Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AI	Artificial Intelligence
API	Application Programming Interface
APP	Application
AWS	Amazon Web Services
BSI	Bundesamt für Sicherheit in der Informationstechnik
bzw.	beziehungsweise
ChatGPT	Chat Generative Pre-trained Transformer
CRM	Customer Relationship Management
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DML	Data Manipulation Language
EU	European Union
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
INC	Incorporated
IQ	Intelligenzquotient
IT	Informationstechnologie
KI	Künstliche Intelligenz
LLM	Large Language Model
LWC	Lightning Web Components
SaaS	Software as a Service
SOC2	Service Organization Control 2
SOQL	Salesforce Object Query Language
TLS	Transport Layer Security
UK	United Kingdom
URL	Uniform Resource Locator
USA	United States of America

VS-Code	Visual Studio Code
XML	Extensible Markup Language
z.B.	zum Beispiel

1 Integration von Künstlicher Intelligenz zur Optimierung von Entwicklungsprozessen

1.1 Motivation, Ziele und Inhalt

Qualität und Effizienz waren schon immer zentrale Aspekte in der Entwicklung von Softwareprojekten. Mit der fortschreitenden Digitalisierung und der zunehmenden Bedeutung von Massenproduktion sind diese Faktoren noch entscheidender geworden. Durch die Steigerung der Abstraktion und die Nutzung neuer Tools wurden immer komplexere Probleme bewältigbar: Von den frühen Tagen der Programmierung in Assembler über die Nutzung höherer Programmiersprachen bis hin zu modernen Entwicklungsplattformen wie Visual Studio. Heute suchen Unternehmen nach neuen Wegen, um Effizienz und Qualität noch weiter zu verbessern.

Es lässt sich also darauf schließen, dass beide Faktoren immer mehr an Bedeutung gewinnen. Salesforce, als eine der führenden Plattformen für Customer Relationship Management (CRM), wird in Unternehmen weltweit genutzt um Geschäftsprozesse zu verwalten und zu optimieren. Trotz der umfangreichen Funktionalitäten und der hohen Flexibilität von Salesforce stoßen viele Unternehmen an Grenzen, wenn es um die Verbesserung von Effizienz und Automatisierung ihrer Entwicklungsprozesse geht. Auch mit der Einführung von ChatGPT bietet Künstliche Intelligenz (KI) immer mehr Potenzial, um repetitive Aufgaben zu automatisieren, Entscheidungsprozesse zu unterstützen und die Gesamtproduktivität zu steigern.

Ein Beispiel für ein globales Problem, das die Relevanz dieser Arbeit unterstreicht, ist der allgemeine Mangel an Fachkräften im Informationstechnologie (IT)-Bereich. Dies führt dazu, dass Unternehmen Wege suchen, um ihre vorhandenen Ressourcen bestmöglich zu nutzen und gleichzeitig die Qualität ihrer Entwicklungsarbeit aufrechtzuerhalten. Die Integration von KI in Salesforce-Umgebungen kann einen entscheidenden Beitrag dazu leisten, genau diese Herausforderung zu meistern, indem sie Entwicklern ermöglicht, sich auf komplexere und wertschöpfendere Aufgaben zu konzentrieren.

Das Hauptziel dieser Bachelorarbeit ist es, die Potenziale der Integration von KI zur Optimierung von Entwicklungsprozessen in Salesforce Umgebungen zu untersuchen. Dabei werden spezifisch die KI-Tools Codeium und Einstein for Developers betrachtet. Es soll evaluiert werden, inwieweit diese Tools dazu beitragen können, Entwicklungsprozesse effizienter zu gestalten und typische Probleme in Softwareprojekten mit proprietären Programmiersprachen zu lösen.

Die zentralen Forschungsfragen, die in dieser Arbeit beantwortet werden sollen, sind:

1. In welchen Bereichen der Salesforce-Entwicklungsprozesse können Codeium und Einstein for Developers eine Optimierung bewirken?
2. Wie unterscheiden sich die beiden Tools in ihrer Herangehensweise und Effektivität?
3. Welche messbaren Verbesserungen können durch den Einsatz dieser KI-Tools erzielt werden?

1.2 Methodik und Vorgehensweise

Die Methodik dieser Arbeit basiert auf einem systematischen und strukturierten Ansatz zur Untersuchung der Potenziale von KI in Salesforce Entwicklungsprozessen. Im Mittelpunkt stehen die KI-Tools Codeium und Einstein for Developers. Die Untersuchung erfolgt in mehreren Schritten. Zunächst wird eine umfassende Literaturrecherche durchgeführt. Diese umfasst die Analyse der vorhandenen wissenschaftlichen Literatur zu Salesforce und KI. Ziel ist es, ein fundiertes theoretisches Fundament zu schaffen und bestehende Erkenntnisse zu diesen Themen zu sammeln. Daraufhin erfolgt die Erstellung und Durchführung einer Umfrage, sowie deren Analyse. Diese wird unter Entwicklern durchgeführt, um deren Einblicke in die aktuellen Problemstellungen und Bedürfnisse der Praxis geben.

Ein wesentlicher Bestandteil soll auch der systematische Vergleich der KI-Tools Codeium und Einstein for Developers sein. Verschiedene Kriterien wie Funktionalität, Benutzerfreundlichkeit, Integration in Salesforce sowie die tatsächliche Verbesserung der Entwicklungsprozesse werden untersucht. Ziel ist es, die Stärken und Schwächen der beiden Tools herauszuarbeiten und ihre jeweilige Eignung für die Optimierung von Entwicklungsprozessen zu bewerten. Die gesammelten Daten werden analysiert, um die Hauptprobleme in den Entwicklungsprozessen zu identifizieren und die Nützlichkeit der KI-Tools zu bewerten. Die Ergebnisse aus der Funktionsanalyse und der Umfrage werden gegenübergestellt, um die Vor- und Nachteile der beiden Tools zu ermitteln. Des weiteren sollen die beiden Tools im Sinne der IT-Sicherheit betrachtet werden. Dabei werden einige Strategien zur Risikominderung und Nutzung der beiden Tools erarbeitet. Den Abschluss der Arbeit bildet die Zusammenfassung der wichtigsten Erkenntnisse. Diese werden kritisch diskutiert und ihre Bedeutung für die Praxis wird herausgestellt. Schließlich werden Vorschläge für zukünftige Forschungs- und Anwendungsfelder sowie Empfehlungen für die Implementierung der Tools in Salesforce-Entwicklungsprozesse gegeben.

2 Künstliche Intelligenz, Salesforce und Apex

In diesem Kapitel werden die Grundlagen von KI sowie die Bedeutung von Salesforce und Apex für die Entwicklung von Softwareprojekten behandelt. Zunächst wird KI definiert, ihre Funktionen sowie Anwendungsbereiche erläutert. Dabei wird auch ein kurzer historischer Überblick gegeben, um den aktuellen Stand von KI-Modellen zu verdeutlichen um damit die Grenzen auch für diese Arbeit aufzuzeigen. Anschließend wird auf Salesforce und Apex eingegangen. Es wird erklärt, was Apex ist und welche Funktionen die Programmiersprache bietet. Zudem wird die Bedeutung von Apex aus Entwicklersicht beleuchtet und anhand eines Beispiels von dotSource illustriert. Dieses Kapitel dient generell dazu, ein grundlegendes Verständnis von KI, Salesforce und Apex zu vermitteln. Es legt den Grundstein für die weiteren Untersuchungen und ermöglicht es den Lesern, die Zusammenhänge zwischen diesen Bereichen besser zu verstehen. Dieses Wissen ist von entscheidender Bedeutung, um die Potenziale der Integration von KI in Salesforce-Umgebungen zu erfassen und zu bewerten.

2.1 Definition und Funktionen von Künstlicher Intelligenz

KI hat in den letzten Jahren einen enormen Aufschwung erlebt und ist zu einem zentralen Thema in vielen Bereichen der Technologie geworden. Da viele Menschen jedoch nicht genau wissen, was KI ist und wie sie funktioniert, ist es wichtig eine klare Definition und Erklärung zu geben. Dazu ist es sinnvoll, sich zunächst die beiden Begriffe die in KI vorkommen, anzuschauen: Was ist Intelligenz und was genau ist künstlich?

Intelligenz wird unterschiedlich definiert: als Fähigkeit, Probleme zu lösen, zu lernen oder sich an neue Situationen anzupassen. Psychologen sehen Intelligenz als Sammelbegriff für kognitive Leistungsfähigkeit, die sich durch logisches, sprachliches oder mathematisches Problemlösen auszeichnet. Dabei konnte jedoch keine einheitliche Zustimmung gefunden werden wie Intelligenz genau definiert wird, da die kognitiven-Fähigkeiten bei Individuen unterschiedlich stark ausgeprägt sind. Trotzdem können einige Ergebnisse festgehalten werden[1]:

Wie bereits erwähnt unterscheiden sich Individuen voneinander durch ihre Herangehensweise an Probleme, wie sie lernen oder Hindernisse durch Denken überwinden. Konzepte wie das Intelligenzquotient (IQ)-Modell versuchen, Intelligenz darzustellen, erfassen aber nicht die ganze Wahrheit.

[1] Vgl. [Pos23], S.9-11

Menschliche Intelligenz ist nicht nur auf kognitive Fähigkeiten beschränkt, sondern eher als ein Prozess zu sehen der durch Motivation und Selbstbewusstsein beeinflusst wird. Dadurch erhält der Mensch die Fähigkeit Konzepte zu bilden, Muster zu erkennen, zu planen oder Sprache zur Kommunikation einzusetzen. Auch emotionale, soziale und kollektive Intelligenz zählen dazu, die nicht nur bei Menschen erkennbar sind.[2]

Intelligenz kann nach verschiedenen Ausprägungen unterschieden werden, darunter:[3]

- Sprachliche / Linguistische Intelligenz
- Musikalische Intelligenz
- Logisch-mathematische Intelligenz
- Räumliche Intelligenz
- Körperlich-kinästhetische Intelligenz
- Interpersonale Intelligenz
- Natürliche Intelligenz
- Existentielle / Spirituelle Intelligenz
- Kreative Intelligenz

Die Ausprägungen von Intelligenz können in fünf Stufen unterteilt werden. Dabei gilt als niedrigste Stufe das Denken und die Deduktion, also das logische Denken. Die nächste Stufe ist das Lernen und die Induktion, welches das Lernen aus Erfahrungen und damit die Mustererkennung zur Problemlösung beinhaltet. Danach folgt die Stufe der Kreativität und Kognition, die das Schaffen von Neuem umfasst. Die vierte Stufe ist das Bewusstsein und die Wahrnehmung der Umwelt. Die höchste Stufe ist das Selbstbewusstsein und die Selbstwahrnehmung, also das Erkennen und Verstehen der eigenen Person, auch als Selbstkritik bekannt. Alle Menschen besitzen diese Fähigkeiten in unterschiedlichen Ausprägungen. Derzeit erreicht KI nur 3-4 dieser Aspekte; ihr fehlen die Weltsicht und ein tiefes Verständnis der Zusammenhänge in der Welt. Es wird noch viele Jahre dauern, bis diese erreicht werden können.[4]

[2] Vgl. [Pos23], S.9-11
[3] Vgl. [Kre23], S.5-6
[4] Vgl. ebenda, S.7

Nun kann man grob erkennen was Intelligenz ist und wie weit das Thema nach aussen geht. Dies ist aber nicht Gegenstand dieser Arbeit. Zu klären ist noch was genau nun eine KI ist. Es wurde versucht, diese oben genannte Intelligenz möglichst erfolgreich, also künstlich, nachzubilden (Siehe Abschnitt weiter unten). Dadurch entsteht die Künstliche Intelligenz. Das 'intelligente' Verhalten wird erforscht, also wie Probleme gelöst werden. Daraus entsteht danach die intelligente Lösung KI. Dabei steht aber nicht nur der Mensch im Vordergrund sondern auch, wie andere Individuen diese Probleme lösen würden.[5]

Aber wie hat diese Forschung angefangen? Im Sommer 1956 wurde in der Dartmouth Conference die akademische KI geboren. Die folgenden Jahre waren dann zwar von einigen Fortschritten geprägt, zum Beispiel (z.B.) die Entwicklung eines Übersetzungssystems um einen Vorteil im kalten Krieg zu haben, konnten aber wegen fehlender Rechenleistung und Hintergrundwissen sowie Finanzieller Unterstützung durch den Staat nicht weiter verfolgt beziehungsweise (bzw.) umgesetzt werden. Erst 1972 wurde in der Standford University ein erstes Expertensystem Namens MYCIN entwickelt, welches die Diagnose von bakteriellen Infektionen unterstützen sollte. Aber auch dieses konnte nicht in die Praxis umgesetzt werden, da ethische und rechtliche Fragen aufkamen. Dennoch gilt es heute als Vorlage für spätere Expertensysteme wie R1/XCON. Erst in den 90er / 2000er Jahren gelang es dann beispielsweise Systeme zu erstellen welche mit einheitlichen mathematischen Notationen viele komplexe Probleme lösen konnten. 2011 schrieb die Unternehmensberatung McKinsey, dass Big Data eine Grundlage für die Innovation, Kompensation und Produktivität sein wird.[6]

Heute sieht man, dass sich diese Vorhersage bewahrheitet hat. KI wird heute von den größten Firmen unter anderem Microsoft (mit Copilot), Google (mit Bard) oder Facebook (mit Meta AI) vorangetrieben und uns erreichen immer wieder neue Schlagzeilen zu diesem Thema. Die Einsatzgebiete sind dabei beinahe unbegrenzt (Siehe Abbildung 1, eine Statistik von schon 2021). Ob nun in der Medizin, der Automobilindustrie oder der Finanzbranche. Überall wird KI immer wichtiger. Es ist also nicht verwunderlich, dass auch Salesforce in das Thema investiert und mit Einstein for Developers den Start in die KI-gestützte Entwicklung wagt.

[5] Vgl. [Kre23], S.8
[6] Vgl. [Wen20], S.2-9

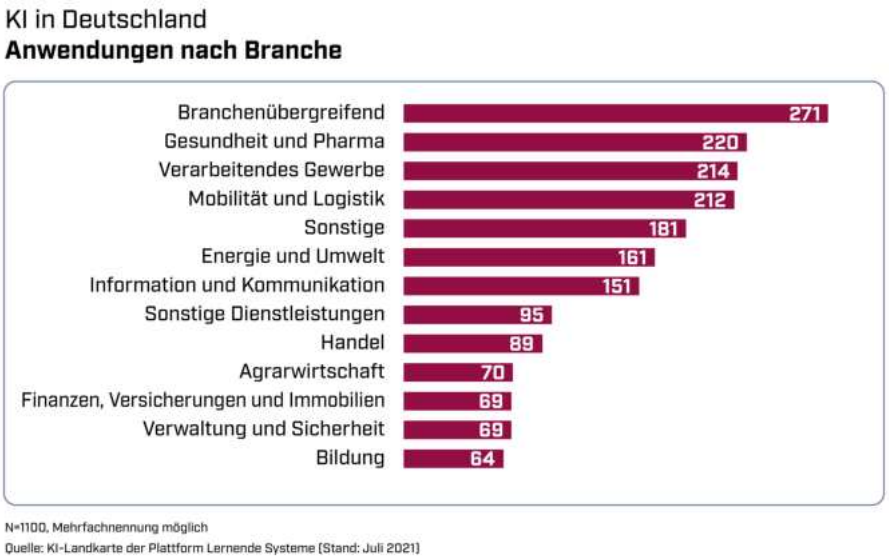

Abb. 1: Nutzung von KI nach Branche (2021)

Zu klären ist nun noch die Frage, wie KI funktioniert. Meistens tauchen im Zusammenhang mit KI die Begriffe maschinelles Lernen, neuronale Netze oder Deep Learning auf. Aber was genau ist der Unterschied bzw. was bedeuten diese Begriffe im Kontext von KI?

Geklärt wurde bereits was eine KI ist, nämlich das Nachahmen menschlicher Intelligenz durch Maschinen. Maschinelles lernen ist dabei der gesamte Prozess, wie die Maschine lernt, Probleme löst und erst zur KI wird. Dies wird durch die Analyse und Verarbeitung großer Datenmengen angetrieben. In diesem Prozess des Lernens wird in zwei Lernmethoden unterschieden. Überwachtes Lernen (Supervised learning) und unüberwachtes Lernen (Unsupervised learning). Bei dem überwachten Lernen sind die Ergebnisse bereits bekannt und werden der Maschine als Lerngrundlage präsentiert. Dies hilft der Maschine, aus Fehlern zu lernen und ihre Leistung zu verbessern. Ein typisches Beispiel hierfür ist die Bilderkennung, bei der die Maschine trainiert wird Bilder zu identifizieren indem sie korrekte Ergebnisse zuvor gezeigt bekommt. Vorerst wird also eine große Datenmenge von richtigen Daten benötigt (zum Beispiel Bilder von Hunden und Katzen). Die Maschine lernt dann aus diesen Daten und verbessert ihre Fähigkeit, Hunde und Katzen zu identifizieren.[7]

[7] Vgl. [Wen20], S.10-37

Bei dem unüberwachten Lernen sind die Ergebnisse nicht im Voraus bekannt. Die Maschine versucht hier, aus den ihr zur Verfügung gestellten Daten selbstständig Muster zu erkennen und diese durch Methoden wie Clustering und Segmentierung zu gruppieren. Empfehlungssysteme, wie sie bei Streaming-Diensten zum Einsatz kommen, sind ein klassisches Anwendungsfeld dieser Lernmethode. Ein entscheidender Aspekt Bei dem maschinellen Lernen ist die Qualität der Daten. Nur hochwertige, gut aufbereitete Daten ermöglichen es der KI, zuverlässige und präzise Ergebnisse zu liefern. Fehlerhafte oder schlecht aufbereitete Daten hingegen führen zu ungenauen Ergebnissen und können die Leistung der KI erheblich beeinträchtigen[8].

Deep Learning versucht die Struktur des menschlichen Gehirns nachzubilden. Dies geschieht durch den Einsatz multipler Schichten von Neuronen in einem neuronalen Netzwerk Siehe Abbildung 2, wobei Informationen zwischen den Neuronen ausgetauscht werden. Beispielsweise kann ein solches Netzwerk dazu trainiert werden, Hunde auf Bildern zu erkennen, indem es zunächst mit Bildern trainiert wird, die als Hund oder kein Hund gekennzeichnet sind. Das neuronale Netz lernt aus jeder Interaktion und verbessert seine Fähigkeit Hunde zu identifizieren kontinuierlich. Dieser Lernprozess und die daraus resultierende Leistungssteigerung sind allerdings stark von der gewählten Technologie, dem Algorithmus und der verfügbaren Datenmenge abhängig.

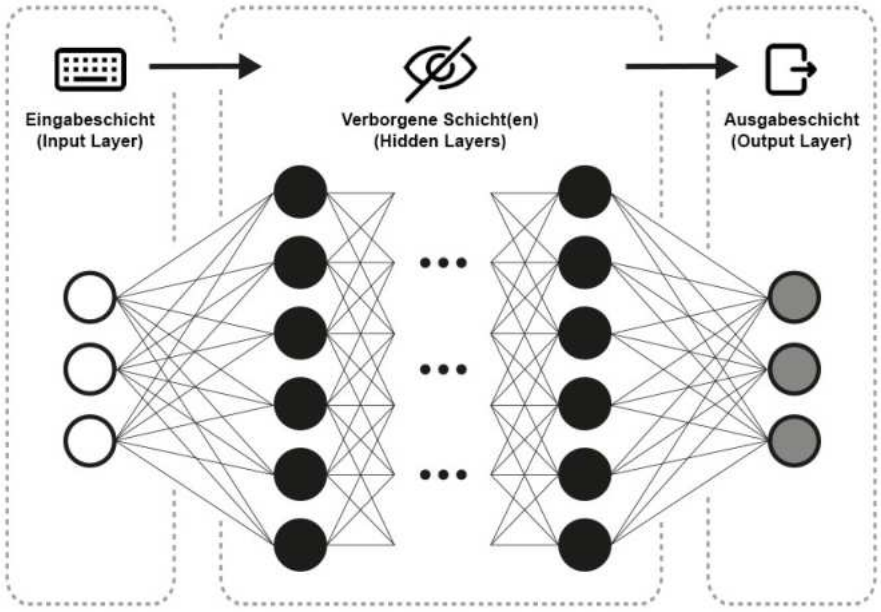

Abb. 2: Neuronale Netze und Deep Learning

Quelle: [Son24]

[8] Vgl. [Wen20], S.10-37

Obwohl es noch viele weitere Facetten und Methoden gibt wie neuronale Netze lernen können, beschränkt sich dieser Überblick darauf ein grundlegendes Verständnis zu vermitteln, ohne zu sehr in die Tiefe technischer Details zu gehen. Der Fokus dieser Arbeit liegt auf den relevanten und verständlichen Aspekten von KI und maschinellem Lernen für die Thematik dieser Arbeit.

Ein Beispiel für ein KI-System ist ChatGPT von OpenAI. ChatGPT bedeutet Generative Pre-trained Transformer und ist ein Large Language Model (LLM), das mit einem riesigen Datensatz aus unmarkierten Texten vorab trainiert (Deep Learning) wurde und nach einer Transformer-Architektur aufgebaut ist. Ein GPT kann aus diesen trainierten Inhalten über ein neuronales Netzwerk neue Texte generieren. ChatGPT kann sowohl direkt per Texteingabe über die OpenAI-Plattform, also über den Chat, als auch über eine Application Programming Interface (API) angesprochen werden. Damit können Drittanbieter-Programme auf die Funktionalitäten von ChatGPT zugreifen und diese in ihre eigenen Anwendungen integrieren oder direkt das Modell nutzen, um eine eigene KI zu trainieren. Beispiele hierfür sind Microsoft Copilot oder GitHub Copilot.

Die allgemeinen Konzepte lassen sich direkt auf die spezifische Forschungsfrage dieser Arbeit anwenden. Eine der Verkaufsargumente von Salesforce ist die Zusammenführung und Konsolidierung von Kundendaten um in der Kundenkommunikation alle relevanten Informationen auf einen Blick zu haben, was es zu einer idealen Plattform für den Einsatz solcher Tools macht.[9] KI könnte dabei helfen Entwicklungsprozesse zu optimieren, indem sie automatisierte Tests, Fehlererkennung oder Performance-Optimierungen in Echtzeit ermöglicht. Dies könnte nicht nur zu einer höheren Produktivität, sondern auch zu einer verbesserten Qualität der entwickelten Anwendungen führen. Ob sich dies in der Praxis realisieren lässt, soll in den nächsten Kapiteln genauer untersucht werden.

2.2 Salesforce und Apex

Salesforce ist eine führende CRM Plattform die Unternehmen dabei unterstützt, ihre Kundenbeziehungen effizient zu verwalten. Die cloudbasierte Lösung bietet eine breite Palette an Tools und Funktionen, die darauf abzielen, den Vertrieb, den Kundenservice, das Marketing oder andere geschäftskritische Prozesse zu optimieren. Durch die Bereitstellung einer einheitlichen Plattform für alle kundenbezogenen Aktivitäten ermöglicht Salesforce Unternehmen, ihre Produktivität zu steigern und eine tiefere Beziehung zu ihren Kunden aufzubauen.

[9] Vgl. [Sal24d]

Salesforce wurde 1999 von Marc Benioff und Parker Harris in San Francisco, Kalifornien, gegründet. Die Vision der Gründer war es, Software as a Service (SaaS) zu revolutionieren und Unternehmen eine kostengünstige, flexible und skalierbare Lösung für das Kundenbeziehungsmanagement zu bieten.

Salesforce war eines der ersten Unternehmen die vollständig auf Cloud-Technologie setzten, was zu einer schnellen Anpassung und Akzeptanz im Markt führte.Im Laufe der Jahre hat Salesforce seine Produktpalette stetig erweitert und durch strategische Akquisitionen und Innovationen seine Marktposition gestärkt. Bedeutende Meilensteine in der Unternehmensgeschichte umfassen die Einführung der Sales Cloud und Service Cloud, die Übernahme von MuleSoft zur Verbesserung der Integration von Daten und Systemen sowie die Entwicklung von Salesforce Einstein, einer KI-Plattform, die maschinelles Lernen und prädiktive Analysen in die Salesforce-Umgebung integriert.[10]

Heute ist Salesforce nicht nur im CRM-Sektor führend, sondern hat auch bedeutende Marktanteile in Bereichen wie Marketing-Automatisierung, E-Commerce und Business Intelligence erobert. Die Plattform wird von Unternehmen jeder Größe und Branche genutzt, von kleinen Start-ups bis hin zu großen multinationalen Konzernen.

Ein Argument für die Nutzung Salesforce ist die Anpassbarkeit und Erweiterbarkeit. Die Plattform bietet eine Vielzahl von APIs und Integrationsmöglichkeiten, die es Entwicklern ermöglichen, maßgeschneiderte Lösungen zu erstellen und Salesforce nahtlos in bestehende Systeme zu integrieren. Mit der AppExchange, einem Marktplatz für Salesforce-Anwendungen, können Unternehmen zusätzliche Funktionen und Anwendungen von Drittanbietern hinzufügen, um ihre spezifischen Anforderungen zu erfüllen. Wie erfolgt nun die Entwicklung in Salesforce? Den Entwicklungsprozess kann man definieren als den Prozess, den die dotSource durchläuft, um z.B. einen Shop auf der Commerce-Cloud zu entwickeln. Dabei benutzen die Entwickler die in den folgenden Abschnitten definierten und erklärten Technologien.

Salesforce nutzt die proprietäre[11] Programmiersprache Apex, die einen Java-ähnlichen Syntax aufweist. Apex ermöglicht es Entwicklern, Code in Systemereignisse wie Button-Klicks oder Aktualisierungen von Datensätzen zu integrieren. Dabei ist sie tief mit den Salesforce-Datenmodellen und -Prozessen vernetzt und kann diese direkt beeinflussen.

Apex unterstützt verschiedene Datentypen, einschließlich des Salesforce-spezifischen Datentyps sObject[12], sowie Sammlungstypen wie Listen, Sets und Maps. Darüber hinaus nutzt Apex Prinzipien der objektorientierten Programmierung, welche die Wiederverwendung von Code durch Klassen und Methoden ermöglicht.

[10] Vgl. [Sal24d]

[11] Eine Programmiersprache, welche speziell für eine Plattform (In dem Falle Salesforce) entwickelt wurde

[12] Eine Klasse, welche ein Objekt wie z.B. Account oder Contact definiert

Diese Methoden können von Auslösern (Triggern) und anderen Klassen aufgerufen werden. Trigger sind Codeblöcke, die auf bestimmte Ereignisse in Salesforce reagieren, wie das Erstellen, Aktualisieren oder Löschen von Datensätzen. Apex ermöglicht auch die Nutzung von Salesforce Object Query Language (SOQL) Statements innerhalb des Codes um Daten abzufragen. SOQL ist ähnlich zu SQL (Structured Query Language) aber speziell für Salesforce-Objekte.

Für die Entwicklung mit Apex stehen mehrere Werkzeuge zur Verfügung. Dazu gehören die Salesforce Extensions (Illuminated Cloud) für die Integrated Development Environment (IDE)s Visual Studio Code (VS-Code) und IntelliJ/WebStorm, sowie die Salesforce-Benutzeroberfläche (Developer Console) im Browser. Diese Werkzeuge bieten Funktionen zur Code-Vervollständigung, Fehlererkennung und Debugging.

Über die IDEs kann auch das Salesforce Command Line Interface verwendet werden. Damit können Entwickler verschiedene Aufgaben von der Kommandozeile aus erledigen, wie das Erstellen und Verwalten von Scratch-Orgs, das Ausführen von Tests und das Deployment von Metadaten. Zudem wird Syntaxüberprüfung und die Integration von Einstein-KI ermöglicht.

Neben Apex können Entwickler auch die Lightning Plattform nutzen, insbesondere die Lightning Web Components (LWC)s. Diese ermöglicht die Entwicklung moderner Webanwendungen unter Verwendung von Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript. Dieses Framework bietet eine hohe Leistung und eine einfachere Entwicklung durch die Nutzung standardisierter Webtechnologien. Salesforce bietet zudem eine Vielzahl von APIs, die den Zugriff auf Salesforce-Daten und -Funktionen ermöglichen. Diese APIs können genutzt werden um Salesforce in andere Systeme zu integrieren oder Daten zwischen Salesforce und externen Anwendungen auszutauschen.

Ein wichtiger Aspekt des Entwicklungsprozesses ist die Versionskontrolle. Projekte und Salesforce-Umgebungen können in ein Versionskontrollsystem wie Git integriert werden. Dies ermöglicht eine effektive Verwaltung von Codeänderungen und Zusammenarbeit im Team. Entwickler können Änderungen verfolgen, Branches erstellen und Pull Requests nutzen, um Codeüberprüfungen zu erleichtern. Salesforce bietet zudem ein integriertes Testframework, das es Entwicklern ermöglicht, automatisierte Tests für Apex-Code zu schreiben und auszuführen. Dies gewährleistet, dass der Code zuverlässig und fehlerfrei ist. Tests können direkt in der Developer Console oder über Pipelines ausgeführt werden, was die Qualität und Stabilität der Anwendungen sicherstellt.

Die Kombination all dieser Werkzeuge, Programmiersprachen und Frameworks ermöglicht leistungsstarke, skalierbare und effiziente Anwendungen auf der Salesforce-Plattform zu erstellen. Von der Entwicklung über die Integration bis hin zum Testen und Deployment bietet Salesforce eine umfassende Umgebung, die alle Aspekte des Entwicklungsprozesses abdeckt und kontinuierliche Verbesserungen sowie eine hohe Produktivität fördert.

3 Herausforderungen in Software-Projekten

Die Frage wo genau KI in Entwicklungsprozesse integriert werden kann stellt das Hauptziel dieser Arbeit dar. Um diese Frage zu unterstützen wird zunächst geklärt, welche Teile des Entwicklungsprozesses KI genau unterstützen kann. Dafür soll unter Entwicklern eine Umfrage durchgeführt werden.

Diese Umfrage zielt darauf ab ein detailliertes Verständnis der aktuellen Praktiken, Wahrnehmungen und Erwartungen bezüglich der Nutzung von KI in der Entwicklung in Salesforce-Projekten zu erlangen. Dabei wird im ersten Abschnitt geklärt, wie sich die Umfrage zusammensetzt, warum Fragen so gewählt und Methodik benutzt wurden. Anschließend werden die Ergebnisse dieser Umfrage ausgewertet und sollen als Grundlage für die weiteren Untersuchungen in den nächsten Kapiteln dienen und letztendlich dazu beitragen, konkrete Empfehlungen für die Implementierung von KI-Lösungen in Salesforce-Projekten abzuleiten.

3.1 Methodik der Umfrage

Durch das Verständnis der Herausforderungen kann untersucht werden, inwiefern die Integration von KI-Technologien, speziell Einstein und Codeium, diese Probleme effektiv adressieren kann.

Ein Ziel der Umfrage ist es außerdem eine Basis für den späteren praktischen Test zu schaffen. Dieser Test sollen zeigen, ob und wie die genannten KI-Lösungen in der Lage sind die Probleme in realen Szenarien zu lösen. Durch solche direkten Anwendungen können die Stärken und Limitationen von Einstein und Codeium in der Praxis ermittelt werden. Darüber hinaus ermöglicht die Umfrage den Rückgriff auf die Meinungen und Erfahrungen von erfahrenen Entwicklern. Ihre Einsichten sind unverzichtbar, um die Wirksamkeit der Integration zu bewerten und um zu verstehen, wie diese Technologien den Entwicklungsprozess verbessern können.

Die Umfrage wird aus einer Kombination von quantitativen Daten Fragen bestehen. Die quantitativen Fragen werden in Form von Multiple-Choice-Fragen gestellt, um klare und präzise Antworten zu erhalten. Dabei gliedert sich die Umfrage in mehrere Themenblöcke:

1. Allgemeine Fragen: Dient als Screening-Abschnitt. Hier werden die Teilnehmer ausgeschlossen, welche keine Entwickler sind. (Siehe Anhang 11)

2. Allgemeine Entwicklung: Hier werden allgemeine Probleme bei der Programmierung erfragt sowie die Meinung zur Notwenigkeit von KI bei diesen Problemen

(Siehe Anhang 13)

3. Nutzung von KI: Es wird erfragt, wie Entwickler zur Nutzung von KI stehen (Wo und wie der Einsatz sinnvoll ist). Außerdem ob sie bereits KI nutzen, die Einschätzung der Qualität sowie wie sie die IT-Sicherheit einschätzen (Siehe Anhang 14,15 und 16)

4. Salesforce und Apex: Fragen bezüglich des Erfahrungsstandes mit Salesforce und Apex, sowie die Nutzung von Apex in der Entwicklung (Siehe Anhang 12)

Wie im Anhang zu sehen bildet die Grundlage der einzelnen Multiple-choice und Bewertungs-Fragen eine Auswahl an Antwortmöglichkeiten, welche vorab definiert wurden. Diese Antwortmöglichkeiten wurden auf der Basis von eigenen Erfahrungen und Recherchen erstellt. Die einzelnen Bewertungs-Fragen wurden mit einer individuellen Skala von 1-5 definiert.

Die Umfrage wird online mittels Microsoft-Forms[13] durchgeführt. Diese Plattform wurde gewählt da Umfragen schnell und einfach erstellt werden können. Auch wird den Teilnehmern eine einfache Zugänglichkeit und Handhabung ermöglicht, da die Umfrage direkt über eine Uniform Resource Locator (URL) aufgerufen im Browser ausgefüllt werden kann. Die Einladung zur Teilnahme an der Umfrage wird breit gestreut: Sie wird per E-Mail über universitäre Verteiler, innerhalb der dotSource sowie an den Kreis einiger Freunde versendet.

Zusätzlich wird die Umfrage auf der Plattform 'PollPool' geteilt, um auch potenzielle Teilnehmer außerhalb des direkten beruflichen und akademischen Umfelds zu erreichen und somit eine diverse Teilnehmerschaft zu sichern. Die Website bietet an die Umfrage in bestimmten Zielgruppen zu teilen.[14]

Um die Qualität und Verständlichkeit der Umfrage zu gewährleisten, wird eine kleine Testgruppe bestehend aus fünf erfahrenen Salesforce-Entwicklern vorab befragt. Diese Gruppe wird die Umfrage vor der allgemeinen Aussendung ausfüllen und Feedback geben. Dieses Vorgehen dient dazu, mögliche Unklarheiten oder technische Probleme zu identifizieren und zu beheben bevor die Umfrage einer größeren Öffentlichkeit zugänglich gemacht wird. Die Umfrage wurde Ende Mai 2024 durchgeführt und wird über einen Zeitraum von zwei Wochen bis Mitte Juni offen sein. Dieser Zeitrahmen soll den Teilnehmern genügend Gelegenheit bieten die Umfrage auszufüllen, während gleichzeitig eine zügige Auswertung der Ergebnisse sichergestellt wird.

[13] Vgl. [Mic24]
[14] Vgl. [Pol24]

3.2 Auswertung der Umfrage

Insgesamt nahmen 198 Personen an der Umfrage teil, von denen 104 aufgrund der Screening-Frage ausgeschlossen wurden, da sie keine Entwickler waren. Somit blieben 94 Teilnehmer übrig, die relevante Antworten zur Auswertung beisteuerten.

Häufige Probleme bei reinem Coding: Die Umfrage ergab, dass die häufigsten Probleme bei dem Coding die Einhaltung von Standards und Best Practices (64 Nennungen) sowie die Fehlerbehebung (63 Nennungen) sind. Das Erstellen von Tests (59 Nennungen) und Dokumentationen (58 Nennungen) wurden ebenfalls als zeitaufwendig und herausfordernd empfunden. Refactoring und Codeerklärungen erhielten beide über 50 Nennungen, während die restlichen Bereiche im 30er-Bereich lagen (Siehe Abbildung 3).

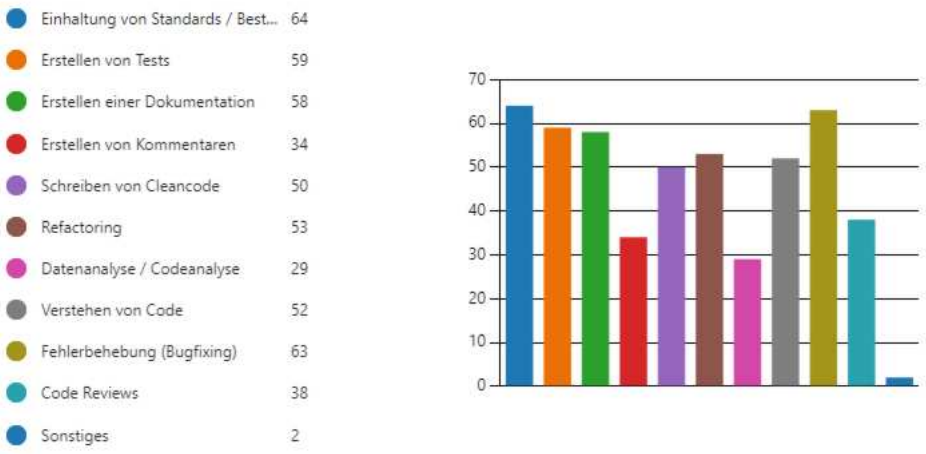

Abb. 3: Häufige Probleme bei dem Coding

Wunsch nach Unterstützung in verschiedenen Bereichen: In allen abgefragten Bereichen gab es eine deutliche Nachfrage nach Unterstützung, wobei die Mehrheit der Befragten starke oder sehr starke Unterstützung bevorzugte. Besonders wichtig sind dabei die Einhaltung von Standards und Best Practices, das Erstellen von Tests sowie das Schreiben von Dokumentationen. Das Schreiben von Cleancode und Refactoring wurden ebenfalls als hohe Prioritäten identifiziert. Weniger Unterstützung wurde bei Aufgaben wie dem Schreiben von Kommentaren und Code-Reviews gewünscht, wobei hier mehr Teilnehmer als bei den anderen Fragen keine Unterstützung benötigten (Siehe Abbildung 4).

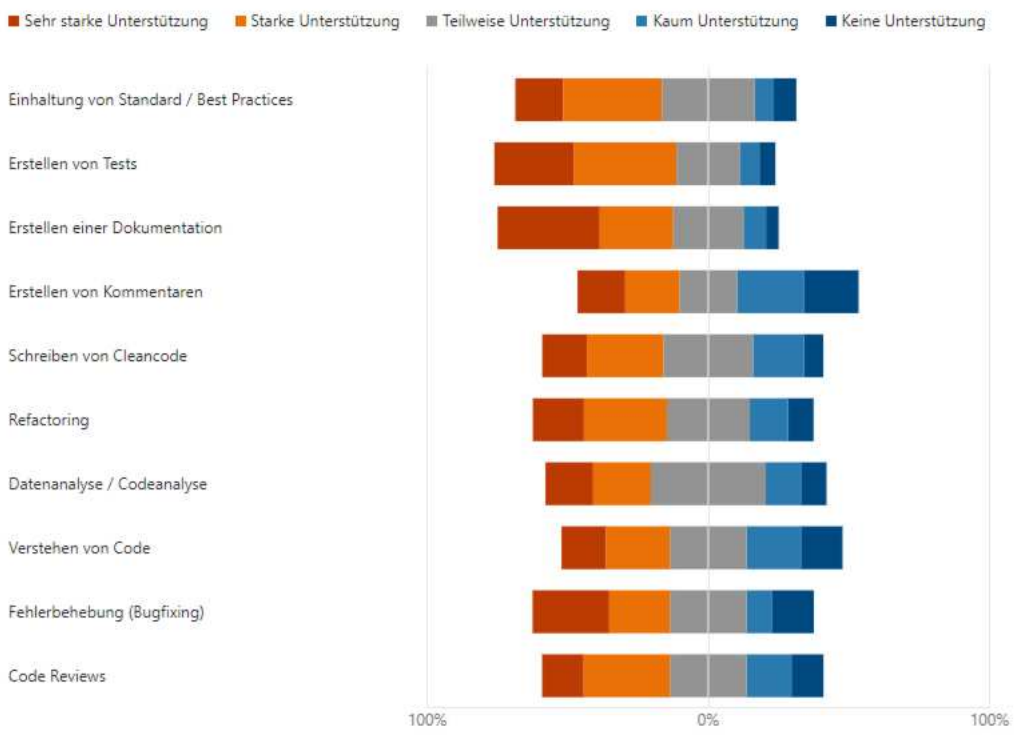

Abb. 4: Wunsch eines Copiloten in verschiedenen Bereichen

Sinnhaftigkeit von Copilot: Die Mehrheit der Befragten sieht KI als Copilot als hilfreich an, wobei die Bewertungen 8, 9 und 10 insgesamt 53 Nennungen umfassen. Es gibt jedoch auch einige Skeptiker, mit Bewertungen von 0 bis 5, die insgesamt 14 Nennungen ausmachen. Dies zeigt eine überwiegend positive Haltung gegenüber der Verwendung von KI als Copilot, jedoch sind einige Bedenken oder geringere Erwartungen vorhanden.

Einsatzgebiete von KI: Die häufigsten Bereiche, in denen KI unterstützen kann, sind das Erstellen von Tests (75 Nennungen), Dokumentation (66 Nennungen) und Kommentare (58 Nennungen). Auch Refactoring (58 Nennungen), Einhaltung von Standards (58 Nennungen) und das Verstehen von Code (55 Nennungen) wurden als wichtige Unterstützungsgebiete identifiziert. Code-Reviews wurden mit 37 Nennungen am wenigsten als unterstützungswürdig angesehen (Siehe Abbildung 5).

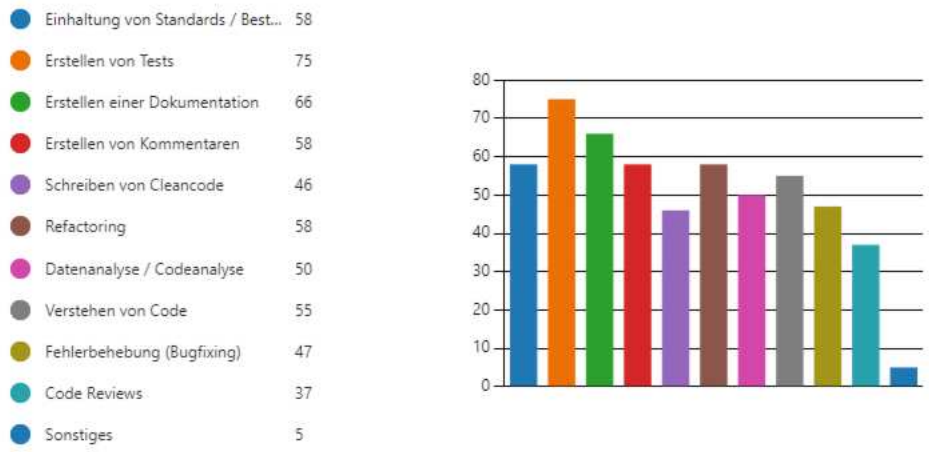

Abb. 5: Einsatzgebiete von KI

Bewertung der Unterstützung durch KI in verschiedenen Bereichen: Die meisten Bereiche wurden mehrheitlich als Gut bewertet, allerdings gab es viele Unentschiedene ('Weiß nicht'), was darauf hindeutet, dass viele Befragte keine klare Meinung oder keine ausreichende Erfahrung mit KI-Unterstützung in diesen Bereichen haben. Besonders hoch waren die Zustimmungswerte im Bereich Erstellen von Tests und Erstellen einer Dokumentation. Die höchste Anzahl an sehr schlecht-Bewertungen fand sich im Bereich Fehlerbehebung (Bugfixing) (Siehe Abbildung 6).

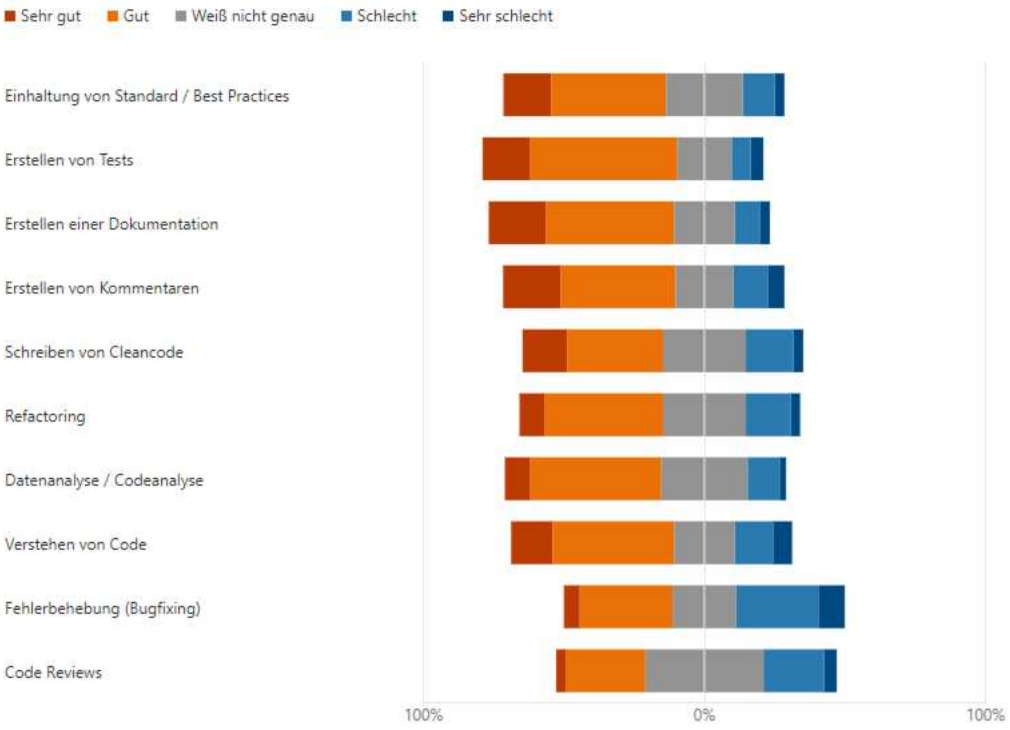

Abb. 6: Qualität von KI verschiedenen Bereichen

Nutzung von KI Die Mehrheit der Befragten nutzt bereits KI in ihren Entwicklungsprozessen, entweder gelegentlich (39 Nennungen) oder regelmäßig (35 Nennungen). Eine kleinere Gruppe von Befragten hat bisher keine KI genutzt, wobei die Hälfte dieser Gruppe plant, dies in Zukunft zu tun (9 Nennungen). 9 Befragte wollen auch in Zukunft keine KI nutzen (Siehe Abbildung 7). ChatGPT ist das am häufigsten genutzte KI-Tool (55 Nennungen), gefolgt von Codeium (44 Nennungen). Es gibt eine Vielzahl von Tools, die genutzt werden, einschließlich spezieller interner Lösungen und Kombinationen verschiedener Tools. Einstein und einige spezifische Lösungen wie GitHub Copilot und Microsoft Copilot wurden ebenfalls genannt, jedoch seltener.

Abb. 7: Nutzung von KI-Tools

Aufgaben, die mit KI optimiert werden: Die meisten Aufgaben werden nicht oder nur selten mit KI optimiert, insbesondere Code Reviews und das Erstellen von Kommentaren. Einige Aufgaben wie das Schreiben von Cleancode, Refactoring und das Verstehen von Code werden von einer signifikanten Anzahl von Befragten zumindest teilweise mit KI optimiert. Es gibt einen kleinen, aber wichtigen Anteil von Befragten, die viele dieser Aufgaben oft oder sehr oft mit KI optimieren (Siehe Abbildung 8).

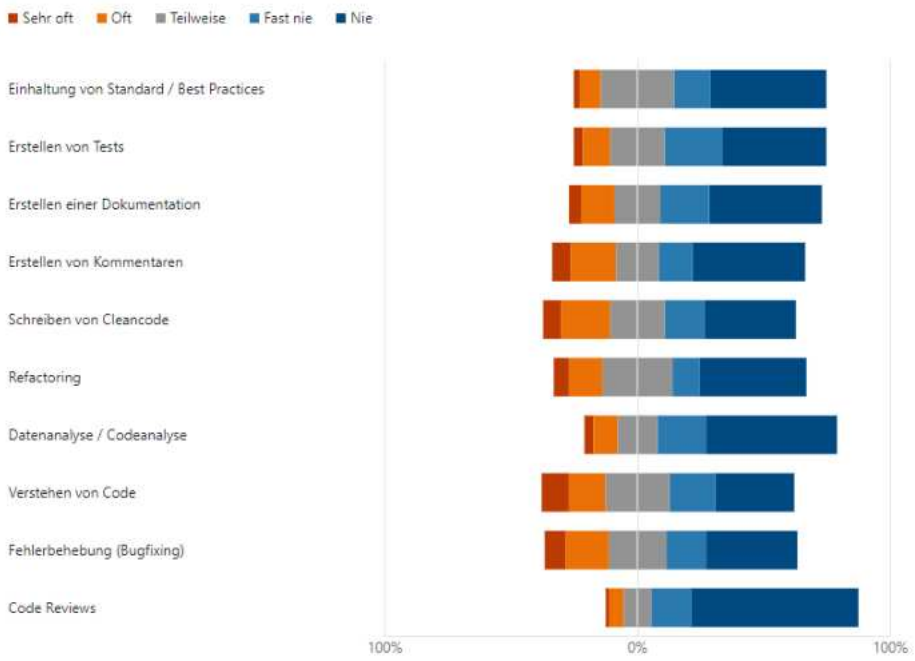

Abb. 8: Aufgaben die mit KI optimiert werden

Bewertung der optimierten Aufgaben durch KI: Gut ist die am häufigsten gewählte Bewertung, was darauf hindeutet, dass die Mehrheit der Befragten die Qualität der KI-Hilfe als positiv empfindet. Es gibt eine signifikante Anzahl von Weiß nicht-Antworten, was darauf hinweist, dass viele Befragte keine klare Meinung oder keine ausreichende Erfahrung mit der Qualität der KI-Hilfe in diesen Bereichen haben. Besonders hoch sind die Zustimmungswerte im Bereich Erstellen von Tests und Erstellen einer Dokumentation. Einige Bereiche wie Fehlerbehebung (Bugfixing) und Code Reviews, haben eine höhere Anzahl von Sehr schlecht-Bewertungen (Siehe Abbildung 9).

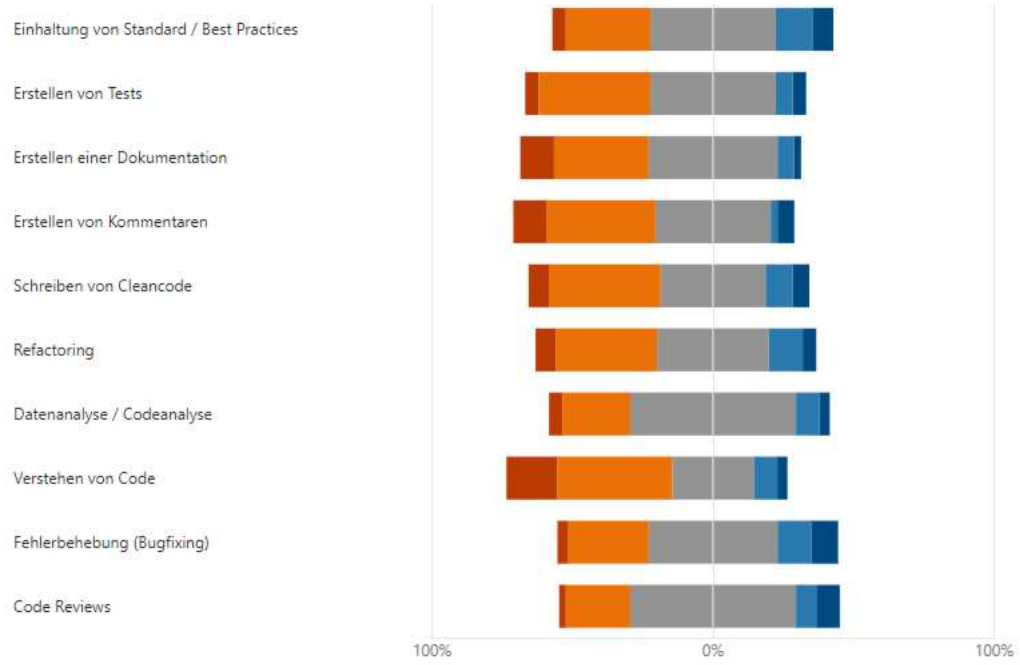

Abb. 9: Qualität der optimierten Aufgaben durch KI

Die Auswertung der Umfrage zeigt eine überwiegend positive Einstellung der Entwickler gegenüber der Nutzung von KI in ihren Arbeitsprozessen. Eine signifikante Mehrheit der Befragten nutzt KI regelmäßig oder gelegentlich in ihren Entwicklungsprozessen. Diese breite Akzeptanz verdeutlicht, dass die Vorteile der KI bereits erkannt und genutzt werden. Eine kleinere Gruppe von 18 Befragten nutzt derzeit keine KI, wobei jedoch 9 dieser Personen planen, dies in Zukunft zu tun. Die häufigsten Probleme in den Entwicklungsprozessen betreffen die Einhaltung von Standards, die Fehlerbehebung (Bugfixing), das Erstellen von Tests und die Dokumentation. Diese Problembereiche korrelieren stark mit den Bereichen, in denen die Unterstützung durch KI besonders gewünscht wird. Diese Korrelation zeigt, dass KI gezielt zur Lösung dieser Herausforderungen eingesetzt werden kann. Auch in genau diesen Bereichen gibt es eine klare Nachfrage nach KI-Unterstützung, was auf die Komplexität und den hohen Arbeitsaufwand dieser Aufgaben hinweist.

Die Qualität der KI-Hilfe wird in den meisten Bereichen als gut bewertet, insbesondere bei dem Erstellen von Tests und Dokumentationen. Diese positiven Bewertungen bestätigen die Effektivität der KI in diesen Bereichen. Obwohl die Bewertungen sehr gut seltener sind, unterstreichen sie dennoch das Potenzial der KI hohe Standards zu erfüllen. Allerdings haben Bereiche wie Fehlerbehebung und Code Reviews relativ viele schlecht und sehr schlecht Bewertungen erhalten, was auf Verbesserungsbedarf hinweist. Die hohe Anzahl an weiß nicht Antworten deutet darauf hin, dass viele Befragte entweder keine Erfahrung mit KI in diesen Bereichen haben oder unsicher über deren Fähigkeiten sind, was einen Bedarf an weiterer Aufklärung und Schulung signalisiert.

Bei den genutzten KI-Tools sind ChatGPT und Codeium am beliebtesten, was ihre Benutzerfreundlichkeit und Nützlichkeit unterstreicht. Es gibt eine Vielzahl von KI-Tools die genutzt werden, einschließlich spezialisierter Tools wie Microsoft Copilot und Einstein for Developers, was eine breite Akzeptanz und Diversität in der Nutzung zeigt. Viele Aufgaben, wie das Schreiben von Clean Code, Refactoring und das Verstehen von Code, werden bereits mit KI optimiert. Dies zeigt, dass diese in diesen Bereichen bereits effektiv eingesetzt wird. Aufgaben wie Code Reviews und das Erstellen von Kommentaren werden noch selten mit KI optimiert, was auf Potenzial für weitere Implementierungen deutet.

Unternehmen und Entwicklungsteams sollten die Nutzung von KI in weiteren Entwicklungsbereichen fördern, insbesondere in den Bereichen, die derzeit selten optimiert werden, wie Code Reviews und Kommentieren. Die Implementierung von KI in den genannten Problembereichen kann dazu beitragen, die Effizienz und Qualität der Entwicklungsprozesse zu steigern. Dies beweist die Sinnhaftigkeit dieser Arbeit. Es sollte in umfassende Schulungsprogramme investiert werden, um das Wissen und das Vertrauen der Entwickler in die Fähigkeiten der KI zu stärken. Dies kann die Anzahl der weiß nicht Antworten reduzieren und die Akzeptanz der KI erhöhen. Schulungen sollten sich auf die praktische Anwendung von KI-Tools in realen Entwicklungsprojekten konzentrieren, um den Entwicklern zu zeigen, wie sie die KI effektiv nutzen können. Trotzdem könnte dies auch auf den aktuellen Stand der Technik zurückzuführen sein, da viele KI-Tools noch in der Entwicklung sind und noch nicht alle Anforderungen erfüllen.

Falls eigene Tools entwickelt werden sollte die Entwicklung und Verbesserung dieser sich außerdem auf die Bereiche konzentrieren, die derzeit als weniger effektiv bewertet werden, wie Fehlerbehebung und Code Reviews. Hier besteht ein klarer Bedarf an besseren Lösungen. Benutzerfeedback sollte genutzt werden, um die Funktionalität und Benutzerfreundlichkeit der Tools kontinuierlich zu verbessern.

Unternehmen sollten KI-Tools gezielt in den Bereichen implementieren, die als besonders problematisch identifiziert wurden, solange diese Tools die Anforderungen der Entwickler erfüllen. Dies umfasst die Einhaltung von Standards, Bugfixing und das Erstellen von Tests. Die gezielte Implementierung kann durch Pilotprojekte und Testläufe unterstützt werden, um die Wirksamkeit der KI-Tools in spezifischen Szenarien zu evaluieren.

Die Analyse zeigt, dass KI ein großes Potenzial zur Optimierung von Entwicklungsprozessen hat und bereits in vielen Bereichen positiv bewertet wird. Durch gezielte Schulung, Erweiterung der Nutzung, kontinuierliche Verbesserung der Tools und eine problemorientierte Implementierung kann die Effizienz und Qualität der Entwicklungsarbeit weiter gesteigert werden. Die Befragten sehen in der KI einen wertvollen Copiloten, der sie in ihren täglichen Aufgaben unterstützen und entlasten kann. Die positive Einstellung und die bereits weit verbreitete Nutzung von KI-Tools sind starke Indikatoren für die zukünftige Relevanz und den Erfolg der KI in der Softwareentwicklung.

4 Künstliche Intelligenz in Salesforce

Die Integration von KI in Salesforce bietet ein enormes Potenzial welches in diesem Kapitel genauer Untersucht werden soll. In der Salesforce-Umgebung stehen mit Einstein for Developers und Codeium zwei spezialisierte KI-Tools zur Verfügung, die Entwicklern bei verschiedenen Aufgaben unterstützen sollen. Dieses Kapitel untersucht die spezifischen Funktionen und Anwendungsbereiche dieser Tools, vergleicht ihre Leistungsfähigkeit in praktischen Tests und bewertet ihre Eignung zur Optimierung von Entwicklungsprozessen.

4.1 Einstein for Developers

Einstein for Developers ist ein KI-gestütztes Entwicklertool von Salesforce welches auf einem angepassten LLM namens CodeGen-2.5 basiert. Dieses Tool unterstützt den Entwicklungsprozess indem es auf anonymisierte Codemuster trainiert wurde und generative KI-Techniken anwendet. Obwohl sich Einstein for Developers derzeit in der Beta-Version befindet, bietet es bereits wertvolle Unterstützung für Entwickler. Salesforce schreibt dazu, dass das Model auf ihren fünf grundlegende Prinzipien basiert:[15]

- Genauigkeit: Die Modelle von Einstein for Developers legen großen Wert auf Genauigkeit, Präzision und Rückruf. Die Modellergebnisse werden nach Möglichkeit mit Erklärungen und Quellen untermauert. Es wird empfohlen,dass ein Mensch die Modellergebnisse überprüft, bevor sie an Endbenutzer weitergegeben werden.
- Sicherheit: Salesforce arbeitet daran, Verzerrungen, Toxizität und schädliche Ergebnisse in den Modellen mithilfe verschiedener Techniken zu verringern. Der Schutz der Privatsphäre persönlich identifizierbarer Informationen (PII) wird durch Schutzmaßnahmen um diese Daten herum gewährleistet.
- Transparenz: Salesforce stellt sicher, dass die Daten, die in den Modellen verwendet werden, ihre Herkunft respektieren und dass die Zustimmung zur Verwendung der Daten vorliegt.
- Empowerment: Die Modelle werden so entwickelt, dass sie die menschliche Beteiligung in den Arbeitsablauf einbeziehen, wann immer dies möglich ist.
- Nachhaltigkeit: Salesforce ist bestrebt, Modelle der richtigen Größe zu bauen, bei denen die Genauigkeit im Vordergrund steht und der CO2-Fußabdruck reduziert wird.

[15] Vgl. [Sal24b]

Diese Prinzipien sollen sicherstellen, dass Einstein for Developers nicht nur ein leistungsstarkes, sondern auch ein verantwortungsbewusstes Werkzeug für Salesforce-Entwickler ist.

Einstein for Developers kann über verschiedene Entwicklungsumgebungen integriert werden. Entwickler können das Tool als VS-Code Extension oder über JetBrain-IDEs mit Illuminated Cloud nutzen. Allerdings ist es derzeit nicht in der European Union (EU) verfügbar. Nutzbar ist das Tool für die Salesforce-Editionen Developer, Enterprise, Partner Developer, Performance und Unlimited. Für diejenigen, die Einstein for Developers testen möchten, kann das Tool unentgeltlich über eine Trailhead-Org genutzt werden. Um Einstein for Developers in einer Salesforce-Org zu aktivieren, muss im Setup-Bereich die Option „Einstein for Developers“ aktiviert werden. Danach ist das Tool in beiden Extensions verfügbar und kann genutzt werden.[16]

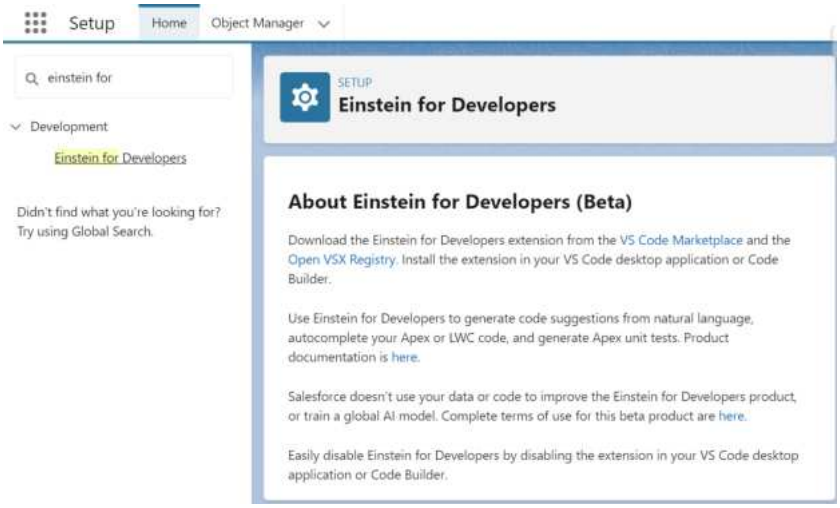

Abb. 10: Einstein for Developers in Salesforce

Einstein for Developers bietet eine Vielzahl von Funktionen die den Entwicklungsprozess erheblich erleichtern. Zu diesen Funktionen gehören einerseits die Codegenerierung. Entwicklern wird ermöglicht Apex-Code aus Eingabeaufforderungen in natürlicher Sprache zu generieren. Über ein Chat-Fenster kann eine Anforderungen formuliert werden und das Tool erstellt den entsprechenden Code. Zu dem werden auch Statische Analysen und Sicherheitsscans angeboten. Dies trägt dazu bei, die Codequalität zu verbessern und Sicherheitsrisiken zu minimieren. Auch Inline-Autovervollständigung ist für Apex und LWCs vorhanden. Zuletzt besteht auch die Möglichkeit Unit-Tests zu Generieren. Der Projektkontext kann über einen manuellen Befehl in der Commandozeile von Salesforce hinzugefügt werden. Wie diese Funktionen im Detail funktionieren und wie sie in der Praxis eingesetzt werden können, wird im nächsten Kapitel genauer untersucht.[17]

[16] Vgl. [Sal24b]
[17] Vgl. ebenda

4.2 Codeium

Codeium ist ein KI-gestütztes Toolkit das von Exafunction Incorporated (INC) entwickelt wurde um die Effizienz und Produktivität von Entwicklern zu steigern. Dieses Tool kann über eine Erweiterung unter anderem in VS-Code, IntelliJ/WebStorm und über 40 weiteren IDEs genutzt werden. Codeium bietet Autocomplete an die auf dem Kontext des geöffneten Projekts basieren. Außerdem existiert ein integrierter Chat sowie eine semantische Suchtechnologie die es erlaubt ein gesamtes Repository zu durchsuchen und Code-Snippets, Dokumentationen und Dateien einfach zu finden. Unterstützt werden über 70 Programmiersprachen.[18]

Codeium ist in drei verschiedenen Versionen erhältlich, um den Bedürfnissen unterschiedlicher Benutzergruppen gerecht zu werden. Die unentgeltliche Version bietet grundlegende Funktionen für Einzelentwickler und kleine Teams. Die Teams-Version, die für 12 Euro pro Benutzer und Monat verfügbar ist, richtet sich an mittlere bis große Entwicklungsteams und bietet erweiterte Kollaborationsfunktionen (Dort wird auch eine gesonderte GPT-4 Version angeboten). Für große Organisationen bietet die Enterprise-Version maßgeschneiderte Lösungen mit Selbst-Hosting, Feinabstimmung und speziellem Training, um den Anforderungen der Organisation gerecht zu werden.[19]

Die zugrunde liegende generative KI basiert auf einem LLM, die mit Daten unter der General Public License trainiert wurden. Unternehmen haben die Möglichkeit, Codeium mit ihrem eigenen permissiven Code zu trainieren, um noch relevantere Ergebnisse zu erzielen.[20] Auf Grundlage der gegebenen Informationen wird für den praktischen Test eine erste These festgelegt: Codeium wird bei der Erstellung von Code besser sein als Einstein for Developers. Dies wird begründet durch das Modell, auf dem Codeium basiert. Es wird angenommen, dass ein GPT-4 Modell bessere Ergebnisse liefert als ein CodeGen-2.5 Modell.

4.3 Praktischer Test und Aufbau

Das Hauptziel dieses Tests ist es, die Effektivität und die spezifischen Einsatzgebiete der beiden KI-Tools Codeium und Einstein for Developers bei der Optimierung von Entwicklungsprozessen in Salesforce-Umgebungen zu vergleichen. Die Untersuchung soll aufzeigen, in welchen Bereichen diese Tools besonders nützlich sind und welche Vorteile sie den Entwicklern bei der Umsetzung von Salesforce-Features bieten können.

[18] Vgl. [Exa24]
[19] Vgl. ebenda
[20] Vgl. ebenda

Durch diesen Vergleich sollen präzise Empfehlungen für die Integration und Nutzung dieser Tools in der Praxis abgeleitet werden. In den vorherigen Kapiteln wurden die beiden KI-Tools inklusive ihrer Funktionalitäten bereits ausführlich beschrieben. Im folgenden Abschnitt wird der Testaufbau und die Methodik zur Durchführung des Tests erläutert.

4.3.1 Methodik

Da es grundsätzlich um Salesforce-Entwicklung handelt wird eine Sandbox als Testumgebung genutzt. In dieser Sandbox können die Funktionen der beiden Tools in realen Szenarien getestet werden. Die drei Disziplinen die im Fokus dieses Tests stehen sind Tests (Apex Unittests), Trigger und LWCs. Jede dieser Disziplinen beinhaltet mehrere Aufgaben: das Erstellen des Codes (also das Generieren eines Triggers, einer LWC oder eines Tests), das Korrigieren fehlerhaften Codes, das Dokumentieren und Beschreiben sowie das Neu-Generieren von Code. Diese Aufgaben zielen darauf ab, die in der Umfrage identifizierten Probleme anzusprechen, also die Einhaltung von Standards, Fehlerbehebung, Erstellen von Tests, Codeanalyse und Dokumentation. Die Bewertung erfolgt anhand mehrerer Kriterien:

1. Fehlerbehebung: Hierbei wird überprüft, ob der fehlerhafte Code korrekt behoben wurde. Dies erfolgt durch das Ausführen von Tests im Hintergrund, die bestätigen, dass keine Fehler mehr vorhanden sind.

2. Codequalität: Die Lesbarkeit und Qualität des Codes wird durch statische Codeanalyse Tools bewertet. Diese beinhalten die Überprüfung des Codes auf Einhaltung von Best Practices und Standards ohne Ausführung des Programms.

3. Zeitmessung: Die Zeit, die für jede Aufgabe benötigt wird wird gemessen, um die Effizienz der Tools zu bewerten.

4. Fehlerfreiheit: Die Fehlerfreiheit des Codes wird durch die Ausführung von Tests überprüft, die sicherstellen, dass der Code wie erwartet funktioniert.

5. Dokumentation: Es wird bewertet, ob der Code gut beschrieben und dokumentiert ist, um die Verständlichkeit und Wartbarkeit des Codes zu gewährleisten.

Theoretisch muss aufgrund einer objektiven Bewertung diese Aufgaben zu tausenden durchgeführt werden um einen aussagekräftigen Durchschnitt zu finden. Da aufgrund der zeitlichen Beschränkungen eine solche Anzahl von Messungen nicht möglich sind, wird eine stichprobenartige Untersuchung mit je 5 Durchläufen durchgeführt.

Die Entscheidung basiert auf der schnellen Entwicklung der KI-Tools. In dieser Arbeit wird nur der aktuelle Stand dieser Untersucht. Dies kann sich aber in den nächsten Wochen oder Monaten drastisch ändern. Um weitere Best Practices im Promting zu untersuchen, wird jede Aufgabe in den Kategorien mit unterschiedlichen Prompts durchgeführt. Dabei werden folgende Varianten verwendet:

1. 5x mit dem Prompt 'Du bist ein Junior Entwickler'
2. 5x mit dem Prompt 'Du bist ein Senior Entwickler'
3. 5x mit dem Prompt 'Du bist ein Trainee Entwickler'
4. 5x ohne spezifischen Promp

Aus dieser Disziplin entsteht für den praktischen Test folgende zweite These: Das Promting wird einen signifikanten Einfluss auf die Qualität und Geschwindigkeit der Ergebnisse haben. Die Bewertung aller Disziplinen, abgesehen von der Zeit, erfolgt anhand von Kriterien, welche bei den einzelnen Aufgaben beschrieben und festgelegt werden. Ist eins der Kriterien erfüllt, bekommt die Disziplin einen Punkt. Außerdem werden die Ergebnisse hinsichtlich der Syntax und Semantik des generierten Codes bewertet:

1. 0 Punkte: Mehr als 2 Fehler in Syntax und Semantik
2. 1 Punkt: Mehr als 2 Fehler in einem Bereich
3. 2 Punkte: Ein Fehler in beiden oder einem Bereich
4. 3 Punkte: Keine Fehler

4.3.2 Vorbereitung und Durchführung

Um die Tests durchzuführen wurden mehrere vorbereitende Maßnahmen getroffen um eine optimale Testumgebung zu gewährleisten. Zunächst wurde eine Salesforce Trailhead-Organisation erstellt, die als einfache Testumgebung diente. Diese Umgebung kann kostenlos über die Lernplattform Trailhead von Salesforce erstellt und abgerufen werden.[21] Eine Lightning App wurde konfiguriert, um die Integration der LWC zu ermöglichen. Diese Application (APP) war so eingerichtet, dass sie die LWCs nach jedem Push automatisch aktualisierte, was eine sofortige Überprüfung der Änderungen ermöglichte.

Für das Test-Szenario wurden zwei benutzerdefinierte Objekte erstellt: Employee__c und Task__c. Beide Objekte wurden mit mehreren benutzerdefinierten Feldern versehen. Zusätzlich wurden 50 Datensätze aus diesen Objekten in die Testumgebung eingefügt um ausreichende Testdaten bereitzustellen. Dies dient zur Überprüfung, ob auch die Nutzung von Custom Data für die KI möglich ist.

Die eigentlichen Tests wurden in VS-Code durchgeführt. Diese IDE wurde gewählt, da Codeium und Einstein for Developers hier bereits vollständig integriert und auf dem neuesten Stand waren. Einstein ist in IntelliJ nicht in der aktuellsten Version verfügbar, weshalb VS-Code die bevorzugte Wahl war. Die Umgebung wurde durch das Klonen des Repositories und die Autorisierung der Salesforce-Organisation vorbereitet. Beide Tools wurden als Plugins heruntergeladen und installiert.

Zur Sicherstellung der Codequalität wurden die spezifischen Linter ESLint für die LWCs und ApexPMD für den Apex-Code hinzugefügt. Außerdem wurde eine detaillierte Excel-Tabelle vorbereitet, um die Bewertungskriterien systematisch zu erfassen und die Ergebnisse der Tests zu dokumentieren. Diese kann im Anhang Tabelle 2 - 6 eingesehen werden.

Die Durchführung des Tests erfolgte nacheinander nach den einzelnen Kategorien - Tests (Apex Unittests), Trigger und Lightning Web Components (LWC). Für jede Aufgabe wurde ein spezifischer Prompt vorbereitet damit beide Tools die gleichen Eingaben haben. Angefangen mit dem allgemeinen Promts, welche bereits erwähnt wurden, gefolgt von den einzelnen spezifischen Promts. Bei dem abschicken der jeweiligen Aufgabe wurde die Zeit mittels einer Stoppuhr gemessen. Ist die Aufgabe abgeschlossen, wurde der generierte Code überprüft und bewertet. Die Ergebnisse wurden in der vorbereiteten Excel-Tabelle festgehalten.

[21] Vgl. [Sal24c]

Lightning Web Components

Der erste Test bestand aus folgendem Promt: 'Erstelle mir eine Komponente welche eine Liste von Employees (Employee__c) aus der Salesforce-Datenbank anzeigt. Jeder Mitarbeiter soll mit Name (Name__c), Position (Position__c) und Kontaktinformationen (Email__c, Phone__c) dargestellt werden. Dabei soll zum anzeigen die lightning-card-komponente genutzt werden. Außerdem soll es möglich sein durch die Mitarbeiter zu blättern, falls die Liste über 10 Einträge hat.' Die Kriterien für die Bewertung umfassen: ob die Liste angezeigt wurde, ob sie korrekt befüllt war, ob die lightning-card-Komponente verwendet wurde, ob die Liste blätterbar war und ob die Extensible Markup Language (XML)-Datei korrekt angegeben wurde. Maximal konnten also fünf Punkte erreicht werden.

Codeium Ergebnisse[22]:

1. Beste Ergebnisse: Der Prompt 'Du bist ein Senior Entwickler' führte zu den besten Ausgaben. Die Komponente wurde in allen Fällen korrekt erstellt und die Aufgabe am schnellsten abgeschlossen. Der Senior-Prompt erzielte konstant funktionierende Komponenten mit vollständiger Implementierung der Paging-Funktion und korrektem Controller.
2. Fehleranalyse: Der Linter schlug bei allen erstellten Komponenten an, insbesondere bei der Nutzung alter HTML-Templates und fehlenden Create, Read, Update, Delete (CRUD) Berechtigungsvalidierungen vor Data Manipulation Language (DML)-Anweisungen, falls ein Apex-Controller erstellt wurde.
3. Komponentenfunktionalität: Bei Verwendung des Senior- und Junior-Prompts funktionierten die Komponenten am häufigsten korrekt. Bei den Prompts ohne spezifische Entwicklerrolle und dem Trainee-Prompt war die Erfolgsrate nur bei 60 Prozent. Ein häufiges Problem war das Fehlen eines Controllers bei Trainee-Prompts und unvollständige Paging-Funktionen bei Junior-Prompts.

Einstein Ergebnisse[23]:

1. Beste Ergebnisse: Der Trainee-Prompt führte zu den besten Ergebnissen. Hier wurden sowohl HTML- als auch JavaScript-Dateien erstellt und die Komponente funktionierte am häufigsten korrekt. Der Senior-Prompt erzielte ebenfalls gute Ergebnisse, war jedoch weniger konsistent.

[22] Vgl. Spalte 5-8 Tabelle 2
[23] Vgl. Spalte 5-8 Tabelle 5

2. Fehleranalyse: Im Durchschnitt wiesen die erstellten Komponenten 2,6 syntaktische oder semantische Fehler auf. Der Linter schlug bei keiner Komponente an.

3. Komponentenfunktionalität: Bei dem Senior-Prompt wurde die Komponente am häufigsten korrekt erstellt, jedoch gab es häufig Probleme mit der Erstellung von HTML-Dateien trotz spezifischer Hinweise im Prompt. Trainee-Prompts führten zu vollständigeren Ergebnissen mit korrekt erstellten HTML- und JavaScript-Dateien. Junior-Prompts gaben manchmal Aura-Komponenten anstelle von LWCs aus.

Die nächste Aufgabe beinhaltet die bestehenden Fehler in einer LWC zu korrigieren. Der Promt lautet: 'Bitte korrigiere den Code so, dass die Komponente wieder funktioniert und liste mir alle Fehler auf, die du gefunden hast.' Die Kriterien zur Bewertung waren: das Erkennen und Korrigieren von Fehlern im HTML (falscher Variablenname und Methodenaufruf). Beheben einer Fehlermeldung, die nicht angezeigt wurde. Korrektur eines Fehlers, bei dem ein then-Aufruf fehlerhaft verwendet wurde. Behebung einer Racing Condition in der LoadRecords-Methode.

Codeium Ergebnisse[24]:

1. Beste Ergebnisse: Die Prompts Du bist ein Senior Entwickler und Du bist ein Junior Entwickler führten zu den besten Ergebnissen. Beide erzielten nahezu identische Resultate, wobei jeweils nur zwei Ausgaben -1 Punkt erhielt. Zeitlich waren alle Antworten ähnlich zwischen 34 und 37 Sekunden.

2. Fehleranalyse: Keiner der generierten Codes wies syntaktische oder semantische Fehler auf und der Linter schlug nie an. Allerdings korrigierten 50 Prozent der Versuche die HTML-Fehler nicht, besonders bei den Prompts Trainee und ohne spezifischen Prompt. Der Senior-Prompt lieferte die besten Erklärungen und Optimierungsvorschläge, während Junior- und Trainee-Prompts oft nur unvollständige Korrekturen lieferten oder lediglich beschrieben, was zu tun sei.

3. Komponentenfunktionalität: Die Komponente funktionierte am häufigsten mit dem Trainee-Prompt, während sie bei Verwendung des Senior-Prompts nur zu 60 Prozent funktionierte.

[24] Vgl. Spalte 9-12 Tabelle 2

Einstein Ergebnisse[25]:

1. Beste Ergebnisse: Der Senior-Prompt lieferte die besten Ergebnisse, während die Prompts Trainee und ohne spezifischen Prompt die schlechtesten Resultate erzielten. Die Zeit lag zwischen 5 und 8 Sekunden.
2. Fehleranalyse: Die Anzahl der semantischen und syntaktischen Fehler war eher gering, wobei der Trainee und Junior-Prompt die wenigsten Fehler aufwies.
3. Komponentenfunktionalität: Die Komponente funktionierte am häufigsten mit dem Junior-Prompt, während die anderen Prompts weniger konsistente Ergebnisse lieferten.

Der dritte Test bestand darin den Code so zu erneuern. Der Promt lautete: 'Erneure den Code so, dass er besser wartbar und lesbar wird. Achte dabei auf die Lesbarkeit und Wiederverwendbarkeit.' Die Kriterien zur Bewertung waren die Implementierung von Schutzmaßnahmen gegen Mehrfachklicks. Änderungen im HTML welche sich nicht durch lightning-input auszeichneten. Das Hinzufügen von Datenvalidierungen.

Codeium Ergebnisse[26]:

1. Beste Ergebnisse: Der Senior-Prompt lieferte die besten Ergebnisse, während der Prompt ohne spezifische Entwicklerrolle die schlechtesten Resultate erzielte. Zeitlich lagen alle 4 Promts bei 33 - 38 Sekunden, am Schnellsten war jedoch der Senior.
2. Fehleranalyse: Der Senior-Prompt hatte die wenigsten syntaktischen und semantischen Fehler (nur ein Fehler), während die anderen Prompts mehr Fehler aufwiesen. Der Linter sprang nie an.
3. Komponentenfunktionalität: Die Komponente funktionierte in 75 Prozent der Fälle, am häufigsten jedoch bei dem Trainee-Prompt. Der Senior-Prompt wählte oft komplexere Ansätze, die nicht immer funktionierten.

Einstein Ergebnisse[27]:

1. Beste Ergebnisse: Der unspezifische Promt lieferte die besten Resultate, während der Trainee und Senior-Prompt die schlechtesten Resultate erzielte. Jedoch war der Senior am schnellsten.

[25] Vgl. Spalte 9-12 Tabelle 5
[26] Vgl. Spalte 1-4 Tabelle 2
[27] Vgl. Spalte 1-4 Tabelle 5

2. Fehleranalyse: Die Anzahl der semantischen und syntaktischen Fehler war gering, wobei der Trainee und Senior-Prompt die wenigsten Fehler aufwies. Der Linter sprang nicht an.

3. Komponentenfunktionalität: Die Komponente funktionierte nur bei dem Prompt ohne spezifische Entwicklerrolle korrekt, da bei den anderen Prompts das JavaScript oft nicht richtig geändert wurde.

Der vorletzte Test umfasst, den vorhandenen Code zu analysieren und die Schritte zu erklären, was im Code passiert. Der verwendete Prompt lautete: Ànalysiere mir den vorhandenen Code und gib mir in Schritten an, was in diesem Code passiert.Ídie Kriterien zur Bewertung umfassten die Genauigkeit und Vollständigkeit der Analyse.

Codeium Ergebnisse[28]:

1. Beste Ergebnisse: Die Senior und Junior-Prompts führten zu den besten Ergebnissen. Beide Prompts lieferten ähnliche Resultate hinsichtlich der Geschwindigkeit und Genauigkeit.

2. Fehleranalyse: Die Zeit für die Analyse lag stets zwischen 29 und 39 Sekunden, wobei der Trainee-Prompt am langsamsten war. Ein neuer Chat-Kontext führte oft zu unterschiedlichen Antworten, was darauf hinweist, dass der vorherige Kontext nicht immer berücksichtigt wurde. Die Analysen beschränkten sich meist auf den connectedCallback und die Imports, während die eigentlichen Funktionen selten detailliert analysiert wurden. Junior- und Trainee-Prompts neigten dazu, die Funktionen genauer zu erklären.

Einstein Ergebnisse: Einstein kann auf Grund seiner Funktionalität den Code nicht beschreiben. Wenn die Aufgabe dennoch gestellt wird, versucht das Tool gelegentlich Kommentare hinzuzufügen was jedoch oft zu Missverständnissen führt. Daher Fällt diese Aufgabe (auch bei den anderen Kategorien) bei dieser KI weg.

Der letzte Test beinhaltet, eine Dokumentation des Codes zu erstellen. Der verwendete Prompt lautete: 'Erstelle mir eine Dokumentation des Codes.' Die Kriterien zur Bewertung umfassten die Auflistung aller Funktionen, Events und Variablen sowie die Erklärung des Zusammenspiels der Komponenten.

[28] Vgl. Spalte 13-14 Tabelle 2

Codeium Ergebnisse[29]:

1. Beste Ergebnisse: Der Senior-Prompt lieferte die besten Ausgaben mit einem deutlichen Abstand von durchschnittlich einem Punkt. Der Prompt ohne spezifische Entwicklerrolle erzielte die schlechtesten Ergebnisse, wobei die Zeit für alle Prompts zwischen 29 und 35 Sekunden lag.

2. Fehleranalyse: Die schnellsten Ergebnisse wurden ohne spezifischen Prompt erzielt. Der Unterschied zwischen den Prompts für Trainee und Senior betrug nur zwei Sekunden. Geöffnete Dokumente wurden immer priorisiert, während andere oft ignoriert wurden, selbst wenn sie geöffnet waren. Das Zusammenspiel der Komponenten wurde oft nicht berücksichtigt, es sei denn HTML wurde ausdrücklich erwähnt, was die Genauigkeit jedoch verringerte. Markdown-Dokumentationen hatten manchmal Fehler, was durch den Neustart des Kontexts behoben werden konnte.

Schlussfolgerung
Die durchgeführten Tests in den Kategorien Erstellen, Korrigieren, Refactoring, Analyse und Dokumentation der Lightning Web Components zeigen deutliche Unterschiede in der Leistung und Effektivität der beiden Tools.

Bei dem Erstellen von LWC zeigte sich, dass der Senior-Prompt bei Codeium die besten und schnellsten Ergebnisse lieferte mit häufigen Fehlern, die vom Linter erkannt wurden. Jedoch ist zu erwähnen, dass diese Fehler bei allen Ausgaben zu finden war. Der Junior-Prompt erzeugte stabilere und weniger fehlerhafte Komponenten, aber nicht immer vollständig funktionale. Bei Einstein führte der Trainee-Prompt zu den besten und vollständigsten Ergebnissen, während der Senior-Prompt ebenfalls gute, aber weniger konsistente Resultate lieferte. Ein klarer Einfluss des Promtings auf die Qualität der Ergebnisse war zu beobachten.

Im Korrigieren von bestehenden Fehlern in LWCs erwies sich der Junior-Prompt bei Codeium als am effektivsten indem er qualitativ hochwertige und schnelle Korrekturen lieferte. Doch sieht man, dass auch der Senior-Promt nur in der Funktionalität zurück steht. Auch gibt dieser Promt tiefere Einblicke in eventuelle Optimierungspotenziale. Der Trainee-Prompt erzielte häufiger funktionale Korrekturen, jedoch ohne immer die HTML-Fehler zu beheben. Bei Einstein gibt der Senior-Prompt die besten Ergebnisse und die schnellsten Korrekturen, während die anderen Prompts oft unvollständige oder fehlerhafte Korrekturen lieferten.

[29] Vgl. Spalte 15-16 Tabelle 2

Das Refactoring von Code zeigte, dass der Senior-Prompt bei Codeium qualitativ hochwertige und schnelle Antworten ausgab, jedoch komplexere Ansätze wählte die nicht immer funktionierten. Der Trainee-Prompt erzielte stabilere und funktionale Komponenten. Bei Einstein erwies sich der Prompt ohne spezifische Entwicklerrolle als am effektivsten, während der Senior-Prompt schnelle, aber oft unzureichende Änderungen lieferte. Einstein zeigte Schwierigkeiten, mehrere Programmiersprachen gleichzeitig zu bearbeiten und gibt ab und zu unzureichende Antworten aus. Ein Beispiel ist im folgenden Code zu sehen:

```
<template>
    !-- Hier wird die Liste der Mitarbeiter angezeigt -->
</template>
```

In der Analyse des Codes zeigte Codeium, dass der Senior-Prompt die schnellsten und qualitativ hochwertigsten Analysen lieferte, während Junior- und Trainee-Prompts genauere, aber langsamere Analysen boten. Einstein war in dieser Kategorie nicht in der Lage, zufriedenstellende Ergebnisse zu liefern, da es Schwierigkeiten hatte, den Code als Text auszugeben oder zu kommentieren. Dies ist auf die Funktionalitäten des Tools zurückzuführen.

Auch bei der Dokumentation des Codes lieferte der Senior-Prompt bei Codeium wieder die detailliertesten und besten Ergebnisse. Jedoch ist es ratsam, die Dokumente einzeln beschreiben zu lassen und anschließend das Zusammenspiel der Komponenten zu dokumentieren, da das Tools manchmal Probleme mit dem Kontext von mehreren Dateien hatte.

Abschließend ist festzuhalten, dass Codeium in allen Kategorien insgesamt besser abschnitt als Einstein, insbesondere bei der Verwendung des Senior-Prompts. Die Wahl des richtigen Prompts hatte einen erheblichen Einfluss auf die Qualität und Geschwindigkeit der Ergebnisse. Einstein zeigte deutliche Schwächen in der Ausgabe von mehreren Programmiersprachen und konnte die Anforderungen in diesen Bereichen nicht zufriedenstellend erfüllen.

Erstellen eines Apex Triggers

Die erste Aufgabe in dieser Disziplin wurde mit folgendem Promt durchgeführt: Erstelle einen Apex Trigger, der bei dem Erstellen eines neuen Mitarbeiters (Employee__c) automatisch ein Custom Field befüllt und eine Willkommens E-Mail an den Mitarbeiter schickt. Der Trigger soll ausgelöst werden, wenn ein neuer Employee-Datensatz erstellt wird. Dann soll ein Custom Field (Email_send__c) auf True gesetzt werden. Folgende Anforderungen soll der Trigger haben: Der Trigger sollte nur bei dem Einfügen eines neuen Mitarbeiters ausgelöst werden und der Trigger sollte sicherstellen, dass die E-Mail-Adresse vorhanden ist, bevor er versucht, das Feld zu befüllen und zu senden. Die Bewertungskriterien sind dabei wie folgt: Before/After Insert wurde verwendet, Trigger Funktioniert, Custom Field wird beschrieben, Fehlervalidierung vorhanden und er schlägt vor einen Controller für die Mail zu erstellen / macht es direkt.

Codeium Ergebnisse[30]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Resultate, während der Prompt ohne spezifische Entwicklerrolle die schlechtesten Ergebnisse lieferte. Die anderen Prompts lagen nur einen Punkt auseinander.

2. Fehleranalyse: Auch hier erzeugte er keine syntaktischen oder semantischen Fehler, während die anderen Prompts im Durchschnitt knapp 1,5 Fehler aufwiesen. Der Linter schlug stets bei DML CRUD-Berechtigungen an. 90 Prozent der semantischen Fehler betrafen einen Methodenaufruf (SendSafeMail), der nicht existierte.

3. Komponentenfunktionalität: Der Trigger funktionierte auch nur bei diesem Prompt immer korrekt, während er bei den anderen Prompts nur zu 60% und bei dem Trainee-Prompt zu 40% funktionierte.

Einstein Ergebnisse[31]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Ergebnisse und war der schnellste, während der Trainee-Prompt die schlechtesten Resultate lieferte.

2. Fehleranalyse: Es gab grundlegend keine syntaktischen oder semantischen Fehler, jedoch 3 bei Junior und dem unspezifischen-Promt. Der Linter schlug nie an.

3. Komponentenfunktionalität: Der Trigger funktionierte bei dem Trainee und Senior-Prompt immer, 40% bei dem Prompt ohne spezifische Entwicklerrolle.

[30] Vgl. Spalte 5-8 Tabelle Tabelle 3
[31] Vgl. Spalte 5-8 Tabelle 6

Die nächste Aufgabe bestand darin, den vorhandenen Trigger zu korrigieren und alle gefundenen Fehler aufzulisten. Der genutzte Promt ist gleich zu dem welcher bei der LWC genutzt wurde. Die Kriterien zur Bewertung umfassten die Hinzufügung von Fehlerbehandlung, die Entfernung von DML in Schleifen, die Hinzufügung von Datenvalidierungen und je die Korrektur von einem Syntaxfehler.

Codeium Ergebnisse[32]:

1. Beste Ergebnisse: Die Prompts für Junior und Senior erzielten die besten Ausgaben, gefolgt von Trainee und dem unspezifischen Promt.
2. Fehleranalyse: Der Trainee-Prompt war am schnellsten, jedoch nur etwa eine Sekunde schneller als der Senior. Es gab keine syntaktischen oder semantischen Fehler, und der Linter schlug erneut bei DML und Vermeidung von Logik im Trigger an.
3. Komponentenfunktionalität: Der Trigger funktionierte bei allen Prompts.

Einstein Ergebnisse[33]:

1. Beste Ergebnisse: Der Junior-Prompt erzielte die besten Ergebnisse, gefolgt von den Prompts für Trainee und Senior. Der Prompt ohne spezifische Entwicklerrolle schnitt am schlechtesten ab. Der Senior-Prompt war am schnellsten, mit Zeiten zwischen 2 und 6 Sekunden.
2. Fehleranalyse: Der Trainee-Prompt wies keine syntaktischen oder semantischen Fehler auf, während Senior und Junior im Durchschnitt 2,2 Fehler hatten.
3. Komponentenfunktionalität: Der Trigger funktionierte bei dem Trainee-Prompt immer, bei dem Senior nur zu 40% und ohne spezifische Entwicklerrolle nie.

Die Aufgabe beinhaltet den bestehenden Trigger so anzupassen, dass Wartbarkeit und Lesbarkeit verbessert werden. Der Promt ist der LWC zu entnehmen. Die Kriterien zur Bewertung umfassten die Trennung der Logik vom Trigger, die Datenvalidierung im Trigger und die Vermeidung von E-Mail-Sendungen in Schleifen.

Codeium Ergebnisse[34]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Resultate, gefolgt von den Prompts für Trainee und Junior. Der Prompt ohne spezifische Entwicklerrolle war am schnellsten, aber die Unterschiede zu den anderen Prompts lagen nur bei etwa einer Sekunde (33-35 Sekunden).

[32] Vgl. Spalte 9-12 Tabelle 3
[33] Vgl. Spalte 9-12 Tabelle 6
[34] Vgl. Spalte 1-4 Tabelle 3

2. Fehleranalyse: Die Semantik und Syntax war bei den Prompts für Senior und Trainee am besten. Der Linter schlug bei keinem Prompt an.

3. Komponentenfunktionalität: Bei allen Prompts funktionierte der Trigger einmal nicht.

Einstein Ergebnisse[35]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Ergebnisse, gefolgt von den Prompts für Trainee und ohne spezifische Entwicklerrolle. Der Junior-Prompt erzielte keine Punkte. Der Senior-Prompt war am schnellsten, jedoch nur 0,2 Sekunden schneller als der Trainee-Prompt (alle zwischen 6 und 7 Sekunden).

2. Fehleranalyse: Die Semantik und Syntax war bei dem Senior-Prompt am besten, gefolgt vom Trainee-Prompt. Der Junior-Prompt hatte nur Fehler. Der Linter schlug bei keinem Prompt an.

3. Komponentenfunktionalität: Der Senior-Promt hat immer funktioniert, bei den anderen nur zu 60%. Der Junior hat nie funktioniert.

Die vorletzte Aufgabe bestand wieder darin, den vorhandenen Code zu analysieren und die Schritte zu erklären. Die Kriterien zur Bewertung umfassten das Erkennen von DML in Schleifen, die falsche Verwendung von before statt after und die allgemeine Erkennung des Codes.

Codeium Ergebnisse[36]:

1. Beste Ergebnisse: Alle Prompts erzielten gleich gute Ausgaben. Der Senior-Prompt war am schnellsten (30-35 Sekunden).

2. Fehleranalyse: Alle Prompts beschrieben die Fehler, listeten sie jedoch nicht auf.

Die letzte Aufgabe, die Dokumentation des Codes, hatte folgende Kriterien zur Bewertung: die Auflistung aller Funktionen, Events und Variablen sowie die Erklärung der Komponente.

Codeium Ergebnisse[37]:

1. Beste Ergebnisse: Der Junior-Prompt erzielte die besten Resultate, gefolgt von den Prompts für Senior und ohne spezifische Entwicklerrolle. Der Trainee-Prompt schnitt am schlechtesten ab. Der Junior-Prompt war am schnellsten, der Senior-Prompt am langsamsten (39-50 Sekunden).

[35] Vgl. Spalte 1-4 Tabelle Tabelle 6
[36] Vgl. Spalte 13-14 Tabelle 3
[37] Vgl. Spalte 15-16 Tabelle 3

2. Fehleranalyse: Junior und Trainee-Prompts listeten nicht alle Funktionen auf und der Prompt ohne spezifische Entwicklerrolle hielt teilweise Hilfsfunktionen für Beispielmethoden. Der Senior-Prompt war am ausführlichsten und begründete oft, warum die Implementierung gut war.

Schlussfolgerung
Bei dem Erstellen eines Apex-Triggers zeigte Codeium, dass der Senior-Prompt die besten und schnellsten Ergebnisse lieferte. Der Trigger funktionierte bei diesem Prompt immer korrekt und es gab keine syntaktischen oder semantischen Fehler. Bei den anderen Prompts funktionierten die Trigger nur zu 60%, wobei häufig Methodenaufrufe verwendet wurden, die nicht existierten. Einstein erzielte mit dem Trainee-Prompt die besten Resultate, während die anderen Prompts ähnlich abschnitten. Der Senior-Prompt war der schnellste und alle erstellten Trigger funktionierten korrekt. Insgesamt zeigte sich, dass der Senior-Prompt bei Codeium und der Trainee-Prompt bei Einstein die besten Resultate lieferten. Somit kann auch hier die These bestätigt werden, dass das Promting einen Einfluss auf die Ergebnisse hat.

Bei dem korrigieren des fehlerhaften Triggers erzielte der Trainee-Prompt bei Codeium die besten Ausgaben, dicht gefolgt vom Senior-Prompt. Die Trigger funktionierten bei beiden Prompts immer korrekt, während der Prompt ohne spezifische Entwicklerrolle nur zu 40% funktionierte. Der Trainee-Prompt beschrieb die Fehler detailliert und gab genaue Anleitungen zur Korrektur. Bei Einstein erzielte der Junior-Prompt die besten Korrekturen, während der Senior-Prompt schneller war, aber häufiger Fehler machte. Die meisten Fehler wurden erkannt, aber nicht immer behoben.

Das Refactoring eines Apex-Triggers zeigte, dass der Senior-Prompt bei Codeium qualitativ hochwertige Refaktorisierungen lieferte, während der Trainee-Prompt die stabilsten funktionalen Resultate erzielte. Der Senior-Prompt war der schnellste, aber auch der fehleranfälligste. Bei Einstein lieferte der Senior-Prompt konsistente und funktionale Refaktorisierungen, während der Junior-Prompt die Aufgabe nicht zufriedenstellend erfüllte.

Bei der Analyse und Dokumentation des Apex-Triggers erzielten alle Prompts bei Codeium ähnliche Ergebnisse, wobei der Senior-Prompt am schnellsten war. Doch war der Senior-Prompt wieder qualitativ hochwertiger. Auch bei dieser Disziplin zeigte sich wieder, dass Codeium Einstein in den Ausgaben überlegen ist. Trozdessen ist festzuhalten, dass Einstein bei dieser Kategorie besser abschnitt, als bei der Erstellung von LWCs.

Testing
Der verwendete Prompt lautete: 'Erstelle mir einen Test für folgenden Apex-Trigger.' Die Kriterien zur Bewertung umfassten die Erstellung von Tests für 'Inactive' und 'Active' Zustände, sowie mindestens ein Edgecase und einen Error-Test.

Codeium Ergebnisse [38]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Resultate, gefolgt von den drei anderen. Alle Promts zwaren ähnlich schnell (24-27 Sekunden).
2. Fehleranalyse: Alle Promts wiesen keine syntaktischen und semantischen Fehler auf. Der Linter schlug bei keinem Prompt an.
3. Komponentenfunktionalität: Die Tests funktionierten bei dem Senior-Prompt immer korrekt während sie bei den anderen Prompts nur zu 80% funktionierten.

Einstein Ergebnisse [39]:

1. Beste Ergebnisse: Der Trainee-Prompt erzielte die besten Ergebnisse, alle anderen waren ähnlich. Zeitlich lagen alle zwischen 5-6 Sekunden.
2. Fehleranalyse: Alle Prompts wiesen keine syntaktischen oder semantischen Fehler auf.
3. Komponentenfunktionalität: Alle Tests funktionieren.

Die Kriterien für die nächste Aufgabe umfassten: die Korrektur von Syntaxfehlern wie '' in Apex, doppelte Insert-Befehle, falsche Vergleichsoperatoren, DML in Schleifen und fehlende Klammern.

Codeium Ergebnisse [40]:

1. Beste Ergebnisse: Der Trainee-Prompt erzielte die besten Ausgaben, knapp gefolgt vom Senior-Prompt. Der Prompt ohne spezifische Entwicklerrolle lieferte die schlechtesten Ergebnisse. Der Senior-Prompt war am schnellsten (30-35 Sekunden).
2. Fehleranalyse: Der Trainee-Prompt wies die wenigsten syntaktischen und semantischen Fehler auf. Der Linter schlug bei keinem Prompt an.

[38] Vgl. Spalte 4-8 Tabelle 4
[39] Vgl. Spalte 4-8 Tabelle 7
[40] Vgl. Spalte 9-12 Tabelle 4

3. Komponentenfunktionalität: Die Tests funktionierten bei dem Senior- und Trainee-Prompt immer korrekt, während sie bei dem Prompt ohne spezifische Entwicklerrolle nur zu 40% und bei dem Junior-Prompt zu 80% funktionierten.

Einstein Ergebnisse [41]:

1. Beste Ergebnisse: Der Senior- und der Prompt ohne spezifische Entwicklerrolle erzielten die besten Ausgaben, während der Trainee-Prompt die schlechtesten Resultate lieferte.

2. Fehleranalyse: Der Senior-Prompt funktionierte immer korrekt, während die anderen Prompts nur zu 60 schnellsten, jedoch nur 0,2 Sekunden schneller als der Senior-Prompt (alle zwischen 6 und 7 Sekunden). Der Linter schlug bei keinem Prompt an.

3. Komponentenfunktionalität: Der Senior-Prompt lieferte konsistent funktionale und qualitativ hochwertige Korrekturen, obwohl der Handler-Code manchmal auch unter dem Trigger abgelegt wurde. Der Junior-Prompt erfüllte die Aufgabe nicht zufriedenstellend.

Die Kriterien zur Refaktorisierungen umfassten die Prüfung von alten Werten, das Hinzufügen einer Setup-Methode und das Entfernen von wiederholtem Code.

Codeium Ergebnisse [42]:

1. Beste Ergebnisse: Der Senior-Prompt erzielte die besten Ergebnisse, gefolgt vom Trainee-Prompt. Mit einem Punkt weniger sind die anderen Promts dahinter. Der Senior und unspezifische -Prompt waren am schnellsten (30-35 Sekunden).

2. Fehleranalyse: Der Senior-Prompt wies die meisten syntaktischen und semantischen Fehler auf. Der Linter schlug bei keinem Prompt an.

3. Komponentenfunktionalität: Die Tests funktionierten bei dem Trainee-Prompt am häufigsten, während sie bei dem Prompt ohne spezifische Entwicklerrolle nur zu 20% und bei den anderen Prompts zu 60% funktionierten.

Einstein Ergebnisse [43]:

1. Beste Ergebnisse: Der Senior und unspezifische-Prompt erzielte die besten Resultate, während der Trainee-Prompt die schlechtesten Resultate lieferte. Die Zeit lag zwischen 8 und 10 Sekunden.

[41] Vgl. Spalte 9-12 Tabelle 7
[42] Vgl. Spalte 1-4 Tabelle 4
[43] Vgl. Spalte 1-4 Tabelle 7

2. Fehleranalyse: Die Anzahl der semantischen und syntaktischen Fehler war eher gering, wobei der Senior die wenigsten Fehler aufwies. Der Linter schlug nicht an.

3. Komponentenfunktionalität: Die Komponente funktionierte am häufigsten mit dem Senior-Prompt, während die anderen Prompts weniger konsistente Ergebnisse lieferten.

Die Kriterien für die vorletzte Aufgabe beinhalten die Erkennung von ineffizienten Praktiken und die richtige Zuordnung von Testfällen.

Codeium Ergebnisse [44]:

1. Beste Ergebnisse: Der Senio-Prompt erzielten die besten Ergebnisse. Zu erwähnen ist jedoch das alle 4 ähnliche Ergebnisse erzielt haben

2. Fehleranalyse: Der unspezifische-Prompt war am schnellsten. Der Linter schlug bei keinem Prompt an.

Die Kriterien der letzten Aufgabe umfassten die Erklärung des Test Setups und die Auflistung und Erklärung aller Tests mit ihrer Funktionalität.

Codeium Ergebnisse [45]:

1. Beste Ergebnisse: Alle vier erzielten die gleichen Resultate.

2. Fehleranalyse: Es gab keine Syntaktischen oder Semantischen Fehler. Der Linter schlug bei keinem Prompt an.

Schlussfolgerung
Bei dem Erstellen eines Tests für den Apex-Trigger zeigte Codeium, dass der Senior-Prompt die besten und schnellsten Ausgaben lieferte. Die Tests funktionierten bei diesem Prompt immer korrekt und es gab keine syntaktischen oder semantischen Fehler. Einstein erzielte mit dem Trainee-Prompt die besten Resultate, wobei alle erstellten Tests funktional und fehlerfrei waren.

Bei dem Korrigieren eines fehlerhaften Tests erzielte der Trainee-Prompt die besten Korrekturen, während der Senior-Prompt schneller war, aber weniger detaillierte Antworten gab. Die Tests funktionierten bei beiden Prompts immer korrekt. Bei Einstein erzielte der Junior-Prompt die besten Korrekturen, während der Senior-Prompt schneller, aber fehleranfälliger war.

[44] Vgl. Spalte 13-14 Tabelle 4
[45] Vgl. Spalte 15-16 Tabelle 4

Das Refactoring eines Tests zeigte, dass der Senior-Prompt bei Codeium qualitativ hochwertige Refaktorisierungen lieferte, während der Trainee-Prompt die stabilsten funktionalen Resultate erzielte. Der Senior-Prompt war der schnellste, aber auch der Fehleranfälligste. Bei Einstein lieferten der Senior-Prompt und der Prompt ohne spezifische Entwicklerrolle die besten Ergebnisse, wobei der Senior-Prompt die konsistentesten und funktionalsten Refaktorisierungen erzielte.

Bei der Analyse des Testcodes erzielten alle Prompts bei Codeium ähnliche Ergebnisse, wobei der Senior-Prompt am schnellsten war. Einstein konnte den Code nicht zufriedenstellend analysieren, da es Schwierigkeiten hatte, die Schritte im Code korrekt zu beschreiben.

Bei der Dokumentation des Testcodes lieferten alle Prompts bei Codeium qualitativ hochwertige Dokumentationen, wobei der Senior-Prompt am schnellsten war. Einstein konnte den Code nicht zufriedenstellend dokumentieren.

4.4 Wie kann KI Entwickler unterstützen?

In der vorliegenden Arbeit stand das Ziel im Vordergund zu Untersuchen wie Künstlicher Intelligenz zur Optimierung von Entwicklungsprozessen in Salesforce-Umgebungen eingesetzt werden kann. Hierfür wurde eine Umfrage unter Entwicklern sowie ein praktischer Test mit den KI-Tools Codeium und Einstein for Developers durchgeführt um zu evaluieren, wie diese Tools die Effizienz und Qualität der Entwicklungsprozesse verbessern können. Dieses abschließende Kapitel widmet sich der Beantwortung dieses Ziels und der zentralen Forschungsfragen:

1. In welchen Bereichen der Salesforce-Entwicklungsprozesse können Codeium und Einstein for Developers eine Optimierung bewirken?
2. Wie unterscheiden sich die beiden Tools in ihrer Herangehensweise und Effektivität?
3. Welche messbaren Verbesserungen können durch den Einsatz dieser KI-Tools erzielt werden?

Basierend auf der ersten Forschungsfrage gibt es vier mögliche Szenarien wie man diese beantworten kann:

- Beide Tools sind ineffektiv und sollten nicht eingesetzt werden
- Nur eines der beiden Tools ist nützlich

- Beide Tools sind in verschiedenen Bereichen gut und ergänzen sich
- Beide Tools sind effektiv und können unabhängig voneinander eingesetzt werden

Im Folgenden werden die Ergebnisse und Schlussfolgerungen zusammengefasst, um diese Fragen zu beantworten und Empfehlungen für die Praxis abzuleiten.

Im praktischen Test zeigte sich das Codeium in vielen Bereichen überlegen ist. Codeium bietet Inline-Autocomplete für über 70 Programmiersprachen und ist in mehr als 40 IDEs verfügbar. Es verfügt über ein Chatfenster mit Kontext, Chatverlauf und Einstellungen zum Promting, was eine flexible und benutzerfreundliche Interaktion ermöglicht. Der generierte Code kann direkt eingefügt werden, was die Effizienz erhöht. Codeium zeigt außerdem eine hohe Kontextsensitivität und bietet eine projektspezifische, semantische Suche. Im Chat gibt es zwei Versionen: eine normale, schnelle Version (etw a 7-10 Sekunden) und eine GPT-4-Version (etwa 20-50 Sekunden), die nur in der Teams-Version verfügbar ist. Codeium basiert auf dem GPT-4-Modell von OpenArtificial Intelligence (AI).

Einstein ist nur für IntelliJ (über ein weiteres Plugin namens Illuminated Cloud) und VS Code verfügbar, was die Nutzung komplizierter und unflexibler macht. Die Funktionen umfassen Autocomplete für LWCs, Apex und verfügen über ein einfaches Chatfenster mit den Optionen Senden und Clear. Es basiert auf dem Gen Model 2.5 von Salesforce und ist speziell auf Apex und Salesforce trainiert.

Beide Tools sind als Erweiterungen für VS Code und IntelliJ IDEs verfügbar und bieten Inline-Autocomplete sowie ein Chatfenster, das die Interaktion erleichtert. Die Ergebnisse der Tests zeigen deutliche Unterschiede zwischen den beiden untersuchten Tools. Unabhängig vom Promting zeigte Codeium bei der Arbeit mit LWCs, dass oft veraltete HTML-Elemente verwendet wurden, die von Apex-PMD erkannt und angezeigt werden. Auch wenn ein Controller erstellt wurde, zeigte der Linter grundsätzlich CRUD-Berechtigungsprobleme an. [46]. Die Kontextsensitivität war in den Ausgaben deutlich erkennbar, da HTML und JavaScript gut zusammenspielten. Bei dem Korrigieren und Refaktorisieren wurde diese Kontextsensitivität manchmal nicht genutzt beziehungsweise der Kontext nicht erkannt.

Einstein hatte Schwierigkeiten mit mehrsprachigem Code (HTML und JavaScript) und gab oft nur eine der beiden Sprachen aus. Bei der Korrektur und Refaktorisierung zeigte sich, dass oft nicht alle Fehler behoben wurden. Die Analyse und Dokumentation war mit Einstein problematisch, da das Tool nicht über die Fähigkeit verfügt solche Antworten zu geben.

[46] Diese Fehlermeldung weist in Apex darauf hin, dass bestimmte Sicherheitsanforderungen geprüft und validiert werden müssen.

Bei der Arbeit mit Triggern war die Semantik bei Codeium oft fehlerhaft da Funktionsaufrufe wie SendSafeMail genutzt wurden, die nicht existierten. Diese Methode ist Recherchen zufolge nicht und war auch nicht in Apex integriert. Daher kann dieser Fehler auf die nicht vorhandene Erfahrung mit Apex zurückgeführt werden. Apex PMD schlug auch hier wieder bei CRUD-Berechtigungen relativ oft an. Bei Einstein waren die Ergebnisse deutlich besser was auf das spezifische Training auf Apex zurückzuführen ist. Falls eine Auslagerung von Logik vorhanden war, war diese qualitativ okay, obwohl der Code manchmal im Trigger belassen wurde, was wiederum einen Verstoß gegen die Best Practices darstellt. Die Untersuchung bezüglich der Tests ergab, dass die Ergebnisse bei Codeium gut waren, jedoch wurden keine Edgecases oder Error-Tests erstellt. Einstein zeigte ähnliche Ergebnisse.

Das Promting machte bei Codeium einen großen Unterschied. Bei dem Senior-Prompt wurden beispielsweise oft Controller erstellt oder vorgeschlagen, während Bei dem Trainee- oder Junior-Prompt dies nicht der Fall war. Der Junior-Prompt bereitete manchmal nur Funktionen vor (Paging). Der Senior-Prompt gab oft Optimierungsvorschläge und kompliziertere Antworten auf Kosten der Funktionalität. Junior und Trainee erklärten mehr und detaillierter. Unspezifische Prompts fixierten selten etwas. Bei der Dokumentation und Analyse waren die Antworten bei spezifischen Prompts genauer.

Bei Einstein hatte das Promting nicht so einen großen Einfluss wie bei Codeium. Die Antworten waren oft ähnlich, unabhängig vom Prompt. Der Junior-Prompt erstellte zum Beispiel oft keine vollständigen Trigger, sondern nur Methoden. Dies war auch bei den anderen drei Prompts zu beobachten. Der spezifische Kontext half selten, und die Ergebnisse waren oft gleich.

Wenn der Kontext bei Codeium nicht funktioniert, sollte er neu zugeordnet, in die Seite geklickt oder die Erweiterung neu gestartet werden. Die Markdown-Formatierung hat wies gelegentlich Fehler auf, welche mit einem Neustart behoben werden konnten. Ein neuer Chat führt zu neuen Antworten, während der gleiche Chat ähnliche Antworten gibt. Es hilft, die Dateien einzeln zu analysieren und zu dokumentieren, da dies genauere Antworten liefert. GPT-4 ist langsamer als Einstein (20-50 Sekunden), die normale Version gibt jedoch Antworten schneller aus. Syntax und Semantik waren im Durchschnitt sehr gut (2,5 von 3 Punkten), wobei der Prompt einen signifikanten Einfluss auf die Qualität hatte. Die durchschnittliche Funktionalität der Komponenten lag bei 0,82.

Einstein gibt nur eine Sprache aus, was nachteilig für den Kontext und Zusammenhänge ist. Es gibt keine Vorschläge oder umfassende Erklärungen, was die Nutzung erschwert. Manchmal werden falsche Ausgaben generiert. Dokumentation und Analyse sind nur über Kommentare möglich, was unzureichend ist. Mehrsprachiger Kontext verwirrt das Modell eher.

Zwar ist das Modell sehr schnell (meist unter 10 Sekunden), aber das Promting macht selten einen Unterschied. Syntax und Semantik bei Apex waren gut (2,21 von 3). Die durchschnittliche Funktionalität lag bei 0,51.

Schwere Fehler wie Anführungszeichen welche in Apex verboten sind, werden von beiden Tools eher selten erkannt, was durch die zusätzliche Nutzung von APEX PMD behoben werden kann. Funktionen, deren Inhalt nicht bekannt ist, werden oft nur geschätzt. Wichtige Erwähnungen sind, dass Einstein, anders als Codeium, noch in der Beta-Phase ist. Einstein ist nur auf Apex und Salesforce trainiert, während Codeium über 70 Programmiersprachen unterstützt. Trotz dieser Einschränkungen ist Codeium in vielen Aspekten besser. Dies zeigt sich in den Ergebnissen, die in den Anhängen dokumentiert sind.

Allgemein kann festgehalten werden, dass Codeium in den meisten Kategorien überlegen ist. Insbesondere wenn ein Senior-Prompt benutzt wurde, wurden schnelle und qualitativ hochwertige Ergebnisse geliefert. Der Trainee-Prompt erzielte in einigen Fällen die stabilsten funktionalen Ergebnisse. Einstein konnte in den Kategorien Erstellen und Korrigieren von Tests mithalten, zeigte jedoch Schwächen in der Analyse und Dokumentation des Codes, da diese nicht in seinen Funktionsbereich fallen. Der richtige Prompt hat einen signifikanten Einfluss auf die Qualität und Geschwindigkeit der Ergebnisse , wobei der Senior-Prompt bei Codeium konsistent die besten objektiven Resultate lieferte. Damit kann die zweite Forschungsfrage beantwortet werden.

Die Umfrage bestätigt die praktischen Testergebnisse in vielerlei Hinsicht. Entwickler haben eine positive Einstellung gegenüber der Nutzung von KI in ihren Arbeitsprozessen und viele nutzen diese regelmäßig oder gelegentlich. Dies spiegelt die breite Akzeptanz und die bereits erkannten Vorteile der KI wider. Eine kleinere Gruppe von Entwicklern könnte durch die positiven Ergebnisse überzeugt werden, die Vorteile von KI-Tools in ihren Prozessen zu erkennen und zu nutzen.

Die häufigsten Probleme in den Entwicklungsprozessen – die Einhaltung von Standards, Fehlerbehebung (Bugfixing), das Erstellen von Tests und die Dokumentation – sind genau die Bereiche, in denen Unterstützung gewünscht wird und in denen die getesteten KI-Tools hilfreich sein könnten. Es zeigte sich, dass Codeium besonders beliebt ist und regelmäßig genutzt wird, was die Benutzerfreundlichkeit und Nützlichkeit unterstreicht. Insbesondere für die Einhaltung von Standards und Best Practices sowie das Erstellen von Tests und Dokumentationen wird das Tool genutzt. Viele Nutzer bewerten die Qualität der Antworten als 'Gut' was darauf hindeutet, dass Codeium in vielen Fällen hilfreiche und qualitativ hochwertige Antworten liefert, auch wenn es bei einigen spezifischen Aufgaben gelegentlich Unsicherheiten gibt. Dies wird durch den praktischen Test bestätigt.

Diese positiven Bewertungen, insbesondere bei dem Erstellen von Tests und Dokumentationen, bestätigen außerdem das Potenzial der KI hohe Standards zu erfüllen.

Dies kann auch durch die Anzahl der syntaktischen/semantischen Fehler bestätigt werden. Umfrageteilnehmer welche Einstein for Developers bereits nutzen haben in diesen Bereichen leider keine verwertbaren Antworten gegeben, da Anzahl nur zwei betrug. Trotzdem hat sich auch bei dem zweiten Tool gezeigt, dass der Einsatz durchaus Sinnvoll sein kann.

Für die Beantwortung der ersten Forschungsfrage lässt sich sagen, dass die Integration von KI-Tools wie Codeium und Einstein for Developers ein großes Potenzial bietet. Codeium zeigte in den meisten Bereichen eine überlegene Leistung, insbesondere bei der Verwendung von Senior-Prompts, die schnelle und qualitativ hochwertige Ergebnisse lieferten. Der richtige Prompt spielt also eine entscheidende Rolle bei der Qualität und Geschwindigkeit der Ergebnisse, wobei spezifische Prompts konsistent bessere Resultate lieferten. Einstein konnte in einigen Bereichen mithalten, insbesondere bei dem Erstellen und Korrigieren von Tests, zeigte jedoch große Schwächen in der Kategorie LWCs und bei den Aufgaben Dokumentation und Analyse. Einstein kann für die schnelle Erstellung von einfachen Tests und Triggern, sowie die Fehlerbehebung in diesen Bereichen genutzt werden. Trotz dessen haben bei der Suche nach Fehlern beide Tools ihre Defizite. Vor allem bei Apex werden kleine Syntaxfehler wie die typischen Anführungszeichen oft von beiden Tools nicht erkannt. Trotzdem muss festgehalten werden, dass Standards im großen und ganzen von beiden eingehalten werden. Codeium sollte für die Arbeit mit LWCs, sowie die Analyse, Dokumentation und das Refactoring von Code bevorzugt werden.

Die Beantwortung der letzten Forschungsfrage steht stark im Zusammenhang mit dem Ziel der Arbeit. Praktisch kann diese Frage durchaus aus den letzten Abschnitten beantwortet werden. Messbar jedoch ist es nötig eine andere Herangehensweise zu wählen. Dafür kann die insgesamte Zeit, die ein Entwickler für die Aufgaben braucht im Vergleich mit den beiden Tools ein guter Indikator sein. Die praktische Überlegenheit von Codeium spielt hierbei aber keine Rolle. Um die nötige Zeit zu messen wurde im Rahmen dieses Kapitels eine Planning Poker Runde durchgeführt.

Planning Poker ist eine Disziplin von SCRUM und wird verwednet um Aufgaben gemeinschaftlich im Team zu schätzen. Dabei wird die Zeit, die ein Entwickler für eine Aufgabe benötigt geschätzt. Jeder Entwickler bekommt einen kurzen Zeitraum um die Aufgabe zu lesen und zu verstehen. Danach wird die Zeit anhand vorgefertiger Karten geschätzt. Die Schätzungen werden dann zusammengezählt und überprüft bei welcher Zeit der Median liegt. Haben einige der Entwickler weit abweichende Meinungen wird kurz über die Gründe diskutiert und die Schätzung wiederholt. Sind alle Schätzungen ähnlich wird diese Zeit als Schätzung für die Aufgabe genommen.

Diese Art der Schätzung bietet für die vorhanden Aufgaben eine gute Möglichkeit die Zeit zu messen und gleichzeitig einen praktischen Einblick zu bekommen.

Die auszuwählenden Zeiten werden nach der Fibunachhi-Reihe gewählt. Diese Reihe wird oft bei dieser Disziplin verwendet um Aufwände zu schätzen. Die Zeiten welche zur Auswahl stehen sind 0, 0.5,1, 2, 3, 5, 8, 13, 20, 40, 100 (in Minuten). Die Zeit, welche Bei dem Test gemessen wurde wird dann an die nächst höhere Zahl in der verwendeten Skala aufgerundet. Damit wird die Schätzung etwas genauer. Teilgenommen haben insgesamt 7 Junior-Entwickler. Die Dokumentation ist in Tabelle 8 im Anhang zu finden.

Für den Vergleich zu den beiden KI-Tools wird die Zeit der einzelnen Aufgaben (also die Generierung der Antwort), die Zeit für das Überarbeiten der Antwort und die Zeit für das Erstellen des Prompts genutzt und geschätzt. Die erste Zeit wurde durch den praktischen Test schon messbar Dokumentiert. (Siehe Spalte 2 der Tabelle 8) Bei den anderen beiden Werten muss eine Schätzungen aus eigener Erfahrung genommen werden. Diese Zeiten zu untersuchen ist fast unmöglich, da hier eine vielzahl von Entwicklern oder Probanten nötig ist. Diese müssten dann die Aufgaben mit den Tools bearbeiten und die Zeit messen. Dies ist aber einerseits im Rahmen dieser Arbeit nicht möglich und anderseits generell schwer zu messen.

Das Erstellen und Verfassen eines Promts hängt von der Erfahrung, dem Tool, der Qualität und Komplexität der zugrundeliegenden Aufgabe (z.B. das Ticket) und vielen weiteren Faktoren (wie die Tippgeschwindigkeit) ab. Festgehalten werden kann aber, dass es sich bei den zugrundeliegenden Aufgaben um leichte Aufgaben handelt, welche nicht viel Aufwand benötigen. Beide Tools haben einfache Chatfenster, in dem der Promt eingegeben werden kann. Die Erstellung der Promts wird auf ungefähr 1-4 Minuten festgelegt. Da der Promt zu Generierung die meisten Anforderungen hat, wird hier eine generelle Zeit von 4 Minuten geschätzt. Die Zeit für die Fehlerbehebung und Dokumentation sind leicht und schnell verfasst. Daher wird hier eine Zeit von 1 Minute geschätzt. Die Zeit für die Refaktorisierung und Analyse ist etwas komplexer und benötigt mehr Zeit, da genau geprüft werden muss auf was Analysiert und wie genau Refaktorisiert werden muss. Daher wird hier eine Zeit von 3 Minuten für die Analyse und 2 Minuten für die Refaktorisierung geschätzt.

Die Zeit für die Überarbeitung ist schwieriger zu schätzen. Dies kann aber Anhand der Qualität der Ausgaben beider Tools festgemacht werden. Bei Codeium sollte für die Überarbeitung generell weniger Zeit benötigt werden, da die Antworten deutlich ausführlicher und qualitativ Hochwertiger sind. Daher kann man hier eine Zeit von 3-5 Minuten festlegen. Das erstellen von LWCs sollte mehr und ausführlicher Analysiert werden, da es hier um die Integration von HTML, JavaScript und gegebenenfalls einem Apex Controller geht. Daher wird hier eine Zeit von 6 Minuten angelegt. Die Erstellung eines Triggers und Tests ist weniger komplex im Bezug auf die Schwierigkeit der zugrundeliegenden Aufgaben, daher schätzt der Autor eine Zeit von 5 Minuten für beide Disziplinen.

Eine Dokumentation, Analyse und Fehlerbehebung erfordert bei einfachen Aufgaben keine tiefen Überprüfungen, wenn die Antwort qualitativ hochwertig ist und die Antwort nach eigener Erfahrung gut Einschätzen kann. Bestärkt wird dies durch die Erklärungen von Codeium. Daher wird hier eine Zeit von 3 Minuten festgehalten. Für die Fehlerbehebung wird außerdem für LWCs aufgrund der Komplexität 1 Minute extra angesetzt.

Anders als bei Codeium sind die Antworten von Einstein nicht so ausführlich und qualitativ hochwertig. Daher erfordert die Überarbeitung mehr Zeit. Da das Tool bei der Refaktorisierung und Fehlerbehebung von Triggern und Tests relativ gut abgeschnitten hat, wird auch hier die gleiche Zeit wie bei Codeium angesetzt. Dies gilt auch für das Erstellen, obwohl der Autor aufgrund der Qualität eine extra Minute eingebaut hat. Dies kann jedoch nicht für LWCs gelten. Da die KI meistens nur eine der beiden Sprachen ausgegangen kann, sollte bei der Refaktorisierungen das doppelte und Fehlerbehebung das 2,5-fache angesetzt werden. Insgesamt wurde außerdem aufgrund der Schätzungen eine Pufferzeit von 2 Minuten eingebaut.

Wie in der Tabelle 8 zu sehen ist, ist die Zeit für die Überarbeitung bei beiden Tools geringer. Die Entwickler haben bei jeder Aufgabe die gesamte Zeit höher eingeschätzt, als das was die jeweilige KI im Test mit der zusätzlichen Zeit benötigt hat. Vor allem die Dokumentation sticht in der Zeit heraus. Einige Aufgaben wie die Fehlerbehebung und Refaktorisierung sind von den Zeiten relativ sehr nah beieinander, wo sich Schlussfolgern lässt, dass eventuell hier die KI keine ausschaltkräftegen Antworten beziehungsweise Hilfestellungen gibt. Damit lässt sich letztendlich die Forschungsfrage beantworten. Die KI-Tools können die Effizienz und Qualität verbessern und damit die der Entwicklungsprozesse optimieren. Dies hat einerseits die eben erfolgte theoretische Schätzung und andererseits die praktischen Tests gezeigt.

5 Strategien zur Risikominderung bei KI-Tools

IT-Sicherheit bezieht sich auf den Schutz von Informationssystemen vor Diebstahl, Beschädigung oder Störung der Hardware, Software und der darauf gespeicherten Daten. Sie umfasst Maßnahmen und Kontrollen, die darauf abzielen, die Vertraulichkeit, Integrität und Verfügbarkeit von Informationen zu gewährleisten (Schutzziele). Vertraulichkeit bedeutet, dass Informationen nur autorisierten Personen zugänglich sind. Integrität stellt sicher, dass Informationen unverändert und vollständig bleiben, und Verfügbarkeit bedeutet, dass Informationen und Systeme bei Bedarf zugänglich sind.

Auch im Rahmen dieser Arbeit sollte dieses Thema betrachtet werden. Vor allem Entwicklungsprozesse, welchen nicht ausreichend abgesichert sind, können dazu führen, dass Schwachstellen in der Software entstehen, die von Angreifern ausgenutzt werden können. Dies kann schwerwiegende Konsequenzen haben, wie Datenverlust, finanzielle Schäden und Reputationsverlust. Daher ist es wichtig, Sicherheitsmaßnahmen in allen Phasen des Entwicklungsprozesses zu implementieren, von der Planung und Designphase bis hin zur Implementierung, Testung und Wartung. Zur Hilfe kann hierbei die Artikel, Studien und Empfehlungen des Bundesamt für Sicherheit in der Informationstechnik (BSI) herangezogen werden. Diese geben eine gute Übersicht über die wichtigsten Sicherheitsrisiken und Maßnahmen, die bei der Entwicklung und Nutzung von verschiedensten Tools beachtet werden sollten. Besonders wichtig ist dabei das IT-Grundschutz Kompendium. Es gibt standardisierte Sicherheitsmaßnahmen und Best Practices vor, welche zur Identifizierung und Minimierung von IT-Sicherheitsrisiken dienen.

In den folgenden Kapiteln wird zuerst auf die spezifischen Sicherheitsrisiken von KI-Tools eingegangen. Darauffolgend werden spezifische Herausforderungen in Salesforce und Apex betrachtet und beide in dieser Arbeit vorgestellten KI-Tools Codeium und Einstein for Developers im Rahmen der IT-Sicherheit bewertet. Abschließend werden allgemeine Empfehlungen zur Nutzung von KI-Tools in der Softwareentwicklung gegeben.

5.1 Spezifische Sicherheitsrisiken von KI-Tools

Wie bereits geklärt wurde spielt in der modernen Softwareentwicklung KI und maschinelles Lernen eine zunehmend wichtige Rolle. Die Tools bieten zahlreiche Vorteile, darunter die Automatisierung von Aufgaben, die Verbesserung der Effizienz und die Bereitstellung tiefgehender Analysen[47].

[47] Vlg. Kapitel 2

Diese Technologien werden in einer Vielzahl von Anwendungsbereichen eingesetzt, von der Datenanalyse und dem Kundensupport bis hin zur Optimierung von Geschäftsprozessen und der Entwicklung neuer Produkte. Trotz ihrer Vorteile stellt sich die Frage, welche Risiken KI- Tools mit sich bringen. Vor allem im Bezug auf die IT-Sicherheit.

Die Sicherheit ist aus mehreren Gründen von zentraler Bedeutung. Erstens sind KI-Modelle oft auf große Mengen an Daten angewiesen, darunter auch sensible und personenbezogene Informationen. Der Schutz dieser Daten vor unbefugtem Zugriff, Missbrauch und Verlust ist entscheidend, um die Privatsphäre und die Rechte der betroffenen Personen zu wahren. Die Tools dürfen also nicht einfach diese Daten ausgeben (Obwohl Sie dies durch aus wissen können). Datenschutzverletzungen können nicht nur rechtliche Konsequenzen haben, sondern auch das Vertrauen der Kunden und Partner in das Unternehmen erheblich schädigen.[48]

Zweitens können Schwachstellen in KI-Modellen und -Systemen von böswilligen Akteuren ausgenutzt werden, um Angriffe durchzuführen. Solche Angriffe können darauf abzielen, die Integrität und Verfügbarkeit der Systeme zu beeinträchtigen oder die Modelle zu manipulieren, um falsche oder schädliche Ergebnisse zu erzeugen. Auch ist die Transparenz und Nachvollziehbarkeit von KI-Entscheidungen ein wichtiger Aspekt der IT-Sicherheit. Viele KI-Modelle, insbesondere tiefere neuronale Netzwerke, sind als 'Black Boxes' bekannt, deren interne Entscheidungsprozesse schwer zu interpretieren sind. Dies kann die Identifizierung und Behebung von Sicherheitslücken erschweren und zu unvorhersehbaren und potenziell schädlichen Modellverhalten führen.[49]

Angesichts dieser Herausforderungen ist es unerlässlich, umfassende Sicherheitsmaßnahmen bei der Entwicklung und Nutzung dieser Tools zu implementieren. Dies umfasst technische, organisatorische und prozessuale Maßnahmen, die darauf abzielen, die Sicherheit der Daten, Modelle und Systeme zu gewährleisten. Die Implementierung von Best Practices und die Einhaltung von Sicherheitsstandards wie den Empfehlungen des BSI sind dabei von zentraler Bedeutung. Nun aber genauer dazu, welche Risiken auftreten können. Dazu gibt das BSI folgende Kategorien vor: Risiken im Rahmen der ordnungsgemäßen Nutzung, Risiken durch eine missbräuchliche Nutzung von LLM und Risiken in folge von Angriffen.[50]

Risiken im Rahmen der ordnungsgemäßen Nutzung Die ordnungsgemäße Nutzung von LLMs birgt verschiedene Risiken, die sich aus deren stochastischer Natur, der Qualität und Zusammensetzung der Trainingsdaten sowie der Bereitstellung durch externe Unternehmen ergeben.

[48] Vgl. [Bun24a], S.9
[49] Vgl. ebenda, S.14 ff.
[50] Vgl. ebenda, S.9

LLMs können aufgrund ihrer Trainingsdaten persönliche oder urheberrechtlich geschützte Informationen sowie diskriminierende oder falsche Inhalte wiedergeben. Unausgewogenheiten in den Daten können zu Verzerrungen (Bias) im Modell führen. Weiterhin gibt es keine Garantie für die Richtigkeit oder Qualität der generierten Inhalte. Modelle können 'halluzinieren', also glaubhafte, aber falsche Informationen erzeugen, insbesondere durch Verweise auf erfundene Quellen. LLMs ohne Echtzeitzugriff können keine aktuellen Ereignisse verarbeiten und neigen dazu, falsche Informationen über aktuelle Themen zu erzeugen.[51]

Programmcode welcher durch so ein Modell generiert wurde können schadhaften Code oder Sicherheitslücken enthalten, da sie auf unsicheren Trainingsdaten basieren können. Außerdem sind sie sehr empfindlich gegenüber Veränderungen in den Eingaben, was schnell zu fehlerhaften Ausgaben führen kann, insbesondere bei untypischen oder absichtlich manipulierten Eingaben. Die hohe sprachliche Qualität der generierten Texte kann zu einem übermäßigen Vertrauen in die Aussagen der Modelle führen, was falsche Schlüsse oder ungeprüfte Übernahmen zur Folge haben kann. Auch birgt die Nutzung von LLMs über Internetdienste das Risiko des ungewollten Datenabflusses, sowohl durch die Datenübertragung als auch durch den Zugriff des betreibenden Unternehmens. Dazu zählt auch, dass die Nutzung dieser Modelle auf der Infrastruktur externer Unternehmen zu einer großen Abhängigkeit führt. Sowohl in Bezug auf die Verfügbarkeit als auch auf die Entwicklung und Sicherheitsmechanismen der Modelle.[52]

Risiken durch eine missbräuchliche Nutzung Missbräuchliche Nutzung von KI birgt vielfältige Risiken, die sowohl für Individuen als auch für Organisationen erhebliche Schäden verursachen können. Ein mögliches Beispiel ist die Verbreitung von Falschinformationen. Kriminelle können hochqualitative Textausgaben von LLMs nutzen, um gezielt Falschmeldungen, Propaganda oder Hassnachrichten zu generieren. Diese Texte können auf sozialen Medien oder anderen Plattformen verbreitet werden, um die öffentliche Meinung zu manipulieren, das Vertrauen in Institutionen zu untergraben oder gesellschaftliche Spannungen zu verstärken. Auch gefälschte Produktbewertungen können Verbraucher täuschen und Marktverzerrungen verursachen. Solche Anwendungen zeigen, wie die sprachliche Qualität und der einfache Zugang zu LLMs es Kriminellen ermöglichen, Informationen in großem Maßstab zu manipulieren und zu missbrauchen.[53]

[51] Vgl. [Bun24a], S.9
[52] Vgl. ebenda, S.9
[53] Vgl. ebenda, S.11-14

Ein weiteres erhebliches Risiko stellt das Social Engineering dar. Hierbei nutzen potentielle Täter LLMs, um personalisierte Phishing-E-Mails oder betrügerische Nachrichten zu erstellen. Diese Nachrichten können täuschend echt wirken und durch die Einbindung öffentlich verfügbarer Informationen sogar den Schreibstil bestimmter Personen oder Organisationen nachahmen. Dies erhöht die Wahrscheinlichkeit, dass Opfer auf solche Angriffe hereinfallen, vertrauliche Informationen preisgeben, Geldüberweisungen tätigen oder schadhafte Software installieren. Besonders gefährlich sind Angriffe bei dem Angreifer vorgeben, ein hochrangiger Firmenvertreter zu sein und Mitarbeitende zu finanziellen Transaktionen verleiten. Auch die Re-Identifizierung von Personen aus anonymisierten Daten durch die Kombination verschiedener Datensätze stellt eine Bedrohung dar, da so die Privatsphäre von Individuen verletzt werden kann.[54]

Risiken in Folge von Angriffen entstehen durch sogenannte Privacy Attacks, Evasion Attacks und Poisoning Attacks. Privacy Attacks zielen darauf ab, sensible Informationen aus einem LLMs zu extrahieren. Eine Form davon ist die Rekonstruktion von Trainingsdaten, bei der Angreifer durch gezielte Abfragen an das Modell herausfinden können, welche Daten im Training verwendet wurden. Dies kann besonders kritisch sein, wenn sensible oder personenbezogene Daten im Trainingsdatensatz enthalten sind. Ein weiteres Risiko ist die Embedding Inversion, bei der Angreifer versuchen, ursprüngliche Eingabetexte aus den Embeddings zu rekonstruieren, was vor allem in Anwendungen relevant ist, die auf externe Vektordatenbanken zugreifen. Zudem besteht die Gefahr des Modelldiebstahls, bei dem ein existierendes LLM genutzt wird, um ein Klonmodell zu erstellen, das das Verhalten des ursprünglichen Modells imitiert. Dies kann zur Vorbereitung weiterer Angriffe, wie Evasion Attacks, verwendet werden.[55]

Evasion Attacks zielen darauf ab, die Eingabe an ein LLMs so zu manipulieren, dass es unvorhersehbare oder gewünschte Ausgaben generiert. Dies kann durch geringfügige Veränderungen in der Eingabe geschehen, wie Rechtschreibfehler oder den Einsatz ähnlicher Zeichen. Ein Beispiel ist die Manipulation durch Perturbation, bei der kleine Änderungen im Text das Modell täuschen können, um unerwünschte Inhalte zu verschleiern. Eine andere Form sind Prompt Injections, bei denen Angreifer durch spezifische Eingaben das LLM dazu bringen, aus seiner vorgesehenen Rolle auszubrechen und unerwünschte Inhalte zu generieren. Indirect Prompt Injections gehen noch einen Schritt weiter, indem Manipulationen in Drittquellen versteckt werden, die das LLM nutzt. Dies kann zu einer Veränderung des Chatverlaufs zwischen Nutzern und dem Modell führen und potenziell schadhafte Aktionen auslösen.[56]

[54] Vgl. [Bun24a], S.11-14
[55] Vgl. ebenda, S.14-19
[56] Vgl. ebenda, S.14-19

5.2 Herausforderungen in Salesforce und Apex

Die Sicherheit in der Salesforce-Plattform ist von entscheidender Bedeutung, da diese Plattform häufig sensible und geschäftskritische Daten verwaltet. Durch die zentrale Rolle, die Salesforce in den Geschäftsvorgängen vieler Unternehmen spielt, ist es unverzichtbar, dass diese Daten vor unbefugtem Zugriff, Verlust und Missbrauch geschützt werden. Die Plattform arbeitet auf einem multitenancy-Modell, bei dem mehrere Kunden dieselbe Infrastruktur nutzen. Dies bedeutet, dass die Daten eines Unternehmens streng von den Daten anderer Unternehmen isoliert sein müssen. Es erfordert strikte Maßnahmen zur Isolierung von Daten zwischen verschiedenen Mandanten, um sicherzustellen, dass die Daten eines Unternehmens nicht von anderen Mandanten eingesehen oder manipuliert werden können. Die Herausforderungen bei der Datentrennung und -sicherheit in einem solchen Modell sind komplex, da sie nicht nur technische Lösungen, sondern auch strenge Zugangskontrollen und Überwachungsmechanismen erfordern. Sicherheitslücken in diesem Modell könnten zu schweren Datenschutzverletzungen führen, die nicht nur rechtliche Konsequenzen haben, sondern auch das Vertrauen der Kunden und Partner in die betroffenen Unternehmen erheblich beeinträchtigen könnten.[57]

Salesforce bietet auch eine Vielzahl von APIs und Integrationen an, die es Entwicklern ermöglichen maßgeschneiderte Anwendungen zu erstellen und bestehende Systeme zu erweitern. Diese müssen jedoch sicher konfiguriert und verwaltet werden, um Sicherheitsrisiken zu minimieren. Darüber dürfen nur autorisierte Benutzer Zugriff auf sensible Daten haben, und dies erfordert eine sorgfältige Konfiguration von Rollen, Profilen und Berechtigungen innerhalb der Plattform. Die Sicherheitsarchitektur umfasst dabei mehrere Ebenen, um die Sicherheit, den Zugriff und die Sichtbarkeit von Daten zu gewährleisten. Dies beginnt bei der grundlegenden Plattformsicherheit, die Mechanismen zur Authentifizierung, Autorisierung und Zugriffskontrolle umfasst. Salesforce verwendet außerdem rollenbasierte Zugriffskontrollen, Berechtigungssätze und Profile, um den Zugang zu Daten und Funktionen zu steuern. Zudem bietet Salesforce Technologien wie Transport Layer Security (TLS) für die sichere Datenübertragung. Die Verwaltung von Datenzugriffen basiert auf einem mehrschichtigen Modell, das von der Organisationsebene über Objekte bis hin zu einzelnen Datensätzen reicht. Standardmäßig werden die meisten Daten als privat behandelt, und der Zugriff wird nur nach Bedarf gewährt. Dies wird durch eine Kombination aus rollenbasierter Hierarchie, Freigaberegeln und manueller sowie programmatischer Freigabe erreicht.[58]

[57] Vgl. [Hut23], S.22 ff.
[58] Vgl. ebenda, S.171 ff.

Auch die Sicherheit von Apex-Code ist von zentraler Bedeutung, da unsichere Codierungspraktiken zu Schwachstellen führen können, die von Angreifern ausgenutzt werden könnten. Entwickler müssen sich bewährter Sicherheitspraktiken bewusst sein und diese anwenden, um sicherzustellen, dass der Code robust und sicher ist. Dies umfasst die Validierung und Sanitierung von Benutzereingaben, die Vermeidung von Angriffen und die regelmäßige Durchführung von Sicherheitstests.

Salesforce bietet umfangreiche Verschlüsselungsoptionen, um Daten sowohl im Ruhezustand als auch bei der Übertragung zu schützen. Die Shield Platform Encryption bietet eine erweiterte Verschlüsselung auf Feldebene, die den Schutz sensibler Daten gewährleistet. Zusätzlich zur Verschlüsselung kann auch die Tokenisierung eingesetzt werden, bei der sensible Daten durch nicht umkehrbare Tokens ersetzt werden, um das Risiko von Datenmissbrauch weiter zu minimieren.

Ein weiterer wichtiger Aspekt der Salesforce-Sicherheitsarchitektur ist die kontinuierliche Überwachung von Systemereignissen und die Aufzeichnung von Audit-Trails. Diese Funktionen ermöglichen es Administratoren, alle Zugriffe und Änderungen an Daten und Konfigurationen nachzuverfolgen und bei Bedarf detaillierte Berichte zu erstellen. Dadurch wird nicht nur die Integrität der Daten gewährleistet, sondern es werden auch potenzielle Sicherheitsvorfälle frühzeitig erkannt und gemeldet.

Es zeigt sich, dass die Salesforce-Plattform umfassende Sicherheitsmechanismen und -praktiken bietet, die darauf abzielen, die Vertraulichkeit, Integrität und Verfügbarkeit von Daten zu gewährleisten. Diese Maßnahmen sind unerlässlich, um den Schutz sensibler Informationen zu gewährleisten und den vielfältigen Bedrohungen im digitalen Zeitalter wirksam zu begegnen.

5.3 Sicherheitsbewertung von Codeium und Einstein for Developers

Codeium sammelt personenbezogene Daten wie Name, E-Mail-Adresse, Jobtitel und andere Kontakt- oder Authentifizierungsdaten, die Benutzer bei der Registrierung oder Interaktion mit der Website angeben. Zusätzlich werden automatisch Informationen wie IP-Adresse, Browser- und Geräteeigenschaften sowie Nutzungsdaten gesammelt. Diese Informationen sind notwendig, um die Sicherheit und den Betrieb der Dienstleistungen zu gewährleisten und für interne Analysen und Berichte verwendet zu werden.

Diese Daten werden für verschiedene Geschäftszwecke verwendet, einschließlich der Bereitstellung und Wartung der Dienstleistungen, der Verwaltung von Benutzerkonten, der Anforderung von Feedback und der Kommunikation über administrative Informationen.

Codeium nutzt diese Daten auch, um seine Dienstleistungen sicher und geschützt zu halten sowie zur Einhaltung gesetzlicher und regulatorischer Anforderungen. Marketing- und Werbemitteilungen werden nur nach ausdrücklicher Zustimmung des Benutzers versendet, und es besteht jederzeit die Möglichkeit, sich von diesen Mitteilungen abzumelden. Codeium gibt personenbezogene Daten nur unter bestimmten Bedingungen weiter, darunter mit Zustimmung des Benutzers, zur Vertragserfüllung, zur Einhaltung gesetzlicher Verpflichtungen oder zum Schutz von Rechten und Interessen. Eine Weitergabe an Drittanbieter erfolgt ausschließlich, wenn dies zur Bereitstellung der Dienstleistungen notwendig ist. Es implementiert technische und organisatorische Sicherheitsmaßnahmen, um die Sicherheit der verarbeiteten personenbezogenen Daten zu gewährleisten. Dies umfasst die Verschlüsselung von Daten während der Übertragung mittels SSL und den Schutz von Daten im Ruhezustand durch große Cloud-Anbieter. Trotz dieser Maßnahmen wird darauf hingewiesen, dass keine elektronische Übertragung oder Speichermethode 100 Prozent sicher ist, sodass keine absolute Sicherheit garantiert werden kann.[59]

Codeium verschlüsselt alle Kommunikationen und stellt sicher, dass alle Daten sowohl im Ruhezustand als auch während der Übertragung geschützt sind. Es werden keine externen Machine-Learning-APIs verwendet, um Inferenz- oder Verarbeitungsaufgaben durchzuführen, was das Risiko einer Datenweitergabe oder -leckage über diese Kanäle eliminiert. Weiterhin werden Telemetriedaten wie Latenz, Nutzung von Funktionen und Annahme von Vorschlägen gesammelt. Diese Daten dienen der Verbesserung der Dienstqualität und werden nicht an Dritte verkauft oder weitergegeben. Benutzer haben die Möglichkeit, sich jederzeit von der Telemetriedatenerfassung abzumelden, um ihre Präferenzen zu respektieren und ihre Datenkontrolle zu stärken.[60]

Außerdem ist das Tool Service Organization Control 2 (SOC2) Typ II-konform, was bedeutet, dass es strenge Sicherheits- und Datenschutzstandards einhält. Diese Compliance stellt sicher, dass Codeium als führend in der Privatsphäre und Sicherheit von AI-Entwicklungstools anerkannt wird.[61] Codeium zeigt ein hohes Maß an Engagement für den Datenschutz und die Sicherheit der Benutzerdaten. Die umfassenden Datenschutzrichtlinien, starken Sicherheitsmaßnahmen und die Einhaltung von SOC2 Typ II-Standards bieten Benutzern Vertrauen in die sichere und verantwortungsvolle Handhabung ihrer Daten. Besonders hervorzuheben ist die Möglichkeit der Benutzer, sich von bestimmten Datenerfassungspraktiken abzumelden und die klare Kommunikation über die Nutzung und den Schutz ihrer Daten.

[59] Vgl. [Exa24]
[60] Vgl. ebenda
[61] Vgl. ebenda

Einstein for Developers erfasst und verwendet verschiedene Arten von Daten, abhängig von der Nutzung innerhalb der unterstützten IDEs. Hierzu gehören eingereichte Codes, die Beschreibungen der Code-Logik oder Ziele in Klartext sowie der dazugehörige Code, der zur Bereitstellung zusätzlicher Kontextinformationen verwendet wird. Zudem wird das Feedback der Benutzer zu Einstein for Developers gesammelt, wobei Salesforce eine weltweit gültige, unwiderrufliche und zeitlich unbegrenzte Lizenz zur Nutzung und Speicherung dieses Feedbacks eingeräumt wird. Die Verarbeitung und Speicherung dieser Daten erfolgt über die Infrastruktur von Amazon Web Services (AWS) in den United States of America (USA).[62]

Benutzer sollten sicherstellen, dass eingereichte Codes und Feedback keine personenbezogenen Daten enthalten. Es wird empfohlen, konstruktives Feedback zu geben, um die Weiterentwicklung der Dienste zu unterstützen. Zudem sollte sichergestellt werden, dass die Nutzung der Dienste im Einklang mit allen geltenden Gesetzen und Vorschriften steht. Ein generierter Code bleibt Eigentum des Erstellers. Ausgaben, die das Tool generiert, können Ähnlichkeiten mit Trainingsdaten aufweisen, aber die eingegebenen Daten werden nicht weitergegeben oder zum Training verwendet. Dennoch wird in den häufig gestellten Fragen angedeutet, dass der CodeGen-Service das Knowhow aus anonymisierten Daten nutzt, welche aber vorher durch eine Anonymisierung entfernt wurden.[63]

Salesforce hat eine Reihe technischer und organisatorischer Maßnahmen implementiert, um die Sicherheit der verarbeiteten Daten zu gewährleisten. Alle Kommunikationen und Datenübertragungen werden mit branchenüblichen Verschlüsselungstechnologien wie TLS geschützt. Der Zugriff auf die Dienste erfordert eine gültige API-Schlüssel/Geheimnis/Token, die verschlüsselt übertragen werden. Salesforce überwacht die Dienste auf unbefugte Zugriffe mit verschiedenen Erkennungsmechanismen. Eingereichte Codes und Feedback werden nicht an Dritte weitergegeben oder zur Verbesserung des Dienstes oder zum Training eines globalen Modells verwendet. Einstein for Developers ist durch verschiedene Sicherheits- und Datenschutzstandards zertifiziert, darunter die APEC Privacy Recognition for Processors, die EU und United Kingdom (UK) Binding Corporate Rules für Prozessoren, die ISO 27001/27017/27018 Zertifizierung sowie TRUSTe Zertifizierung und das Information System Security Management and Assessment Program.[64]

Einstein for Developers wird 'wie besehen' ohne jegliche Garantien bereitgestellt. Die Nutzung erfolgt auf eigenes Risiko des Benutzers.

[62] Vgl. [Sal24b]
[63] Vgl. ebenda
[64] Vgl. [Sal24d], Trusted AI

Salesforce haftet in keinem Fall für direkte, indirekte, spezielle, zufällige, strafbare oder Folgeschäden, einschließlich, aber nicht beschränkt auf entgangenen Gewinn, Datenverlust oder Nutzungsausfall.[65] Gemäß der AI Acceptable Use Policy besteht eine Kennzeichnungspflicht für KI-generierten Code. Zudem ist die Verwendung in risikobehafteten Bereichen laut Salesforce untersagt. Vertrauliche Informationen bleiben durch Verschwiegenheitsklauseln geschützt. Da die Daten außerhalb der EU verarbeitet werden, unterliegen sie anderen Datenschutzgesetzen, welches möglicherweise zu datenschutzrechtlichen Bedenken führen kann. Dazu zählt beispielsweise die unbefugte Speicherung und Weitergabe von personenbezogenen Daten.[66]

Salesforce gibt mit ihrem KI-Tool viel für Datenschutz und IT-Sicherheit. Durch umfassende Sicherheitsmaßnahmen und Compliance-Zertifizierungen bietet das Tool ein solides Sicherheitsniveau. Benutzer sollten jedoch beachten, dass keine absolute Sicherheit garantiert werden kann und die Nutzung auf eigenes Risiko erfolgt. Die Verantwortung für den sicheren und gesetzeskonformen Umgang mit den Daten liegt Bei dem Benutzer.

5.4 Strategien zur Risikominderung

Die Implementierung von KI-Tools in Salesforce bietet sichtbare Vorteile, die in den vorgangenen Kapiteln dieser Arbeit beleuchtet und erkannt wurden. Um diese Vorteile jedoch sicher nutzen zu können, ist es entscheidend, potenzielle Risiken zu identifizieren und geeignete Maßnahmen zur Risikominderung zu ergreifen.

Identifikation und Analyse von Risiken: Der erste Schritt zur Risikominderung besteht in der Durchführung einer detaillierten Risikoanalyse. Diese Analyse dient dazu, Schwachstellen und Bedrohungen zu erkennen, die sich aus der Nutzung von KI-Tools in Salesforce ergeben könnten. Eine gründliche Risikoanalyse umfasst einerseits die Erkennung von Schwachstellen und Bedrohungen. Dazu gehören potenzielle Datenschutzverletzungen, unbefugter Zugriff auf sensible Daten und die Manipulation von KI-Modellen. Nachdem die Risiken identifiziert wurden, müssen ihre potenziellen Auswirkungen auf die Sicherheit und Integrität der Daten bewertet werden. Dies umfasst die Analyse der möglichen Konsequenzen für die Vertraulichkeit, Integrität und Verfügbarkeit der Daten.

Um die identifizierten Risiken zu minimieren, sollten KI-Tools nach Best Practices integriert werden. Dazu gehört unter anderem das die Tools eingängig auf die Einhaltung von Datenschutz- und Sicherheitsvorschriften geprüft werden.

[65] Vgl. [Sal24b]
[66] Vgl. [Sal24a]

Es muss sichergestellt werden, dass alle eingesetzten KI-Tools den geltenden Datenschutz- und Sicherheitsvorschriften entsprechen. Dies beinhaltet die Einhaltung der Datenschutz-Grundverordnung (DSGVO) und anderer relevanter rechtlicher Rahmenbedingungen. Dies sollte regelmäßig durchgeführt werden. Vor allem ist es bei Tools wie Codeium wichtig die Funktionsweisen und die Verarbeitung von Daten zu überprüfen. Diese können, wie auch in dem Fall von Codeium, das gesamte Projekt als Kontext beziehen und damit eventuell sensible Daten verarbeiten. Dazu zählt beispielsweise eine geöffnete Datenbank mit Kundendaten, Authentifizierungsdaten oder andere sensible Informationen. Ein Datenschutzbeauftragter kann dabei unterstützen, die Einhaltung dieser Regelungen sicherzustellen und unbefugte Zugriffe oder Verarbeitungen zu verhindern[67].

Ein wichtiger Aspekt der Prüfung ist, dass die Betreiber dieser Tools klar und transparent darlegen sollten, wie Nutzerdaten, einschließlich Eingaben und generierten Ausgaben, verarbeitet werden und welche Risiken damit verbunden sind. Ebenso sollten sie die Limitierungen und Schwächen des Systems klar kommuniziert werden.[68] Ist dies nicht der Fall, ist die Nutzung dieses Tools womöglich nicht empfehlenswert, da die Risiken nicht transparent sind und somit nicht eingeschätzt werden können.

Auf Seite der Nutzer sollten diese mit der Preisgabe von persönlichen Daten achtsam umgehen, sowohl bei der Anmeldung als auch bei Eingaben von Daten. Sie sollten in der Lage sein, die Ausgaben von diesen Tools kritisch zu hinterfragen, auf Wahrheitsgehalt und Manipulation zu prüfen und entsprechend zu handeln. Dazu zählt beispielsweise das Überprüfen von generierten Code auf Sicherheitslücken oder auf Urheberrechtsverletzungen. Auch sollten sie sich bewusst sein, dass die generierten Ausgaben möglicherweise nicht immer korrekt oder vertrauenswürdig sind und daher nicht ohne weiteres übernommen werden sollten. Sie sollten regelmäßig geschult werden, um sicherzustellen, dass sie die Best Practices und Sicherheitsrichtlinien für den Umgang mit KI-Tools kennen und anwenden können. Dabei sollte außerdem Schulungsmaterialien und Guidelines entwickelt werden, die die Nutzer klar und verständlich über diese Themen informieren. Die Sensibilisierung der Nutzer hinsichtlich der Fähigkeiten und Schwächen von KI sowie möglicher Angriffsvektoren und daraus resultierender Bedrohungen ist entscheidend zur Risikominderung[69].

Dabei sollten die Anforderungen an Sensibilisierung und Schulung in allen Arbeitsumgebungen berücksichtigt und in andere Schulungsthemen durch die Personalabteilung oder das Weiterbildungsmanagement integriert werden.

[67] Vgl. [Bun24b], CON.2 - CON.3
[68] Vgl. [Bun24a], S. 22ff
[69] Vgl. ebenda, S. 22ff

Unwissenheit über Regelungen, fehlende Sensibilisierung, mangelnde Unterstützung und Ressourcen, sowie fehlende Akzeptanz von Sicherheitsmaßnahmen sind Gefährdungen, die durch eine geeignete Schulung minimiert werden können. Es ist wichtig, dass die Leitungsebene sensibilisiert ist und Schulungen unterstützt, um eine Vorbildfunktion einzunehmen. Mitarbeiter sollten in Richtlinien eingewiesen und regelmäßig in ihren Aufgaben, Verantwortlichkeiten und Vorgehensweisen nach IT-Grundschutz geschult werden. Online-Kurse des BSI und Zertifizierungen können hierfür genutzt werden.[70]

Auch die Auswahl eines geeigneten Vorgehensmodells für die Entwicklung ist von großer Bedeutung. Ein ungeeignetes Modell kann den Entwicklungsverlauf und das Projektmanagement stören und relevante Sicherheitsaspekte vernachlässigen. Ein geeignetes Modell muss Sicherheitsanforderungen integrieren und das Personal in der Methodik des Modells geschult werden. Die Entwicklungsumgebung sollte auf Basis klar definierter Auswahlkriterien ausgewählt werden, um Fehlfunktionen und Sicherheitslücken zu vermeiden. Ein sicheres Systemdesign, die Verwendung externer Bibliotheken aus vertrauenswürdigen Quellen und entwicklungsbegleitende Tests sind weitere Maßnahmen, die die Sicherheit der Software erhöhen. Eine geeignete Versionsverwaltung des Quellcodes und die Überprüfung externer Komponenten auf Schwachstellen und Konflikte sind ebenfalls wichtig.[71] Dazu zählt auch die Untersuchung der genutzten KI-Tools und deren Integration in die IDEs.

[70] Vgl. [Bun24b], ORP3
[71] Vgl. ebenda, CON.8

6 Zusammenfassung und Ausblick

Die vorliegende Arbeit untersuchte die Potenziale und Herausforderungen der Integration von Künstlicher Intelligenz (KI) in Salesforce-Entwicklungsprozesse. Der Fokus lag auf den beiden Tools Codeium und Einstein for Developers. Dabei unterlag diese Arbeit folgenden drei Zielen:

1. In welchen Bereichen der Salesforce-Entwicklungsprozesse können Codeium und Einstein for Developers eine Optimierung bewirken?
2. Wie unterscheiden sich die beiden Tools in ihrer Herangehensweise und Effektivität?
3. Welche messbaren Verbesserungen können durch den Einsatz dieser KI-Tools erzielt werden?

Um diese Forschungsfragen zu beantworten wurde vorerst eine Umfrage unter Entwicklern durchgeführt um die eigentlichen Probleme in Programmierung herauszufinden, welchen die Entwickler jeden Tag begegnen. Die Ergebnisse zeigen einerseits, dass die Hauptprobleme die Einhaltung von Coding-Standards und Best-Practices, die Dokumentation von Code, Fehlerbehebung und das Erstellen von Tests sind. In alle diesen Bereichen suchen Entwickler Unterstützung. Aufbauend darauf wurden die zwei KI-Tools, Codeium und Einstein for Developers, in praktischen Tests in der IDE VS-Code evaluiert. Diese wurden in den Hauptdisziplinen von Salesforce (Trigger, Lightning Web Components (LWC) oder Tests erstellen) getestet und bewertet. Die Bewertung umfasste einerseits die Qualität der Ergebnisse durch Kriterien welche vorab festgelegt wurden: die Geschwindigkeit, die Funktionalität und Fehleranfälligkeit bewertet durch statische Codeanalyse Tools. Dadurch wurde eine erste These festgehalten: Die Überlegenheit von Codeium im praktischen Test. Alle eingegebenen Promts wurden außerdem mit einem spezifischen, einleitenden Satz begonnen um zu untersuchen wie sich das Promting auf die Ergebnisse auswirkt. Dabei wurde entschieden der KI jeweils mitzuteilen, dass sie entweder ein Junior, Senior oder Trainee Entwickler ist. Woraus sich die zweite These ergab: Das Promting wird einen signifikanten Einfluss auf die Qualität und Geschwindigkeit der Ergebnisse haben.

Die Ergebnisse des Tests zeigen, dass Codeium und Einstein for Developers signifikant zur Optimierung von Entwicklungsprozessen beitragen können. Codeium erwies sich in den meisten getesteten Kategorien als überlegen, insbesondere bei der Arbeit mit LWCs und der Dokumentation von Code wodurch die erste These belegt wurde. Einstein hingegen war zwar oftmals schneller als Codeium, jedoch waren die Antworten qualitativ nicht auf dem Niveau des ersten Tools.

Trotz dessen ist es geeignet für das Erstellen von einfachen Triggern und Tests. Die Untersuchung der Forschungsfragen ergab, dass beide Tools in den verschiedenen Bereichen wertvolle Unterstützung bieten können, wobei der richtige Einsatz und das geeignete Promting entscheidend für die Qualität der Ergebnisse sind. Damit wurde auch die zweite These belegt. Die Umfrage bestätigte die positiven Erfahrungen aus den praktischen Tests, zeigte aber auch, dass die Entwickler unterschiedlich stark auf KI-Tools setzen. Die Bewertung der Tools durch die Entwickler verdeutlichte, dass es eine breite Akzeptanz für den Einsatz von KI-Tools in der Softwareentwicklung gibt, aber auch Herausforderungen und Unsicherheiten bestehen. Folgend aus der Auswertung entstanden folgende Best Practices, welche bei der Nutzung der beiden Tools beachtet werden sollten:

- Kontextbezogenheit von Codeium: Funktioniert dieser Kontext bei Codeium nicht, sollte ein Neustart, ein neuer Chat oder eine Neuzuordnung des Kontexts durchgeführt werden.
- Für detaillierte Ergebnisse (mehr Erklärungen und einfacher) in der Code-Erstellung, sollte einen Trainee- und Junior-Prompt benutzt werden. Für funktionale Ergebnisse mit Optimierungsvorschlägen sollte der Senior-Prompt benutzt werden. (Codeium)
- Bei der Dokumentation sollte grundsätzlich ein Senior-Prompt benutzt werden, da dieser detaillierter ist.
- Die Nutzung einer statischen Codeanalyse (Wie ESLint oder Apex PMD) ist Empfohlen, da beide Tools Fehleranfällig für syntaktische Fehler (wie Anführungszeichen in Apex) sind.

Trotz der positiven Ergebnisse dieser Arbeit gibt es auch Punkte, die es zu beachten gilt. Auf der einen Seite hätte die durchgeführte Umfrage von einem größeren Umfragekreis profitieren können, um repräsentativere und belastbarere Ergebnisse zu erhalten. Auch ist im praktischen Teil der Arbeit die Qualität der Antworten schwer zu messen, da sie subjektiv sind und von den individuellen Anforderungen und Erwartungen der Benutzer abhängen. Auch ist die Bewertung der Zeit durchaus schwer, da der Vergleich zwischen berechneter Zeit (Gemessen + Pufferzeit + Prompting + Überarbeitung) und der Schätzung im Planning Poker sehr ungenau ist. Die Zeit, einen Prompt zu erstellen, variiert von Entwickler zu Entwickler und hängt von der Qualität der gestellten Aufgabe (z.B. Ticketbeschreibungen), der Erfahrung oder dem Tool ab. Auch die Überarbeitung individuell unterschiedlich, was die Ergebnisse eher subjektiv macht.

Auf einer anderen Seite ist die Subjektivtät durch den zeitlichen Rahmen und Aufwand der Arbeit zu begrenzen. Angefangen mit der Durchführung des praktischen Tests. Die gestellten Aufgaben sind unterschiedlich und könnten nicht unbedingt den Anforderungen der Praxis entsprechen. Auch sind diese subjektiv als leicht eingestuft wurden. Die Sinnhaftigkeit der Antworten ist sehr variabel, und eine objektive Bewertung ist schwer. Diese anhand der gewählten Kriterien zu bewerten ist eher nur beispielhaft. Eine solche Bewertung erfordert eine umfangreichere und detailliertere Betrachtung.

Für zukünftige Untersuchungen sind umfangreichere Tests erforderlich. Es reicht bei diesem Thema nicht, die selbe Aufgabe mit gleichen Promps sehr oft durchzuführen. Eher sollte eine große, unterschiedliche Anzahl an Promts getestet werden um eventuelle Verbesserungen in den Ausgaben festzustellen. Eine Automatisierbarkeit der Disziplinen und festgelegter Kriterien wäre wünschenswert. Dies könnte theoretisch durch die Nutzung von APIs ermöglicht werden, um beide Tools ohne großen Aufwand zu testen. Es sollten auch komplexere Aufgaben untersucht werden, welche beispielsweise im Rahmen eines Projektes anfallen. Auch könnte die Untersuchung der zweiten Version von Codeium sinnvoll sein. Durch den Beleg der zweiten These kann eine Standardisierung der Prompts ebenfalls in Betracht gezogen werden. Eine Tabelle mit vorgefertigten Prompts zum Kopieren könnte die Nutzung und Vergleichbarkeit der Ergebnisse verbessern.

Die Arbeit sollte daher als Grundlage für weitere Untersuchungen genutzt werden und durch objektivere Methoden ergänzt werden. Doch bietet sie einen Einblick in die Funktionsweise und Sinnhaftigkeit der beiden Tools. Entwickler, welche in der Umfrage eher negativ vom Thema KI eingestellt waren, sollten sich dennoch mit den Tools auseinandersetzen und diese Arbeit als Grundlage nutzen um die Vorteile beider zu erkennen. Unternehmen und Entwicklungsteams sollten die Nutzung von KI in weiteren Entwicklungsbereichen fördern, insbesondere in den Bereichen, die derzeit selten optimiert werden. Durch gezielte Schulung, Erweiterung der Nutzung und kontinuierliche Verbesserung der Tools kann die Effizienz und Qualität der Entwicklungsarbeit weiter gesteigert werden. Abschließend lässt sich festhalten, dass die Integration von KI-Tools in Salesforce-Entwicklungsprozesse großes Potenzial zur Effizienzsteigerung und Qualitätsverbesserung bietet.

Literaturverzeichnis

[Bun24a] Bundesamt für Sicherheit in der Informationstechnik: „Generativen KI-Modelle“, Juni 2024.
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/KI/Generative_KI-Modelle.pdf?__blob=publicationFile&v=5

[Bun24b] Bundesamt für Sicherheit in der Informationstechnik: „IT-Grundschutz-Kompendium 2023“, Juli 2024.
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/IT_Grundschutz_Kompendium_Edition2023.pdf?__blob=publicationFile&v=4

[Exa24] Exafunction, Inc.: „Codieum AI“, Mai 2024.
https://codeium.com/

[Hut23] Hutcherson, D. J. J. A.: „Handbuch für Salesforce-Architekten: Ein umfassender Leitfaden für End-to-End-Lösungen“, Apress / Springer Berlin Heidelberg / Springer Vieweg / Springer, Berlin, Juni 2023

[Kre23] Kreutzer, R. T.: „Künstliche Intelligenz verstehen: Grundlagen - Use-Cases - unternehmenseigene KI-Journey“, 2., vollständig überarbeitete und erweiterte Auflage. Springer Fachmedien Wiesbaden / Springer Gabler / Springer, Berlin, Juni 2023

[Mic24] Microsoft Corporation: „Microsoft Forms“, Mai 2024.
https://www.microsoft.com/de-de/microsoft-365/online-surveys-polls-quizzes

[Pol24] PollPool GbR.: „PollPool“, Mai 2024.
https://www.poll-pool.com/

[Pos23] Posthoff, C.: „Computer und Künstliche Intelligenz: Vergangenheit - Gegenwart - Zukunft“, 1. Aufl. Springer Vieweg, Juni 2023

[Sal24a] Salesforce Inc.: „AI Acceptable Use Policy“, 2024.
https://www.salesforce.com/content/dam/web/en_us/www/documents/legal/Agreements/policies/ai-acceptable-use-policy.pdf

[Sal24b] Salesforce Inc.: „Salesforce Developer Documentation“, Mai 2024.
https://developer.salesforce.com/

[Sal24c] Salesforce Inc.: „Trailhead“, Juli 2024.
https://trailhead.salesforce.com/de/users/profiles/orgs

[Sal24d] Salesforce Inc.: „Was ist Salesforce?“, Juni 2024.
https://www.salesforce.com/de

[Son24] Sonja Cechak: „Deep Learning, Neuronale Netze und Marketing Automation“, Juni 2024.
https://imbstudent.donau-uni.ac.at/aihubyou/ai-grundlagen/deep-learning-neuronale-netze-und-marketing-automation/

[Wen20] Wennker, P.: „Künstliche Intelligenz in der Praxis: Anwendung in Unternehmen und Branchen: KI wettbewerbs- und zukunftsorientiert einsetzen“, 1. Aufl. Springer Fachmedien Wiesbaden;Springer Gabler, Juni 2020