2021	0,90 (1)	0,90 (1)
12.10.2021	0,90 (1)	0,90 (1)
28.10.2021	0,90 (1)	0,90 (1)
05.11.2021	0,90 (2)	0,90 (2)
14.11.2021	0,90 (1)	0,90 (1)
27.11.2021	0,90 (1)	0,90 (1)
27.12.2021	0,88 (2)	0,88 (2)

Quelle: Eigene Tabelle.

Anhang 3.6 Zu Abbildungen 14 bis 16 und Anhang 1.2

Die folgenden Tabellen zeigen die Änderungen der Code Ownership Werte der einzelnen Entwickler:innen in Bezug auf die Technologien.

Tabelle: Technologiebezogene Abhängigkeiten (ARM)

Entwickler:in	Jahr	
	2020	2021
1	-	0,521
10	1,000	0,479

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (Docker)

Entwickler:in	Jahr	
	2020	2021
1	0,903	0,742
10	0,097	0,258

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (DynamoDB)

Entwickler:in	Jahr	
	2020	2021
1	0,775	0,616
2	0,076	0,043
9	-	0,021
10	0,121	0,272
12	-	0,039
14	-	0,002
16	0,028	0,007

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (CloudFormation)

Entwickler:in	Jahr	
	2020	2021
1	0,904	0,913
3	0,015	0,011
4	-	0,002
10	-	0,014
16	0,081	0,06

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (TypeScript)

Entwickler:in	Jahr	
	2020	2021
1	0,477	0,354
2	0,266	0,109
3	0,038	0,015
9	-	0,021
10	0,088	0,15
11	-	0,006
12	0,100	0,097
13	0,032	0,215
14	-	0,004
15	-	0,016
16	-	0,013

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (Vue.js)

Entwickler:in	Jahr	
	2020	2021
1	0,446	0,323
2	0,282	0,115
3	0,040	0,016
9	-	0,022
10	0,093	0,157
11	-	0,006
12	0,106	0,102
13	0,034	0,225
14	-	0,004
15	-	0,017
16	-	0,013

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (Terraform)

Entwickler:in	Jahr	
	2020	2021
1	0,292	0,305
2	0,134	0,097
3	0,021	0,014
5	0,264	0,197
6	0,005	0,003
7	-	0,001
8	0,002	0,001
9	-	0,003
10	0,232	0,302
12	0,009	0,028
13	-	0,002
15	-	0,005
16	0,043	0,042

Quelle: Eigene Tabelle.

Tabelle: Technologiebezogene Abhängigkeiten (Python)

Entwickler:in	Jahr	
	2020	2021
1	0,515	0,482
2	0,074	0,043
3	0,022	0,009
9	-	0,007
10	0,293	0,369
12	-	0,015
14	-	0,001
16	0,097	0,075

Quelle: Eigene Tabelle.

Anhang 3.7 Zu Abbildung 17

Die folgende Tabelle zeigt die monatliche Entwicklungsaktivität (Anzahl Commits), kategorisiert nach Technologien. Ein Commit kann dabei mehreren Technologien zugeordnet sein.

Tabelle: Entwicklungsaktivität nach Technologie

Monat	Technologie							
	Python	Terraform	CloudFormation	ARM	Docker	Vue.js	TypeScript	DynamoDB
Mrz 2020	0	103	4	0	6	0	0	0
Apr 2020	0	12	14	0	3	0	0	0
Mai 2020	9	30	17	0	9	0	0	0
Jun 2020	74	85	31	4	0	4	4	0
Jul 2020	101	275	12	1	1	15	23	0
Aug 2020	44	130	33	9	5	84	84	0
Sep 2020	45	112	30	4	4	4	4	0
Okt 2020	132	219	28	0	4	16	16	23
Nov 2020	98	230	3	3	2	49	49	46
Dez 2020	55	89	4	0	2	5	5	12
Jan 2021	153	103	11	0	0	17	17	58
Feb 2021	93	141	0	0	0	11	11	19
Mrz 2021	45	62	22	0	4	22	22	25
Apr 2021	55	61	5	0	0	32	32	30
Mai 2021	61	49	4	0	0	19	19	27
Jun 2021	61	74	0	0	0	22	22	26
Jul 2021	36	116	0	0	5	33	41	17
Aug 2021	98	35	3	0	0	14	16	9
Sep 2021	40	50	1	0	4	19	23	15
Okt 2021	53	69	7	7	0	31	31	15
Nov 2021	25	19	0	5	0	15	15	9
Dez 2021	0	3	7	0	0	0	0	0

Quelle: Eigene Tabelle.

Anhang 3.8 Zu Abbildung 18 und Anhang 2.5

Die folgenden Tabellen zeigen die Anzahl der Aufgaben mit einer bestimmten Anzahl Sprints bis zu ihrem Abschluss in Assoziation mit dem Code Ownership des Projekts.

Tabelle: Verzögerungen in der Entwicklung (Kategorien)

Bereich	Anzahl Sprints						
	0	1	2	3	4	5	10
0 - 10 %	0	0	0	0	0	0	0
11 - 20 %	0	0	0	0	0	0	0
21 - 30 %	0	2	2	0	0	0	0
31 - 40 %	1	40	10	10	2	3	0
41 - 50 %	0	84	37	14	1	5	0
51 - 60 %	1	42	11	4	0	1	0
61 - 70 %	0	27	8	8	2	4	0
71 - 80 %	0	21	10	1	0	2	0
81 - 90 %	0	33	11	3	2	2	0
91 - 100 %	1	123	35	15	2	2	2
Summe	3	372	124	55	9	19	2

Quelle: Eigene Tabelle.

Tabelle: Verzögerungen in der Entwicklung (Mittelwerte)

Anzahl Sprints	Code Ownership
0	0,62
1	0,69
2	0,67
3	0,64
4	0,68
5	0,62
10	1

Quelle: Eigene Tabelle.

Tabelle: Verzögerungen in der Entwicklung (Detail)

Anzahl Sprints	Code Ownership (+Anzahl)
0	0,32 (1); 0,53 (1); 1,00 (1)
1	0,23 (1); 0,26 (1); 0,31 (5); 0,32 (10); 0,33 (3); 0,34 (4); 0,35 (2); 0,37 (3); 0,38 (4); 0,39 (5); 0,40 (9); 0,41 (2); 0,42 (2); 0,43 (7); 0,44 (10); 0,45 (12); 0,46 (7); 0,47 (2); 0,48 (10); 0,49 (25); 0,50 (10); 0,51 (1); 0,52 (3); 0,53 (2); 0,54 (6); 0,55 (3); 0,56 (1); 0,57 (7); 0,58 (6); 0,59 (2); 0,60 (3); 0,61 (2); 0,62 (1); 0,63 (4); 0,64 (1); 0,65 (3); 0,66 (3); 0,67 (4); 0,68 (5); 0,69 (4); 0,71 (2); 0,73 (1); 0,74 (1); 0,75 (1); 0,76 (5); 0,77 (2); 0,78 (4); 0,79 (3); 0,80 (3); 0,81 (1); 0,82 (1); 0,83 (3); 0,84 (5); 0,85 (6); 0,86 (1); 0,87 (3); 0,88 (5); 0,89 (3); 0,90 (7); 0,91 (8); 0,92 (5); 0,93 (8); 0,94 (10); 0,95 (6); 0,96 (6); 0,98 (2); 0,99 (2); 1,00 (73)
2	0,26 (1); 0,27 (1); 0,31 (1); 0,32 (4); 0,33 (1); 0,34 (1); 0,39 (1); 0,40 (3); 0,41 (2); 0,42 (2); 0,43 (1); 0,44 (5); 0,45 (4); 0,46 (4); 0,47 (1); 0,48 (4); 0,49 (10); 0,50 (4); 0,51 (3); 0,52 (2); 0,53 (2); 0,54 (2); 0,56 (1); 0,60 (2); 0,64 (1); 0,65 (2); 0,68 (1); 0,69 (1); 0,70 (1); 0,71 (1); 0,72 (1); 0,73 (2); 0,74 (1); 0,75 (1); 0,77 (1); 0,78 (1); 0,79 (2); 0,83 (2); 0,85 (1); 0,86 (1); 0,87 (3); 0,88 (1); 0,89 (2); 0,90 (5); 0,91 (1); 0,92 (2); 0,93 (5); 0,94 (4); 0,95 (2); 0,96 (3); 0,97 (1); 0,98 (1); 1,00 (12)
3	0,32 (2); 0,35 (1); 0,36 (2); 0,37 (1); 0,38 (1); 0,39 (2); 0,40 (1); 0,41 (1); 0,42 (1); 0,43 (1); 0,45 (1); 0,46 (1); 0,48 (2); 0,49 (7); 0,52 (1); 0,56 (1); 0,57 (1); 0,59 (1); 0,60 (1); 0,61 (1); 0,62 (1); 0,63 (1); 0,64 (1); 0,66 (1); 0,69 (2); 0,70 (1); 0,80 (1); 0,83 (1); 0,87 (1); 0,91 (2); 0,93 (3); 0,95 (1); 1,00 (9)
4	0,32 (1); 0,33 (1); 0,48 (1); 0,61 (1); 0,65 (1); 0,88 (1); 0,90 (1); 0,95 (1); 0,97 (1)
5	0,34 (1); 0,36 (1); 0,39 (1); 0,46 (1); 0,47 (2); 0,48 (1); 0,49 (1); 0,53 (1); 0,60 (1); 0,62 (1); 0,64 (1); 0,69 (1); 0,73 (1); 0,76 (1); 0,87 (2); 0,94 (1); 1,00 (1)
10	1,00 (2)

Quelle: Eigene Tabelle.

Anhang 3.9 Zu Abbildungen 19 und 20 und Anhängen 1.3 und 2.6

Die folgenden Tabellen zeigen die Entwicklungsaktivität (Anzahl Commits - C) und die Anzahl der Reviews (R) der Entwickler:innen, aufgeschlüsselt nach Technologien.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 1)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
16.03.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
17.03.2020	-	-	3	-	3	-	-	-	3	-	-	-	-	-	-	-
19.03.2020	-	-	1	-	-	-	-	-	3	-	-	-	-	-	-	-
23.03.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
25.03.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
17.04.2020	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-
20.04.2020	-	-	6	-	5	-	-	-	-	-	-	-	-	-	-	-
28.04.2020	-	-	-	-	6	-	-	-	3	-	-	-	-	-	-	-
29.04.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
08.05.2020	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-
11.05.2020	-	-	-	1	6	-	-	-	-	-	-	-	-	-	-	-
12.05.2020	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-
15.05.2020	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
18.05.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
19.05.2020	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-
20.05.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
25.05.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
26.05.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
27.05.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
28.05.2020	8	-	3	-	3	-	-	-	4	-	-	-	-	-	-	-
29.05.2020	1	-	1	-	-	-	-	-	5	-	-	-	-	-	-	-
04.06.2020	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.06.2020	11	-	9	-	2	-	-	-	-	-	-	-	-	-	-	-
08.06.2020	6	-	17	-	8	-	-	-	-	-	-	-	-	-	-	-
10.06.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15.06.2020	6	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
16.06.2020	2	-	4	-	2	-	-	-	-	-	-	-	-	-	-	-
17.06.2020	6	-	5	-	7	-	-	-	-	-	-	-	-	-	-	-
18.06.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
19.06.2020	-	-	2	-	3	-	-	-	-	-	-	-	-	-	-	-
22.06.2020	-	-	-	-	8	-	-	-	-	-	-	-	-	-	-	-
26.06.2020	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
29.06.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
01.07.2020	2	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
02.07.2020	8	-	12	-	8	-	-	-	-	-	-	-	-	-	-	-
03.07.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
06.07.2020	21	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-
07.07.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
08.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
28.07.2020	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-
29.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	1	-	-	-
30.07.2020	-	-	14	-	-	-	-	-	-	-	-	-	3	-	-	-
31.07.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
03.08.2020	-	-	-	-	12	-	-	-	-	-	-	-	-	-	-	-
07.08.2020	3	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
10.08.2020	5	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
12.08.2020	-	-	16	-	1	-	-	-	-	-	-	-	-	-	-	-
13.08.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14.08.2020	4	-	2	-	1	-	-	-	-	-	-	-	-	-	-	-
17.08.2020	-	-	7	-	4	-	-	-	5	-	-	-	-	-	-	-
18.08.2020	-	-	4	-	1	-	-	-	-	-	-	-	-	-	-	-
19.08.2020	-	-	3	-	-	-	-	-	-	-	3	-	3	-	-	-
20.08.2020	-	-	-	-	9	-	-	-	-	-	6	-	6	-	-	-
21.08.2020	1	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
24.08.2020	2	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
25.08.2020	-	-	-	-	-	-	-	-	-	-	4	-	4	-	-	-
26.08.2020	2	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-
31.08.2020	5	-	7	-	-	-	-	-	-	-	6	-	6	-	-	-
01.09.2020	5	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
02.09.2020	6	-	-	-	1	-	-	-	-	-	2	-	2	-	-	-
03.09.2020	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
04.09.2020	-	-	4	-	2	-	-	-	-	-	-	-	-	-	-	-
09.09.2020	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-
10.09.2020	-	-	-	-	17	-	-	-	-	-	-	-	-	-	-	-
11.09.2020	-	-	2	-	2	-	-	-	-	-	-	-	-	-	-	-
15.09.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.09.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
18.09.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
21.09.2020	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
22.09.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
05.10.2020	-	-	3	1	-	-	-	-	-	-	-	-	-	-	3	-
08.10.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
09.10.2020	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-
12.10.2020	2	-	-	-	2	-	-	-	-	-	-	-	-	-	2	-
15.10.2020	5	-	-	-	-	-	-	-	4	-	-	-	-	-	5	-
16.10.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19.10.2020	3	-	5	-	-	-	-	-	-	-	-	-	-	-	4	-
20.10.2020	3	-	2	-	-	-	-	-	-	-	-	-	-	-	1	-
21.10.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
23.10.2020	10	-	6	-	-	-	-	-	-	-	-	-	-	-	4	-
26.10.2020	-	-	3	-	-	-	-	-	-	-	2	-	2	-	-	-
27.10.2020	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
28.10.2020	3	-	16	-	-	-	-	-	-	-	-	-	-	-	-	-
29.10.2020	8	-	18	-	-	-	-	-	-	-	-	-	-	-	-	-
30.10.2020	-	1	4	1	-	-	-	-	-	-	-	-	-	-	-	-
02.11.2020	-	-	5	4	-	-	-	-	-	-	-	-	-	-	-	-
03.11.2020	-	-	4	-	-	-	-	-	2	-	2	-	2	-	-	-
04.11.2020	3	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
05.11.2020	3	-	3	1	-	-	-	-	-	-	-	-	-	-	6	-
06.11.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
09.11.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
10.11.2020	-	-	10	1	-	-	-	-	-	-	-	-	-	-	-	-
11.11.2020	2	-	8	-	-	-	-	-	-	-	-	-	-	-	7	-
12.11.2020	5	1	-	-	-	-	-	-	-	-	-	-	-	-	5	1
13.11.2020	3	-	1	1	-	-	-	-	-	-	-	-	-	-	3	-
16.11.2020	3	-	7	1	-	-	-	-	-	-	-	-	-	-	3	-
17.11.2020	3	-	3	-	-	-	-	-	-	-	-	-	-	-	3	-
18.11.2020	6	-	1	-	-	-	-	-	-	-	-	-	-	-	4	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
19.11.2020	2	-	4	-	1	-	-	-	-	-	-	-	-	-	-	-
20.11.2020	1	-	5	2	2	-	-	-	-	-	-	-	-	-	-	-
24.11.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.11.2020	-	-	5	-	-	-	-	-	-	-	6	-	6	-	-	-
26.11.2020	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-	-
27.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.11.2020	5	-	5	-	-	-	-	-	-	-	-	-	-	-	5	-
01.12.2020	1	1	-	1	-	-	-	-	-	-	-	-	-	-	-	1
02.12.2020	-	-	2	1	-	-	-	-	-	-	-	-	-	-	-	-
03.12.2020	6	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
08.12.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
10.12.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11.12.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
14.12.2020	-	-	-	-	4	-	-	-	-	-	-	1	-	1	-	-
15.12.2020	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
18.12.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
21.12.2020	4	-	1	-	-	-	-	-	2	-	-	-	-	-	-	-
22.12.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
05.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
06.01.2021	2	1	5	-	-	-	-	-	-	-	-	-	-	-	2	-
07.01.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
08.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
11.01.2021	7	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
13.01.2021	3	-	-	1	2	-	-	-	-	-	-	-	-	-	3	-
14.01.2021	9	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
18.01.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
19.01.2021	-	2	10	-	-	-	-	-	-	-	-	-	-	-	-	-
20.01.2021	6	1	-	-	-	-	-	-	-	-	-	-	-	-	6	1
21.01.2021	9	1	13	-	-	-	-	-	-	-	-	-	-	-	9	1
22.01.2021	4	1	3	-	-	-	-	-	-	-	-	-	-	-	1	-
25.01.2021	4	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
26.01.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
27.01.2021	2	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-
28.01.2021	1	-	2	-	1	-	-	-	-	-	-	-	-	-	-	-
29.01.2021	2	-	4	-	2	-	-	-	-	-	-	-	-	-	-	-
01.02.2021	1	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
02.02.2021	2	1	6	-	-	-	-	-	-	-	-	-	-	-	-	-
03.02.2021	1	1	-	-	-	-	-	-	-	-	-	-	-	-	1	1
05.02.2021	2	2	-	1	-	-	-	-	-	-	-	-	-	-	-	1
08.02.2021	3	2	3	-	-	-	-	-	-	-	-	-	-	-	-	-
09.02.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10.02.2021	1	-	1	-	-	-	-	-	-	-	-	-	-	-	1	-
11.02.2021	3	-	5	1	-	-	-	-	-	-	-	1	-	1	3	-
16.02.2021	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.02.2021	-	-	3	1	-	-	-	-	-	-	-	-	-	-	-	-
18.02.2021	5	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
22.02.2021	1	-	4	-	-	-	-	-	-	-	-	-	-	-	1	-
24.02.2021	4	-	5	-	-	-	-	-	-	-	-	-	-	-	1	-
01.03.2021	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
05.03.2021	2	-	4	1	-	-	-	-	-	-	-	-	-	-	2	-
08.03.2021	12	2	8	-	-	-	-	-	-	-	-	-	-	-	12	-
09.03.2021	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
10.03.2021	2	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
11.03.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
12.03.2021	5	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
14.03.2021	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
16.03.2021	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-
19.03.2021	8	-	6	1	-	-	-	-	-	-	-	1	-	1	8	-
24.03.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
25.03.2021	3	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
29.03.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30.03.2021	-	-	-	1	-	-	-	-	-	-	3	1	3	1	-	-
31.03.2021	-	-	-	-	12	-	-	-	-	-	1	1	1	1	-	-
01.04.2021	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-
06.04.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
07.04.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
08.04.2021	2	-	-	-	-	-	-	-	-	-	1	-	1	-	2	-
09.04.2021	12	-	-	-	-	-	-	-	-	-	1	1	1	1	1	-
12.04.2021	25	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-
13.04.2021	-	-	29	-	-	-	-	-	-	-	1	-	1	-	-	-
15.04.2021	-	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-
22.04.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1
26.04.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
28.04.2021	4	-	1	1	-	-	-	-	-	-	4	-	4	-	5	-
29.04.2021	1	-	4	-	-	-	-	-	-	-	-	-	-	-	4	-
30.04.2021	5	-	4	-	-	-	-	-	-	-	1	-	1	-	4	-
03.05.2021	11	-	8	-	1	-	-	-	-	-	-	-	-	-	-	-
04.05.2021	5	1	11	-	1	-	-	-	-	-	-	1	-	1	-	1
06.05.2021	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-
17.05.2021	-	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-
27.05.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
28.05.2021	3	1	-	3	-	-	-	-	-	-	-	-	-	-	3	1
31.05.2021	-	1	-	-	-	-	-	-	-	-	4	1	4	1	-	1
01.06.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
02.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
07.06.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
08.06.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
09.06.2021	-	1	-	1	-	-	-	-	-	-	3	-	3	-	-	-
10.06.2021	4	-	3	-	-	-	-	-	-	-	-	-	-	-	4	-
11.06.2021	1	6	1	5	-	-	-	-	-	-	-	-	-	-	1	2
14.06.2021	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
17.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.06.2021	-	1	-	-	-	-	-	-	-	-	1	-	1	-	-	1
23.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
24.06.2021	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	1
28.06.2021	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
30.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
01.07.2021	-	-	-	2	-	-	-	-	-	1	-	1	1	1	-	-
02.07.2021	-	-	1	1	-	-	-	-	-	1	1	-	4	-	-	-
05.07.2021	-	-	1	1	-	-	-	-	-	-	-	-	4	-	-	-
06.07.2021	-	1	13	-	-	-	-	-	-	-	2	-	2	-	-	-
07.07.2021	2	1	1	-	-	-	-	-	-	-	2	-	2	-	-	-
08.07.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
09.07.2021	1	-	10	1	-	-	-	-	-	-	-	-	-	-	7	-
12.07.2021	1	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-
13.07.2021	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
14.07.2021	-	-	1	2	-	-	-	-	-	-	-	-	-	-	-	-
15.07.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
16.07.2021	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.08.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
06.08.2021	3	-	7	1	-	-	-	-	-	-	-	-	-	-	-	-
09.08.2021	-	-	2	1	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
11.08.2021	1	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
12.08.2021	2	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
16.08.2021	-	-	-	-	-	-	-	-	-	-	6	-	6	-	-	-
17.08.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
19.08.2021	90	-	-	-	-	-	-	-	-	-	-	-	-	-	8	-
20.08.2021	-	-	6	-	3	-	-	-	-	-	-	-	-	-	-	-
26.08.2021	-	-	1	-	-	-	-	-	-	-	-	-	2	-	-	-
27.08.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
30.08.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
01.09.2021	2	-	1	-	-	-	-	-	-	-	-	-	4	-	2	-
02.09.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
03.09.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
07.09.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
21.09.2021	-	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-
23.09.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
29.09.2021	-	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-
30.09.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
01.10.2021	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-
07.10.2021	-	8	-	-	-	-	-	-	-	-	3	-	3	-	-	1
11.10.2021	1	4	4	-	-	-	-	-	-	-	1	-	1	-	-	-
12.10.2021	-	-	2	2	-	-	3	2	-	-	1	1	1	1	-	-
13.10.2021	2	-	1	-	4	-	-	-	-	-	1	-	1	-	2	-
14.10.2021	4	-	2	-	-	-	-	-	-	-	2	-	2	-	1	-
15.10.2021	10	-	8	1	-	-	-	-	-	-	-	-	-	-	5	-
18.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
19.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
20.10.2021	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	1
21.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
29.10.2021	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
02.11.2021	-	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-
15.11.2021	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-
16.11.2021	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-
24.11.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
29.11.2021	-	2	-	-	-	-	-	-	-	-	1	-	1	-	-	-
20.12.2021	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 2)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
29.06.2020	-	-	16	-	-	-	-	-	-	-	-	-	-	-	-	-
30.06.2020	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
03.07.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
08.07.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
13.07.2020	-	-	14	-	-	-	-	-	-	-	-	-	-	-	-	-
14.07.2020	-	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
15.07.2020	-	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-
17.07.2020	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
21.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
22.07.2020	-	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
29.07.2020	-	-	-	-	-	-	-	-	-	-	4	-	4	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
30.07.2020	-	-	2	-	-	-	-	-	-	-	5	-	5	-	-	-
31.07.2020	-	-	1	-	-	-	-	-	-	-	3	-	3	-	-	-
03.08.2020	-	-	6	-	-	-	-	-	-	-	2	-	2	-	-	-
04.08.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
05.08.2020	-	-	-	-	-	-	-	-	-	-	10	-	10	-	-	-
06.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
07.08.2020	-	-	2	-	-	-	-	-	-	-	7	-	7	-	-	-
11.08.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
12.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
17.08.2020	-	-	1	-	-	-	-	-	-	-	6	-	6	-	-	-
18.08.2020	-	-	8	-	-	-	-	-	-	-	3	-	3	-	-	-
19.08.2020	-	-	-	-	-	-	-	-	-	-	5	-	5	-	-	-
02.10.2020	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
05.10.2020	4	-	12	-	-	-	-	-	-	-	-	-	-	-	-	-
08.10.2020	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
12.10.2020	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
13.10.2020	2	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
14.10.2020	8	-	14	-	-	-	-	-	-	-	-	-	-	-	-	-
29.10.2020	-	-	3	-	-	-	-	-	-	-	4	-	4	-	-	-
01.11.2020	-	-	5	-	-	-	-	-	-	-	6	-	6	-	-	-
02.11.2020	-	-	20	-	-	-	-	-	-	-	1	-	1	-	-	-
11.11.2020	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
12.11.2020	2	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
16.11.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
17.11.2020	-	-	3	-	-	-	-	-	-	-	3	-	3	-	-	-
19.11.2020	4	-	1	-	-	-	-	-	-	-	2	-	2	-	-	-
20.11.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
23.11.2020	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
24.11.2020	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
27.11.2020	3	-	15	-	-	-	-	-	-	-	-	-	-	-	-	-
01.12.2020	1	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
02.12.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
03.12.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
07.12.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
08.12.2020	2	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
09.12.2020	1	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
10.12.2020	1	-	15	-	-	-	-	-	-	-	1	-	1	-	-	-
11.12.2020	7	1	-	-	-	-	-	-	-	-	-	-	-	-	7	-
14.12.2020	6	-	5	-	-	-	-	-	-	-	1	-	1	-	2	-
07.01.2021	7	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
08.01.2021	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
11.01.2021	1	-	1	-	-	-	-	-	-	-	-	-	-	-	1	-
12.01.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
13.01.2021	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
18.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
19.01.2021	7	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-
20.01.2021	7	-	7	-	-	-	-	-	-	-	-	-	-	-	4	-
22.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
26.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
27.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
28.01.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
29.01.2021	-	1	-	-	-	1	-	-	-	-	-	-	-	-	-	-
01.02.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 3)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
21.07.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
22.07.2020	2	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
27.07.2020	2	-	4	-	2	-	-	-	-	-	-	-	-	-	-	-
03.08.2020	-	-	2	-	-	-	-	-	-	-	2	-	2	-	-	-
06.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
07.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
10.08.2020	-	-	6	-	-	-	-	-	-	-	4	-	4	-	-	-
28.08.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
31.08.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
03.09.2020	2	-	5	-	3	-	-	-	-	-	-	-	-	-	-	-
04.09.2020	4	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
09.10.2020	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
15.10.2020	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
20.10.2020	-	-	-	-	-	-	-	-	-	-	1	1	1	1	-	-
19.11.2020	-	1	-	-	-	-	-	-	-	-	1	1	1	1	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 4)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
29.03.2021	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
31.03.2021	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 6)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
21.08.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
31.08.2020	-	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 7)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
16.07.2021	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 5)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
02.03.2020	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
04.03.2020	-	-	15	-	-	-	-	-	-	-	-	-	-	-	-	-
05.03.2020	-	-	29	-	-	-	-	-	-	-	-	-	-	-	-	-
06.03.2020	-	-	25	-	-	-	-	-	-	-	-	-	-	-	-	-
10.03.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
17.03.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
18.03.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
19.03.2020	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
23.03.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
24.03.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
26.03.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.03.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
03.04.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
06.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
07.07.2020	-	-	12	-	-	-	-	-	-	-	-	-	-	-	-	-
08.07.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
09.07.2020	-	-	19	-	-	-	-	-	-	-	-	-	-	-	-	-
10.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
27.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
29.07.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
19.08.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
21.08.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
25.08.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 9)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
23.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
04.10.2021	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
13.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
14.10.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
15.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
18.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
19.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
20.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
28.10.2021	1	-	-	-	-	-	-	-	-	-	2	-	2	-	1	-
02.11.2021	1	-	-	-	-	-	-	-	-	-	1	-	1	-	1	-
03.11.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
11.11.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
12.11.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 8)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
26.10.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
27.10.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.10.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 10)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
27.04.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.04.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
05.05.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
11.05.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
09.06.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10.06.2020	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.06.2020	9	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
15.06.2020	24	-	16	-	-	-	-	-	-	-	-	-	-	-	-	-
29.06.2020	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-
06.07.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
07.07.2020	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08.07.2020	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
09.07.2020	27	-	30	-	-	-	-	-	-	-	-	-	-	-	-	-
10.07.2020	1	-	6	-	-	-	-	-	1	-	-	-	-	-	-	-
13.07.2020	6	-	29	-	-	-	1	-	-	-	-	-	-	-	-	-
17.07.2020	5	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
22.07.2020	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23.07.2020	12	-	15	-	-	-	-	-	-	-	-	-	-	-	-	-
27.07.2020	9	-	13	-	-	-	-	-	-	-	-	-	-	-	-	-
30.07.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
31.07.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
03.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
04.08.2020	-	-	-	-	-	-	-	-	-	-	7	-	7	-	-	-
05.08.2020	8	-	12	-	-	-	-	-	-	-	2	-	2	-	-	-
06.08.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
07.08.2020	-	-	6	-	-	-	-	-	-	-	4	-	4	-	-	-
26.08.2020	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
27.08.2020	11	-	13	-	-	-	8	-	-	-	-	-	-	-	-	-
02.09.2020	-	-	2	-	-	-	-	-	4	-	-	-	-	-	-	-
03.09.2020	-	-	4	1	-	-	-	-	-	-	-	-	-	-	-	-
04.09.2020	3	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-
07.09.2020	3	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
09.09.2020	-	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
11.09.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2020	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
17.09.2020	-	-	2	1	-	-	-	-	-	-	-	-	-	-	-	-
21.09.2020	9	-	23	-	-	-	-	-	-	-	-	-	-	-	-	-
25.09.2020	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
15.10.2020	7	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
19.10.2020	3	-	14	-	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
20.10.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
22.10.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23.10.2020	1	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
27.10.2020	6	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
28.10.2020	2	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
29.10.2020	9	-	30	-	-	-	-	-	-	-	-	-	-	-	-	-
30.10.2020	15	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
02.11.2020	1	-	15	-	-	-	3	-	-	-	2	1	2	1	-	-
03.11.2020	2	-	4	-	-	-	-	-	-	-	1	1	1	1	-	-
04.11.2020	3	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
05.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
06.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
09.11.2020	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
10.11.2020	-	-	3	1	-	-	-	-	-	-	-	-	-	-	-	-
11.11.2020	-	1	-	2	-	-	-	-	-	-	-	-	-	-	-	2
12.11.2020	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
13.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
16.11.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
18.11.2020	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
20.11.2020	-	-	12	-	-	-	-	-	-	-	-	-	-	-	-	-
23.11.2020	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
25.11.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
26.11.2020	1	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-
27.11.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
30.11.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
01.12.2020	2	-	-	1	-	-	-	-	-	-	-	-	-	-	2	-
07.12.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
08.12.2020	1	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-
09.12.2020	2	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
10.12.2020	2	-	3	2	-	-	-	-	-	-	-	-	-	-	-	-
11.12.2020	8	-	17	-	-	-	-	-	-	-	-	-	-	-	1	-
14.12.2020	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15.12.2020	6	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
04.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.01.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
06.01.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
07.01.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11.01.2021	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
12.01.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13.01.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14.01.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
15.01.2021	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
18.01.2021	3	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
19.01.2021	6	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
20.01.2021	1	1	-	-	-	-	-	-	-	-	-	-	-	-	1	1
21.01.2021	5	-	1	-	-	-	-	-	-	-	-	-	-	-	4	-
22.01.2021	9	-	14	1	-	-	-	-	-	-	-	-	-	-	-	-
25.01.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.01.2021	4	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
27.01.2021	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
29.01.2021	2	1	-	-	-	1	-	-	-	-	-	-	-	-	1	-
01.02.2021	2	-	-	1	-	-	-	-	-	-	-	1	-	1	1	-
02.02.2021	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
03.02.2021	7	2	5	-	-	-	-	-	-	-	-	-	-	-	6	2
04.02.2021	3	-	3	-	-	-	-	-	-	-	-	-	-	-	1	-
05.02.2021	12	-	5	-	-	-	-	-	-	-	-	-	-	-	4	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
08.02.2021	20	1	32	-	-	-	-	-	-	-	-	-	-	-	-	-
09.02.2021	4	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
10.02.2021	-	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-
11.02.2021	-	1	2	1	-	-	-	-	-	-	-	-	-	-	-	1
16.02.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
17.02.2021	-	-	3	-	-	-	-	-	-	-	1	-	1	-	-	-
19.02.2021	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
22.02.2021	4	1	1	1	-	-	-	-	-	-	-	-	-	-	-	1
23.02.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
24.02.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
02.03.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.03.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
08.03.2021	3	2	7	1	-	-	-	-	-	-	-	-	-	-	-	2
09.03.2021	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
12.03.2021	-	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-
17.03.2021	-	1	-	-	2	1	-	-	-	-	-	-	-	-	-	1
01.04.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
12.04.2021	-	12	2	-	-	-	-	-	-	-	-	-	-	-	2	1
13.04.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
14.04.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	2	-
15.04.2021	-	-	-	-	-	-	-	-	-	-	4	1	4	1	-	-
16.04.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
19.04.2021	3	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
20.04.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	1	-
22.04.2021	-	-	7	-	-	-	-	-	-	-	-	-	-	-	3	-
26.04.2021	-	-	3	-	-	-	-	-	-	-	3	-	3	-	-	-
28.04.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.04.2021	-	2	-	2	-	-	-	-	-	-	-	1	-	1	-	2
03.05.2021	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
04.05.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.05.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
06.05.2021	-	1	-	-	-	1	-	-	-	-	-	-	-	-	-	-
10.05.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.05.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
14.05.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
17.05.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
26.05.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
27.05.2021	-	2	1	-	-	-	-	-	-	-	-	-	-	-	-	1
28.05.2021	3	-	4	1	-	-	-	-	-	-	1	-	1	-	3	-
31.05.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
01.06.2021	1	-	-	-	-	-	-	-	-	-	2	-	2	-	1	-
02.06.2021	2	-	3	-	-	-	-	-	-	-	-	-	-	-	2	-
07.06.2021	3	-	1	-	-	-	-	-	-	-	-	-	-	-	2	-
08.06.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09.06.2021	4	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
10.06.2021	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	1
11.06.2021	22	-	15	-	-	-	-	-	-	-	-	-	-	-	6	-
14.06.2021	3	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
16.06.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
17.06.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
18.06.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-
21.06.2021	2	-	5	-	-	-	-	-	-	-	-	-	-	-	2	-
22.06.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
23.06.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
24.06.2021	6	-	11	-	-	-	-	-	-	-	-	-	-	-	2	-
28.06.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
30.06.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
01.07.2021	-	-	3	-	-	-	-	-	4	-	-	1	-	1	-	-
02.07.2021	-	-	4	-	-	-	-	-	1	-	-	1	-	1	-	-
06.07.2021	3	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
07.07.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
08.07.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
09.07.2021	-	-	4	3	-	-	-	-	-	-	-	-	-	-	-	2
12.07.2021	7	-	1	-	-	-	-	-	-	-	-	-	-	-	3	-
14.07.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
15.07.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.07.2021	7	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
19.07.2021	-	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-
21.07.2021	-	-	12	-	-	-	-	-	-	-	-	-	-	-	1	-
22.07.2021	-	-	-	1	-	-	-	-	-	-	2	-	2	-	-	-
28.07.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
29.07.2021	12	-	9	-	-	-	-	-	-	-	1	-	1	-	3	-
30.07.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
03.08.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
04.08.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
05.08.2021	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
06.08.2021	-	1	4	-	-	-	-	-	-	-	-	-	-	-	-	-
09.08.2021	-	-	3	1	-	-	-	-	-	-	-	-	-	-	-	-
01.09.2021	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-
03.09.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
06.09.2021	13	1	-	-	-	-	-	-	4	-	-	-	-	-	2	1
07.09.2021	-	-	11	-	-	-	-	-	-	-	2	1	2	1	-	-
09.09.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
14.09.2021	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
15.09.2021	2	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.09.2021	1	-	2	-	-	-	-	-	-	-	-	-	-	-	1	-
20.09.2021	1	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
21.09.2021	5	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
22.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
23.09.2021	-	-	3	2	-	-	-	-	-	-	-	-	-	-	-	-
28.09.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
29.09.2021	1	-	3	-	-	-	-	-	-	-	-	1	-	1	1	-
30.09.2021	4	1	-	-	-	-	-	-	-	-	-	-	-	-	2	1
01.10.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
04.10.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
06.10.2021	5	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
07.10.2021	16	-	9	-	-	-	1	-	-	-	-	-	-	-	2	-
08.10.2021	5	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
11.10.2021	7	-	13	1	-	-	-	-	-	-	-	-	-	-	-	-
12.10.2021	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-
13.10.2021	-	-	-	1	-	2	-	-	-	-	-	-	-	-	-	-
14.10.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
15.10.2021	-	3	-	1	-	-	-	-	-	-	-	-	-	-	-	3
20.10.2021	-	-	11	-	-	-	-	-	-	-	-	-	-	-	3	-
21.10.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
22.10.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
26.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
02.11.2021	8	2	5	-	-	-	-	-	-	-	3	1	3	1	8	2
04.11.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
05.11.2021	2	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
10.11.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
18.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
22.11.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
23.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
24.11.2021	-	-	-	-	-	-	-	-	-	-	2	1	2	1	-	-
25.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.11.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09.12.2021	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
10.12.2021	-	-	1	-	1	-	-	-	-	-	-	-	-	-	-	-
13.12.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 11)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
04.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
22.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
27.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 12)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
26.10.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
27.10.2020	-	-	-	-	-	-	-	-	-	-	4	-	4	-	-	-
28.10.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
30.10.2020	-	-	2	-	-	-	-	-	-	-	2	-	2	-	-	-
02.11.2020	-	-	3	-	-	-	-	-	-	-	1	-	1	-	-	-
04.11.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
05.11.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
06.11.2020	-	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
11.11.2020	-	-	1	-	-	-	-	-	-	-	6	1	6	1	-	-
12.11.2020	-	-	1	-	-	-	-	-	-	-	2	1	2	1	-	-
13.11.2020	-	-	3	-	-	-	-	-	-	-	4	-	4	-	-	-
18.01.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
20.01.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
27.01.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
01.02.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
11.02.2021	-	1	-	2	-	-	-	-	-	-	5	-	5	-	-	1
17.02.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
02.03.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
04.03.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
05.03.2021	-	-	4	1	-	-	-	-	-	-	-	-	-	-	-	-
10.03.2021	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-	-
17.03.2021	2	1	-	-	-	1	-	-	-	-	4	-	4	-	2	1
19.03.2021	-	1	-	-	-	-	-	-	-	-	2	-	2	-	-	1

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
25.03.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
26.03.2021	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
29.03.2021	-	1	-	-	-	-	-	-	-	-	1	-	1	-	-	-
30.03.2021	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
31.03.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
06.04.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
08.04.2021	-	1	-	-	-	-	-	-	-	-	1	-	1	-	-	1
13.04.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
14.04.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1
15.04.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
16.04.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
27.04.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
28.04.2021	-	-	-	-	-	-	-	-	-	-	2	1	2	1	-	-
29.04.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1
30.04.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
03.05.2021	8	2	-	-	-	-	-	-	-	-	-	-	-	-	8	-
04.05.2021	2	-	-	2	-	-	-	-	-	-	-	-	-	-	2	-
07.05.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
12.05.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17.05.2021	-	1	-	-	-	-	-	-	-	-	-	2	-	2	-	1
27.05.2021	7	-	6	-	-	-	-	-	-	-	-	-	-	-	5	-
28.05.2021	1	1	8	-	-	-	-	-	-	-	1	-	1	-	1	1
14.06.2021	-	1	-	2	-	-	-	-	-	-	-	-	-	-	-	-
16.06.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
21.06.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
22.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
25.06.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
28.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
29.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
01.07.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
06.07.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
07.07.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.07.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
13.07.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
16.07.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
21.07.2021	-	-	1	-	-	-	-	-	-	-	-	1	-	1	-	-
22.07.2021	-	-	15	13	-	-	-	-	-	-	-	-	-	-	1	1
23.07.2021	-	-	-	-	-	-	-	-	-	-	1	2	1	2	-	-
26.07.2021	-	-	6	-	-	-	-	-	-	-	-	1	-	1	-	-
29.07.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
30.07.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
05.08.2021	-	-	-	1	-	-	-	-	-	-	-	1	-	1	-	-
09.08.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
10.08.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
11.08.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
03.09.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
14.09.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
15.09.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.09.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
27.09.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
28.09.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
29.09.2021	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
30.09.2021	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
04.10.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
06.10.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
12.10.2021	-	-	3	1	-	-	-	-	-	-	1	1	1	1	-	-
14.10.2021	-	-	2	-	-	-	-	-	-	-	-	1	-	1	-	-
15.10.2021	-	1	6	3	-	-	-	-	-	-	-	-	-	-	-	-
22.10.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
02.11.2021	-	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-
04.11.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
11.11.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
26.11.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
29.11.2021	6	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-
10.12.2021	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
13.12.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 14)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
05.07.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
03.09.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
07.09.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
29.09.2021	-	-	-	-	-	-	-	-	-	-	1	1	1	1	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 15)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
12.07.2021	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-
14.07.2021	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
19.07.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.07.2021	-	-	-	-	-	-	-	-	-	-	5	-	5	-	-	-
23.07.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
27.08.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
30.08.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
01.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
03.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
16.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
17.09.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
20.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
21.09.2021	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
23.09.2021	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
24.09.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
28.09.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 13)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
02.11.2020	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
05.11.2020	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
09.11.2020	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
11.11.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
12.11.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
13.11.2020	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
19.11.2020	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
08.01.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
11.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
13.01.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
21.01.2021	-	1	-	-	-	-	-	-	-	-	-	1	-	1	-	1
27.01.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
23.02.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
02.03.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
15.04.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
29.04.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
04.05.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
11.05.2021	-	-	-	-	-	-	-	-	-	-	4	-	4	-	-	-
17.05.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
27.05.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
09.06.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
15.06.2021	-	-	-	-	-	-	-	-	-	-	4	-	4	-	-	-
16.06.2021	-	-	6	-	-	-	-	-	-	-	2	-	2	-	-	-
18.06.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
06.07.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
07.07.2021	-	-	-	1	-	-	-	-	-	-	2	1	2	1	-	-
08.07.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
20.07.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
22.07.2021	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
26.07.2021	-	-	-	1	-	-	-	-	-	-	4	-	4	-	-	-
30.07.2021	-	-	-	-	-	-	-	-	-	-	1	1	1	1	-	-
04.08.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
05.08.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
17.08.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
20.08.2021	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-
01.09.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
07.09.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
09.09.2021	-	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-
08.10.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
12.10.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-

Quelle: Eigene Tabelle.

Tabelle: Entwicklungstätigkeit und Reviews nach Technologie (Entwickler:in 16)

Datum	Python		Terraform		CloudFormation		ARM		Docker		Vue.js		TypeScript		DynamoDB	
	C	R	C	R	C	R	C	R	C	R	C	R	C	R	C	R
07.09.2020	4	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
08.09.2020	5	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-
09.09.2020	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
29.09.2020	3	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
01.10.2020	-	-	11	-	-	-	-	-	-	-	-	-	-	-	-	-
02.10.2020	2	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
05.10.2020	-	-	3	-	1	-	-	-	-	-	-	-	-	-	-	-
07.10.2020	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
09.10.2020	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.10.2020	3	-	-	-	10	-	-	-	-	-	-	-	-	-	-	-
16.10.2020	-	-	-	-	7	-	-	-	-	-	-	-	-	-	-	-
21.10.2020	1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
26.10.2020	3	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
27.10.2020	4	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
28.10.2020	8	-	13	-	3	-	-	-	-	-	-	-	-	-	-	-
12.11.2020	5	-	1	-	-	-	-	-	-	-	-	-	-	-	4	-
13.11.2020	3	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
16.11.2020	16	-	15	-	-	-	-	-	-	-	-	-	-	-	-	-
17.11.2020	3	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
18.11.2020	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
22.01.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
25.01.2021	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
26.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
28.01.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
29.01.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
02.02.2021	3	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
08.02.2021	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
09.02.2021	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
10.02.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12.02.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
15.02.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
16.02.2021	4	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-
05.03.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
08.03.2021	3	1	4	-	-	-	-	-	-	-	-	-	-	-	-	1
25.03.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
29.03.2021	-	-	-	-	-	-	-	-	-	-	3	-	3	-	-	-
30.03.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
31.03.2021	-	-	-	-	-	-	-	-	-	-	1	1	1	1	-	-
01.04.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
06.04.2021	-	-	-	-	-	-	-	-	-	-	1	-	1	-	-	-
09.04.2021	-	-	-	-	-	-	-	-	-	-	2	-	2	-	-	-
03.05.2021	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05.05.2021	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
06.05.2021	5	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-
07.05.2021	3	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-
09.06.2021	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Quelle: Eigene Tabelle.

Anhang 4 Quellcode zur Datenextraktion

Im Folgenden sind Ausschnitte der Anwendung zur Extraktion der Daten aufgeführt. Der Quellcode und die darin enthaltenen Anmerkungen sind nach gängiger Entwicklungspraxis in Englisch verfasst. Neben der Datenextraktion wurde auch die Datenanalyse innerhalb der Anwendung umgesetzt. Dieser Teil wird in Anhang 5 näher betrachtet.

Die unten abgebildete Ordnerstruktur dient als Einstieg in den Aufbau der Quellcodes. Die hervorgehobenen Teile sind im Nachfolgenden abgebildet. Die Anwendung als Gesamtes ist im digitalen Anhang zu finden. Sie wurde gänzlich eigenständig implementiert.

Aufbau der Anwendung zur Fallstudie

- **main.py** ..... *Einstiegspunkt, Kapselung als Konsolenanwendung*
- README.md ..... *Dokumentation zur Ausführung der Anwendung*
- requirements.txt ..... *Abhängigkeiten zu externen Bibliotheken*
- _config/
  - configuration.yaml ..... *Konfiguration der Anwendung*
  - secrets.yaml ..... *Zugangsdaten*
- _input/
  - jira.csv ..... *Exportierte Daten aus Jira*
- analyze/
  - **...** ..... *Quellcode zur Analyse der Daten, siehe **Anhang 5***
- database/
  - _db.json ..... *Ablage der Daten*
  - **database.py** ..... *Schnittstellen zur Datenbank*
  - **models.py** ..... *Klassenrepräsentionen der Datentypen*
- extract/
  - **cloudwatch.py** ..... *Extraktion der Daten aus AWS CloudWatch*
  - **gitlab.py** ..... *Extraktion der Daten aus GitLab*
  - **jira.py** ..... *Extraktion der exportierten Daten aus Jira*
- helpers/
  - concurrency.py ..... *Funktion zur Nebenläufigkeit der Extraktionsprozesse*
  - config.py ..... *Zugriff auf Konfiguration und Zugangsdaten*

Anhang 4.1 Einstiegspunkt der Anwendung: main.py

```
from typing import List
from pathlib import Path
from datetime import datetime
from tabulate import tabulate
from tinydb.queries import where
from argparse import ArgumentParser
from multiprocessing import Process, Manager

from extract.jira import Jira
from extract.gitlab import GitLab
from extract.cloudwatch import CloudWatch

from helpers.concurrency import visualize
from helpers.config import Configuration, Secrets

from database.database import Database
from database.models import Sources, Account, LogGroup, ErrorLog
from database.models import User, Commit, Merge, Project, Group, Story


# Resolve local paths for required files.
DIR = Path().resolve()
INPUT_DIR = DIR.joinpath("_input")
CONFIG_DIR = DIR.joinpath("_config")
JIRA_PATH = INPUT_DIR.joinpath("jira.csv")
DATABASE_PATH = DIR.joinpath("database").joinpath("_db.json")


if __name__ == "__main__":

    start = datetime.now()

    #---                 Initialize console application                  ---#
    # Provide options to extract data from specific sources or all at once. #

    parser = ArgumentParser(
        description="Analyze effects of Collective Code Ownership on Software Development",
        epilog="Developed by Joscha Nassenstein"
    )

    parser.add_argument("-c", "--cloudwatch", dest="cloudwatch",
        action="store_true", help="Extract from Cloudwatch")
    parser.add_argument("-g", "--gitlab", dest="gitlab",
        action="store_true", help="Extract from GitLab")
    parser.add_argument("-j", "--jira", dest="jira",
        action="store_true", help="Extract from Jira")
    parser.add_argument("-a", "--all", dest="all",
        action="store_true", help="Extract from all sources")
    args = parser.parse_args()

    config = Configuration.from_input_file(f"{CONFIG_DIR}/configuration.yaml")
    secrets = Secrets.from_input_file(f"{CONFIG_DIR}/secrets.yaml")


    #---                       Optional: Extract data (if specified)                       ---#
    #  The application starts a process for each source and also each AWS Account separately. #

    if any([args.all, args.cloudwatch, args.gitlab, args.jira]):
        producers: List[Process] = []
        database_queue = Manager().Queue()
        database = Database(DATABASE_PATH)

        if args.cloudwatch or args.all:
            database.clear(Sources.CLOUDWATCH)
            for accountid in config.get_account_ids():
                producers.append(Process(
```

```
                name=f"CloudWatch[{accountid}]",
                target=CloudWatch.initiate_process,
                args=(accountid, config, secrets, database_queue)))

    if args.gitlab or args.all:
        database.clear(Sources.GITLAB)
        producers.append(Process(
            name="GitLab",
            target=GitLab.initiate_process,
            args=(config, secrets, database_queue)))

    if args.jira or args.all:
        database.clear(Sources.JIRA)
        producers.append(Process(
            name="Jira",
            target=Jira.initiate_process,
            args=(JIRA_PATH, database_queue)))

    for process in producers:
        process.start()

    consumer = Process(
        name="Database",
        target=Database.initiate_process,
        args=(DATABASE_PATH, database_queue, True))
    consumer.start()

    visualize(producers)

    for process in producers:
        process.join()

    database_queue.put((None, None))
    consumer.join()


#---                 Basic analysis                 ---#
# This provides an overview of all the extracted data. #

database = Database(DATABASE_PATH, cached=True)
users = database.all(User.LABEL)

print()
print(tabulate(
    headers=["Platform", "Metric", "Value"],
    tabular_data=[
        ["Gitlab", "Groups", len(database.all(Group.LABEL))],
        ["Gitlab", "Projects", len(database.all(Project.LABEL))],
        ["Gitlab", "Commits", len(database.all(Commit.LABEL))],
        ["Gitlab", "Merges", len(database.all(Merge.LABEL))],
        ["Gitlab", "Merges with user stories",
            len(database.query(Merge.LABEL, where("story_id") != None))],
        ["Gitlab", "Merges with review",
            len(database.query(Merge.LABEL, where("contributor_ids") != []))],
        ["Gitlab", "Contributors", len(users)],
        ["Jira", "Stories", len(database.all(Story.LABEL))],
        ["Cloudwatch", "Accounts", len(database.all(Account.LABEL))],
        ["Cloudwatch", "Dev: Log groups",
            len(database.query(LogGroup.LABEL, where("account") == "development"))],
        ["Cloudwatch", "Dev: Error logs",
            len(database.query(ErrorLog.LABEL, where("account") == "development"))],
        ["Cloudwatch", "Dev: Assigned Error logs", len(database.query(ErrorLog.LABEL,
            (where("author_id") != None) & (where("account") == "development")))],
        ["Cloudwatch", "Prod: Log groups",
            len(database.query(LogGroup.LABEL, where("account") == "production"))],
        ["Cloudwatch", "Prod: Error logs",
            len(database.query(ErrorLog.LABEL, where("account") == "production"))],
        ["Cloudwatch", "Prod: Assigned Error logs", len(database.query(ErrorLog.LABEL,
            (where("author_id") != None) & (where("account") == "production")))]
    ],
```

```
        tablefmt="presto"
    ))
    print()
    print(tabulate(
        headers=["User", "Commits", "Lines Changed", "Merges"],
        tabular_data=[[
            userid,
            len(database.query(Commit.LABEL, where("author_id") == userid)),
            sum([commit.changed_loc
                for commit in database.query(Commit.LABEL, where("author_id") == userid)]),
            len(database.query(Merge.LABEL, where("author_id") == userid))
        ] for userid in [user.id for user in users]],
        tablefmt="presto"
    ))

    print(f"\n\nExecution time: {datetime.now() - start}")
```

Anhang 4.2 Schnittstelle zur Datenbank: database.py

```
from queue import Queue
from typing import List, Dict
from tinydb import TinyDB, Query
from tinydb.table import Table
from tinydb.storages import JSONStorage
from tinydb.middlewares import CachingMiddleware

from database.models import SOURCE_MAP, TYPE_MAP, ALL_TYPES


class Database:
    """This class provides all required operations on the TinyDB database."""
    def __init__(self, filepath: str, cached: bool = False) -> None:
        storage = CachingMiddleware(JSONStorage) if cached else JSONStorage
        self.db: TinyDB = TinyDB(filepath, sort_keys=True, indent=4, storage=storage)
        self.tables: Dict[str,Table] = {label:self.db.table(label) for label in TYPE_MAP.keys()}

    @staticmethod
    def initiate_process(filepath: str, queue: Queue, cached: bool = False) -> None:
        """Initiate a database object and listen on a queue.
            This is intended to be used to asynchronously write data to the database.
        """
        Database(filepath, cached).__listen(queue)

    def insert(self, label: str, document: dict) -> int:
        """Insert a document to a specific table in the database."""
        return self.tables[label].insert(document)

    def query(self, label: str, query: Query) -> List[ALL_TYPES]:
        """Perform a query on a specific table."""
        return [TYPE_MAP[label](**result) for result in self.tables[label].search(query)]

    def all(self, label: str) -> List[ALL_TYPES]:
        """Get all items of a specific type / from a specific table."""
        return [TYPE_MAP[label](**result) for result in self.tables[label].all()]

    def clear(self, source: str) -> None:
        """Empty the database of a specific source. A source specifies a collection of types."""
        for label in SOURCE_MAP[source]:
            self.tables[label].truncate()

    def __listen(self, queue: Queue) -> None:
        """Listen to a queue and insert documents accordingly.
            This is intended to be used to asynchronously write data to the database.
        """
```

```
        with self.db:
            while True:
                label, document = queue.get()
                if label != None:
                    self.insert(label, document)
                else:
                    return
```

Anhang 4.3 Klassenrepräsentionen der Datentypen: models.py

```
import re
from dataclasses import dataclass, field
from datetime import datetime
from typing import Dict, List, ClassVar, Tuple, Union
from hashlib import shake_128

from helpers.config import Configuration


ERRORMSG_RE = re.compile(r"(/var/task/)([A-Za-z0-9_/.]+)(.*\n[\u00a0]{4})(.+)(\n[\u00a0]{2}|$)")
SPRINT_RE = re.compile(r"\d+")


#--- Sources ---#

class Sources:
    CLOUDWATCH = "cloudwatch"
    GITLAB = "gitlab"
    JIRA = "jira"


#--- CloudWatch ---#

@dataclass
class Account:
    """Dataclass representing an AWS Account."""
    name: str
    region: str
    id: int | None = None

    LABEL: ClassVar[str] = "account"

@dataclass
class LogGroup:
    """Dataclass representing an AWS CloudWatch log group."""
    name: str
    project_id: int
    account: str

    LABEL: ClassVar[str] = "loggroup"

    @staticmethod
    def hash_name(name: str) -> str:
        """Return a hash to mask the name of the log group."""
        return shake_128(name.encode("UTF-8")).hexdigest(8)

    @staticmethod
    def transform_mapping(name: str, project_id: int, account: str) -> Dict[str,str|int]:
        """Transform Configuration mappings into a dictionary representation of this class."""
        return {"name": LogGroup.hash_name(name), "project_id": project_id, "account": account}

@dataclass
class ErrorLog:
    """Dataclass representing an AWS CloudWatch error log."""
```

```
    loggroup: str
    account: str
    timestamp: int
    message: str | None = None
    author_id: int | None = None

    LABEL: ClassVar[str] = "errorlog"

    @staticmethod
    def log_timestamp_to_epoch(timestamp: str):
        """Transform the AWS log timestamp format to a unix / epoch timestamp."""
        return int(datetime.strptime(timestamp, "%Y-%m-%d %H:%M:%S.%f").timestamp())

    @staticmethod
    def query_result_extract_group(data: dict) -> str:
        """Extract the log group name from a CloudWatch query result."""
        return data[0]["value"].split(":")[1]

    @staticmethod
    def query_result_transform_timestamp(data: dict) -> str:
        """Transform the AWS log timestamp format directly from a query result."""
        return ErrorLog.log_timestamp_to_epoch(data[1]["value"])

    @staticmethod
    def transform_query_result(data: dict, account: str) -> Dict[str,str|int]:
        """Transform a CloudWatch query result into a dictionary representation of the class."""
        loggroup = ErrorLog.query_result_extract_group(data)
        return {
            "loggroup": LogGroup.hash_name(loggroup),
            "account": account,
            "timestamp": ErrorLog.query_result_transform_timestamp(data),
            "message": data[2]["value"]
        }

    @staticmethod
    def from_query_result(data: dict, account: str) -> "ErrorLog":
        """Transform a CloudWatch query result into an instance of this class."""
        return ErrorLog(**ErrorLog.transform_query_result(data, account))

    @staticmethod
    def timestamp_to_zulustr(timestamp) -> str:
        """Parse a timestamp to a Zulu-style datetime string."""
        return f"{datetime.fromtimestamp(timestamp).replace(microsecond=0).isoformat()}Z"

    @staticmethod
    def get_error_info(message) -> List[Tuple[str,str]]:
        """Extract the source code lines from the message which potentially caused the error."""
        potential_errors = []
        for prefix, file, infix, statement, suffix in ERRORMSG_RE.findall(message):
            potential_errors.insert(0, (file, statement))
        return potential_errors


#--- GitLab ---#

@dataclass
class User:
    """Dataclass representing a GitLab user."""
    id: str

    LABEL: ClassVar[str] = "user"

    @staticmethod
    def hash_id(id: str) -> str:
        """Return a hash to mask the ID of the user."""
        return shake_128(id.encode("UTF-8")).hexdigest(4)

    @staticmethod
    def transform_user_id(user_id: str) -> Dict[str,str]:
        """Transform a user ID into a dictionary representation of this class."""
```

```
        return {
            "id": User.hash_id(user_id)
        }

    @staticmethod
    def from_user_id(user_id: str) -> "User":
        """Transform a user ID into an instance of this class."""
        return User(**User.transform_user_id(user_id))

    @staticmethod
    def transform_merge_participants_input(data: dict) -> Dict[str,str]:
        """Transform merge participants data into a dictionary representation of this class."""
        return {
            "id": User.hash_id(data["username"])
        }

    @staticmethod
    def from_merge_participants_input(data: dict) -> "User":
        """Transform merge participants data into an instance of this class."""
        return User(**User.transform_merge_participants_input(data))

@dataclass
class Commit:
    """Dataclass representing a Git commit in GitLab."""
    id: int
    short_id: str
    timestamp: int
    changed_loc: int
    project_id: int
    author_id: int

    LABEL: ClassVar[str] = "commit"

    @staticmethod
    def transform_input(data: dict, project_id: int, author_id: int) -> Dict[str,str|int]:
        """Transform GitLab commit data into a dictionary representation of this class."""
        return {
            "id": data["id"],
            "short_id": data["short_id"],
            "timestamp": int(datetime.fromisoformat(data["authored_date"]).timestamp()),
            "author_id": author_id,
            "changed_loc": data["stats"]["total"],
            "project_id": project_id
        }

    @staticmethod
    def from_input(data: dict, project_id: int, author_id: int) -> "Commit":
        """Transform GitLab commit data into an instance of this class."""
        return Commit(**Commit.transform_input(data, project_id, author_id))

@dataclass
class Merge:
    """Dataclass representing a merge request in GitLab."""
    id: int
    internal_id: int
    project_id: int
    author_id: int
    timestamp: int
    contributor_ids: List[int] = field(default_factory=lambda:[])
    story_id: int | None = None

    LABEL: ClassVar[str] = "merge"

    @staticmethod
    def transform_input(data: dict, project_acronym: str) -> Dict[str,str|int]:
        """Transform GitLab merge request data into a dictionary representation of the class."""
        story_result = re.search(
            fr"([{project_acronym}]{{5,}}[\s_-]*)(\d+)",
            data["title"]+data["description"], flags=re.I)
        story_id = int(story_result.group(2)) if story_result else None
```

```
        return {
            "id": data["id"],
            "internal_id": data["iid"],
            "project_id": data["project_id"],
            "author_id": User.hash_id(data["author"]["username"]),
            "timestamp": int(datetime.fromisoformat(data["merged_at"]).timestamp()),
            "story_id": story_id if story_id and story_id < 1000 else None
        }

    @staticmethod
    def from_input(data: dict, project_acronym: str) -> "Merge":
        """Transform GitLab merge request data into an instance of this class."""
        return Merge(**Merge.transform_input(data, project_acronym))

@dataclass
class Project:
    """Dataclass representing a GitLab project."""
    id: int
    default_branch: str
    log_groups: List[str]
    platforms: List[str]
    technologies: List[str]
    group_id: int | None = None

    LABEL: ClassVar[str] = "project"

    @staticmethod
    def transform_input(data: dict, config: Configuration) -> Dict[str,str|int|List[str]]:
        """Transform GitLab project data into a dictionary representation of this class."""
        project_id = data["id"]
        log_groups = [LogGroup.hash_name(name)for name in
            config.get_log_groups_for_project(project_id)]
        return {
            "id": project_id,
            "default_branch": data["default_branch"],
            "log_groups": log_groups,
            "platforms": config.get_platforms_for_project(project_id),
            "technologies": config.get_technologies_for_project(project_id),
            "group_id": data.get("namespace", {}).get("id", None)
        }

    @staticmethod
    def from_input(data: dict, config: Configuration) -> "Project":
        """Transform GitLab project data into an instance of this class."""
        return Project(**Project.transform_input(data, config))

@dataclass
class Group:
    """Dataclass representing a GitLab group."""
    id: int
    parent_id: int | None = None

    LABEL: ClassVar[str] = "group"

    @staticmethod
    def transform_input(data: dict) -> Dict[str,str|int]:
        """Transform GitLab group data into a dictionary representation of this class."""
        return {
            "id": data["id"],
            "parent_id": data["parent_id"]
        }

    @staticmethod
    def from_input(data: dict) -> "Group":
        """Transform GitLab group data into an instance of this class."""
        return Group(**Group.transform_input(data))


#--- Jira ---#
```

```
@dataclass
class Story:
    """Dataclass representing a user story in Jira."""
    id: int
    sprints: List[str] = field(default_factory=lambda:[])

    LABEL: ClassVar[str] = "story"

    @staticmethod
    def transform_input(data: List[str]) -> Dict[str,str|int|List[int]]:
        """Transform exported jira data into a dictionary representation of this class."""
        sprints = []
        for i in range(1, len(data)):
            sprints.append(data[i])

        return {
            "id": int(data[0]),
            "sprints": [int(SPRINT_RE.search(sprint).group()) for sprint in sprints if sprint]
        }

    @staticmethod
    def from_input(data: List[str]) -> "Story":
        """Transform exported jira data into an instance of this class."""
        return Story(**Story.transform_input(data))


#--- Helpers ---#

SOURCE_MAP = {
    Sources.CLOUDWATCH: {Account.LABEL, LogGroup.LABEL, ErrorLog.LABEL},
    Sources.GITLAB: {User.LABEL, Commit.LABEL, Merge.LABEL, Project.LABEL, Group.LABEL},
    Sources.JIRA: {Story.LABEL}
}

TYPE_MAP = {
    Account.LABEL: Account,
    LogGroup.LABEL: LogGroup,
    ErrorLog.LABEL: ErrorLog,
    User.LABEL: User,
    Commit.LABEL: Commit,
    Merge.LABEL: Merge,
    Project.LABEL: Project,
    Group.LABEL: Group,
    Story.LABEL: Story
}

ALL_TYPES = Union[Account, LogGroup, ErrorLog, User, Commit, Merge, Project, Group, Story]
```

Anhang 4.4 Extraktion der Daten aus CloudWatch: cloudwatch.py

```
from queue import Queue
from typing import List
from boto3.session import Session
from botocore.exceptions import ClientError

from extract.gitlab import GitLab
from helpers.config import Configuration, Secrets
from database.models import Account, LogGroup, ErrorLog


QUERY_LIMIT = 10000
ERROR_LOGS_QUERY = """
fields @log, @timestamp, @message
```

```
| sort @timestamp asc
| filter @message like 'ERROR'
"""

class QueryStatus:
    IN_PROGRESS = 0
    THROTTLED = 1
    LIMIT_REACHED = 2
    NOT_FOUND = 3

class CloudWatchError(Exception):
    pass

class CloudWatch:
    """Class encapsulating data extraction operations on AWS CloudWatch."""
    def __init__(self, account_id: int, config: Configuration, secrets: Secrets,
        queue: Queue) -> None:

        self.account = Account(**config.get_account_information(account_id), id=account_id)
        self.config = config
        self.queue = queue
        self.gitlab = GitLab(config, secrets)

        self.client = Session(
            profile_name=str(account_id), region_name=self.account.region).client("logs")

    @staticmethod
    def initiate_process(account_id: str, config: Configuration, secrets: Secrets,
        queue: Queue) -> None:
        """Initiate a CloudWatch object and start extracting data.
            This is intended to be used to anynchronously fetch and forward data.
        """
        CloudWatch(account_id, config, secrets, queue).__extract()


    def __start_error_query(self, log_group: str, start: int, limit: int) -> str | int:
        """Start an error query on a specific log group (+ time interval) in CloudWatch."""
        try:
            return self.client.start_query(
                logGroupNames=[log_group],
                startTime=start,
                endTime=limit,
                queryString=ERROR_LOGS_QUERY,
                limit=QUERY_LIMIT
            )["queryId"]
        except ClientError as error:
            if error.response["Error"]["Code"] == "LimitExceededException":
                return QueryStatus.LIMIT_REACHED
            elif error.response["Error"]["Code"] == "ResourceNotFoundException":
                return QueryStatus.NOT_FOUND
            else:
                raise

    def __get_query_results(self, query_id: str) -> List[dict] | int:
        """Try to get the result of a previously started query."""
        try:
            result = self.client.get_query_results(queryId=query_id)
        except ClientError as error:
            if error.response["Error"]["Code"] == "ThrottlingException":
                return QueryStatus.THROTTLED
            else:
                raise

        if result["status"] == "Complete":
            return result["results"]
        elif result["status"] in ("Scheduled","Running"):
            return QueryStatus.IN_PROGRESS
        else:
            raise CloudWatchError(f"Error running query {query_id}: Status {result['status']}")
```

```
    def __handle_query_results(self, results: List[dict], log_group: str) -> None:
        """Extract information from a query result and forward it to the database."""
        for result in results:
            log = ErrorLog.transform_query_result(result, self.account.name)
            log["author_id"] = self.gitlab.blame(log, log_group)
            self.queue.put((ErrorLog.LABEL, {k:log[k] for k in log.keys() if k!="message"}))

    def __query_logs(self, log_groups: List[str]):
        """Repeatedly start queries and extract the results once they have finished.
            This spins up as many queries as CloudWatch allows on a single account.
        """
        log_groups = {log_group: (self.config.start_timestamp, self.config.limit_timestamp)
            for log_group in log_groups}
        queries = {}

        while True:
            while log_groups:
                log_group, time_restriction = next(iter(log_groups.items()))
                query = self.__start_error_query(
                    log_group, time_restriction[0], time_restriction[1])

                if query == QueryStatus.LIMIT_REACHED:
                    break
                elif query == QueryStatus.NOT_FOUND:
                    pass
                else:
                    queries[log_group] = query
                log_groups.pop(log_group)

            for i in range(len(queries)-1, -1, -1):
                log_group, query = list(queries.items())[i]

                results = self.__get_query_results(query)

                if results == QueryStatus.IN_PROGRESS:
                    continue
                elif results == QueryStatus.THROTTLED:
                    continue
                else:
                    self.__handle_query_results(results, log_group)

                if len(results) >= QUERY_LIMIT:
                    last_timestamp = ErrorLog.query_result_transform_timestamp(results[-1])
                    log_groups[log_group] = (last_timestamp, self.config.limit_timestamp)

                queries.pop(log_group)

            if not log_groups and not queries:
                break

    def __add_account(self) -> None:
        """Forward the information about the AWS account itself to the database."""
        self.queue.put((Account.LABEL,
            {i:self.account.__dict__[i] for i in self.account.__dict__ if i!="id"}))

    def __add_loggroups(self) -> None:
        """Forward the information about the log groups themselves to the database."""
        for mapping in self.config.get_log_group_mappings():
            self.queue.put((LogGroup.LABEL,
                LogGroup.transform_mapping(*mapping, self.account.name)))

    def __extract(self) -> None:
        """Wrapper for the entire process of fetching data from CloudWatch."""
        self.__add_account()
        self.__add_loggroups()
        self.__query_logs(self.config.log_groups)
```

Anhang 4.5 Extraktion der Daten aus GitLab: gitlab.py

```python
import requests
from queue import Queue
from typing import Any, Dict, Iterable, List, Tuple
from dataclasses import dataclass
from cachetools import cached
from cachetools.keys import hashkey
from urllib.parse import quote_plus

from helpers.config import Configuration, Secrets
from database.models import ErrorLog, User, Commit, Group, Merge, Project


class GitLabError(Exception):
    pass

@dataclass
class GitLab:
    """Class encapsulating data extraction operations on GitLab."""
    config: Configuration
    secrets: Secrets
    queue: Queue | None = None

    @staticmethod
    def initiate_process(config: Configuration, secrets: Secrets, queue: Queue) -> None:
        """Initiate a GitLab object and start extracting data.
            This is intended to be used to anynchronously fetch and forward data.
        """
        GitLab(config, secrets, queue).__extract()

    @cached(cache={}, key=lambda self, url: hashkey(url))
    def __get(self, url: str) -> requests.Response:
        """Cached function for an authorized call to a specific resource on the GitLab API."""
        response = requests.get(url, headers = {"PRIVATE-TOKEN": self.secrets.gitlab_token})

        if response.ok:
            return response
        else:
            raise GitLabError(f"Requesting {url} failed: {response.text}")

    def __fetch(self, uri: str, additional_params: Dict[str, Any] = {}) -> requests.Response:
        """Prepare the URL for the GitLab API with specified parameters and fetch data."""
        url = f"{self.secrets.gitlab_host}/api/v4{uri}"
        if additional_params:
            url = self.__add_to_querystring(url, additional_params)
        return self.__get(url)

    def __fetch_single(self, uri: str, additional_params: Dict[str, Any] = {}) -> Dict | List:
        """Fetch a resource from GitLab which does not use pagination."""
        return self.__fetch(uri, additional_params).json()

    def __fetch_all(self, uri: str, additional_params: Dict[str, Any] = {}) -> Iterable[Dict]:
        """Fetch a resource from GitLab which uses pagination.
            This will call the API until there is no more information available.
        """
        while True:
            response = self.__fetch(uri, additional_params)
            for dataset in response.json():
                yield dataset

            if next_page := response.headers.get("X-Next-Page", None):
                additional_params["page"] = next_page
            else:
                break

    def __add_to_querystring(self, url: str, params: Dict[str, Any]) -> str:
        """Prepare a URL with specified query parameters."""
```

```
        updated = requests.PreparedRequest()
        updated.prepare_url(url, params)
        return updated.url

    @cached(cache={}, key=lambda self, user: hashkey(user["id"]))
    def __add_user(self, user: Dict) -> None:
        """Forward user data to the database."""
        self.queue.put((User.LABEL, user))

    def __fetch_groups(self, base_group_id: int) -> Iterable[str]:
        """Fetch group data from GitLab based on a parent group and forward to the database."""
        self.queue.put((
            Group.LABEL,
            Group.transform_input(self.__fetch_single(f"/groups/{base_group_id}/"))
        ))

        yield base_group_id

        for group in self.__fetch_all(f"/groups/{base_group_id}/descendant_groups"):
            group = Group.transform_input(group)
            self.queue.put((Group.LABEL, group))
            yield group["id"]

    def __fetch_projects(self, group_id: int) -> Iterable[str]:
        """Fetch projects contained in a GitLab group and forward them to the database."""
        for project in self.__fetch_all(f"/groups/{group_id}/projects", {"archived": "false"}):
            if (project["created_at"] >= self.config.start_time
                and project["created_at"] <= self.config.limit_time):
                project = Project.transform_input(project, self.config)
                self.queue.put((Project.LABEL, project))
                yield project["id"]

    def __fetch_commits_and_authors(self, project_id: int) -> None:
        """Fetch commits and authors of a GitLab project and forward them to the database."""
        for commit in self.__fetch_all(
            f"/projects/{project_id}/repository/commits",{
                "with_stats": "true",
                "since": self.config.start_time,
                "until": self.config.limit_time }):

            userid = self.__get_user_identifiers(commit["author_email"])
            user = User.transform_user_id(userid)

            self.__add_user(user)

            self.queue.put(
                (Commit.LABEL, Commit.transform_input(commit, project_id, user["id"])))

    def __fetch_merges_and_contributors(self, project_id: int) -> None:
        """Fetch merge requests and contributors of a GitLab project and forward the data."""
        for dataset in self.__fetch_all(
            f"/projects/{project_id}/merge_requests", {
                "state": "merged",
                "created_after": self.config.start_time,
                "updated_before": self.config.limit_time }):

            merge = Merge.from_input(dataset, self.config.project_acronym)

            for contributor in self.__fetch_single(
                f"/projects/{project_id}/merge_requests/{merge.internal_id}/participants"):

                user = User.transform_merge_participants_input(contributor)
                self.__add_user(user)
                if user["id"] != merge.author_id:
                    merge.contributor_ids.append(user["id"])

            self.queue.put((Merge.LABEL, merge.__dict__))

    def __fetch_most_recent_commit(self, project_id: int, time: str) -> str:
        """Fetch the most recent commit of a project in GitLab."""
```

```
        return self.__fetch_single(
            f"/projects/{project_id}/repository/commits",
            {"with_stats": "true", "until": time})[0]["id"]

    @cached(cache={}, key=lambda self, project_id, file, time: hashkey(project_id, file, time))
    def __fetch_blame(self, project_id: int, file: str, time: str) -> None:
        """Fetch blame information from a GitLab project based on the file and the timestamp."""
        last_commit = self.__fetch_most_recent_commit(project_id, time)

        return self.__fetch_all(
            f"/projects/{project_id}/repository/files/{quote_plus(file)}/blame",
            {"ref": last_commit})

    @cached(cache={}, key=lambda self, mail: hashkey(mail))
    def __get_user_identifiers(self, mail: str) -> str:
        """Fetch user information from GitLab and return the username."""
        try:
            data = self.__fetch_single(f"/search", {"scope": "users", "search": mail})[0]
            return data["username"]
        except IndexError:
            return self.config.get_user_id(mail)

    def __extract(self) -> None:
        """Wrapper for the entire process of fetching data from GitLab."""
        for group_id in self.__fetch_groups(self.config.parent_group):
            for project_id in self.__fetch_projects(group_id):
                self.__fetch_commits_and_authors(project_id)
                self.__fetch_merges_and_contributors(project_id)

    def blame(self, log: dict, loggroup: str) -> int | None:
        """Attempt to fetch blame information for a CloudWatch error log."""
        time = ErrorLog.timestamp_to_zulustr(log["timestamp"])
        for file, statement in ErrorLog.get_error_info(log["message"]):
            try:
                for item in self.__fetch_blame(
                    self.config.get_project_for_log_group(loggroup), file, time):

                    if statement in [line.strip() for line in item["lines"]]:
                        userid = self.__get_user_identifiers(item["commit"]["author_email"])
                        return User.hash_id(userid)

            except GitLabError:
                continue
```

Anhang 4.6 Extraktion der Daten aus Jira: jira.py

```python
from csv import reader
from queue import Queue
from dataclasses import dataclass

from database.models import Story


@dataclass
class Jira:
    """Class encapsulating data extraction operations on a file exported from Jira."""
    filepath: str
    queue: Queue

    @staticmethod
    def initiate_process(filepath: str, queue: Queue) -> None:
        """Initiate a Jira object and start extracting data from an exported file.
            This is intended to be used to anynchronously read and forward data.
        """
        Jira(filepath, queue).__extract()

    def __extract(self) -> None:
        """Continuously read and forward data from the Jira export."""
        with open(self.filepath, "r") as jira_input:
            issue_reader = reader(jira_input, delimiter=";")

            for index, row in enumerate(issue_reader):
                if index == 0:
                    continue
                self.queue.put((Story.LABEL, Story.transform_input(row)))
```

Anhang 5 Quellcode zur Datenanalyse

Die Grundlage der Datenanalyse in Abschnitt 6 bilden verschiedene Abbildungen, welche teilweise im Text sowie im Anhang eingebunden wurden. Diese wurden auf Basis der extrahierten Daten programmatisch angelegt und anschließend exportiert. Jede Datei beinhaltet dabei neben den Abbildungen selbst auch die entsprechenden Werte in Tabellenform, welche als Basis für die Wertetabellen in Anhang 3 genutzt wurden.

Die Analysen sind in sogenannte Jupyter Notebooks gekapselt. Diese stellen eine web-basierte Plattform dar, welche Funktionen für das Ausführen von Code (hier mittels Python implementiert), der Ausgabe und Speicherung der Ergebnisse sowie zusätzlicher Dokumentation vereinen. Für jede der durchgeführten Analysen wurde eine einzelne Datei angelegt. Diese folgen einem stringenten Aufbau. Zunächst werden die benötigten Daten aus der Datenbank gelesen, welche die Ergebnisse der Datenextraktion erhält. Anschließend werden die Daten in verschiedenen Formen visualisiert. Im letzten Schritt werden die Wertetabellen zu den Visualisierungen angelegt. Zusätzlich zum Quellcode sind die Schritte direkt im Jupyter Notebook dokumentiert. Diese Annotationen ist im Gegensatz zur Dokumentation in der Extraktionsanwendung auf Deutsch verfasst. Die gesamte Anwendung wurde eigenständig implementiert.

In Zusammenspiel mit dem Quellcode aus Anhang 4 komplettiert dies die Anwendung zur quantitativen Datenanalyse, welche im Rahmen dieser Masterthesis durchgeführt wurde. Da die Ergebnisse der Auswertung in Form von Abbildungen und Wertetabellen bereits im Textteil enthalten sind, befindet sich der entsprechende Quellcode zur Datenanalyse lediglich im digitalen Anhang. Auf der nächsten Seite ist dessen Aufbau dargestellt.

Aufbau der Datenanalyse

- **analyze/** — *Unterverzeichnis der Anwendung*
  - activity.ipynb — *Entwicklungsaktivität (Abbildung 8)*
  - activity_technologies.ipynb — *Entwicklungsaktivität nach Technologien (Abbildung 17)*
  - delay.ipynb — *Verzögerungen in der Entwicklung (Abbildung 18)*
  - dependencies.ipynb — *Abhängigkeiten (Abbildungen 14 bis 16 und Anhang 1.2)*
  - errors.ipynb — *Fehlerprotokolle (Abbildung 9)*
  - expertise.ipynb — *Expertise der Entwickler:innen (Abbildung 19)*
  - ownership.ipynb — *Code Ownership der Teilprojekte (Abbildung 10)*
  - ownership_errors.ipynb — *Fehlerprotokolle nach Code Ownership (Abbildung 11)*
  - ownership_errors_authors.ipynb — → *personalisiert (Abbildungen 12 und 13 und Anhang 1.1)*
  - reviews.ipynb — *Code Reviews (Abbildung 20 und Anhang 1.3)*
  - **appendix/** — *Abbildungen in größerem Maßstab als Teil des Anhangs*
    - activity_detail.ipynb — *Entwicklungsaktivität (Anhang 2.1)*
    - activity_errors_detail.ipynb — *Fehlerprotokolle (Anhang 2.2)*
    - delay_detail.ipynb — *Verzögerungen in der Entwicklung (Anhang 2.5)*
    - expertise_detail.ipynb — *Expertise der Entwickler:innen (Anhang 2.6)*
    - ownership_detail.ipynb — *Code Ownership der Teilprojekte (Anhang 2.3)*
    - ownership_errors_detail.ipynb — *Code Ownership und Fehlerprotokolle (Anhang 2.4)*
    - *paths.py* — *Auflösung von lokalen Ordnerpfaden*
  - **dedicated/** — *Passgenaue Abbildungen für den Textteil*
    - ownership_errors_example_deviation.ipynb — *Zu Abbildung 12*
    - ownership_errors_example_equivalent.ipynb — *Zu Abbildung 13*
    - *paths.py* — *Auflösung von lokalen Ordnerpfaden*
  - **shared/** — *Geteilte Funktionen der Analyse*
    - *calculation.py* — *Berechnungsfunktionen*
    - *locals.py* — *Alternative Bezeichnungen und Farbgebung*
    - *paths.py* — *Auflösung von lokalen Ordnerpfaden*