HAW HAMBURG

BACHELORARBEIT
Laura Jurgeneit

Vergleich von Slack und Microsoft Teams Bots

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Erstbetreuer: Prof. Dr. Martin Becke
Zweitbetreuer: Prof. Dr. Martin Hübner
Eingereicht am: 31. Januar 2024

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Laura Jurgeneit

Thema der Arbeit

Vergleich von Slack und Microsoft Teams Bots

Stichworte

Slack, Microsoft Teams, Bot, Qualitätsmerkmale, Kollaborationssoftware, ISO/IEC 25010, Wartbarkeit, Effizienz

Kurzzusammenfassung

Diese Arbeit untersucht die Implementierung und Nutzung der Bots von Slack und Microsoft Teams. Es werden die Unterschiede beider Umsetzungen an einem Beispiel herausgearbeitet und bewertet. Das Ziel ist es, eine Aussage darüber zu treffen, ob Slack oder Microsoft Teams besser geeigenet ist, um einen Bot, wie er in dieser Ausarbeitung implementiert wurde, zu betreiben. Hierfür wurde auf beiden Plattformen jeweils ein Bot implementiert, der die gleichen Anforderungen erfüllen soll. Diese Anforderungen stammen aus einer Befragung der Mitarbeiter der Firma CAS AG. Die Implementierungen wurden miteinander verglichen und anhand der Qualitätsmerkmale *Wartbarkeit* und *Effizienz* bewertet. Die Grundlage des Vergleichs ist im ISO/IEC 25010 [61] festgelegt. Des Weiteren wurden die Kosten mit einbezogen, die nötig sind, um die Bots zu betreiben. Der Vergleich zeigt, dass Slack bei der Wartbarkeit und Effizienz Microsoft Teams gegenüber knapp überlegen ist. Bei den Kosten erzielt Microsoft Teams jedoch bessere Ergebnisse. Nach der Diskussion der Ergebnisse wurde das Fazit gezogen, dass Microsoft Teams die bessere Wahl zur Implementierung des Bots ist, da es kostengünstiger ist und in den Punkten der Wartbarkeit und Effizienz nur geringfügig schlechtere Werte als Slack gemessen hat und sich somit auf sehr ähnlichem Niveau befindet.

Laura Jurgeneit

Bachelor thesis title

Comparison of Slack and Microsoft Teams bots

Keywords

Slack, Microsoft Teams, bot, quality models, collaboration software, ISO/IEC 25010, maintainability, efficiency

Abstract

This paper examines the implementation and use of bots from Slack and Microsoft Teams. The differences between the two implementations are worked out and evaluated. The aim is to come to a conclusion as to whether Slack or Microsoft Teams is better suited to operate a bot as implemented in this paper. For this purpose, a bot was implemented on both platforms, which should fulfill the same requirements. These requirements originate from a survey of CAS AG employees. The implementations were compared with each other and evaluated based on the quality criteria of maintainability and efficiency. The basis of the comparison is defined in ISO/IEC 25010 [61]. The costs required to operate the bots were also included. The comparison shows that Slack is slightly superior to Microsoft Teams in terms of maintainability and efficiency. In terms of costs, however, Microsoft Teams achieves better results. After discussing the results, the conclusion was drawn that Microsoft Teams is the better choice for implementing the bot, as it is more cost-effective and it measured maintainability and efficiency was only slightly worse than Slack and therefore ends up at a very similar level.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielsetzung . . . 2
1.3 Gliederung . . . 2

2 Verwandte Arbeiten **3**

3 Grundlagen **5**
3.1 Slack und Microsoft Teams . . . 5
3.2 Bots . . . 5
3.3 ISO/IEC 25010 . . . 7

4 Konzept **8**
4.1 Anforderungen . . . 8
4.1.1 Stakeholder . . . 8
4.1.2 Systemkontext . . . 9
4.1.3 Anwendungsfälle . . . 11
4.2 Beschreibung der Software . . . 14
4.2.1 Architektur . . . 14
4.2.2 Verteilungsdiagramm . . . 17
4.3 Funktionsweise . . . 19
4.3.1 UC-01 - Verfügbare Mitarbeiter ermitteln . . . 19
4.3.2 UC-02 - Gemeinsame freie Termine ermitteln . . . 22
4.3.3 UC-03 - Kanal Text Suche anfordern . . . 25
4.3.4 UC-04 - Daten beschaffen . . . 27
4.3.5 UC-05 - Daten aufbereiten . . . 28

5 Bewertung der Implementierung **29**
5.1 Wartbarkeit . . . 29
5.1.1 Lines of Code (LoC) . . . 31
5.1.2 Duplizierter Code . . . 32
5.1.3 Zyklomatische Komplexität . . . 32
5.1.4 Halstead-Metrik . . . 34
5.1.5 Software Maintainability Index . . . 35
5.1.6 Verwendung der Metriken zur Bewertung der Wartbarkeit . . . 36

5.2 Effizienz . . . 36
5.2.1 Begriffsdefinitionen . . . 36
5.2.2 Anwendung auf diese Ausarbeitung . . . 39

6 Experimente **41**
6.1 Wartbarkeit . . . 41
6.1.1 Forschungsfrage . . . 42
6.1.2 Erwartungen . . . 43
6.1.3 Aufbau . . . 44
6.1.4 Durchführung . . . 44
6.1.5 Ergebnisse . . . 45
6.1.5.1 Implementierung des Slack Bots . . . 45
6.1.5.2 Implementierung des Slack Bot Frameworks . . . 47
6.1.5.3 Implementierung des Microsoft Teams Bots . . . 48
6.1.5.4 Implementierung des Microsoft Teams Bot Frameworks . . . 48
6.1.5.5 Zusammenfassung . . . 49
6.2 Effizienz . . . 50
6.2.1 Forschungsfrage . . . 50
6.2.2 Erwartungen . . . 52
6.2.3 Aufbau . . . 53
6.2.4 Durchführung . . . 54
6.2.5 Ergebnisse . . . 58

7 Diskussion **61**
7.1 Ergebnisse der Experimente . . . 61
7.1.1 Wartbarkeit . . . 62
7.1.2 Effizienz . . . 67
7.2 Kosten . . . 71
7.3 Programmiersprachen . . . 73

8 Zusammenfassung **75**
8.1 Fazit . . . 75
8.2 Relexionen . . . 75
8.3 Ausblick . . . 77

Literatur **I**

Glossar **VIII**

Abbildungsverzeichnis **XIII**

Tabellenverzeichnis **XIII**

Selbstständigkeitserklärung **XV**

1 Einleitung

1.1 Motivation

In jedem Unternehmen gibt es Aufgaben, die repetitiv sind und automatisiert werden können. Einige davon hängen von Informationen zu anderen Mitarbeitern ab. Das Suchen, Vergleichen und Evaluieren dieser Informationen ist zeitintensiv. Insbesondere, wenn diese aus mehreren Quellen bezogen werden müssen. Beispielsweise ist es zeitaufwendig, einen gemeinsamen freien Termin mit zwei weiteren Kollegen zu finden, da in diesem Fall drei Kalender nacheinander Termin für Termin verglichen werden müssen. Heute gibt es kollaborative Programme, die eine Lösung dafür bieten, indem sie eine Schnittstelle zur Verfügung stellen, mit welcher ein Bot implementiert werden kann, der von allen Nutzern, die zum Unternehmen gehören und das Programm verwenden, ansprechbar ist. Zwei Kollaborationsprogramme, die solch eine Schnittstelle anbieten [53] [38] und laut Statista [62] einen nennenswerten Marktanteil haben, sind Microsoft mit 23,7 % und Slack mit 5,1 %. Das Unternehmen *CAS AG* [4] nutzt genau diese zwei Programme zur Kollaboration und hat Anforderungen an einen Bot formuliert, die den Arbeitsalltag erleichern sollen. Aus diesem Grund werden die beiden genannten Programme in dieser Ausarbeitung untersucht. Es wurden die folgenden zwei Qualitätsmerkmale ausgewählt, die zum Vergleich beider Bots genutzt werden: Wartbarkeit und Effizienz. Das Merkmal Warbarkeit ist relevant, da der Bot von Mitarbeitern der CAS AG gewartet wird. Der Bot ist nur ein Werkzeug, um die Mitarbeiter in ihrem Alltag zu unterstützen und die Wartung sollte wenig Zeit in Anspruch nehmen. Je einfacher der Bot zu warten ist, desto geringer ist das Risiko, dass der Wartungsbeauftragte einen Programmierfehler verursacht. Das Merkmal Effizienz ist ebenfalls von Relevanz, da es Zeit und in folge dessen Geld spart. Je weniger Zeit ein Bot benötigt, um die Anfrage eines Nutzer zu beantworten, desto eher kann dieser mit der Antwort des Bots weiter arbeiten und kommt somit schneller zum Ziel. Des Weiteren ist sparsame Ressourcennutzung von Vorteil, da diese freien Ressourcen so an anderen benötigten Stellen verwendet werden können und weniger Verschleiß entsteht. Beispielsweise sollte eine Anfrage an einen Bot nicht den Arbeitsspeicher des Servers in der Art auslasten, dass der Server nicht mehr in der Lage ist, andere Prozesse parallel auszuführen. Ebensowenig sollte der Nutzercomputer durch starke Ressourcennutzung des Bots in seiner Arbeit behindert werden oder die Hardware des Computers durch übermäßige Beanspruchung verschleißen. Letztendlich sind die Kosten zur Betreibung der Bots noch von Bedeutung. Da dieses Unterstützungswerkzeug

keine Einnahmen generiert, sollte es bei voller Funktion möglichst wenig Ausgaben verursachen. Um von den zwei Kollaborationsprogrammen das für die CAS AG geeignetste zu ermitteln, müssen die beiden Bot Schnittstellen untersucht und mit den Anforderungen des Unternehmens abgeglichen werden.

1.2 Zielsetzung

Ziel dieser Ausarbeitung ist es, die Bot Schnittstellen der Kollaborationsprogramme Slack und Microsoft Teams anhand eines Beispiels mit einander zu vergleichen und mit Hilfe der Ergebnisse zu bestimmen, welcher Bot in diesem Fall die geeignetere Wahl ist. Der Bot erfüllt Anforderungen, die aus einer Umfrage in dem Unternehmen CAS AG [4] stammen.

1.3 Gliederung

Zunächst werden verwandte Arbeiten vorgestellt. Anschließend werden Grundlagen etabliert. Danach wird ein Konzept vorgestellt, welches bestimmte Anforderungen an eine Bot Implementierung vorgibt. Dieses wird die Grundlage der Implementierung als Nachweis der Machbarkeit sein. Im nächsten Schritt werden Experimente durchgeführt, um im Folgenden die Wartbarkeit sowie die Effizienz zu diskutieren. Danach wird ein Blick auf die Kosten geworfen, die zur Betreibung der Bots nötig sind. Zum Schluss werden die Ergebnisse der Vergleiche zusammengefasst.

2 Verwandte Arbeiten

Eine Herausforderung dieser Arbeit ist es, einen Vergleich zwischen dem Slack und Microsoft Teams Bot zu schaffen. Somit ist es hilfreich zu schauen, welche bereits existierenden Arbeiten es gibt, die Software bewerten, miteinander vergleichen oder die Eigenschaften anhand von ISO Qualitätsmerkmalen messen. In diesem Abschnitt werden Artikel vorgestellt, von denen diese Ausarbeitung in Bezug auf die Messung und Bewertung der Wartbarkeit sowie der Effizienz inspiriert ist.

Performanz Vergleich: Nurdayenti und Amiruddin [52] haben drei Passwort Manager auf die Qualitätsmerkamle Usability, Performanz und Sicherheit getestet. Der Abschnitt, der für diese Ausarbeitung relevant ist, ist der, in dem es um die Messung der Performanz geht, denn diese entspricht dem, was in dieser Ausarbeitung als Effizienz bezeichnet wird. Hierzu wurden die zwei Applikationen Process Monitor und SysGauge verwendet, um die Performanz in Bezug auf Zeitverhalten, Ressourcennutzung und Kapazität zu überwachen. Um das Zeitverhalten zu testen, wurde für jeden Passwort Manager die Zeit gemessen, die er benötigt hat, um ein Passwort zu speichern. Dieses Experiment wurde für jeden Passwort Manager 100 Mal mit dem gleichen Passowrt durchgeführt. Aus den Ergebnissen wurde ein Mittwlwert gebildet, um sie anschließend miteinander zu vergleichen. Um die Ressourcennutzung zu messen, wurde der CPU- und Arbeitsspeicherverbrauch gemessen, der während des Speicherns eines Passwort benötigt wird. Auch dieses Experiment wurde 100 Mal mit dem selben Passwort durchgeführt, um danach die Mittelwerte miteinander zu vergleichen. Bezüglich der Kapazität wurde die Größe der Datei gemessen, die die Passwortdatenbank enthält, nachdem 100 Mal das gleiche Passwort neu gespeichert wurde. Je kleiner die Datei ist, desto größer ist die Speicherkapazität.

Die von Nurdayenti und Amiruddin [52] genutzte Vorgehensweise zur Messung der Performanz wird zur Orientierung verwendet, um in dieser Ausarbeitung die Effizient zu messen. Es wird falls nötig auf Software zurückgegriffen, um das Zeitverhalten sowie die Ressourcennutzung zu messen. Die Experimente werden mehrfach durchgeführt und es wird aus den Ergebnissen ein Mittwelwert gebildet, um das Risiko zu mindern, dass Störungen den Ergebniswert beeinflussen.

Metriken aus Qualitätskriterien formulieren: Steidl et al. [65] beschreiben in ihrem Artikel die Qualitätskontrolle von Software in Bezug auf Wartbarkeit, die auf Metriken, manuellen Handlungen und enge Zusammenarbeit mit Qualitätsingeneuren, Entwicklern und Managern basiert, da automatisierte Verfahren oft nicht ausreichend seien. Nachdem die Qualitätsziele definiert wurden, schreibt der Qualitätsingeneur in regelmäßigen Abständen einen Qualitätsbericht, welcher eine Übersicht über den aktuellen Qualitätsstatus gibt und die Richtung beschreibt, in welche die Qualität sich seit dem letzten Bericht entwickelt hat. Basierend darauf werden Aufgaben zur Refaktorisierung erstellt, die an das Management und die Entwickler weitergegeben werden. Steidl et al. [65] haben die folgenden vier Kriterien verwendet, um Metriken zu erzeugen:

- Klone: Es sollte möglichst wenig doppelter Code vorkommen.
- Dateilänge: Die Datei sollte eine definierte Größe nicht überschreiten.
- Methodenlänge: Eine Methode sollte eine definierte Zahl an Codezeilen nicht überschreiten.
- Verschachtelungstiefe: Eine bestimmte Tiefe der Verschachtelung von Ausdrücken sollte nicht. überschritten werden

Zwar ist das Ziel dieses Artikels, fortlaufend die Qualität des Codes in Bezug auf Wartbarkeit zu sichern, jedoch lassen sich daraus auch für diese Ausarbeitung, die einmalig zwei Implementierungen betrachtet, Vorgehensweisen ableiten. So ist es sinnvoll, zunächst Metriken zur Prüfung der Wartbarkeit zu definieren. Diese beschreiben, was im Code gemessen wird und in welcher Wertspanne die Ergebnisse liegen müssen, damit der Code im Sinne der Wartbarkeit als positiv bewertet werden kann. Da die Kriterien Klone, Dateilänge, Methodenlänge und Verschachtelungstiefe sich den Unterkategorien des ISO/IEC 25010 Qualitätsmerkmals Wartbarkeit zuordnen lassen, werden diese zur Bewertung des Codes in dieser Ausarbeitung verwendet. In Kapitel 5 werden die Metriken formuliert.

3 Grundlagen

In diesem Abschnitt werden Themen eingeführt, die die Grundlagen dieser Ausarbeitung bilden. Das Unterkapitel 3.1 erläutert, was Slack und Microsoft Teams beinhalten, da diese die Implementierung der Bots ermöglichen. In Kapitel 3.2 wird erklärt, was unter dem Begriff *Bot* verstanden wird. Denn da in dieser Ausabreitung Bots implementiert werden, sollte zuerst abgregrenzt werden, was unter Bots zu verstehen ist. Zum Schluss wird im Unterkapitel 3.3 ausgeführt, was ISO/IEC 25010 ist, da dieses als Grundlage zum Vergleich der implementierten Bots genutzt wird.

3.1 Slack und Microsoft Teams

Slack und Microsoft Teams sind beides Kollaborationssoftwares. Darunter werden Anwendungsprogramme verstanden, deren Zweck es ist, Menschen bei der Arbeit an gemeinsamen Aufgaben und dem Erreichen von Zielen zu unterstützen [16]. Beide bieten für den Anwender eine Schnittstelle an, mit welcher Bots programmiert werden können, die auf die Daten der Kollaborationssoftware zugreifen [53] [38]. Mit einem Marktanteil von 23,7 % bei Microsoft und 5,1 % bei Slack im Jahre 2021 [62] sind beide Programme relevante Leistungsträger im Bereich von Kollaborationssoftwares. Hier ist allerdings zu beachten, dass die 23,7 % sich nicht allein auf Microsoft Teams, sondern auf alle Produkte von Microsoft beziehen. Der Anteil, den Microsoft Teams ausmacht, ist der Quelle nicht zu entnehmen. Wie in Abbildung 1 zu sehen ist, gibt es noch vier weitere Programme, die einen Marktanteil von mindestens 3 % haben: Zoom, Cisco, Google und LogMein. Daraus ist abzuleiten, dass Slack und Microsoft zu den fünf Marktführern gehören und miteinander konkurrieren. Dies macht sie zu einer geeigneten Wahl für diese Ausarbeitung, um sie in Bezug auf die Bot Implementierung miteinander zu vergleichen.

3.2 Bots

Da in dieser Arbeit Bots Gegenstand der Diskussion sind, gilt es zuerst zu klären, was ein Bot ist. Die Authoren Lebeuf et Al. [17] geben in ihrer erarbeiteten Taxonomie eine ausführliche Definition zu dem, was als Bot verstanden wird: Das Wort *Bot* leitet sich aus dem Englischen *robot* (dt. Roboter) ab und wurde in den frühen 1970ern eingeführt.

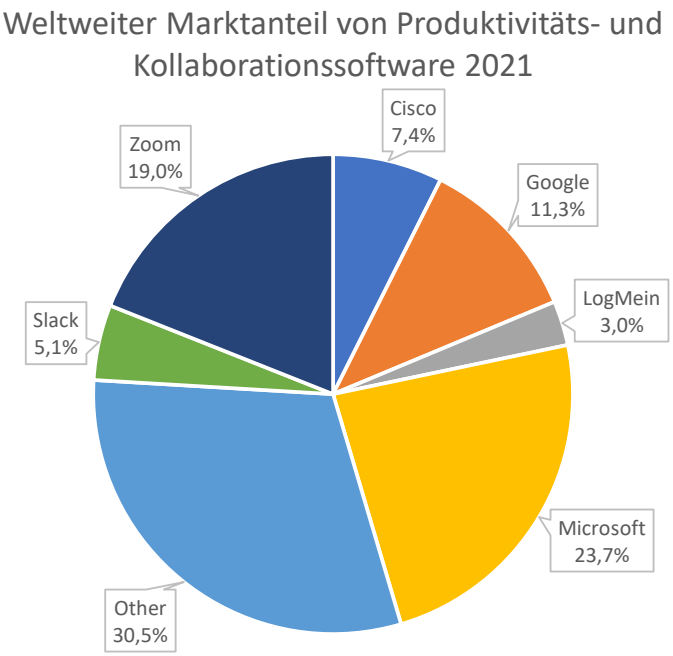

Abbildung 1: Weltweiter Marktanteil von Produktivitäts- und Kollaborationssoftwares nach Statista [62]

Über die Dekaden wurden Bots komplexer und erfüllten verschiedene Arten von Aufgaben, wodurch zahlreiche unterschiedliche Definitionen entstanden. So definieren beispielsweise Storey und Zagalsky [66] einen Bot unter anderem dadurch, dass er Aufgaben automatisch erledigen kann. Zantal-Wiener [74] hingegen schreibt in ihrem Blog, dass ein Bot selbstständig, also ohne Zutun eines Menschen, handeln können muss. Maus [19] wiederum setzt voraus, dass ein Bot ein menschenähnliches Verhalten aufweisen muss. Im Duden wird ein Bot als ein "weitgehend automatisiertes [Schad]programm"[7] bezeichnet. Unter Berücksichtigung dieser Vorgeschichte haben Lebeuf et Al. die folgende Definition erarbeitet: Bots verbinden Nutzer mit Software Services [17]. Dabei können die Nutzer Menschen, Programme, Systeme oder andere Bots sein. Diese Beschreibung trifft auf die Bots zu, die in dieser Arbeit untersucht werden, da sie den Nutzern Informationen liefern, die sie von anderen Diensten anfordern. Des Weiteren bieten sie eigene Dienste an, wie das Ausführen von repetetiven Aufgaben. Die Nutzer dieser Bots sind ausschließlich Menschen.

3.3 ISO/IEC 25010

ISO steht für *International Organization for Standardization*. Dieses Unternehmen entwickelt internationale Normen [15]. In Zusammenarbeit mit der Organisation IEC (International Electronical Commission) hat es Standards für die Softwareentwicklung erarbeitet, darunter das Dokument ISO/IEC 25010. In diesem werden insgesamt dreizehn Charakteristika definiert, die für jedes Software Produkt relevant sind. Mit ihrer Hilfe kann die Quailität einer Software spezifiziert, gemssen und evaluiert werden [14] S. 1.

In dieser Ausarbeitung wird der ISO/IEC 25010 Standard genutzt, um die implementierten Slack und Microsoft Teams Bots anhand ihrer Qualität zu bewerten. Die Charakteristika, mit denen diese gemessen werden sollen, sind *Wartbarkeit* und *Effizienz* [14] S. 10. Es wurden diese zwei ausgewählt, da sie aus Sicht der Stakeholder relevant sind. Die Stakeholder sind zum einen die Nutzer der Bots und zum anderen die Wartungsbeauftragten. Die Nutzer sollen den Bot effizient verwenden können, damit sie keine Abneigung gegen zukünftige Nutzungen entwickeln, sondern den Wunsch haben, den Nutzen des Bots voll auszuschöpfen. Je besser die Effizienz ist, desto eher trifft letzteres zu. Die Wartungsbeauftragten haben in erster Linie ein Interesse daran, dass die Bots leicht zu warten sind, wodurch der Aufwand gering gehalten wird. Folglich ist das Merkmal der Wartbarkeit hoch priorisiert.

4 Konzept

In diesem Kapitel werden alle Anforderungen an die Bots sowie die Strategien zur Implementierung beschrieben. Zuerst werden im Rahmen der Anforderungen die Stakeholder, der Systemkontext und die Anwendungsfälle benannt, die für die Bot Implementierung relevant sind. Im nächsten Schritt wird das zu implementierende Programm in Bezug auf seine Architektur und Verteilung definiert. Zum Schluss wird die Funktionsweise des Programms geschrieben.

Die Implementierung, die für diese Ausarbeitung angefertig wurde, ist auf der Plattform *GitLab* abgelegt. Der Link dahin ist im Anhang in der Tabelle 20 an Position 9 zu finden.

4.1 Anforderungen

In diesem Abschnitt werden die Anforderungen benannt, die an die Implementierungen des Slack und Microsoft Teams Bots gestellt werden. Zuerst werden die Stakeholder und ihre Erwartungen an das Programm genannt. Anschließend wird der Systemkontext beschrieben, der vorgibt, mit welchen anderen Komponenten die Bots interagieren müssen. Zum Schluss werden die Anwendungsfälle genannt, die die beiden Bots bearbeiten können müssen.

4.1.1 Stakeholder

"Ein Stakeholder ist eine Person, eine Gruppe von Personen oder eine Organisation, die ein direktes oder indirektes Interesse (Stake) am zu entwickelnden System hat oder seine Entwicklung beeinflusst“ [3] S. 207. In dieser Ausarbeitung sind die wichtigsten Stakeholder des Systems die Mitarbeiter des Unternehmens CAS AG [4]. Diese wurden in einer Umfrage gebeten, Aufgaben für einen Bot zu nennen, die sie im Arbeitsalltag als nützlich empfinden würden. Diese Aufgaben werden in Kapitel 4.1.3 beschrieben. Als Ersteller der Anforderungen nehmen die Mitarbeiter der CAS AG die Rolle des Nutzers ein, da sie den Bot benutzen. Somit erwarten sie, dass sie jeden Anwednungsfall, der in dieser Ausarbeitung beschrieben wurde, ausführen können. Da der Bot aber auch von der CAS AG betrieben wird, gibt es auch den Stakeholder aus der Wartungsperspektive. Somit gibt es hier die Erwartungen, dass das Programm wartbar ist. Eine weitere Seite,

von welcher aus das System betrachtet wird, ist die finanzielle, da jedes gewinnorientierte Unternehmen daran interessiert ist, die beste Leistung mit den geringsten Kosten zu erzielen. Folglich lassen sich die Mitarbeiter der CAS AG in drei wesentliche Stakeholder unterteilen: Nutzer, Wartungsbeauftragter und Finanzverwalter.

4.1.2 Systemkontext

Um die nötigen Schnittstellen zu ermitteln, die erforderlich sind, um das System zu implementieren, wird in diesem Abschnitt ein fachlicher und ein technischer Systemkontext entworfen. Nach Broy und Kuhrmann [3] S. 46 beschreibt der Systemkontext die Grenzen zwischen einem System und seiner Umwelt. Der Teil der Umwelt, der für das Verhalten des Systems relevant ist, wird als Kontext bezeichnet. Dazu zählen unter anderem Systeme, Nutzer und Prozesse. Da die Bots in der Umgebung des Unternehmens der CAS AG betrieben werden, gibt diese den Systemkontext vor.

Fachliche Schnittstellen

Der fachliche Systemkontext beschreibt, welche Schnittstellen aus Anwendungssicht relevant für das zu entwickelnde System sind [3] S. 46. Für das System, das in dieser Ausarbeitung entwickelt wird, sind die folgenden Schnittstellen vorzufinden:

- Nutzer: Stellt Anfragen mittels des Slack / Microsoft Teams Clients an den Bot.
- Slack / Microsoft Teams Client: Grafische Oberfläche für den Nutzer und Anwednung, die mit den Servern von Slack / Microsoft Teams kommuniziert.
- Meisterplan: Programm zum Koordinieren von Mitarbeitern über Abteilungen und Projekte hinweg [20]. Dieses nutzt der Bot, um verfügbare Zeiten von Mitarbeitern zu ermitteln.
- Microsoft Outlook: Email und Kalenderprogramm [30]. Dieses benutzt der Bot, um freie Termine von Mitarbeitern zu finden.

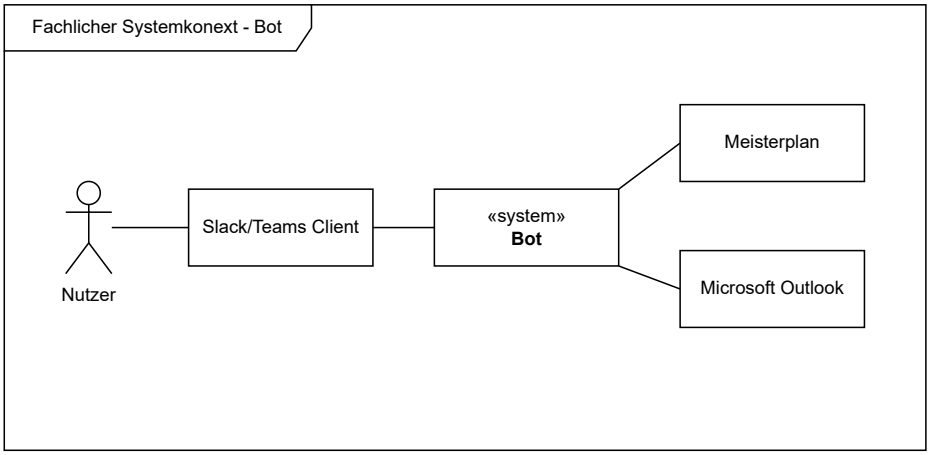

Abbildung 2: Fachlicher Kontext des Bots

Zusammengefasst muss das System mit den Slack / Microsoft Teams Clients, Meisterplan und Microsoft Outlook kommunizieren, um seine Aufgaben aus fachlicher Sicht zu erfüllen.

Technische Schnittstellen

Der technische Systemkontext beschreibt, welche Schnittstellen aus Betriebssicht relevant für das zu entwickelnde System sind [3] S 46. Für das System, das in dieser Ausarbeitung entwickelt wird, sind die folgenden Schnittstellen vorzufinden:

- Slack / Microsoft Teams Bibliotheken: Schnitstellen, die das Programmieren eines Bots ermöglichen, angeboten oder empfohlen von Slack / Microsoft Teams.
- JavaScript https Modul [44]: wird benötigt, um Informationen von den Quellsystemen zu beschaffen.
- NodeJs: Laufzeitumgenung für JavaScript [47].
- Slack / Microsoft Teams Application: Konfigurations-Applikation des Bots. Hier wird der Bot zunächst als App registriert. Anschließend können Voreinstellungen gemacht werden, wie beispielsweise Zugriffsberechtigungen auf Nachrichten. Die Einstellungen für einen Slack Bot können unter der Slack-Konfigurations-URL vorgenommen werden, die im Anhang in der Tabelle 20 in Position eins gefunden

werden kann. Das gleiche kann für den Microsoft Teams Bot unter der Teams-Konfigurations-URL vorgenommen werden, die im Anhang in Tabelle 20 in Position zwei gefunden werden kann.

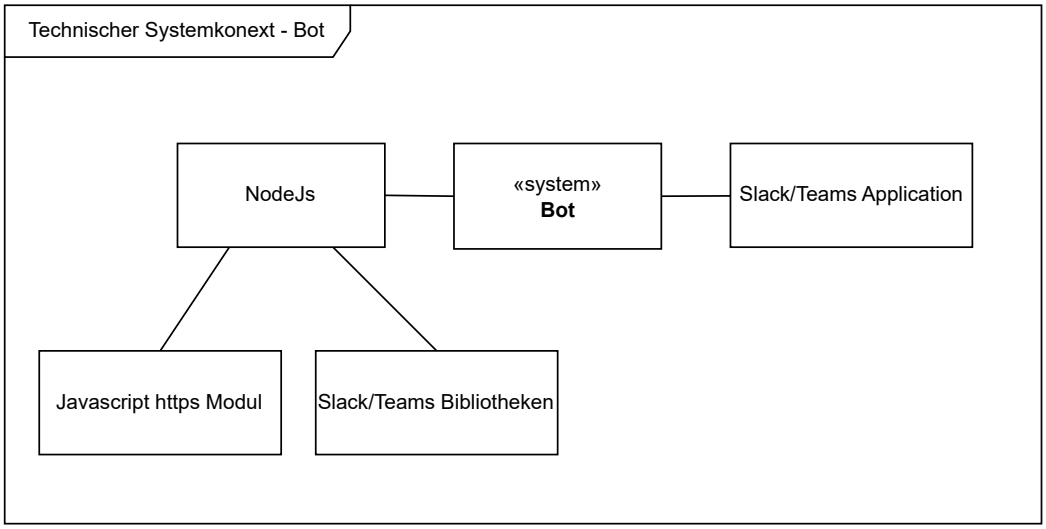

Abbildung 3: Technischer Kontext des Bots

Zusammengefasst muss das System in der Lage sein, die Slack / Microsoft Teams Bibliotheken, das JavaScript https Modul sowie die Programmierumgebung NodeJs zu nutzen. Des Weiteren muss der Bot mit der entsprechenden Konfigurations-Applikation kommunizieren können.

4.1.3 Anwendungsfälle

In dieser Arbeit werden einige ausgewählte Aufgaben festgelegt, die der Bot abdecken soll, die von den Mitarbeitern der CAS AG gewünscht wurden. Die Anwendungsfälle UC-01 bis UC-03 bilden diese ab. Des Weiteren sind die Anwendungsfall UC-04 (Daten beschaffen) und UC-05 (Daten aufbereiten) nötig, um die Aufgaben umzusetzen. Abbildung 4 veranschaulicht dies in einem Anwendungsfall Diagramm. Tabelle 1 beschreibt die Anwendungsfälle näher.

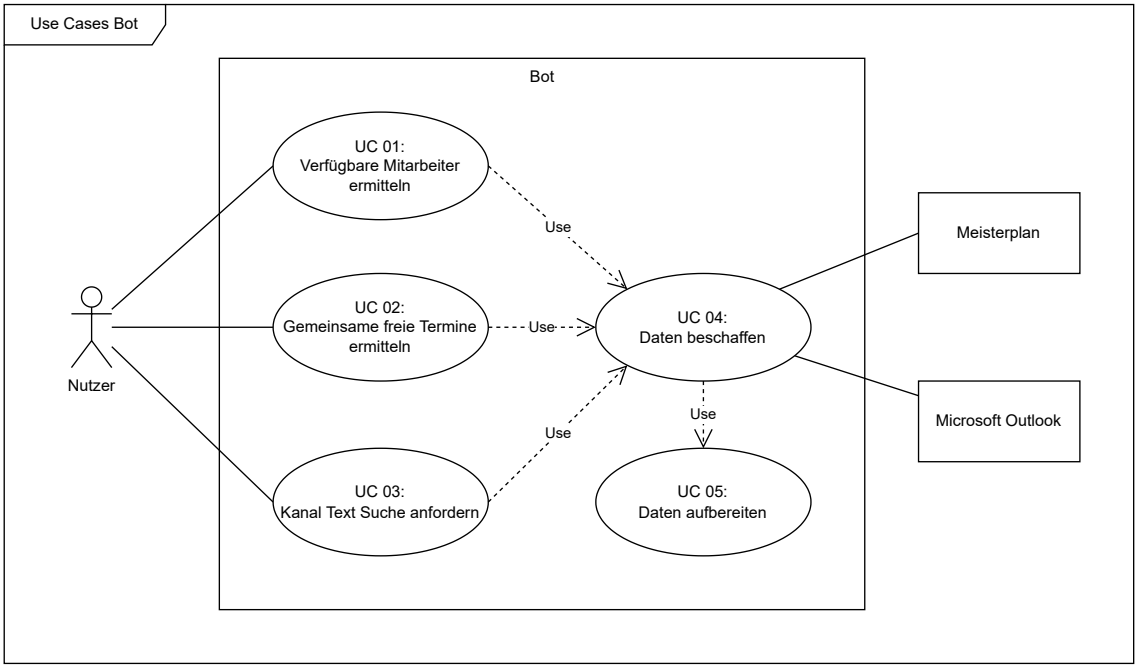

Abbildung 4: Anwendungsfälle des Bots

UC Id	Titel	Beschreibung
UC-01	Verfügbare Mitarbeiter ermitteln	In diesem Fall soll der Bot dem Nutzer sagen, welche Mitarbeiter eine bestimmte Mindestanzahl an Tagen zur Verfügung haben an denen sie noch in keinen Projekten eingeplant sind. Diese Informationen erhält der Bot von der Plattform Meisterplan, wo die Mitarbeiter ihre geplanten Aufwände je Projekt pflegen. Beispiel: Welche Mitarbeiter haben Erfahrung mit Javascript und im Mai noch mehr als 5 Tage freie Zeit
UC-02	Gemeinsame freie Termine ermitteln	In diesem Fall soll der Bot die Kalender von genannten Mitarbeitern analysieren und Zeiträume finden, an denen alle Zeit haben. Beispiel: Sag mir 3 Termine je 2 Stunden an denen Frank Zanker, Andreas Schwarz und Laura Jurgeneit Zeit haben.
UC-03	Kanal Text Suche anfordern	Der Bot durchsucht alle Slack bzw. Microsoft Teams Kanäle darauf, ob ein Thema schon einmal aufgekommen ist. Alle Funde werden als Antwort gegeben. Beispiel Anfrage: Wurde das Thema Kubernetes schon einmal besprochen
UC-04	Daten beschaffen	Aus den vom Nutzer gestellten Anfragen muss ermittelt werden können, woher die Daten zu beschaffen sind, um eine Antwort zu erzeugen. Dieser Anwendungsfall wird zur Realisierung aller vorherigen Anwendungsfälle benötigt.
UC-05	Daten aufbereiten	Die Daten, die aufgrund der Anfrage beschafft wurden, müssen in einem Format aufbereitet werden, dass ein Mensch sie ohne Umstände verstehen kann. Beispielsweise wird ein JSON Objekt in einen natürlich sprachlichen Satz umgewandelt. Dieser Anwendungsfall wird zur Realisierung der Anwendungsfälle uC-01 bis UC-03 benötigt.

Tabelle 1: Liste der Anwendungsfälle

4.2 Beschreibung der Software

In diesem Abschnitt wird die Architektur sowie die Verteilung des Systems beschrieben, die die Implementierung der Bots widerspiegeln. Im Architektur Abschnitt werden die einzelnen Bausteine erläutert, aus denen das System aufgebaut ist. Bei der Verteilung wird gezeigt, wie das System physisch oder virtuell im Netzwerk verteilt ist.

4.2.1 Architektur

Das System, das in dieser Ausarbeitung entwickelt wird, besteht auf oberster aus zwei Bausteinen:

- Bot Logik.
- Business Logik.

Die Abblidung 5 veranschaulicht dies. In Tabelle 2 werden die einzelnen Komponenten und ihre Aufgaben beschrieben. Dies ist die grob granularste Struktur-Ebene der Architektur L0 (Level 0). Im nächsten Schritt werden die Bausteine auf Ebene L1 detaillierter beschrieben.

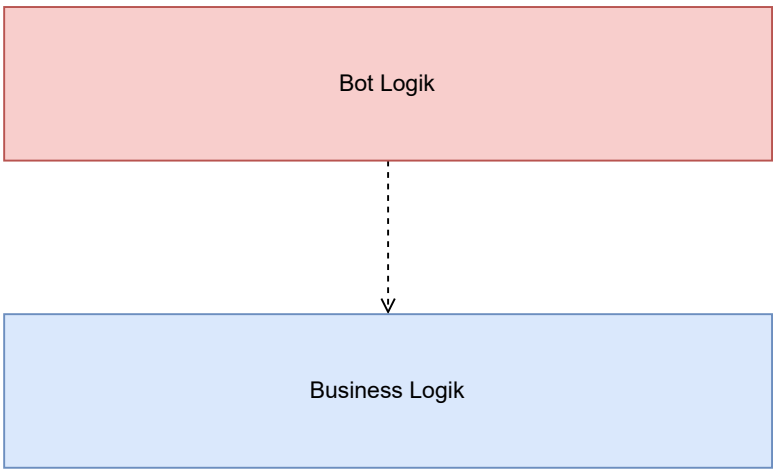

Abbildung 5: Bausteindiagramm L0 des Bot Systems

Komponente	Becshreibung
Bot Logik	Hier ist die Bot Schnittstelle enthalten. In dieser Komponente wird der Bot initialisiert und seine verfügbaren Funktionalitäten werden genutzt, wie beispielsweise das Horchen auf Nachrichten von Nutzern im Slack / Microsoft Teams Client.
Business Logik	Hier werden die Daten, die durch die Bot Komponente weitergegeben wurden, analysiert. Es wird entschieden, welche Aktionen getätigt werden. Beispielsweise wird hier die Nachricht eines Nutzers analysiert und es wird darauf entschieden, welche Schritte gemacht werden müssen, um diese zu beantworten.

Tabelle 2: Beschreibung der Komponenten des Bausteindiagramms Level 0

In Anbetracht der Anforderungen an das System, können die L0 Komponenten detaillierter beschrieben werden. So muss die Komponente *Bot Logik* sowohl den Bot initialisieren und Listeners starten, welche auf vordefinierte Ereignisse horchen. In der *Geschäftslogik Logik* Komponente werden Daten beschafft und verarbeitet, abhängig von dem, was aus der *Bot Logik* Komponente weitergegeben wurde. Die folgende Abbildung 6 veranschaulicht dies. Die Tabelle 3 beschreibt die Komponenten, die zur vorherigen L0 Ansicht hinzugekommen sind.

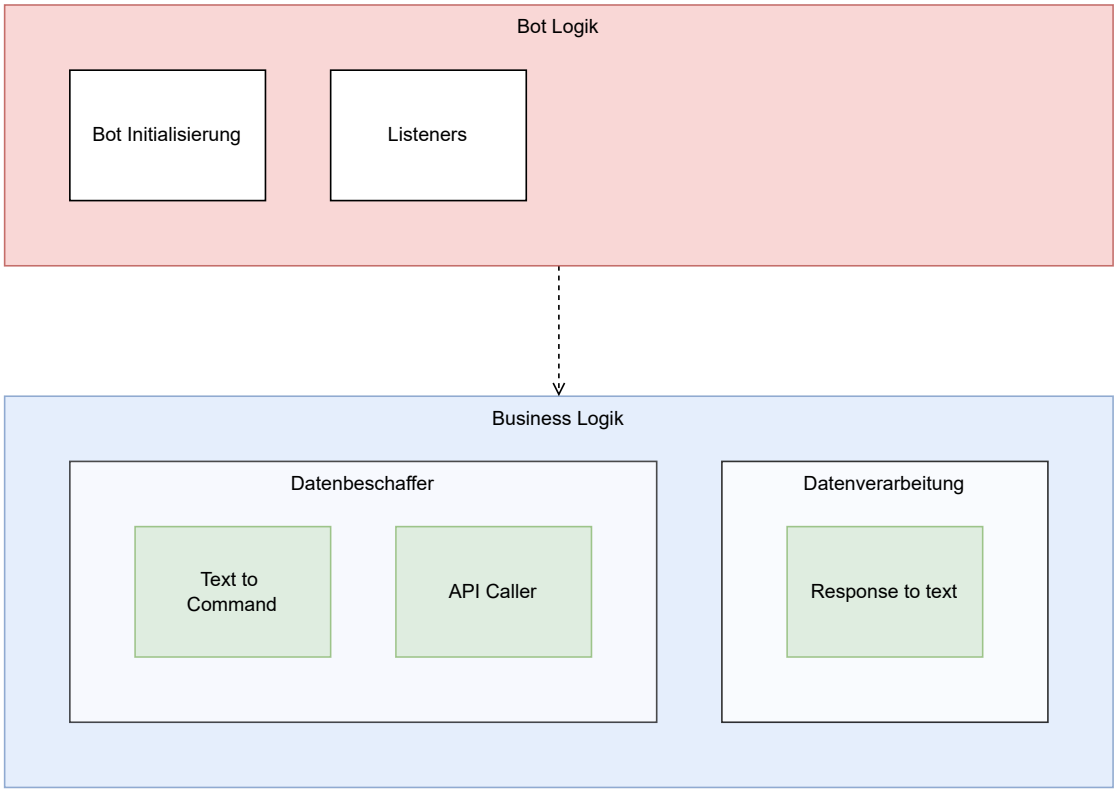

Abbildung 6: Bausteindiagramm L1 des Bot Systems

Komponente	Beschreibung
Bot Initialisierung	Hier wird der Bot initialisiert und gestartet.
Listeners	Hier sind alle Listeners enthalten. Listeners sind Komponenten, die auf bestimmte Ereignisse reagieren. Bots können beispielsweise auf das Ereignis einer eintreffenden Nachricht reagieren. Erfüllen die erhaltenen Daten bestimmte Kriterien, dann werden diese weiter an die Text to Command Komponente weitergeleitet.
Text to Command	Diese Komponente entscheidet durch Analyse der erhaltenen Daten, welche Aktionen getätigt werden. Entweder wird mit den Analyseergebnissen eine Funktion im API Caller aufgerufen oder es wird ein Antowrttext formuliert, der sich auf die erhaltenen Daten bezieht.
API Caller	Diese Komponente sendet HTTPS Anfragen an entfernte Systeme.
Response to text	Diese Komponente bereitet Daten in der Form auf, dass sie an den Slack / Microsoft Teams Client zurück gegeben werden können und dem Nutzer als leserliche Antwort dienen.

Tabelle 3: Beschreibung der Komponenten des Bausteindiagramms Level 1

4.2.2 Verteilungsdiagramm

Ein Verteilungsdiagramm beschreibt, wie einzelne Komponenten eines Systems auf physische oder virtuelle Knoten eines Netzwerkes verteilt sind [3] S. 136. In diesem Abschnitt wird die Verteilung des Systems, das in dieser Ausarbeitung implementiert wird, in Tabelle 4 beschrieben und in Abbildung 7 visualisiert. Dies dient zur Übersicht der Beziehungen zwischen allen Komponenten, die zur Betreibung des Systems notwendig sind. Des Weiteren zeigt es auf, welche Software auf welchen Komponenten vorhanden sein muss.

Knoten	Beschreibung
Server	Dies ist die Hardware Einheit, die das Slack / Microsoft Teams Bot System betreibt. Dieses benötigt eine NodeJs Umgebung sowie Komponenten zur Kommunikation mit den Slack / Microsoft Teams Clients, zur Verabeitung der erhaltenen Daten von den Slack / Microsoft Teams Clients sowie zur Kommunikation über REST API mit den Quellsystemen.
Nutzer Computer	Dies ist die Maschine, auf welcher der Slack / Microsoft Teams Client des Nutzers ausgeführt wird.
Quellsysteme	Diese beinhalten Meisterplan und Microsoft Outlook. Jeder von ihnen bietet eine REST API Schnittstelle an, um Daten auszutauschen.
Slack / Microsoft Teams Application	Diese von Slack / Microsoft Teams bereitgestellte Software registriert und verwaltet die Bots.

Tabelle 4: Komponenten im Verteilungsdiagramm

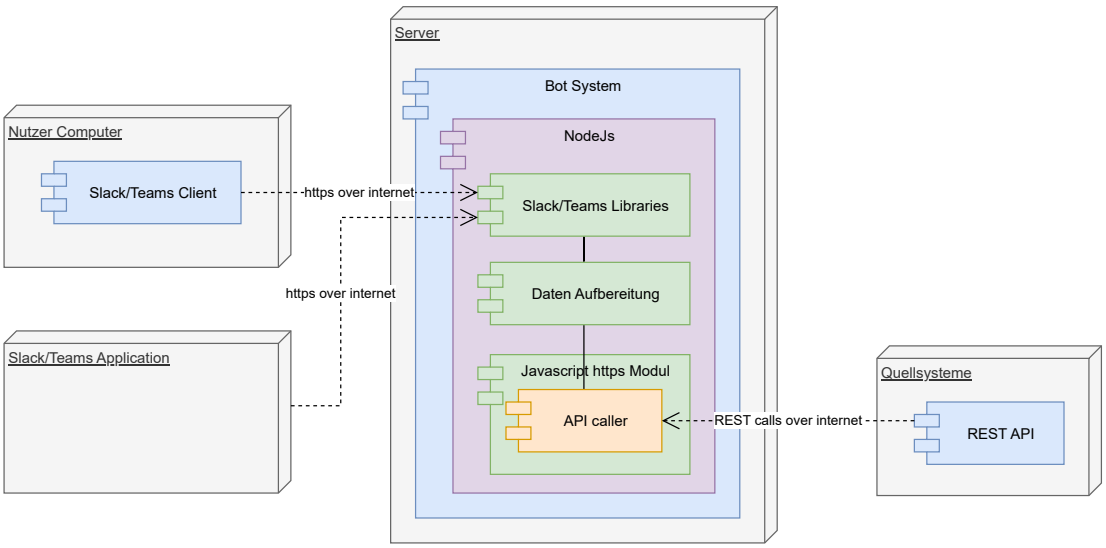

Abbildung 7: Verteilungsdiagramm zur Betreibung des Bots

Zusammengefasst gibt es fünf Komponenten, die zur Betreibung des Systems nötig sind: Server, Nutzer Computer, Quellsyteme und Slack / Microsoft Teams Application. Der Server muss die gesamte Software der Bot Implementierungen enthalten, der Nutzer Computer muss den Slack / Microsoft Teams Client installiert haben und die Quellsysteme müssen eine REST API Schnittstelle anbieten. Die Slack / Microsoft Teams Application ist nötig, um die Bots zu verwalten. Bespielsweise kann damit bestimmt werden, welche

Zugriffsberechtigungen ein Bot hat.

4.3 Funktionsweise

In diesem Abschnitt werden alle fünf Anwendungsfälle in ihrer Funktionsweise beschrieben. Sofern Textbefehle an den Bot beschrieben werden, sind folgende Notationen zu beachten:

- Platzhalter: Spitze Klammern (<...>) dienen als Platzhalter Variablen.
- Optionaler Text: Eckige Klammern ([...]) umschließen Text, der optional ist. Das Hinzufügen oder Fehlen dieses Textes hat keine Auswirkung auf die Anfrage.
- Veroderung: Zwei Pipes (||) innerhalb von eckigen oder runden Klammern trennen verwendbare Texte voneinander, wenn es mehr als eine Text-Option gibt. Es darf nur einer der Texte verwendet werden.
- Erforderlicher Text: Runde Klammern umschließen Text, der in der Anfrage enthalten sein muss. Diese werden genutzt, wenn es mehere Optionen für den erforderlichen Text gibt. Innerhalb dieser runden Klammern werden die Optionen mit den zwei Pipes (||) verodert.
- Fester Text: Texte, die nicht von Klammern umgeben sind, müssen in dieser Form in der Anfrage vorkommen und dürfen somit nicht verändert werden, abgesehen von der Groß- und Kleinschreibung.
- Groß- und Kleinschreibung ist irrelevant.

4.3.1 UC-01 - Verfügbare Mitarbeiter ermitteln

In diesem Fall möchte der Nutzer wissen, welche Mitarbeiter mit bestimmten Fähigkeiten in vorgegeben Monaten eine bestimmte Anzahl an Tagen nicht verplant haben. Die verfügbare Zeit und die Fähigkeiten werden über eine HTTPS Anfrage an die Plattform Meisterplan ermittelt.

Beschreibung der Anfrage
Der Nutzer aktiviert diesen Fall, indem er dem Bot in einer direkten Nachricht im folgenden Format schickt:

> Welche Mitarbeiter haben Erfahrung mit <FÄHIGKEITEN> und im <MONATE> [noch mehr als] <ZAHL> Tage frei[?]

Die Variablen sind wie folgt zu ersetzen:

- <FÄHIGKEITEN>: Gesuchte Fähigkeiten bei Mitarbeitern. Die Eingabe ist Komma getrennt. Anstelle des letzten Kommas kann ein "*und*" geschrieben werden. Wird nur eine Fähigkeit gesucht, dann wird kein Komma oder "*und*" geschrieben.
- <MONATE>: Namen der Monate, in denen die gesuchten Mitarbeiter freie Zeit haben sollen. Die Eingabe ist durch ein Komma getrennt. Anstelle des letzten Kommas kann ein "*und*" geschrieben werden. Wird nur ein Monat gesucht, dann wird kein Komma oder "*und*" geschrieben.
- <ZAHL>: Zahl der freien Tage, die die gesuchten Mitarbeiter mindestens zur Verfügung haben müssen.

Beispiel Anfragen
Kurze Anfrage: Es wird ein Mitarbeiter mit **einer Fähigkeit** gesucht, der in **einem Monat** verfügbar ist:

(1) Welche Mitarbeiter haben Erfahrung mit **Javascript** und im **Mai 5** Tage frei

Anfrage mit **zwei gesuchten Fähigkeiten**:

(2) Welche Mitarbeiter haben Erfahrung mit **Python und Javascript** und im **Mai 5** Tage frei

Komplexe Anfrage: Es werden Mitarbeiter mit **drei Fähigkeiten** gesucht für einen Zeitraum von **zwei Monaten** mit **mindestens** 8 verfügbaren Tagen:

(3) Welche Mitarbeiter haben Erfahrung mit **Python, Java und Javascript** und im **Mai und Juni** noch mehr als **7** Tage frei

Datenquelle

In diesem Anwendungsfall wird die externe Plattform *Meisterplan* [20] als Datenquelle verwendet. Meisterplan ist eine Software, um "Mitarbeiter über Abteilungen und Projekte hinweg" [20] zu koordinieren. Dies bedeutet, dass die Mitarbeiter dort dokumentieren, wann und für wie lange sie in welchen Projekten eingeplant sind. Des Weiteren bietet Meisterplan zwei REST APIs an, mit denen es möglich ist, diese geplanten Arbeitszeiten für jeden Mitarbetier zu lesen. Bei diesen zwei handelt es sich um die *Meisterplan API* [21] und die *Meisterplan Reporting API* [22]. Sowohl der Microsoft Teams als auch der Slack Bot beschaffen die gesuchten Daten über die gleichen REST Anfragen an die Plattform Meisterplan.

Verhalten zur Laufzeit

Im Abbildung 8 wird das simple Verhalten des Bots zur Laufzeit beschrieben, wenn ein Nutzer eine Nachricht im Slack oder Microsoft Teams Client versendet, die dem zuvor beschriebenen Anfrageformat entspricht. Das bedeutet, dass der Bot eine Anfrage vom Nutzer erhält, die entsprechenden Daten zur Beantowrtung der Anfrage von der Plattform Meisterplan beschafft und daraus eine Antwort konstruiert, um sie zurück an den Nutzer zu senden. Es werden dort keine Sonderfälle oder detaillierte Verarbeitungsprozesse abgebildet, um das Diagramm übersichtlich zu gestalten und auf das wesentliche zu fokussieren.

Es beginnt damit, dass der Nutzer, der in dem Sequenzdiagramm als *Actor* bezeichnet wird, einen Text im *Client* schreibt und abschickt. Der *Client* leitet diesen Text weiter an den Server auf welchem der Bot betrieben wird. Dort wird im *Bot Framework* das *onMessage* Ereignis ausgelöst. Dies ist ein Ereignis, das immer ausgelöst wird, wenn in einem Kanal, den der Bot berechtigt ist zu lesen, eine Nachricht gesendet wird. Diese erhaltene Nachricht wird an die *Response Creator* Komponente weitergeleitet, welche analysiert, ob die Nachricht dem Anfrageformat des ersten Anwendungsfalls entspricht. Ist dies der Fall, dann übergibt der *Response Creator* die relevanten Informationen, die aus dem Text analyisiert wurden und benötigt werden, um die Daten zur Beantwortung der Frage zu ermitteln, an den *Data Requester*. Dieser konstruiert aus den gegebenen Daten eine URL, die an *Meisterplan* geschickt wird. *Meisterplan* beantwortet diese URL Anfrage und somit hat der *Data Requester* nun die Daten, die die Antwort auf die Frage des Nutzers enthalten. Da die Daten von *Meisterplan* in einem bestimmten Format zurück kommen, normiert der *Data Requester* diese, bevor es diese an den *Response Creator*

weiter gibt. Der *Response Creator* erzeugt aus diesen Daten eine Antwort in einer Struktur, die der *Client* der entsprechenden Plattform verarbeiten kann. Diese Antwort wird dann über das *Bot Framework* zurück an den *Client* weiter geleitet. Nun sieht der Nutzer eine Antwort auf seine Frage im *Client*.

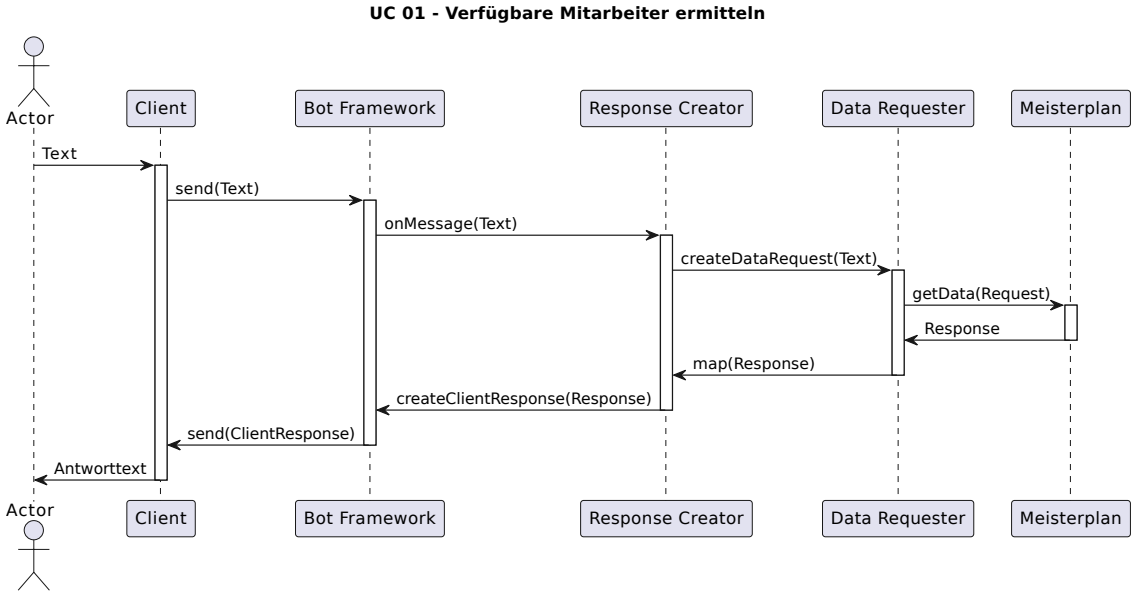

Abbildung 8: Sequenzdiagramm für Anwednungsfall UC-01

Beschreibung der Antwort

Der Bot antwortet in der selben Direktnachricht auf die Anfrage. In der Antwort sind alle gefundenen Mitarbeiter aufgelistet sowie die Anzahl der verfügbaren Tage.

4.3.2 UC-02 - Gemeinsame freie Termine ermitteln

In diesem Fall möchte der Nutzer wissen, welche Mitarbeiter in einem bestimmten Zeitraum für eine gewisse Dauer Zeit haben. Diese Informationen werden aus dem Kalender gelesen.

Beschreibung der Anfrage

Der Nutzer aktiviert diesen Fall, indem er dem Bot in einer direkten Nachricht im folgenden Format schickt:

Sag mir <TERMINANZAHL> Termin[e] (a||je) <ZEIT> (Stunde[n]||Minuten) an (denen||dem) <MITARBEITERNAMEN> Zeit (haben||hat)[.]

Die Variablen sind wie folgt zu ersetzen:

- <TERMINANZAHL>: Anzahl der Termine, an denen alle genannten Mitarbeiter teilnehmen können. Anstelle des letzten Kommas kann ein "*und*" geschrieben werden. Wird nur ein Termin gesucht, dann wird kein Komma oder "*und*" geschrieben.
- <ZEIT>: Dauer der Termine in Stunden oder Minuten.
- <MITARBEITERNAMEN>: Vor- und Nachnamen der Mitarbeiter, die an den Terminen teilnehmen sollen. Die Eingabe ist durch ein Komma zu trennen. Anstelle des letzten Kommas kann ein "*und*" geschrieben werden. Werden Termine nur mit einer Person gesucht, dann wird kein Komma oder "*und*" geschrieben.

Beispiel Anfragen

Simple Anfrage nach einem Termin mit einer Person:

(4) Sag mir **1** Termin je **2** Stunden an dem **Erika Musterfrau** Zeit hat.

Komplexere Anfrage nach vier Terminen mit zwei Personen:

(5) Sag mir **4** Termine je **1** Stunde an denen **Max Mustermann und Erika Musterfrau** Zeit haben.

Komplexere Anfrage nach zwei Terminen mit drei Personen:

(6) Sag mir **2** Termine je **30** Minuten an dem **Max Mustermann, Erika Musterfrau und Lisa Müller** Zeit haben.

Datenquelle

In diesem Anwendungsfall wird davon ausgegangen, dass im Unternehmen Microsoft Outlook als Kalender verwendet wird. Microsoft Outlook ist ein Email und Kalender Programm des Unternehmens Microsoft [30]. Dieses bietet die REST API *Microsoft Graph* [28] an, mit welcher unter anderem die Kalenderdaten je Mitarbeiter ausgelesen werden können [26]. Sowohl der Microsoft Teams als auch der Slack Bot senden die gleichen REST Anfragen an Microsoft Graph, um Kalenderdaten von Mitarbeitern zu erhalten.

Verhalten zur Laufzeit

Im Abbildung 9 wird das simple Verhalten des Bots zur Laufzeit beschrieben, wenn ein Nutzer eine Nachricht im Slack oder Microsoft Teams Client versendet, die dem zuvor beschriebenen Anfrageformat entspricht. Das bedeutet, dass der Bot eine Anfrage vom Nutzer erhält, die entsprechenden Daten zur Beantowrtung der Anfrage von der Software Outlook beschafft und daraus eine Antwort konstruiert, um sie zurück an den Nutzer zu senden. Es werden dort keine Sonderfälle oder detaillierte Verarbeitungsprozesse abgebildet, um das Diagramm übersichtlich zu gestalten und auf das wesentliche zu fokussieren.

Es beginnt damit, dass der Nutzer, der in dem Sequenzdiagramm als *Actor* bezeichnet wird, einen Text im *Client* schreibt und abschickt. Der *Client* leitet diesen Text weiter an den Server auf welchem der Bot betrieben wird. Dort wird im *Bot Framework* das *onMessage* Ereignis ausgelöst. Dies ist ein Ereignis, das immer ausgelöst wird, wenn in einem Kanal, den der Bot berechtigt ist zu lesen, eine Nachricht gesendet wird. Diese erhaltene Nachricht wird an die *Response Creator* Komponente weitergeleitet, welche analysiert, ob die Nachricht dem Anfrageformat des ersten Anwendungsfalls entspricht. Ist dies der Fall, dann übergibt der *Response Creator* die relevanten Informationen, die aus dem Text analyisiert wurden und benötigt werden, um die Daten zur Beantwortung der Frage zu ermitteln, an den *Data Requester*. Dieser konstruiert aus den gegebenen Daten eine URL, die an *Outlook* geschickt wird. *Outlook* beantwortet diese URL Anfrage und somit hat der *Data Requester* nun die Daten, die die Antwort auf die Frage des Nutzers enthalten. Da die Daten von *Outlook* in einem bestimmten Format zurück kommen, normiert der *Data Requester* diese, bevor es diese an den *Response Creator* weiter gibt. Der *Response Creator* erzeugt aus diesen Daten eine Antwort in einer Struktur, die der *Client* der entsprechenden Plattform verarbeiten kann. Diese Antwort wird dann über das *Bot Framework* zurück an den *Client* weiter geleitet. Nun sieht der Nutzer eine Antwort auf seine Frage im *Client*.

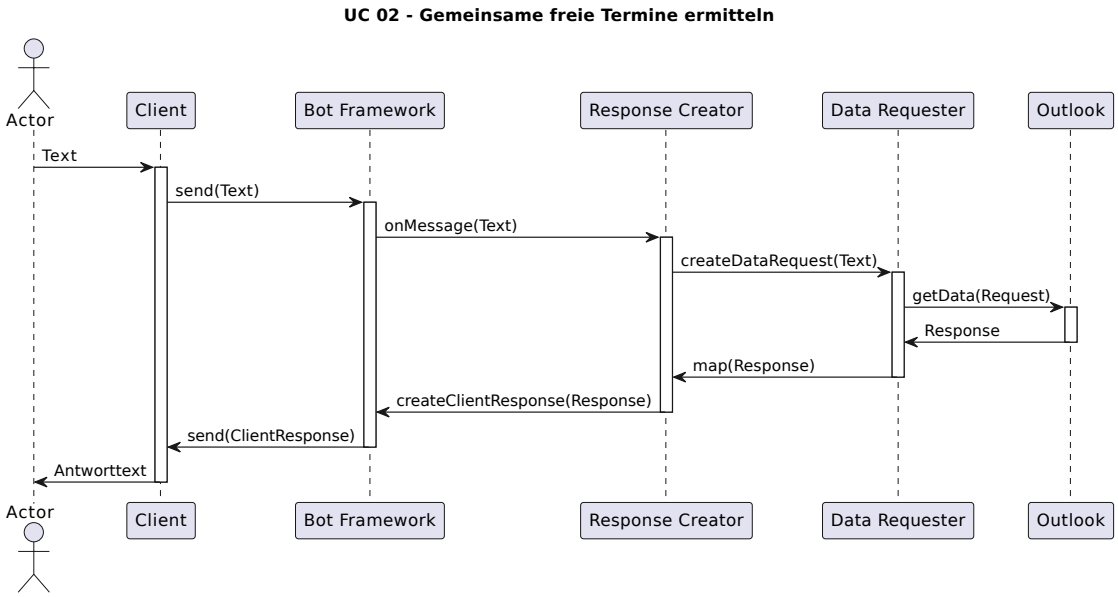

Abbildung 9: Sequenzdiagramm für Anwednungsfall UC-02

Beschreibung der Antwort
Der Bot antwortet in der selben Direktnachricht auf die Anfrage. In der Antwort werden alle gefunden Termine mit Datum und Uhrzeit aufgezählt.

4.3.3 UC-03 - Kanal Text Suche anfordern

In diesem Fall möchte der Nutzer wissen, ob andere Mitarbeiter in gemeinsamen Channeln über ein bestimmtes Thema geredet haben.

Beschreibung der Anfrage
Der Nutzer aktiviert diesen Fall, indem er dem Bot in einer direkten Nachricht im folgenden Format schickt:

(7) Wurde das Thema <THEMA> [schon einmal] besprochen

Die Variablen sind wie folgt zu ersetzen:

- <THEMA>: Name des Themas, das in alen gemeinsamen Channeln gesucht wird.

Beispiel Anfragen
Simple Anfragen für ein Thema:

(8) Wurde das Thema **Docker** schon einmal besprochen

(9) Wurde das Thema **SAP4HANA** besprochen

Datenquelle
In diesem Anwendungsfall sind Slack bzw. Microsoft Teams selber die Datenquelle. Somit sendet der Slack Bot seine Anfrage-URL an die *Web API* von Slack [60]. Der Microsoft Teams Bot sendet die Anfragen an *Microsoft Graph* [28].

Verhalten zur Laufzeit
Im Abbildung 10 wird das simple Verhalten des Bots zur Laufzeit beschrieben, wenn ein Nutzer eine Nachricht im Slack oder Microsoft Teams Client versendet, die dem zuvor beschriebenen Anfrageformat entspricht. Das bedeutet, dass der Bot eine Anfrage vom Nutzer erhält, die entsprechenden Daten zur Beantowrtung der Anfrage von Slack bzw. Microsoft Teams beschafft und daraus eine Antwort konstruiert, um sie zurück an den Nutzer zu senden. Es werden dort keine Sonderfälle oder detaillierte Verarbeitungsprozesse abgebildet, um das Diagramm übersichtlich zu gestalten und auf das wesentliche zu fokussieren.

Es beginnt damit, dass der Nutzer, der in dem Sequenzdiagramm als *Actor* bezeichnet wird, einen Text im *Client* schreibt und abschickt. Der *Client* leitet diesen Text weiter an den Server auf welchem der Bot betrieben wird. Dort wird im *Bot Framework* das *onMessage* Ereignis ausgelöst. Dies ist ein Ereignis, das immer ausgelöst wird, wenn in einem Kanal, den der Bot berechtigt ist zu lesen, eine Nachricht gesendet wird. Diese erhaltene Nachricht wird an die *Response Creator* Komponente weitergeleitet, welche analysiert, ob die Nachricht dem Anfrageformat des ersten Anwendungsfalls entspricht. Ist dies der Fall, dann übergibt der *Response Creator* die relevanten Informationen, die aus dem Text analyisiert wurden und benötigt werden, um die Daten zur Beantwortung der Frage zu ermitteln, an die *Data Requester* Komponente. Diese konstruiert aus den gegebenen Daten eine Anfrage, die an Slack bzw. Microsoft Teams geschickt wird. Die Zielplattform beantwortet diese Anfrage und somit hat der *Data Requester* nun die Daten, die die Antwort auf die Frage des Nutzers enthalten. Da die Daten von Slack bzw.

Microsoft Teams in einem bestimmten Format zurück kommen, normiert der *Data Requester* diese, bevor es diese an den *Response Creator* weiter gibt. Der *Response Creator* erzeugt aus diesen Daten eine Antwort in einer Struktur, die der *Client* der entsprechenden Plattform verarbeiten kann. Diese Antwort wird dann über das *Bot Framework* zurück an den *Client* weiter geleitet. Nun sieht der Nutzer eine Antwort auf seine Frage im *Client*.

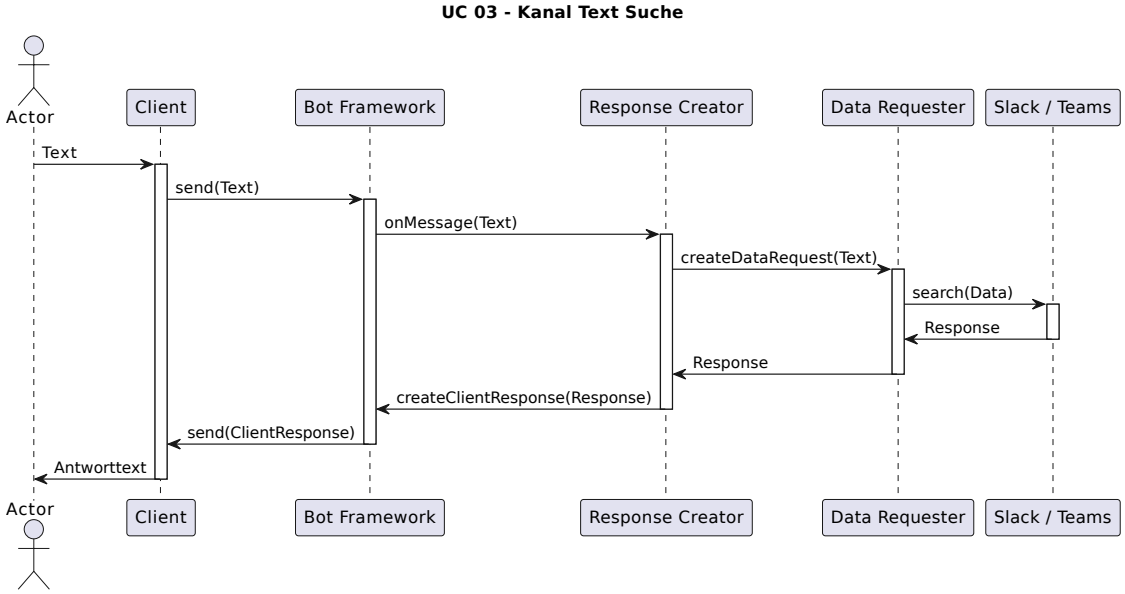

Abbildung 10: Sequenzdiagramm für Anwednungsfall UC-03

Beschreibung der Antwort
Der Bot antwortet in der selben Direktnachricht auf die Anfrage. Es werden alle gefundenen Textnachrichten aufgelistet.

4.3.4 UC-04 - Daten beschaffen

Da der Bot Informationen aus externen Systemen bezieht, die über HTTPS erreicht werden, gibt es eine Komponente im System, die für das Senden von Anfragen verantwortlich ist. Diese wurde in den vorherigen Abschnitten als *Data Requester* bezeichnet. Diese Komponete ist in der Lage, URL Anfragen an die Plattformen *Meisterplan*, *Microsoft Graph* und *Slack Web Api* zu senden.

4.3.5 UC-05 - Daten aufbereiten

Diese Komponente nimmt Daten entgegen und bereitet sie in der Form auf, dass sie an den Bot Client geschickt werden und für den Endnutzer ein verständliches Bild ergeben. Diese wurde in den vorherigen Abschnotten als *Response Creator* bezeichnet.

5 Bewertung der Implementierung

Da es sich bei Bots um Software handelt, können diese mit Hilfe von Qualitätskriterien verglichen werden, die im ISO/IEC 25010 [61] beschrieben sind. Hierzu müssen zunächst auf beiden Plattformen die Bots implementiert werden. Dabei ist es wichtig, dass die Implementierungen dem gleichen Konzept folgen, um eine vergleichabre Grundlage zu schaffen. In dieser Arbeit werden zwei Qualitätsmerkmale zur Hilfe genommen, um beide Bots miteinander zu vergleichen: Wartbarkeit und Effizienz. Im Foglenden werden diese zwei erklärt und es wird beschrieben, wie sie zur Bewertung eines Bots genutzt werden können.

5.1 Wartbarkeit

Das Qualitätsmerkmal der Wartbarkeit beschreibt den Grad an Effektivität und Effizienz, in welchem ein System modifiziert werden kann [61] S. 14. Dieses lässt sich in die fünf Unterkateogirien *Modularität*, *Wiederverwendbarkeit*, *Analysierbarkeit*, *Veränderbarkeit* und *Testbarkeit* aufteilen, die in Tabelle 5 beschrieben werden. In diesem Fall bedeutet das, dass der eigene Bot Code idealerweise modular und redundanzfrei geschrieben werden kann sowie mit wenig Aufwand zu verstehen, zu modifizieren und zu testen ist. Um diese Eigenschaften zu messen und die Ergebnisse durch Zahlen zu beschreiben, gibt es verschiedene Metriken, die angewandt werden können:

- Lines of Code (LoC).
- Zyklomatische Komplexität.
- Halstead-Metrik.
- Software Maintainability Index.

Im Folgenden werden diese beschrieben und es wird erklärt, wie sie genutzt werden können, um die Unterkategorien der Wartbarkeit zu messen.

Unterkategorie	Beschreibung
Modularität	Grad in welchem ein System oder Computer Programm zusammengesetzt ist, sodass Änderungen in einer Komponente einen minimalen Effekt auf andere Komponenten ha.t
Wiederverwendbarkeit	Grad in welchem ein Asset in mehr als einem System benutzt werden kann.
Analysierbarkeit	Grad an Effektivität und Effizienz in dem es möglich ist, bei geplanten Änderungen die Auswirkungen auf ein System abzuschätzen, Fehlerursachen oder Mängel an einem Produkt zu diagnostizieren oder Teile zu identifizieren, die überarbeitet werden müssen.
Veränderbarkeit	Grad in dem ein System effektiv und effizient verändert werden kann ohne die Qualität des Produkts zu mindern und ohne neu Fehler zu erzeugen. Dieses kann durch die Unterkateogrien Modularität und Analysierbarkeit beeinflusst werden.
Testbarkeit	Grad an Effektivität und Effizienz mit dem Testkriterien für ein System etabliert und mit welchem Tests durchgeführt werden können, um zu ermitteln, ob die Kriterien eingehalten wurden.

Tabelle 5: Unterkateogiren der Wartbarkeit nach ISO/IEC 25010 [61] S. 14f

5.1.1 Lines of Code (LoC)

Bei dieser Metrik wird die Anzahl der Programmzeilen inklusive Kommentar- und Leerzeilen gezählt, um den Programmumfang abzuschätzen [3] S. 65. Je größer die gezählte Zahl der Zeilen ist, desto größer ist das Programm und dies impliziert einen höheren Wartungsaufwand. Denn je mehr Codezeilen existieren, desto mehr Aufwand ist erforderlich, um diesen zu lesen und zu verstehen. Da Kommentarzeilen im Code jedoch die Funktion haben, dem menschlichen Leser Abschnitte im Programm natürlichsprachlich zu erklären und Leerzeilen dazu dienen, den Code räumlich zu strukturieren und somit beide zur Verbesserung der Wartbarkeit beitragen, werden diese in den Experimenten zur Messung der Wartbarkeit in Bezug auf *Lines of Code* nicht mit einbezogen. Diese Variation der *Lines of Code* wird *Source Lines of Code* [3] S. 67 genannt. Folglich werden in den Experimenten die *Source Lines of Code* gemessen, um die Wartbarkeit der Bots zu bewerten.

Es ist zu erwähnen, dass diese Metrik und ihre Variationen kritisiert werden, da die Länge von Programmcode abhängig von Programmiersprache ist [3] S. 65. Dies hat zur Folge, dass der Vergleich von Programmen, die in verschiedenen Programmiersprachen implementiert sind, nicht aussagekräftig sind. Denn die Anzahl der *Lines of Code* können sich in diesem Fall trotz gleichem Implementierungskonzept aufgrund der Syntax der Programmiersprache unterscheiden [3] S. 65-67. Da in dieser Ausarbeitung jedoch die gleiche Programmiersprache bei beiden Implementierungen verwendet und nach dem selben Konzept implementiert wurde, sofern das möglich war, trifft dieser Kritikpunkt bei einem Vergleich nicht zu.

Durch die Messung der *Source Lines of Code* werden die Unterkategorien *Analysierbarkeit* und *Veränderbarkeit* der Wartbarkeit nach ISO/IEC 25010 untersucht. Dies ist wie folgt zu begründen: Je umfangreicher ein Programm ist, desto mehr Aufwand ist nötig, die Auswikung von Änderungen abzuschätzen, da potentiell jede Zeile des Programms von der Änderung betroffen sein könnte. D. h., dass mit steigenden Codezeilen der Analyseaufwand steigt. Dadurch steigt ebenfalls das Risiko, dass bei Veränderungen an einer Code Stelle Fehler an einer anderen Stelle entstehen, die aufgrund fehlerhafter oder unzureichender Codeanalyse nicht zum Zeitpunkt der Änderung erkannt wurden.

5.1.2 Duplizierter Code

Wie in Tabelle 5 beschrieben, ist das Ziel der Modularität, dass das Ändern von Programmcodes an einer Stelle möglichst wenig Auswirkung auf andere Code Stellen hat. Dies bedeutet, wenn an einer Stelle Änderungen gemacht werden, dass alle Stellen, die die gleiche Logik verwenden, davon betroffen sind. Existiert im Programm ein doppeltes Codefragment, dann impliziert dies, dass es an jenen Stellen, an denen die gleiche Logik verwendet wird, auch diese anderen Stellen geändert werden müssen. Es fehlt also eine Kapselung des Codes an diesen Stellen. Folglich würde eine Änderung an einer Stelle zu inkonsistenten Ergebnissen führen, wenn die gleiche Logik an anderen Stellen existiert und nicht ebenfalls angepasst wurde. Somit hat diese eine Änderung zum einen die Auswirkung, dass Code in anderen Komponenten ebenfalls geändert werden muss und zum anderen, dass ein Risiko besteht, inkonsistente Daten zu erhalten, sollte eine notwendige Stelle nicht ebenfalls angepasst worden sein. Aus diesem Grund wirkt sich ein duplizierter Code negativ auf die Modularität aus. Das Kapseln von Funktionen unterstützt ebenfalls die Änderbarkeit eines Programms, da somit eine Änderung nur noch an einer Stelle vorgenommen werden muss. Folglich dient die Analyse von doppelten Codes auch zur Bewertung der Änderbarkeit eines Programms.

Es gibt andere Vorschläge von Metriken, die verwendet werden können, um die Modularität von Software zu verwenden. Beispielsweise haben Huynh, Cai, Song und Sullivan [12] S. 415f einen Algorithmus entwickelt, der das Analysieren der Modularität automatisiert. Da es jedoch keine Software gibt, die diese Analyse für JavaScript Programme durchführt und eine eigene Implementierung den Umfang dieser Ausarbeitung überschreiten würde, werden diese Metriken nicht verwendet. Die manuelle Analyse ist ebenfalls keine Option, da dies zu zeitintensiv sowie fehleranfällig wäre.

5.1.3 Zyklomatische Komplexität

Nach Broy und Kuhrmann [3] S. 67-69 wird bei dieser Metrik die Komplexität einer Funktion oder eines Code Fragments bemessen. Es wird dabei angenommen, dass ein Mensch Codes ab einer bestimmten Komplxität nicht mehr ohne Weiteres verstehen kann, wodurch seine Fähigkeit, die Code Qualität zu bewerten, beeinflusst wird. Die Metrik kann auf zwei Arten berechnet werden. Die erste Berechnungsart basiert auf Binärverzweigungen, die im gesamten Programm zu finden sind. Binärverzweigungen sind

Stellen im Code, an denen entschieden wird, welcher von zwei Pfaden zur Programmlaufzeit ausgeführt wird. Ein klassisches Beispiel dafür ist in der Programmierung die if-else Struktur. Mit Hilfe dieser Verzweigungen wird die Komplexität M des Programms wie folgt errechnet:

$$M = b + p \tag{1}$$

In der Formel 1 steht b für bedingte Anweisungen, die genau zwei Zweige haben, wie beispielsweise eine if-else Anweisung. Das p steht für die Anzahl der Kontrollflussgraphen. Ein Kontrollflussgraph bildet die Folge von Aktionen als einen Prozess ab [3] S. 179. Die zweite Berechnungsart basiert auf der Anzahl der Knoten und Kanten in einer Funktion oder in einem Code Fragment. Die Berechnung der Komplexität M lässt sich mit folgender Formel beschreiben:

$$M = e - n + 2p \tag{2}$$

Dabei steht e für die Anzahl der Kanten und n für die Anzahl der Knoten im Kontrollflussgraphen. Das p steht für die Anzahl der Zusammenhangskomponenten im Kontrollflussgraphen.

Diese Metrik ist geeignet, um die zwei Unterkategorien *Wiederverwendbarkeit* und *Testbarkeit* der Wartbarkeit nach ISO/IEC 25010 zu untersuchen. Dies ist wie folgt zu begründen: Da die zyklomatische Komplexität Funktionen besser bewertet, wenn diese wenig komplex sind, bedeutet dies, dass diese Funktionen nicht umfangreich sind und nur sehr kleine Aufgaben erfüllen und somit den Grad der Kohäsion im System erhöhen. Kleine Funktionen, die nur eine Aufgabe erfüllen, sind geeignet dafür, wiederverwendet zu werden. Somit wird bei einer geringen zyklomatischen Komplexität der Wert der Wiederverwendbarkeit höher gewertet. Des Weiteren kann die zyklomatische Komplexität verwendet werden, um eine Zahl an Testfällen zu nennen, die nötig sind, um eine vollständige Zweig- [3] S. 68 sowie Anweisungsabdeckung [11] S. 219 zu erreichen, da diese alle nötigen Pfade errechnet, die durch Tests abgedeckt werden müssen. D. h., dass eine hohe zyklomatische Komplexität gleichbedeutend mit einem höheren Testaufwand ist, da es mehr Testfälle gibt. Zusammengefasst bedeutet dies, dass eine hohe zyklomatische Komplexität den Grad der Testbarkeit des Programms senkt.

5.1.4 Halstead-Metrik

Nach Broy und Kuhrmann [3] S. 69-71 wird diese Metrik verwendet, um den Wartungsaufwand einer Software zu schätzen anhand der Anzahl der Operatoren und Operanden in dem Programm. Dafür werden zuerst zwei Vokabulare n_1 und n_2 definiert. Das Vokabular n_1 beinhaltet alle Operatoren, die gezählt werden können. Zu Operatoren gehören bspw. Schlüsselwörter und Vergleichsoperatoren). Das Vokabular n_2 beinhaltet alle Operanden, die gezählt werden könnnen. Operanden sind Konstanten oder Variablen. Des Weiteren wird die Implementierungslänge N berechnet, welche die Summe aller Operatoren N_1 und Operanden N_2 entspricht, die in dem Programm verwendet wurden. Mit diesen Werten werden die Halstead-Länge (HL) und die Halstead-Volumen (HV) berechnet:

$$HL = n_1 * log_2 n_1 + n_2 * log_2 n_2 \tag{3}$$

$$HV = N * log_2 n \tag{4}$$

Mit diesen zwei Werten können die folgenden drei Kennzahlen abgeschätzt werden, die die Wartbarkeit des Systems bemessen:

- Die Schwierigkeit D (Formel 5). Dieser Wert beschreibt, wie schwierig es ist ein Programm zu schreiben oder es zu verstehen.

$$D = \frac{n_1}{2} * \frac{N_2}{n_2} \tag{5}$$

- Die Aufwandsabschätzung E (Formel 6). Dieser Wert beschreibt den Aufwand, der nötig ist, um das Programm zu implementieren oder zu verstehen.

$$E = D * HV \tag{6}$$

- Die benötigte Zeit zur Implementierung T (Formel 7). Dieser Wert hängt von der Aufwandsschätzung E ab und beschreibt, wie lange es ungefährt dauert, das Programm zu implementieren oder zu verstehen.

$$T = \frac{E}{18} s \tag{7}$$

Diese Metrik ist geeignet, um die zwei Unterkategorien *Analysierbarkeit* und *Veränderbarkeit* der Wartbarkeit nach ISO/IEC 25010 zu untersuchen. Dies ist wie folgt zu begründen: Das Ergebnis des Halstead Volumens basiert auf der Anzahl der Operatoren und Operanden, die verwendet werden können und es beinhaltet auch die Summe der tatsächlich verwendeten Operatoren und Operanden. Somit hängt dieser Wert von der Länge des Codes, also von den Operationen und den Variablen ab. Je mehr Operationen durchgeführt werden und je mehr Variablen beteiligt sind, desto schwierger wird es, den Code zu verstehen. Ähnlich wie in Abschnitt 5.1.1 beschrieben, wirkt die steigende Anzahl an Operationen und Variablen sich negativ auf die Analysierbarkeit und in folge dessen auf die Veränderbarkeit aus, da es für den menschlichen Leser mehr Aufwand erfordert, all diese zu verstehen.

5.1.5 Software Maintainability Index

Nach Broy und Kuhrmann [3] S. 71f hilft diese Metrik dabei einzuschätzen, zu welchem Grad ein Softwaresystem wartbar ist. Die Berechnung verwendet zum einen das Halstead Volumen, die zyklomatische Komplexität und die Source Lines of Code Metriken und zum anderen Konstanten und Gewichtungsfaktoren, welche auf Erfahrungen und historischen Daten basieren. Letztere sind der Grund dafür, dass es mehrere Formeln zur Berechnung des Maintainability Indexes gibt, da diese andere Konstanten und Gewichtungen verwenden. Die orginale Formel zur Berechnung des Maintainability Indexes MI ist in Formel 8 beschrieben.

$$MI = 171 - 5,2 * lnHV - 0,23 * M - 16,2 * lnSLoC + 50 * sin\sqrt{2,4 * Cmt} \qquad (8)$$

Die Variable HV steht für das Halstead Volumen, $SLoC$ steht für die Source Lines of Code und Cmt steht für die zyklomatische Komplexität. Je höher der Maintainability Index ist, desto besser ist die Wartbarkeit des Porgramms. Microsoft Visual Studio hat die Berechnung des Maintainability Indexes in der Art verbessert, dass das Ergebnis eine Zahl zwischen 0 und 100 ist, wodurch deutlich wird, ob das Ergebnis positiv oder negativ zu bewerten ist.

Da diese Metrik die anderen drei Metriken Halstead Metrik, zyklomatische Komplexität und Source Lines of Code verwendet, deckt sie auch die gleichen Unterkategorien der Wartbarkeit nach ISO/IEC 25010 ab: Wiedervewendbarkeit, Analysierbarkeit, Veränderbarkeit und Testbarkeit. Dieser Wert lässt sich jedoch nicht mehr den einzelnen

Unterkategorien zuordnen, sondern ist als Bewertung der Wartbarkeit ungeachtet der Modularität zu betrachten.

5.1.6 Verwendung der Metriken zur Bewertung der Wartbarkeit

Die verfügbaren Metriken lassen sich, wie in Tabelle 6 dargestellt, anwenden, um die fünf Unterkategorien der Wartbarkeit nach ISO/IEC 25010 zu bewerten.

Unterkategorie	**Verwendbare Metrik**
Modularität	Duplizierter Code
Wiederverwendbarkeit	Zyklomatische Komplexität, Software Maintainability Index
Analysierbarkeit	SLoC, Halstead-Metrik, Software Maintainability Index
Veränderbarkeit	SLoC, Halstead-Metrik, Software Maintainability Index
Testbarkeit	Zyklomatische Komplexität, Software Maintainability Index

Tabelle 6: Anwendbare Metriken, mit denen die Unterkategorien des Qualitätsmerkmals Wartbarkeit nach ISO/IEC 25010 in einem System zu bemessen werden können

5.2 Effizienz

Das Qualitätsmerkmal der Effizienz beschreibt die Leistung im Verhältnis zu den genutzen Ressourcen unter vorgegebenen Bedingungen. Zu den Ressourcen zählen auch Software von dritten sowie die Hardware, die das Produkt verwendet [61] S. 11. Dieses Qualitätsmerkmal lässt sich in drei Unterkateogirien aufteilen, die in Tabelle 7 beschrieben werden.

5.2.1 Begriffsdefinitionen

Bevor die Effizienz einer Software getestet werden kann, muss zuerst ein Konsens darüber geschaffen werden, was die Begriffe, die benutzt werden, um die Effizienz zu beschreiben, bedeuten. Aus diesem Grund werden die Fachbegriffe, die in Tabelle 7 genannt wurden, im folgendem Abschnitt erklärt.

Unterkategorie	Beschreibung
Zeitverhalten	Grad in dem die Antwortzeit und Verarbeitungszeit sowie die Durchsatz Rate des Systems vorgegebenen Anforderungen entspricht
Ressourcennutzung	Grad, in dem die Menge und Art der Ressourcen, die ein System bei der Ausführung seiner Funktionen verwendet, den Anforderungen entsprechen. Auch Menschen sind Ressourcen.
Kapazität	Grad, in dem die Höchstgrenzen eines Systemparameters den Anforderungen entsprechen. Beispiele für Parameter: • Anzahl an Objekten, die gespeichert werden können • Anzahl an Nutzern, die das System gleichzeitig nutzen können • Die Kommunikationsbandbreite • Durchsatz von Transaktionen • Größe der Datenbank

Tabelle 7: Unterkateogiren der Effizienz nach ISO/IEC 25010 [61] S. 11

Zeitverhalten - Antwortzeit Nach Fischer und Hofer [8] S. 43 existiert die Antwortzeit in vernetzten Systemen und beschreibt die Zeit zwischen dem Absetzen einer Anfrage bzw. eines Auftrags und dem Eintreffen der Antwort bzw. der Bestätigung. Furrer [9] S. 74 weist darauf hin, dass die Antwortzeit von der aktuellen Belastung im System abhängt also beispielsweise von der Anzahl der Nutzer, die zu einem Zeitpunkt das System gleichzeitig nutzen. Eine akzeptable Antwortzeit, in der das System dem Nutzer keine Rückmeldung bezüglich der Fertigstellung der Verzögerung geben muss, liegt nach Nielsen [43] unter einer Sekunde. Danach wird der Gedankenfluss des Nutzers unterbrochen. Eine Antwortzeit, die mehr als 10 Sekunden beträgt, ist für den Nutzer inakzeptabel, denn in diesem Fall verliert der Nutzer den Fokus.

Zeitverhalten - Verarbeitungszeit Die Verarbeitungszeit bezeichnet die Zeitspanne zwischen dem Beginn und dem Ende eines Programms. D. h. hier ist die Latenz nicht relevant.

Zeitverhalten - Durchsatz Nach Fischer und Hofer [8] S. 212 ist der Durchsatz das Maß für den quantitativ ermittelten Datentransfer auf Bussen, in Kanälen oder in der Datenfernübertragung. Die Angaben sind meist in Bits pro Sekunde (bps) bzw. in Zehnerpotenzen davon angegeben.

Ressourcen Fischer und Hofer [8] S. 753 bezeichnen die Ressourcen als Betriebsmittel. Dazu gehören alle Rohstoff, Nutz-, Hilfs- und Verbrauchsmittel, die zum Betrieb eines Datenverarbeitungssystems nötig sind. Diese werden in vier Gruppen unterteilt:

1. Binär verwaltete Objekte. Dazu gehören bspw. Dateien, Daten oder Betriebssystemkommandos.
2. Hardwarekomponenten. Dazu gehören bspw. der Prozessor oder der Arbeitsspeicher eines Rechners.
3. Menschen.
4. Ressourcen, die zu jedem Zeitpunkt von nur einem Prozess benutzt werden können. Ein Beispiel dafür ist ein Drucker.

Kapazität Nach Fischer und Hofer [8] S. 478 bezeichnet die Kapazität im Allgemeinen ein Fassungsvermögen. Beispiele dafür sind das Fassungsvermögen eines Speichers, das Durchsatzvermögen in Datenkanälen oder das Leistungs- und Rechenvermögen von Prozessoren.

5.2.2 Anwendung auf diese Ausarbeitung

In dieser Ausarbeitung sind keine Anforderungen an das Zeitverhalten, die Ressourcennutzung oder die Kapazität gegeben. Das Ziel ist es, für beide Bot Implementierungen die Effizienz Werte in Experimenten zu messen und im Anschluss miteinander zu vergleichen. Die Experimente untersuchen Merkmale der Effizienz wie folgt:

- Zeitverhalten: Für jeden der drei Anwendungsfälle UC-01 bis UC-03 wird die Antwort- und Verarbeitungszeit ermittelt.
- Ressourcennutzung: Für jeden der drei Anwendungsfälle UC-01 bis UC-03 wird die Auslastung der CPU und des Arbeitsspeichers gemessen.
- Kapazität: Hier wird untersucht, wo die Grenzen der Antwortmöglichkeiten der Bots sind, indem die Frage beantwortet wird, ob es Begrenzungen bezüglich der Nachrichten gibt, die zwischen dem Bot und dem Client ausgetauscht werden. Als Beispiel: Begrenzte Anzahl von erlaubten Satzzeichen je Nachricht. Diese Frage ist von Interesse, da die Größe der Antwortnachrichten des Bots im Anwendungsfall UC-01 linear zur Anzahl der Mitarbeiter wachsen. Im Anwendungsfall UC-02 hängt die Größe der Nachricht davon ab, wie viele Termine der Nutzer anfragt. Und im Anwendungsfall UC-03 hängt die Größe von der Anzahl der Suchergebnisse ab. Mit Beantwortung dieser Frage können die Grenzen der Anwednungsfälle UC-01, UC-02 und UC-03 bestimmt werden.

Im folgenden wird erklärt, welche Merkmale nicht in den Experimenten getestet werden:

- Zeitverhalten - Durchsatz: Da beide Bots unterschiedliche Daten transferieren, unterscheidet sich bei jedem die Anzahl der Bits, die pro Sekunde versendet werden. Folglich wird ein Bot weniger Bits versenden müssen, um eine vollständige Antwort zu geben. Demnach ist die Durchsatzrate bei jenem niedriger als beim anderen. Dies sagt jedoch nicht aus, dass ein Bot eine bessere Durchsatzrate als der andere hat. Somit sind die Durchsatzraten in Bits pro Sekunde nicht miteinander vergleichbar.

- Ressourcennutzung - Binär verwaltete Objekte und Menschen: Diese Arbeit konzentriert sich ausschließlich auf die Ressourcennutzung der Hardware, daher wird diese in Benzug auf den binär verwaltete Objekten und Menschen nicht berücksichtigt.

6 Experimente

Die Experimente in diesem Abschnitt untersuchen die Implementierung beider Bots auf die Qualitätsmerkmale **Wartbarkeit** (Abschnitt 6.1) und **Effizienz** (Abschnitt 6.2). Jedes Experiment beginnt mit der Erläuterung der Forschungsfrage. Das Ziel des Experiments ist es, eine Antwort auf diese Frage zu generieren. Im Anschluss werden die Erwartungen erläutert, die basierend auf Annahmen an die Ergebnisse gestellt werden. Darauf folgenden wird der Aufbau des Experiments beschrieben. Dieser beinhaltet, welche Vorbereitungen nötig sind, um mit den Experimenten beginnen zu können. Dann wird die Durchführung beschrieben. Diese dokumentiert, welche Schritte auszuführen sind, um Ergebnisse zu generieren. Im Anschluss werden die Ergebnisse präsentiert und beschrieben. Diese werden dann im Folgekaptiel 7 diskutiert.

6.1 Wartbarkeit

In diesem Abschnitt wird die Wartbarkeit der Slack und Microsoft Teams Bots gemessen und verglichen. Ein Programm, das einfach zu warten ist, bietet zahlreiche Vorteile. Beispielsweise ist eine gute Änderbarkeit wichtig, wenn das Programm weiter entwickelt wird oder angepasst werden muss. Die Gründe, die eine Änderung am Programm erfordern sind ebenfalls zahlreich. Ein Auslöser könnte zum Beispiel sein, dass die Anforderungen an das Programm aufgrund neuer Technologie oder Sicherheitsproblemen verändert wurden. Ein anderer Auslöser könnte ein Fehler im Programm sein, dessen Behebung schnellstmöglich erfolgen muss. Ein Programm, das gut analysierbar ist, begünstigt die Änderbarkeit, da so die zu ändernden Stellen schneller identifiziert werden können. Des Weiteren führt ein gut analysierbares Programm zu einer langfristigen Kostenersparnis, da die Einarbeitungszeit neuer Entwickler kürzer ist und auch Entwickler, die bereits Erfahrung mit dem Programm haben, Probleme leichter und somit schneller finden und beheben können. Ein gut testbares Programm bietet die Möglichkeit, Fehler zu verhindern und führt somit zu einem zuverlässigen System, welches wiederum zu zufriedenen Endnutzern führt und keine weiteren Wartungskosten verursacht. Ein Programm, welches viele wiederverwendbare Elemente beinhaltet, kann die Entwicklungszeit verkürzen und Änderungen erleichtern, da keine Zeit benötigt wird, um einen Code zu schreiben, der bereits exisitiert, um die notwendige Aufgabe zu erfüllen und weil eine Änderung somit nur an einer Stelle vorgenommen werden muss und nicht an mehreren Stellen an denen der Code dupliziert wurde. Dies begünstigt die Modularität eines Programms, denn durch

die Vermeidung doppelten Codes entsteht eine Kapselung von Aufgaben in Methoden, die durch klare Abgrenzung von Aufgaben einen hohen Wert der Wiederverwendbarkeit haben. Ein modulares Programm bietet den Vorteil, dass eine Änderung an einer Stelle wenig Auswirkung auf andere Stellen im Code hat, wodurch das Risiko neue Fehler zu verursachen minimiert wird. Zusammengefasst ermöglicht ein gut wartbares Programm eine hohe Flexibilität, verrigerten Wartungs- und Entwicklungsaufwand sowie eine hohe Zuverlässigkeit.

6.1.1 Forschungsfrage

In diesem Experiment soll die Frage beantwortet werden, ob Slack oder ob Microsoft Teams in Bezug auf die Wartbarkeit des Programms besser geeignet ist, um einen Bot zu implementieren, der die zuvor genannten Anwendungsfälle (UC-01 bis UC-03) bedient.

Der Untersuchtungsgegenstand ist der Programmcode, der geschrieben wurde, um die Anwednungsfälle durch einen Bot zu verarbeiten sowie der Programmcode, der zur Verfügung steht, um einen Bot für die entsprechende Plattform zu implementieren. Letzeres wird in diesem Experiment als Bot Framework bezeichnet.

Die Eigenschaften der Wartbarkeit, die durch ISO/IEC 25010 definiert wurden, sind in Abschnitt 5.1 erklärt worden. Diese dienen dem Experiment als Grundlage, um die Implementierung der Bots und der Bot Frameworks zu bewerten. Das bedeutet, dass der zu untersuchende Programmcode auf Modularität, Wiederverwendbarkeit, Analysierbarkeit, Veränderbarkeit und Testbarkeit geprüft werden muss. Dies wird in diesem Experiment mit Hilfe von Metriken realisiert. Die folgenden Metriken werden verwendet, um eine oder mehr Wartbarkeitseigenschaften zu bewerten:

- Source Lines of Code bewertet Analysierbarkeit und Veränderbarkeit.
- Duplizierter Code bewertet Modularität.
- Zyklomatische Komplexität bewertet Wiederverwendbarkeit und Testbarkeit.
- Halstead Metrik bewertet Analysierbarkeit und Veränderbarkeit.
- Software Maintainability Index bewertet Analysierbarkeit, Veränderbarkeit, Wiederverwendbarkeit und Testbarkeit.

Was genau jede einzelne der Metriken beinhaltet und warum diese geeignet ist, um eine Wartbarkeitseigenschaft zu bewerten, ist in Kapitel 5.1.1 bis 5.1.5 erläutert.

Die Beantwortung der Forschungsfrage ist relevant, da ein Unternehmen, welches entweder Slack oder Microsoft Teams verwenden möchte, um einen Bot mit den hier genannten Anwendungsfällen zu betreiben und weiter zu entwickeln, ein Interesse daran hat, die Option mit dem geringsten Wartungsaufwand zu wählen, um somit die Wartungs- und Entwicklungskosten möglichst gering zu halten und trotzdem ein zuverlässiges Programm zu haben.

6.1.2 Erwartungen

Es ist anzunehmen, dass die Frameworks eine schlechtere Bewertung der Wartbarkeit erzielen werden als die eigenen Bot Implementierungen, da die Frameworks sehr viel umfangreicher sind. Und da das Microsoft Teams Bot Framework selber noch größer ist als das Slack Bot Framework, ist zu vermuten, dass das Microsoft Teams Bot Framework die schlechtesten Ergebnisse erzielen wird. Diese Annahmen basieren auf der Tatsache, dass die SLoC Metrik sowie das Halstead Volumen von der Größe des Programms abhängen. Je höher der Wert der SLoC oder des Halstead Volumens ist, desto schlechter ist die Wartbarkeit zu bewerten. In Bezug auf die Anzahl duplizierter Code Fragmente in den Programmen, kann ohne vorherige Auseinandersetzung mit dem gesamten Code keine Annahme getroffen werden. Die Größe eines Programms ist jedoch kein Indikator dafür, dass ein Programm duplizierte Code-Fragmente enthält. Zwar würden doppelter Codes dazu beitragen, dass das Programm größer ist, als es sein müsste, aber diese Folgerung kann nicht umgekehrt in dem Sinne angewandt werden, dass ein großes Programm viele doppelte Codes enthält. Auch die zyklomatische Komplexität lässt sich nicht durch die Größe des Programms abschätzen und aus diesem Grund können dies bezüglich keine Vermutungen angestellt werden ohne den Code vorher zu untersuchen. Die Implementierung des Slack und Microsoft Teams Bots sind vom Umfang her sehr ähnlich und nach dem gleichen Konzept implementiert, daher ist zu vermuten, dass beide ähnliche Werte erzeugen werden. Da sie beide wesentlich kleiner als die Frameworks sind, ist anzunehmen, dass ihre Ergebnisse allgemein besser ausfallen werden als die der Frameworks.

Allgemein ist festzulegen, dass je niedriger die SLoC, die Codezeilen-Duplikate, das Halstead Volumen und die zyklomatische Komplexität ist, desto besser ist das Programm in

Bezug auf Wartbarkeit zu bewerten. Der Maintainability Index besitzt einen Wertebereich von 0 bis 100 wobei 100 der bestmögliche Wert ist, der erreicht werden kann und somit ein Programm im Punkt der Wartbarkeit maximal positiv bewertet. Folglich wird ein Programm mit einem Maintainability Index von 0 maximal negativ bewertet.

6.1.3 Aufbau

Es wird die Programmiersprache JavaScript und die Entwicklungsumgebung Visual Studio Code [73] zum Testen verwendet.

Zum Messen des Software Maintainability Indexes, der zyklomatischen Komplexität sowie der Halstead Metrik wird die Software *es6-plato* [49] verwendet. Sie wurde ausgewählt, da sie die folgended Metriken berechnen kann: Source Lines of Code, zyklomatische Komplexität, Halstead-Metrik und Maintainability Index. Um doppelte Codes zu finden, wird das Modul *jscpd* [50] verwendet. Um diese zu installieren, muss zuvor der Packet Manager *npm* installiert werden [71].

6.1.4 Durchführung

Jedes Verzeichnis, das Programmcode enthält, der nötig ist, um den Slack oder den Microsoft Teams Bot zu betreiben, wird durch die Programme *es6-plato* und *jscpd* analysiert. Die betroffenen Verzeichnisse sind in Tabelle 21 von Position 1 bis 12 aufgelistet. Diese enthalten sowohl Programmcode, der explizit für diese Ausarbeitung geschrieben wurde, um die Anwendungsfälle UC-01 bis UC-03 verarbeiten zu können sowie Programmcode, der importiert wurde und als Schnittstelle dient, mit welcher die Bot Implementierung vereinfacht wird. Da ohne diesen Programmcode die Bots nicht betriebsfähig wären, sind all diese Verzeichnisse relevant in Bezug auf die Wartbarkeit der Bots.

Die Abbildung 11 zeigt, wie die Analysefunktion der beiden Programme *es6-plato* und *jscpd* in der Kommandozeile für ein Verzeichnis aufgerufen werden. Dies bedeutet, dass diese Befehlssequenz 12 Mal durchgeführt werden muss, da es 12 Verzeichnisse gibt, die analysiert werden. Die Ergebnisse werden in einem Ordner mit dem Namen *report* gespeichert. Dieser befindet sich nicht in dem Verzeichnis, welches analysiert wurde, sondern dort, wo der Befehl abgesetzt wurde. Des Weiteren werden alle Unterverzeichnisse des zu analysierenden Verzeichnisses, die den Namen *node_ modules* tragen, aus der Analyse ausgeschlossen. Der Grund dafür ist, dass diese keinen Code enthalten, der explizit

```
1    // es6-plato Analysefunktion ausfuehren
2    npx es6-plato --exclude node_modules -r -d report/es6-plato/
     $DIRECTORYPATH
3
4    // jscpd Analysefunktion ausfuehren
5    npx jscpd --ignore **/node_modules --output report/duplicates/
     $DIRECTORYPATH
6
```

Abbildung 11: Nutzung der es6-plato und jscpd Analysefunktionen in der Kommandozeile

für die Implementierung von Slack bzw. Microsoft Teams Bots geschrieben wurde. Alle Dateien, die sich im *node_modules* Ordner befinden, sind installierte Programm Packete, die von unabhängigen Entwicklern geschrieben wurden [70] [72]. Um den Aufruf der Analysefunktionen nachvollziehen zu können, werden die einzelnen Befehlsparameter in Tabelle 8 erläutert.

6.1.5 Ergebnisse

Die Ergebisse der beiden Analysen durch die Software *es6-plato* und *jscpd* sind in den folgenden Verzeichnissen zu finden:

- es6-plato → ./report/es6-plato/
- jscpd → ./report/duplicates/

Diese Verzeichnisse befinden sich an dem Ort, an dem die Befehle zur Analyse abgesetzt wurden. Im Folgenden werden die Analyseergebnisse für die beiden Bot Implementierungen und für die beiden Frameworks präsentiert.

6.1.5.1 Implementierung des Slack Bots

Zur Wartbarkeitsanalyse des Slack Bots wurde das Slack Bot Verzeichnis (siehe Anhang Tabelle 21 Position 1) durch die Programme *es6-plato* und *jscpd* analyisert. Die Tabelle 9 zeigt zusammengefasst die Ergebnisse der Analysen.

Zeile	Parameter	Erklärung
2	es6-plato	Analysefunktion von es6-plato ausführen
2	–exclude node_modules	Verzeichnisse, deren Pfad node_modules enthalten, werden nicht in der Analyse mit einbezogen
2	-r	Alle Dateien in Unterverzeichnissen werden in die Analyse mit einbezogen
2	-d report/es6-plato/	Das Analyseergebnis wird im Verzeichnis report/es6-plato/ gespeichert. Dieses Verzeichnis befindet sich an dem Ort, wo der Befehl abgesetzt wurde
2 & 5	npx	Packet, das mit dem Node Package Manager installiert wurde, ausführen
2 & 5	$DIRECTORYPATH	Pfad des Verzeichnisses, das analysiert wird
5	jscpd	Analysefunktion von jscpd ausführen
5	–ignore **/node_modules	Verzeichnisse, deren Pfad node_modules enthalten, werden nicht in der Analyse mit einbezogen
5	–output report/duplicates/	Das Analyseergebnis wird im Verzeichnis report/duplicates/ gespeichert. Dieses Verzeichnis befindet sich an dem Ort, wo der Befehl abgesetzt wurde

Tabelle 8: Erläuterung der Befehlsparameter bei der Analyse durch die Programme *es6-plato* und *jscpd*

Beschreibung	Gesamt	Durchschnitt
Anzahl an Dateien	7	1
SLoC	234	33,43
Codezeilen-Duplikate	0	0
Halstead Volumen	8.364,65	35,75
Komplexität	36	6,32
Maintainability Index	-	69,15 %

Tabelle 9: Ergebnisse der Wartbarkeitsmessung für die gesamte Slack Bot Implementierung

Der Code umfasst insgesamt 234 Programmzeilen, der auf sieben Dateien aufgeteilt ist. Im Schnitt besteht jede Datei also aus ungefähr 33 Zeilen. Es wurde kein duplizierter Code gefunden. Das Halstead Volumen beläuft sich auf ungefähr 8.364, wodurch der Schnitt ungefähr bei 36 pro Codezeile liegt. Die zyklomatische Komplexität beträgt 36, wodurch im Durchschnitt jede Zeile ungefähr eine zyklomatische Komplexität von 6,32 hat. Der Maintainability Index liegt für das gesamte Programm bei ungefähr 69 % und somit zumindest über 50 %.

6.1.5.2 Implementierung des Slack Bot Frameworks

Zur Wartbarkeitsanalyse des Slack Bot Frameworks wurde das Slack Bot Framework Verzeichnis (siehe Anhang Tabelle 21 Position 2) durch die Programme *es6-plato* und *jscpd* analyisert. Die Tabelle 10 zeigt zusammengefasst die Ergebnisse der Analysen.

Beschreibung	Gesamt	Durchschnitt
Anzahl an Dateien	346	1
SLoC	6.842	19,77
Codezeilen-Duplikate	619	9 %
Halstead Volumen	295.179,71	43,14
Komplexität	1.870	3,66
Maintainability Index	-	79,55 %

Tabelle 10: Ergebnisse der Wartbarkeitsmessung für die gesamte Implementierung

Das Slack Bot Framework besteht aus insgesamt 346 Dateien und 6.842 Programmzeilen. Im Durchschnitt sind das ungefähr 20 Zeilen pro Datei. Es wurden 619 duplizierte

Code Fragmente gefunden. Dies bedeutet, dass ungefähr 9 % des gesamten Codes aus Duplikaten besteht. Das Halstead Volumen beläuft sich auf insgesamt über 295.000 und ungefähr 43 pro Codezeile im Durchschnitt. Die zyklomatische Komplexität beträgt zusammengerechnet 1.870 und im Schnitt 3,66 pro Zeile. Der Maintainability Index liegt mit 79,55 % im oberen Viertel der Werteskala.

6.1.5.3 Implementierung des Microsoft Teams Bots

Zur Wartbarkeitsanalyse des Microsoft Teams Bots wurde das Microsoft Teams Bot Verzeichnis (siehe Anhang Tabelle 21 Position 3) durch die Programme *es6-plato* und *jscpd* analyisert. Die Tabelle 11 zeigt zusammengefasst die Ergebnisse der Analysen.

Beschreibung	**Gesamt**	**Durchschnitt**
Anzahl an Dateien	9	1
SLoC	388	43,11
Codezeilen-Duplikate	0	0
Halstead Volumen	14.223,39	36,66
Komplexität	55	7,05
Maintainability Index	-	68,67 %

Tabelle 11: Ergebnisse der Wartbarkeitsmessung für die gesamte Microsoft Teams Bot Implementierung

Die Microsoft Teams Bot Implementierung besteht aus 9 Dateien und umfasst insgesamt 388 Programmzeilen, wodurch jede Datei im Schnitt ungefähr 43 Zeilen enthält. Es gibt keine doppelten Code Fragmente. Das Halstead Volumen liegt bei über 14.000 und im Schnitt für jede Codezeile bei ungefägr 37. Die zyklomatische Komplexität beläuft sich insgesamt auf 55, was durchschnittlich für jede Zeile eine Komplexität von 7,05 bedeutet. Der Maintainability Index liegt mit 68,67 % in der oberen Hälfte der Werteskala.

6.1.5.4 Implementierung des Microsoft Teams Bot Frameworks

Zur Wartbarkeitsanalyse des Microsoft Teams Bot Frameworks wurden die Microsoft Teams Bot Framework Verzeichnisse (siehe Anhang Tabelle 21 Position 4 bis 12) durch die Programme *es6-plato* und *jscpd* analyisert. Die Tabelle 12 zeigt zusammengefasst die Ergebnisse der Analysen.

Beschreibung	Gesamt	Durchschnitt
Anzahl an Dateien	728	1
SLoC	114.641	157,47
Codezeilen-Duplikate	19.684	17 %
Halstead Volumen	7.377.352,74	64,35
Komplexität	30.542	3,75
Maintainability Index	-	76,47 %

Tabelle 12: Ergebnisse der Wartbarkeitsmessung für die gesamte Microsoft Teams Bot Framework Implementierung

Das Microsoft Teams Bot Framework besteht aus 728 Dateien und umfasst insgesamt 114.641 Zeilen Programmcode, was im Schnitt auf ungefähr 157 Zeilen pro Datei hindeutet. Es wurden insgesamt 19.684 duplizierte Code Fragmente gefunden. Dies bedeutet, dass ungefähr 17 % des Codes aus doppeltem Code besteht, was recht hoch erscheint. Das Halstead Volumen für das Framework beläuft sich insgesamt auf über 7.000.000 und durchschnittlich pro Codezeile auf über ungefähr 64. Für die zyklomatische Komplexität wurde ein Wert von über 30.000 berechnet, was im Durchschnitt für jede Zeile ein Wert von ungefähr 3,75 bedeutet. Der Maintainability Index befindet sich mit 76,47 % im oberen Viertel der Werteskala.

6.1.5.5 Zusammenfassung

In der Tabelle 13 werden alle Ergebnisse der Wartbarkeitsmessungen bei den vier Programmen zusammengefasst. Es ist zu erkennen, dass die Frameworks bei allen Messungen, in denen die Werte akkumuliert wurden, die Werte der Bot Implementierungen weit übersteigen. Von den beiden Frameworks weist das Microsoft Teams Bot Framework in jeder Zeile, d. h. sowohl bei den akkumulierten als auch bei den Durchschnittswerten, mit Ausnahme der Maintainability Index Zeile größere Werte auf als das Slack Framework. Ebenso weist die Microsoft Teams Bot Implementierung in jeder Zeile mit Ausnahme der Maintainability Index Zeile größere Werte als die Slack Bot Implementierung auf. Werden die Durchschnittwerte aller vier Programme betrachtet, so fällt auf, dass das Slack Bot Framework bei den durchschnittliche SLoC, dem durchschnittlichen Halstead Volumen sowie teilweise bei der durchschnittlichen zyklomatischen Komplexität bessere Werte als die beiden Bot Implementierungen erzielt. Der beste Maintainability Index mit einem Wert von 79,55 % gehört zum Slack Bot Framework. Knapp darunter liegt mit

	Slack Bot	Microsoft Teams Bot	Slack Framework	Microsoft Teams Framework
Anzahl an Dateien	7	9	346	728
SLoC	234	388	6.842	114.641
SLoC, Durchschnitt	33,43	43,11	19,77	157,47
Codezeilen-Duplikate	0	0	619	19.684
Codezeilen-Duplikate, Anteil	0 %	0 %	9 %	17 %
Halstead Volumen	8.364,65	14.223,39	295.179,71	7.377.352,74
Halstead Volumen, Durchschnitt pro Zeile	35,75	36,66	43,14	64,35
Komplexität	37	55	1.870	30.542
Komplexität, Durchschnitt pro Zeile	6,32	7,05	3,66	3,75
Maintainability Index	69,15 %	68,67 %	79,55 %	76,47 %

Tabelle 13: Zusammenfassung der Ergebnisse zur Wartbarkeitsmessung

76,47 % das Microsoft Teams Bot Framework. Ungefähr 7 % darunter liegt die Slack Bot Implementierung mit 69,15 %. Und das Programm, das den schlechtesten Maintainability Index erzielt hat, aber nicht einmal 1 % unter dem der Slack Bot Implementierung liegt, ist die Microsoft Teams Bot Implementierung.

6.2 Effizienz

In diesem Abschnitt wird die Effizienz der Slack und Microsoft Teams Bots in Bezug auf Zeitverhalten, Ressourcennutzung und Kapazität gemessen und verglichen. Diese drei Eigenschaften sind Merkmale von effizenten Systemen, die durch ISO/IEC 25010 definiert wurden. Eine Beschreibung dieser Merkmale ist in dieser Ausarbeitung in Abschnitt 5.2 gegeben.

6.2.1 Forschungsfrage

In diesem Experiment soll die Frage beantwortet werden, ob Slack oder ob Microsoft Teams in Bezug auf die Effizienz des Programms besser geeignet ist, um einen Bot zu implementieren, der die zuvor genannten Anwendungsfälle (UC-01 bis UC-03) bedient.

Der Untersuchungsgegenstand sind das Zeitverhalten, die Ressourcennutzung sowie ausgewählte Parameter bezüglich der Kapazität der Slack und Microsoft Teams Bots. Die Messeungen des Zeitverhaltens sind auf die Antwort- und Verarbeitungszeit beschränkt. Die Untersuchung der Ressourcennutzung ist auf die CPU und Arbeitsspeichernutzung eingeschränkt und bezüglich der Kapazität werden die beiden Parameter Nachrichtenlänge und Nachrichtensendelimit untersucht. Dafür müssen sowohl der Slack als auch der Microsoft Teams Bot in Betrieb sein und Nachrichten senden sowie empfangen können.

Das Prüfen des **Zeitverhaltens** bezieht sich in diesem Experiment auf die Antwortzeit und auf die Verarbeitungszeit, die für das Ausführen jeden Anwendungsfalls entstehen. D. h. es wird gemessen, wie lange es dauert bis der Nutzer, der eine Frage, die zu einem Anwendungsfall gehört, an den Bot sendet, eine Antwort erhält (Antwortzeit) und es wird ab dem Zeitpunkt, ab dem der Bot die Nutzeranfrage erhalten hat, gemessen, wie lange es dauert, bis der Bot eine Antwort formuliert hat (Verarbeitungszeit). Die **Ressourcennutzung** wird in Bezug auf Auslastung der CPU sowie des Arbeitsspeichers untersucht, während jeder Ausführung eines Anwendungsfalls. D. h. es wird ermittelt, wie viel Prozent der CPU und wie viele MegaByte an Arbeitsspeicher die Maschine, die den Bot betreibt, zur Verarbeitung eines Anwendungsfalls benötigt. Bezüglich der **Kapazität** werden die Grenzen der Nachrichtensendungen in Bezug auf ihre Länge und ihre Anzahl untersucht. D. h. es wird untersucht, ob gesendete Nachrichten eine bestimmte Größe nicht übersteigen dürfen und ob in vorgegebenen Intervallen eine Anzahl von versendeten Nachrichten nicht überschritten werden darf.

Die Beantwortung der Forschungsfrage ist relevant, da ein Unternehmen, welches entweder Slack oder Microsoft Teams verwenden möchte, um einen Bot mit den hier genannten Anwendungsfällen zu betreiben, ein Interesse daran hat, die Option zu wählen, die den geringsten Ressourcenverbrauch benötigt, am schnellsten Ergebnisse liefert und hohe Kapazitäten im Bereich der Nachrichtensendungen hat. Denn ein geringer Ressourcenverbrauch ist schonend für die Hardware und somit nachhaltig für die Umwelt. Dies bedeutet, dass je weniger die CPU und der Arbeitsspeicher beansprucht werden, desto besser ist dies. Des Weiteren ist es dadurch möglich, beim Mieten von Servern, um den Bot zu betreiben, eine günstige Variante mit wenigen Hardwareressourcen zu wählen. Eine schnelle Antwortzeit ist benutzerfreundlich und ermöglicht dem Nutzer schnellstmöglich mit seiner Arbeit, die die Antwort des Bots benötigt, fortzufahren. Und eine hohe Nachrichtenkapazität ist für ein Unternehmen von Vorteil, da es somit mehreren Nutzern möglich ist, Nachrichten gleichzeitig oder in kurzen Abständen an den Bot zu senden und Antworten zu empfangen, ohne, dass Sendegrenzen überschritten werden.

6.2.2 Erwartungen

Wie in Abschnitt 5.2.1 beschrieben, gilt eine Antwortzeit von maximal 10 Sekunden als akzeptabel. Alle Messungen, die diese 10 Sekunden übersteigen, werden als inakzeptabel bewertet. Sollte die Antwortzeit zwischen einer und zehn Sekunden liegen, dann wird diese als negativ bewertet. Je höher diese Zahl ist, desto negativer ist sie zu berwerten. Antwortzeiten unter einer Sekunde werden positiv bewertet. Sollte die Antwortzeit maximal 0,1 Sekunden betragen, dann wird diese als äußerst positiv bewertet. Das gleiche gilt für die Verarbeitungszeit.

In Bezug auf die CPU- und Arbeitsspeicher-Nutzung gilt, dass je kleiner der Ergebniswert ist, desto besser ist er zu bewerten. Solange die Grenzen der Hardware nicht überschritten werden, ist das Programm funktionsfähig und kann bewertet werden. D. h., dass in diesem Experiment die CPU Auslastung maximal bei 100 % und der Arbeitsspeicher-Nutzung bei maximal 32 GB liegen darf. Da die Bots entweder auf gemieteten oder eigenen Servern, die möglichst günstig und somit weniger Ressourcen zur Verfügung haben, betrieben werden, wird das Ergebnis positiv bewertet, wenn der Wert unter 50 % der Ressourcen eines kleinen Servers beanspruchen würde. In diesem Kontext wird ein Server als klein bezeichnet, wenn er bei einem Serververmietungsportal zum günstigsten Tarif angeboten wird. Die Webseite experte.de listet 12 Anbieter auf und macht Angaben zu der verfügbaren Server Hardware [18]. Durchschnittlich haben die 12 günstigste Server 2,875 GB Arbeitsspeicher und 1,583 CPU Kerne zur Verfügung. Aus diesem Grund werden die CPU Ergebnisse als positiv bewertet, wenn sie einer 1,5 Kerne CPU Auslastung von weniger als 50 % entspricht. Die Arbeitsspeicher Ergebnisse werden positiv bewertet, wenn sie weniger als 1,4375 GB benötigen.

Aufgrund des gemeinsamen Implementierungskonzepts ist zu erwarten, dass beide Programme ähnliche Werte erzielen werden. Da jedoch auch die Frameworks eine tragende Rolle beim Betrieb der Bots haben, besteht die Möglichkeit, dass Unterschiede aufgrund unterschiedlicher Hintergrundprozesse auftreten.

Es ist möglich, dass beim Ausführen des Anwendungsfalls UC-03 Microsoft Teams ein schlechteres Zeitverhalten erzielt wird, da die Beschaffung der Nachrichten für jedes Team, dem der Nutzer angehört, aufwändiger erscheint, als bei Slack. Denn das Slack Bot Framework bietet eine eigene Funktion, um diese Informationen zu erlangen. Das Microsoft Teams Bot Framework tut dies nicht. Stattdessen müssen hier Anfragen an

eine API gesendet werden, wodurch die Antwortzeit der API Anfrage den Prozess verlängert. Allerdings besteht die Möglichkeit, dass Slack ebenfalls die Informationen durch das Anfragen einer API erlangt, wodurch auch hier diese Antwortzeiten das gesamte Zeitverhalten verlängert und zu einem ähnlichen Ergebnis wie bei Microsoft Teams führt.

6.2.3 Aufbau

Folgende Software wurde verwendet, um das Experiment durchzuführen:

- Zeitverhalten: Browser Microsoft Edge und Entwicklertools.
- Ressourcennutzung: Node Package Manager Modul *node-os-utils* [51].

Um das Zeitverhalten messen zu können, muss der Durchführer den Slack bzw. den Microsoft Teams Client im Brwoser öffnen und dort mit dem Bot kommunizieren. Moderne Webbrowser wie Microsoft Edge oder Chrome bieten eine Funktion für Entwickler, um Webseiten zu analysieren. Dies sind die Entwicklertools. Diese sind in diesem Experiment nötig, um das Netzwerk zu analysieren, bei dem das Senden von Nachrichten sowie das Erhalten von Nachrichten aufgezeichnet wird und somit eine zuverlässige Quelle darstellt, um die Zeit zu messen, die zwischen dem Senden und Empfangen einer Nachricht vergangen ist. Um die Ressourcennutzung zu messen, wurde in diesem Experiment Software verwendet, die vom Node Package Manager angeboten wird. Diese Software nennt sich *node-os-utils* und bietet eine Schnittstelle, um den CPU- und Arbeitsspeicher-Verbrauch programmatisch zu messen. Folglich muss der Programmcode des Bots modifiziert werden, um diese Schnittstelle einzubinden.

Die Experimente wurden mit einer Maschine durchgeführt, die folgende Spezifikationen hat:

- Betriebssystem: Windows 11.
- Prozessor: AMD Ryzen 7 3700X, 8 Kerne.
- Arbeitsspeicher: 32 GB.
- NodeJS Version: v18.15.0.
- Datum der Experimentdurchführungen: 02.01.2024.

Da die Ergebnisse der Experimente von der Hardware der ausführenden Maschine abhängen, sind diese Spezifikationen bei der Diskussion der gemessenen Ergebnisse zu berücksichtigen.

6.2.4 Durchführung

Bei allen Experimenten zur Effizienz werden Anwendungsfälle mehrfach ausgeführt, um einen Durchschnittswert zu berechnen. Die Systeme Meisterplan und MsGraph werden in der kostenlosen Entwicklungs- bzw. Testversion verwendet. Die Anfragen an diese zwei Systeme sind wie folgt limiert:

- Meisterplan: 60 Anfragen pro Minute [23]. Da der Anwendungsfall UC-01 Zugriff auf Meisterplan benötigt, wird dieser 50 Mal ausgeführt, um einen Mittelwert zu errechnen.
- MsGraph:
  - Mit der Outlook-Kalender-Ereignisse-URL (siehe Anhang Tabelle 20 Position 3) werden die Kalender Termine der Mitarbeiter ermittelt. Dies wird sowohl für die Slack als auch für die Microsoft Teams Bot Implementierung bei der Ausführung des Anwendungsfalls UC-02 verwendet. Die Anzahl der Anfragen ist auf 10.000 Stück in 10 Minuten begrenzt [29]. Im Testmandanten gibt es 17 Benutzer und für jeden wird einmal die genannte Anfrage URL aufgerufen. Somit kann die UC-02 Anfrage in 10 Minuten ungefähr 588,24 Mal getätigt werden ohne limitiert zu werden. In diesem Experiment wird der Anwendungsfall UC-02 50 Mal ausgeführt, um einen Mittelwert zu errechnen.
  - Mit der Teams-Alle-Nachrichten-URL (siehe Anhang Tabelle 20 Position 4), die bei der Implementierung des Microsoft Teams Bots für den Anwendungsfall UC-03 benötigt wird, dürfen pro Monat 500 Nachrichten ermittelt werden [32]. Da in dem Testfall 22 Nachrichten insgesamt gefunden werden, entstehen pro Ausführung des Anwendungsfalls UC-03 Kosten ihn Höhe von 22 Nachrichten. Folglich können im Monat 22 UC-03 Anfragen gesendet werden. Aufgrund dieser niedrigen monatlichen Anfragengrenze wird der Test zum UC-03 für die Microsoft Teams Bot Implementierung 10 Mal wiederholt, um einen Mittelwert zu errechnen.

Zeitverhalten - Antwortzeit Um die Antwortzeit zu ermitteln, werden die Development Tools im Microsoft Edge Browser verwendet. Diese beinhalten ein Werkzeug, mit welchem die Zeit in Millisekunden gemessen wird, die für Anfragen über den Browser benötigt wird. Sowohl das Senden der Nachricht als auch das Empfangen der Antwort erzeugt eine Anfrage im Browser. Durch das Subtrahieren der Empfangszeit der Startnachricht von der Empfangszeit der Antwort wird die Antwortzeit ermittelt also die Zeit, die ab dem Absetzen der Nachricht bis zum Empfangen der Antwort vergangen ist. Der Ablauf ist wie folgt:

1. Slack im Brwoser Mmicrosoft Edge öffnen.
2. Die Development Tools öffnen, Diese werd standardmäßig mit der Taste F12 gemacht.
3. In dem neu geöffneten Fenster, das zu den Development Tools gehört, den Reiter *Netzwerk* anklicken.
4. Ein Häckchen bei der Einstellung *Cache deaktivieren* setzen.
5. In Slack zum Nachrichten Kanal mit dem Bot navigieren.
6. Die Nachricht für den Anwendungsfall UC-01 abschicken.
7. In dem Netzwerk Reiter der Deveopment Tools wurde eine Slack-PostMessage-URL (siehe Anhang Tabelle 20 Position 5) erzeugt.
8. Sobald die Antwort eintrifft, erscheint im Netzwerkreiter ein Eintrag mit der Slack-Bot-Antwort-URL (siehe Anhang Tabelle 20 Position 6).
9. Nun wird auf den ersten genannten Eintrag, also die Slack-PostMessage-URL, geklickt und zum Reiter *Timing* navigiert.
10. Die Zeit, die als Beginn markiert ist, wird als Startzeit notiert.
11. Nun wird auf den zweiten genannten Eintrag, also die Slack-Bot-Antwort-URL, geklickt und zum Reiter *Timing* navigiert.
12. Die Zeit, die als Beginn markiert ist, wird als Endzeit notiert.
13. Nun wird die notierte Startzeit von der notierten Endzeit subtrahiert. Das Ergebnis ist die Antwortzeit.
14. Die errechnete Antwortzeit wird notiert.

15. Die Schritte 6 - 14 werden 20 mal wiederholt.

16. Dann werden alle 20 notierten Antwortzeiten addiert und die Summe wird durch 20 dividiert. Das Ergebnis ist die durchschnittliche Antwortzeit für den Anwendungsfalls UC-01.

Die Schritte 1 bis 16 werden für die Anwendungsfälle UC-02 und UC-03 wiederholt, um deren Antwortzeiten ebenfalls zu ermitteln. Im Anschluss wird das gleiche für den Microsoft Teams Bot gemacht. Hier ändern sich jedoch die beiden Anforderungs-URLs, die den Start und das Ende der Antwortzeit markieren. Die Slack-PostMessage-URL, die in Schritt 7 beim Absenden der Nachricht entsteht und in Schritt 9 untersucht wird, wird durch die Teams-PostMessage-URL (siehe Anhang Tabelle 20 Position 7) ersetzt. Die Slack-Bot-Antwort-URL, die in Schritt 8 beim Empfangen der Bot Antwort entsteht und in Schritt 11 untersucht wird, wird durch die Teams-Bot-Antwort-URL (siehe Anhang Tabelle 20 Position 8) ersetzt.

Zeitverhalten - Verarbeitungszeit Um die Verarbeitungszeit zu messen, wird bei jeder Bot Implementierung ein Zeitstempel an die Stelle gesetz, an der die Nachrichtenverarbeitung beginnt. Anschließend wird für jeden Anwendungsfall eine Anfrage wiederholt an den Slack und Microsoft Teams Bot gesendet und der Mittelwert errechnet.

Ressourcennutzung - CPU - Zeit und Nutzung Die JavaScript Laufzeitumgebung Nodejs bietet im Modul *process* die Funktion *cpuUsage* an, die die Zeit in Mikrosekunden misst, während der die CPU für den aktuellen Prozess benutzt wurde [45]. Des Weiteren wurde das Node Package Manager Modul *node-os-utils* [51] verwendet, um die prozentuale Nutzung der CPU zu berechnen. Diese Software wird vor Beginn der Nachrichtenverarbeitung durch die Bots und nach dem Senden der Antwort aufgerufen, um die benötigte CPU Zeit und den prozentualen Verwendungsanteil für die Verarbeitung zu berechnen. Für jeden Anwendungsfall wird eine Anfrage wiederholt an den Slack und Microsoft Teams Bot gesendet und der Mittelwert wird aus den Ergebnissen errechnet.

Ressourcennutzung - Arbeitsspeicher Hier wurde ebenfalls das *process* Modul verwendet, um mit der Funktion *memoryUsage* [46] die Anzahl an Bytes zu ermitteln, die für die Ausführung des Prozesses verwendet wurden. Durch mehrfaches Ausführen der Anwendungsfälle wird ein Mittelwert für den genutzten Arbeitsspeicher pro Anwendungsfall ermittelt.

Ausführung der Anwendungsfälle Jeder Anwendungsfall wird mehrfach ausgeführt, damit aus den Ergebnissen der vorherigen Messungen ein Mittelwert errechnen werden kann. Dafür bieten beide Bots die Möglichkeit eine Anfrage mehrfach auszuführen. Dies funktioniert jedoch nicht für die Messung der Antwortzeit, da es in diesem Fall erforderlich ist, dass der Anwendungsfall durch das Senden einer Nachricht im Client beginnt. Die Verarbeitungszeit und die Ressourcennutzung jedoch erfordert lediglich, dass die Prozesse auf dem Server ausgeführt werden und somit kann das wiederholte Ausführen einer Nachricht durch den Nachrichtenpräfix "Experiment - Zahl:" ausgelöst werden. Das Wort *Zahl* muss durch eine ganze Zahl ersetzt werden. Wird dies getan, dann wird die Nachricht, sooft wiederholt, wie die Zahl, die hinter dem Doppelpunkt steht, dies vorgibt. Im Folgenden werden die Nachrichten gezeigt, die zum Testen der Verarbeitungszeit und der Ressourcennutzung verwendet wurden:
UC-01 Anfrage Text:

Experiment - 50: Wer hat Erfahrung mit Oracle und im Dezember noch 5 Tage Zeit (9)

UC-02:

Experiment - 50: Nenn mir 3 Termine im Dezember an denen Laura Jurgeneit 2 Stunden Zeit hat (10)

UC-03 Anfrage Text für Slack:

Experiment - 50: Wurde das Thema SAP besprochen (11)

UC-03 Anfrage Text für Microsoft Teams:

Experiment - 10: Wurde das Thema SAP besprochen (12)

Die niedrige Wiederholungszahl bei Microsoft Teams im UC-03 ist durch die Anfragelimitierung von MsGraph bedingt. Da Slack keinen Zugriff auf MsGraph für UC-03 benötigt, kann dort das Experiment öfter durchgeführt werden.

Kapazität - Nachrichtenmenge und Nachrichtenlänge Diese Informationen werden aus den Dokumentationen zu Slack und Microsoft Teams entnommen.

Kategorie	Testgegenstand	Slack UC-01	Microsoft Teams UC-01
Zeitverhalten	Antwortzeit	1,46 s	1,85 s
	Verarbeitungszeit	0,9 s	0,96 s
Ressourcennutzung	CPU-Nutzung	1,18 %	3,64 %
	Arbeitsspeicher-Nutzung	21,92 mb	48,61 mb

Tabelle 14: Alle Ergebnisse für den Anwendungsfall UC-01 des Experiments zur Effizienz

6.2.5 Ergebnisse

In diesem Abschnitt werden die Ergebnisse zu den Effizienz Experimenten bezüglich Zeitverhalten, Ressourcennutzung und Kapazität vorgestellt.

Zeitverhalten und Ressourcennutzung
Die Ergebnisse, die bezüglich des Anwendungsfalls UC-01 durch dieses Experiment erzielt wurden, sind in Tabelle 14 beschrieben. Dort ist zu erkennen, dass sowohl im Zeitverhalten als auch bei der Ressourcennutzung die Werte bei Microsoft Teams größer sind als bei Slack. Jedoch sind die Unterschiede beim Zeitverhalten minimal. Slack ist bei der durchschnittlichen Antwortzeit 0,39 Sekunden (ungefähr 21 %) und bei der Verarbeitungszeit nur 0,06 Sekunden (ungefähr 6 %) schneller. Trotz der niedrigen Differenz der CPU-Nutzung von nur 2,46 % ist im Verhältnis der Wert von Microsoft Teamsmit 3,64 % ungefähr drei mal so hoch wie der Wert von Slack mit 1,18 %. Ebenso ist die Arbeitsspeicher-Nutzung bei Microsoft Teams mehr als doppelt so hoch wie die bei Slack.

Die Ergebnisse, die bezüglich des Anwendungsfalls UC-02 durch dieses Experiment erzielt wurden, sind in Tabelle 15 beschrieben. Es ist zu erkennen, dass sowohl im Zeitverhalten als auch bei der Ressourcennutzung für Slack niedrigere Werte gemessen wurden. Die Antwortzeit von Slack ist im Schnitt 0,31 Sekunden (ungefähr 19 %) schneller als die von Microsoft Teams. Bezüglich der Verarbeitungszeit ist Slack 0,05 Sekunden (ungefähr 5 %) schneller. Diese Differenzen sind allerdings recht gering. Die Werte CPU- und Arbeitsspeicher-Nutzung sind sowohl bei Slack als auch bei Microsoft Teams gering. Jedoch misst Slack auch hier niedrigere Werte, die bei der CPU-Nutzung einem Drittel und bei der Arbeitsspeicher-Nutzung der Hälfte der Microsoft Teams Messwerte entsprechen.

Kategorie	Testgegenstand	Slack UC-02	Microsoft Teams UC-02
Zeitverhalten	Antwortzeit	1,33 s	1,64 s
	Verarbeitungszeit	0,87 s	0,92 s
Ressourcennutzung	CPU-Nutzung	1,21 %	3,79 %
	Arbeitsspeicher-Nutzung	21,21 mb	48,05 mb

Tabelle 15: Alle Ergebnisse für den Anwendungsfall UC-02 des Experiments zur Effizienz

Kategorie	Testgegenstand	Slack UC-03	Microsoft Teams UC-03
Zeitverhalten	Antwortzeit	2,49 s	4,68 s
	Verarbeitungszeit	1,9 s	3,55 s
Ressourcennutzung	CPU-Nutzung	1,88 %	2,57 %
	Arbeitsspeicher-Nutzung	21,47 mb	45,9 mb

Tabelle 16: Alle Ergebnisse für den Anwendungsfall UC-03 des Experiments zur Effizienz

Die Ergebnisse, die bezüglich des Anwendungsfalls UC-03 durch dieses Experiment erzielt wurden, sind in Tabelle 16 beschrieben. Es ist zu erkennen, dass sowohl das Zeitverhalten als auch die Ressourcen-Nutzung bei Microsoft Teams größere Werte gemessen hat als Slack. Es fällt auf, dass sowohl die Antwort- als auch die Verarbeitungszeit sehr hoch sind, da sie zwischen 1,9 und 4,68 Sekunden liegen. Die CPU-Nutzung ist bei Slack und Microsoft Teams sehr ähnlich. Dennoch benötigt Slack im Schnitt 0,36 % mehr CPU als Microsoft Teams, was im Verhältnis einer Differenz von ungefähr 19 % entspricht. Bei der Arbeitsspeicher-Nutzung benötigt Microsoft Teams wiederum ungefähr doppelt so viele Ressourcen wie Slack.

Kapazität

In diesem Abschnitt werden die Ergebnisse bezüglich der Kapazitätsmerkmale Nachrichtenanzahl, Nachrichtenlänge und API Anfragen präsentiert.

Aus Tabelle 17 ergibt sich, dass die Obergrenze bei Microsoft Teams für das Senden von Nachrichten im gesamten System 50 Mal so hoch ist, wie die von Slack. Wenn es um direkte Nachrichten zwischen dem Bot und einem Nutzer geht, macht Microsoft Teams mehrere Abstufungen, die sich über einen Zeitraum von mehr als einer Sekunde erstrecken. Wird aus diesen Abstufungen der Durchschnitt berechnet, der in "Nachricht pro Sekunde" angegeben wird, dann kann der Microsoft Teams Bot im schlechtesten Fall zwei Nachrichten pro Sekunde senden, was immer noch das doppelte von dem ist, was

Nachrichtenanzahl	
Slack	**Microsoft Teams**
App darf eine Nachricht pro Sekunde senden. Kurzzeitig sind Überschreitungen der Grenzen erlaubt. Dazu macht Slack jedoch keine weiteren Angaben. [56]	Ein Bot darf maximal 50 Nachrichten pro Sekunde im ganzen Mandanten versenden. Pro Konversation darf er in einer Sekunde sieben, in zwei Sekunden acht und in 30 Sekunden 60 Nachrichten senden. Eine Konversation beschreibt den Nachrichtenaustausch zwischen Bot und einem Nutzer, einer Gruppenunterhaltung oder einem Kanal. [31]

Tabelle 17: Grenzen für das Senden von Nachrichten

Slack erlaubt. Für die spätere Diskussion dieser Ergebnisse ist jedoch zu berücksichtigen, dass Slack für einen unbekannten Zeitraum das Überschreiten der Grenzen erlaubt. Ob diese ähnlich zu den Abstufungen von Microsoft Teams sind, ist jedoch nicht bekannt.

Aus Tabelle 18 ergibt sich, dass Nutzer von Microsoft Teams größere Nachrichten versenden dürfen, als die Nutzer von Slack. Bots hingegen dürfen bei Slack größere Nachrichten versenden als die von Microsoft Teams. Sollte jedes Zeichen in der Nachricht genau ein Byte belegen, dann haben beide Bots genau die gleiche Nachrichtengrößengrenze. Für die spätere Diskussion ist zu breücksichtigen, dass Microsoft Teams nicht erläutert, wie sich die Größe von 40 KB errechnet. Es besteht die Möglichkeit, dass genau wie bei Slack 40.000 Zeichen erlaubt sind und bei der Berechnung der Angabe von 40 KB davon ausgegangen wurde, dass jedes Zeichen genau ein Byte belegt. In diesem Fall hätten beide Plattformen die gleichen Grenzen für die Länge von Botnachrichten gesetzt. Des Weiteren macht Slack genauere Angaben zu dem speziellen Nachrichten Format *blocks*. Diese werden in den Antworten aller Anwendungsfälle UC-01 bis UC-03 verwendet und sind somit ebenfalls relevant. Microsoft Teams hat keine *blocks*. Stattdesssen wurden in den Antworten von Microsoft Teams Tabellen verwendet. Jedoch macht Microsoft Teams keine Angaben zu den Grenzen von Zeilen, die pro Tabelle erlaubt sind und folglich sind diese zwei Werte nicht vergleichbar. Es bleibt somit nur der Vergleicht der Gesamtgrößen der Nachrichten je Plattform.

Nachrichtenlänge	
Slack	**Microsoft Teams**
Nutzer dürfen Nachrichten mit maximal 4.000 Zeichen versenden. Ein Zeichen wird durch 1 bis 4 Bytes beschrieben, somit beträgt die maximale Größe einer Nachricht zwischen 4 und 16 KB. [56]. Bots hingegen dürfen Nachrichten mit bis zu 40.000 Zeichen senden [55] und haben somit ein Limit von 40 bis 160 KB. Sind Blöcke in der Nachricht enthalten, so dürfen von diesen maximal 50 Stück verwendet werden [57]. Die Nachrichten haben noch ein Feld *attachments*, welches jedoch veraltet ist und von Slack API nicht mehr zur Verwendung empfohlen wird [58]. Aus diesem Grund wurde dieses nicht in dieser Auswertung einbezogen.	Die Nachrichtengröße ist für Bots auf 40 KB limitiert [25]. Ansonsten ist die Größe auf ungefährt 28 KB beschränkt [27].

Tabelle 18: Grenzen der Nachrichtenlängen

7 Diskussion

In diesem Abschnitt werden die Ergebnisse der Experimente zur Wartbarkeit und zur Effizienz diskutiert. Des Weiteren werden die Kosten zum Betreiben eines Bots analysiert, sowie die Programmiersprachen, die für die Implementierung zur Verfügung stehen. Das Ziel der Diskussion ist es, einzuordnen, ob der Slack oder der Microsoft Teams Bot besser geeignet ist diese Anwendungsfälle UC-01 bis UC-03 abzuarbeiten.

7.1 Ergebnisse der Experimente

In Kapitel 6 wurden Experimente durchgeführt, die den Code, der für beide Plattformen implementiert wurde, auf seine Wartbarkeit untersucht. Des Weiteren wurden Experimente durchgeführt, die während der Ausführung der Anwendungsfälle die Effizient gemessen haben. Die Ergebnisse dieser beiden Experiemente werden in diesem Abschnitt diskutiert.

7.1.1 Wartbarkeit

Im Wartbarkeitsexperiment wurde der Code analyisiert, der nötig ist, um einen Slack bzw. einen Microsoft Teams Bot zu betreiben. Dabei wurde je Plattform zwischen dem Framework Code und dem Code unterschieden, der explizit für die Anwendungsfälle implementiert wurde. Letzterer wird im Folgenden auch *spezifischer Code* genannt. Der Grund dafür ist, dass der Framework und der spezifische Code unterschiedliche Aufgaben haben. Das Framework wird vom Entwickler als Werkzeug verwendet, um sein eigentliches Ziel, also die Verarbeitung von Anwendungsfällen, schneller zu erreichen. Das bedeutet, dass der Framework Code allgemeine Funktionalitäten zur Verfügung stellt, die für verschiedene Arten von Bots nützlich sein können. Der spezifische Code hingegen wurde nur implementiert, um die vorgegeben Anwendungsfälle verarbeiten zu können und ist somit nur in diesem speziellen Kontext verwendbar. Des Weiteren ist der Entwickler des spezifischen Codes nicht der Autor des Frameworks und somit kennt der Entwickler den eigens entwickelten Code besser als den Framework Code. Somit gibt es zwei unterschiedliche Codebasen pro Plattform, die unterschiedliche Entwickler haben und unterschiedliche Aufgaben erfüllen, die getrennt von einander analysiert werden können, um die Diskussionsfrage spezifischer beantworten zu können.

Die Experimentergebnisse werden nun für jeden einzelnen Messparameter diskutiert und am Ende wird, basierend auf der vorherigen Diskussion, eine Einschätzung abgeleitet, die das Ziel verfolgt die Diskussionsfrage zu beantworten. Bei den Messparametern handelt es sich um die **Source Lines of Code** (SLoC), die **Codezeilen-Duplikate**, das **Halstead-Volumen**, die **zyklomatische Komplexität** und den **Maintainability Index**. Abgesehen vom zuletzt genannten Parameter wird für jeden Messparameter noch der Durchschnittwert in die Diskussion mit einbezogen. Da sich die Frameworks von den spezifischen Implementierung hinsichtlich des Umfangs stark voneinander unterscheiden, wird nur das Slack Bot Framework mit dem Microsoft Teams Bot Framework verglichen und der spezifische Code der Slack Bot Implementierung wird nur mit der Microsoft Teams Bot Implementierung verglichen. Das bedeutet, dass die Diskussionsfrage einmal in Bezug auf die Wahl des besseren Frameworks als auch auf die Wahl der spezifischen Bot Implementierung beantwortet wird.

Source Lines of Code

Bezüglich der Source Lines of Code ist im Vorhinein zu erwähnen, dass die Auswirkung der Source Lines of Code auf die Analysierbarkeit und die Änderbarkeit nur implizit

sind. Das bedeutet, dass ein hoher Wert bei den Source Lines of Code auf schlechte Analysierbarkeit und Änderbarkeit hinweist aber kein Beweis dafür ist. Denn auch wenn ein Programm viele Source Lines of Code umfasst, so kann er beispielsweise immer noch gut änderbar sein, wenn dieser unter anderem Aufgaben in Methoden kapselt und es somit nur an einer Stelle einer Änderung bedarf. Nichts desto trotz ist die Länge des Codes ein gutes Indiz, um Rückschlüsse auf die Analysierbarkeit und Änderbarkeit zu ziehen, da potenziell der gesamte Code untersucht werden muss, um sicher zu stellen, dass eine Änderung an einer Stelle an keiner anderen Stelle zu Fehlern führt.

Bei den gesamten Source Lines of Code besitzt das Microsoft Teams Bot Framework mit 114.641 Zeilen die meisten Zeilen. Somit ist dieses Programm das umfangreichste und am schlechtesten zu bewerten, denn dies impliziert, dass die Analysierbarkeit und folglich auch die Veränderbarkeit im Vergleich zum Slack Bot Framework aufwändiger ist, da es potenziell 114.641 Zeilen zu überprüfen gilt. Das Slack Bot Framework besitzt mit 6.842 Source Lines of Code die zweit meisten Zeilen. Dies entspricht im Vergleich zu den Source Lines of Code des Microsoft Teams Bot Framework ungefähr 6 % und ist somit um ein vielfaches kleiner und folglich besser zu bewerten. Auch die durchschnittlichen Source Lines of Code pro Datei sind bei dem Slack Bot Framework mit ungefähr 20 Zeilen fast 8 Mal geringer als bei dem Microsoft Teams Bot Framework, welche im Schnitt ungefähr 157 Source Lines of Code pro Datei besitzen. Somit ist auch in diesem Fall das Slack Bot Framework besser zu bewerten, da der Aufwand zur Analyse und bei Veränderung dadurch vermutlich geringer ist. Daraus ergibt sich, dass in Bezug auf die Source Lines of Code und somit implizit aus Gründen der Analysierbarkeit und der Änderbarkeit das Slack Bot Framework eine bessere Wahl ist als das Microsoft Teams Bot Framework.

Bei der spezifischen Implementierung der Bots besitzt die Microsoft Teams Implementierung mit einem Wert von 388 mehr Source Lines of Code als Slack. Die Slack Bot Implementierung besitzt mit 234 Source Lines of Code ungefähr 40 % weniger Zeilen. Auch bei den durchschnittlichen Zeilen pro Datei schneidet Microsoft Teams schlechter ab als Slack. Zwar sind 388 Zeilen insgesamt oder 43 Zeilen pro Datei noch gut überschaubar für einen Menschen, da es sich hier jedoch um einen direkten Vergleich handelt, kann dennoch die Aussage getroffen werden, dass die Slack in Bezug auf die Source Lines of Code und somit implizit aus Gründen der Analysierbarkeit und der Änderbarkeit eine bessere als Microsoft Teams ist, um einen Bot zu implementieren, der die Anwendungsfälle abdeckt, die in dieser Ausarbeitung umgesetzt wurden.

Somit ergibt sich, dass sowohl das Framework als auch die Bot Implementierung von

Slack besser geeignet sind als die von Microsoft Teams, wenn aufgrund der Source Lines of Code Rückschlüsse auf die Analysierbarkeit und Änderbarkeit eines Programms gezogen werden.

Codezeilen-Duplikate

Bezüglich der Codezeilen-Duplikate ist zu erkennen, dass bei der spezifischen Bot Implementierung sowohl Slack als auch Microsoft Teams keinen duplizierten Code besitzen. Dies ist für beide positiv zu bewerten, da jedoch beide den exakt gleichen Wert haben, ist dies für einen Vergleich nicht relevant, da hier keiner besser als der andere bewertet werden kann. Folglich sind die Codezeilen-Duplikate der Frameworks nun entscheidend dafür, ob Slack oder Microsoft Teams besser in der Wartbarkeit in Bezug auf die Modularität und Änderbarkeit zu bewerten ist. Aus den Experimentergebnissen ist abzulesen, dass das Microsoft Teams Bot Framework weitaus mehr Codezeilen-Duplikate aufweist als Slack. Da das Microsoft Teams Bot Framework aber auch weitaus mehr Codezeilen als Slack hat, könnte angenommen werden, dass diese hohe Zahl aufgrund der hohen Codezeilen entsteht. Folglich muss noch der Wert betrachtet werden, der zeigt, wie viel von dem gesamten Code aus Duplikaten besteht. Bei Slack liegt dieser Wert mit 9 % fast bei der Hälfte vom Microsoft Teams Anteil, der bei 17 % liegt. Somit hat das Microsoft Teams Bot Framework nicht nur ingesamt mehr Codezeilen-Duplikate, sondern auch im Verhältnis zur Gesamtcodemenge einen größeren Anteil an Duplikaten. Daraus ergibt sich, dass in Bezug auf die Modularität und Änderbarkeit Slack die bessere Wahl ist.

An dieser Stelle ist jedoch zu erwähnen, dass die Bewertung der Modularität aufgrund von Codezeilen-Duplikaten nur eine eingeschränkte Sicht auf diese Eigenschaft bietet, da so nicht exakt untersucht wird, ob eine Änderung möglichst wenig Auswirkung auf andere Codefragmente hat, sondern es wird nur untersucht, dass möglichst wenig Änderungen an verschiedenen Codestellen für die gleiche logische Änderung gemacht werden müssen. Wie in Abschnitt 5.1.2 erklärt, gibt es bereits Algorithmen, die versuchen, die Modularität eines Programms zu errechnen, jedoch würde die Implementierung jener den Umfang der Ausarbeitung übersteigen und aufgrund ihrer Komplexität würden sie auch fehleranfällig bei der Implementierung sein. Folglich sind die Codezeilen-Duplikate das einzige Maß, das zur Verargumentierung der Modularität verwendet werden kann.

Halstead-Volumen

Bei den Halstead-Volumen der Frameworks ist zu erkennen, dass Microsoft Teams einen

sehr viel größeren Wert als Slack erzielt, wodurch dieses negativer zu bwerten ist. Da das Halstead-Volumen sich jedoch aus der Anzahl der Operatoren und Operanden zusammensetzt und diese Zahl mit der Zahl der Gesamtcodezeilen steigt, kann als Alternative noch das Halstead-Volumen pro Zeile untersucht werden. Doch auch dort weist Microsoft Teams einen höheren Wert auf und ist damit nach wie vor schlechter zu bewerten als Slack. Bei der spezifischen Implementierung weichen die Werte bezüglich des Halstead-Volumen weniger von einander ab. Während Microsoft Teams einen größeren Insgesamtwert besitzt, der unter anderem der höheren Zahl der Codezeilen geschuldet ist, so ist er im Schnitt pro Codezeile kaum höher als der von Slack. Nichts desto trotz erzielt Slack auch bei der spezifischen Implementierung sowohl beim Insgesamtwert als auch beim Durchschnittswert ein besseres Ergebnis. Ähnlich wie bei den Source Lines of Code wird auch das Halstead-Volumen benutzt, um die Änderbarkeit und Analysierbarkeit zu bewerten und anhand der Ergebnisse ist zu erkennen, dass Slack in Bezug auf diese beiden Eigenschaften eine bessere Wahl für die Implementierung des Bots ist.

Zyklomatische Komplexität

Bei der zyklomatischen Komplexität ist der Gesamtwert des Microsoft Teams Bot Framework wieder sehr viel höher als der des Slack Bot Frameworks. Hier ist jedoch wieder zu berücksichtigen, dass dies akkumulierte Werte sind, die für den gesamten Code summiert wurden. Werden diese beiden Werte jedoch durchschnittlich auf jede Zeile verteilt, so ist der Unterschied nur noch minimal und beträgt nur noch ungefähr 0,09 was nicht einmal mehr 3 % entspricht.

Wie in Abschnitt 5.1.3 erklärt, kann der Wert der zyklomatischen Komplexität genutzt werden, um die Anzahl der Testfälle zu bennen, die nötig sind, um eine vollständige Zweig- und Anweisungsabdeckung zu erreichen. Somit ist auch der akkumilierte Gesamtwert zu verwenden, um die Testbarkeit zu bewerten. In diesem Fall ist das Slack Bot Framework deutlich besser zu bewerten als das Microsoft Teams Bot Framework, da letzteres im Vergleich zum ersteren ungefähr das 16-fache an Testfällen benötigen würde. Der Durchschnittswert ist geeignet, um die Wiederverwendbarkeit zu bewerten, da hier die durchschnittliche Komplexität pro Zeile berechnet wird.

Hier ist zu erwähnen, dass diese Bewertungsmethode einen Mangel aufweist. Denn die zyklomatische Komplexität wird für jede Funktion und nicht für jede Zeile berechnet. Allerdings wurde in den Experimenten nicht die Anzahl der Funktionen gezählt und somit wird hier auf die Anzahl der Codezeilen als nächste Alternative zurückgegriffen.

Ein weiterer Nachteil dieser Bewertungsmethode ist, dass die Wiederverwendbarkeit gut bewertet wird, wenn die Komplexität niedrig ist, da angenommen wird, dass eine niedrige Komplexität eine hohe Wiederverwendbarkeit bedeutet. Eine Studie von Anguswamy und Frakes [1] hat ergeben, dass je höher die Komplexität einer Komponente ist, wobei die Komplexität von der Zahl der Source Lines of Code abhängt, desto schwerer ist es, diese wieder zu verwenden. Das Ergebnis dieser Studie ist den Autoren nach jedoch statistisch nicht aussagekräftig. Mit diesen aufgezählten Schwächen ist das Ergebnis dieser Bewertungsmethode nur als Indikator zu verstehen.

Wie bereits erwähnt, ist der Unterschied zwischen der durchschnittlichen Komplexität beider Frameworks sehr gering. Das Slack Bot Framework hat einen minimal besseren Wert erzielt als das Microsoft Teams Bot Framework. Da der Unterschied jedoch so gering ist, wird hier kein Framework besser als das andere bewertet und somit eignen sich beide Frameworks gut, um den Bot zu implementieren, der die Anwendungsfälle dieser Ausarbeitung abdeckt, wenn in Bezug der Wartbarkeit auf die Wiederverwendbarkeit geachtet wird.

Bei der spezifischen Implementierung der Bots herrscht eine größere Differenz bezüglich der durchschnittlichen zyklomatischen Komplexität und die Slack Bot Implementierung hat dabei den besseren Wert ergeben. Folglich ist Slack in Bezug auf die Wiederverwenbarkeit die bessere Wahl, um einen Bot zu implementieren, der die Anwendungsfälle, die in dieser Ausarbeitung beschrieben wurden, bedient.

Zusammengefasst ist Slack sowohl in Bezug auf die Testbarkeit als auch auf die Wiederverwendbarkeit die bessere Wahl, um einen Bot zu implementieren, der die genannten Anwendungsfälle bedienen kann. Es ist jedoch zu erwähnen, dass Microsoft Teams bei der Wiederverwendbarkeit ebenfalls gute Werte erzielt hat und Slack in diesem Punkt sehr ähnlich ist.

Maintainability Index

Der Maintainability Index der Frameworks liegt mit einer Differenz von 3 % recht nahe beieinander. Wie in Kapitel 5.1.5 beinhaltet der Maintainability Index die Metriken Halstead Metrik, zyklomatische Komplexität und Source Lines of Code und soll die allgemeine Wartbarkeit eines Programms bewerten. Aufgrund der beinhalteten Metriken, werden Rückschlüsse auf die Wartbarkeitseigenschaften *Wiederverwendbarkeit*, *Analysierbarkeit*, *Veränderbarkeit* und *Testbarkeit* gezogen, da diese durch die beinhalteten

Metriken ebenfalls bewertet werden können. Aus den Erperimentergebnissen ist abzulesen, dass sowohl bei dem Framework als auch bei der spezifischen Implementierung Slack bessere Werte ergeben hat. Microsoft Teams liefert jedoch sehr ähnlich Werte, die nur um wenige Prozent schlechter sind.

Zusammenfassung
Zusammenfassend ist festzustellen, dass Slack insgesamt eine bessere Wahl für die Implementierung des Bots in Bezug auf die Wartbarkeit ist. Auch wenn die Ergebnisse von Microsoft Teams in manchen Punkten nahe an denen von Slack heran kamen, so hat Slack dennoch in jedem Vergleich die besseren Ergebnisse erzielt. Aus diesem Grund kann gesagt werden, dass Slack allgemein im Punkt der Wartbarkeit die bessere Wahl ist. Nichts desto trotz hat auch Microsoft Teams gute Wartbarkeitsbewertungen erzielt, es ist jedoch durch seinen großen Umfang vor allem in der Veränderbarkeit, Analysierbarkeit und auch Testbarkeit Slack unterlegen. Dies bezieht sich allerdings hauptsächlich auf das Microsoft Teams Bot Framework. Die spezifische Microsoft Teams Bot Implementierung hingegen ist im Vergleich zur Slack Bot Implementierung nur geringfügig größer. Aus diesem Grund bietet Microsoft Teams eine gute Alternative zu Slack.

7.1.2 Effizienz

Im Effizienzexperiment wurden während der Ausführung der Anwendungsfälle UC-01 bis UC-03 das Zeitverhalten und die Ressourcennutzung gemssen. Die Daten zur Kapazität wurden den Angaben entnommen, die auf den Webseiten der Plattformen zu finden sind. Die Ergebnisse dieser drei Testparameter Zeitverhalten, Ressourcennutzung und Kapazität werden in diesem Abschnitt einzeln diskutiert. Am Ende wird, basierend auf der vorherigen Diskussion, eine Einschätzung abgeleitet, die das Ziel verfolgt die Diskussionsfrage, ob Slack oder Microsoft Teams besser geeignet ist, um einen Bot mit den genannten Anwendungsfällen zu implementieren und zu betreiben, zu beantworten.

Zeitverhalten
Bei den Ergebnissen des Zeitverhaltens ist zu erkennen, dass in jedem Anwendungsfall Slack bessere Werte erzielt hat. Allerdings befinden sich Werte beider Plattformen in einem akzeptablen Bereich, da sie kleiner als 10 Sekunden sind, dennoch liegt keiner von ihnen unter einer Sekunde, was zu einer negativen Bewertung führt. Es ist zu bemerken, dass die Differenz der Antwortzeiten zu UC-01 und UC-02 zwischen Slack und Microsoft

Teams nicht einmal bei einer halben Sekunde liegt. Folglich sind die Unterschiede hier minimal. In Bezug auf die Verarbeitungszeit sind die Differenzen noch geringer und die Werte weichen weniger als 0,1 Sekunden von einander ab. Dies lässt darauf schließen, dass die implementierten Prozesse zur Verarbeitung der Anwendungsfälle UC-01 und UC-02 mit ähnlicher Geschwindigkeit ausgeführt werden. Aufgrund der minimalen Differenzen in der Verarbeitungszeit, die weniger als 0,1 Sekunden betragen, wird weder Slack noch Microsoft Teams als besser oder schlechter bewertet. Denn bei der Messmethode, in welcher aus 50 Iterationen der Mittelwert gebildet wird, ist es möglich, dass ein Wert, der aufgrund von Störungen höher ausgefallen ist, das Gesamtergebnis sehr stark beeinflusst hat und somit für eine minimale Erhöhung verantwortlich ist.

Bei der Antwortzeit weist Microsoft Teams größere und somit schlechtere Werte auf als Slack. Dies inkludiert jedoch auch die Verarbeitungszeit. Wird von der Antwortzeit die Verarbeitungszeit subtrahiert, so ergibt dies die Zeit, die der Bot benötigt, um nach Verarbeitung der Anfrage die Antwort zu senden. Diese Zeit wird im folgenden Sendezeit genannt. In der Sendezeit ist nicht nur die Zeit enthalten, die benötigt wird, um die Datenpakete, die die Nachricht enthalten, vom Bot-Server zum Nutzer-Client zu transportieren, sondern es können davor auch noch weitere Verarbeitungen im Framework geschehen, die als Hilfswerkzeug benutzt wurden, um den Bot zu implementieren. Auch bei der Sendezeit misst Microsoft Teams höhere Werte als Slack. Dies bedeutet, dass der Microsoft Teams Bot nach dem Verarbeiten der Anfrage in jedem Fall mehr Zeit benötigt, um die Antwort an den Nutzer zu senden, als der Slack Bot. Aus diesem Grund wird der Microsoft Teams Bots in Bezug auf die Antwortzeit schlechter bewertet als der Slack Bot.

Eine Besonderheit bei den Ergebnissen ist das Zeitverhalten des UC-03. Während die anderen beiden Anwendungsfälle eine Verarbeitungszeit von unter einer Sekunde und eine Antwortzeit von unter zwei Sekunden haben, so übersteigt UC-03 diese Werte. Zwar wird die Grenze von 10 Sekunden nicht überschritten, jedoch ist fraglich, ob dies nur der Testdatenmenge geschuldet ist. Denn wenn die Verarbeitungszeit linear zu der Menge der Nachrichten, die in diesem Anwendungsfall gefiltert werden sollen, skaliert, dann können die 10 Sekunden überschritten werden. Des Weiteren ist auffällig, dass die zeitlichen Differenzen zwischen den Slack und Microsoft Teams Ergebnissen ebenfalls größer sind. Microsoft Teams benötigt fast die doppelte Verarbeitungs- und Antwortzeit als Slack. Dies könnte der Tatsache geschuldet sein, dass das Microsoft Teams Bot Framework keine Schnittstelle anbietet, um alle Nachrichten aus einem Team zu liefern, in dem der Nutzer die Anfrage nicht gestellt hat. Denn der Nutzer stellt diese Anfrage im privaten

Nachrichtenkanal mit dem Bot und nicht öffentlich in einem Team. Selbst wenn der Nutzer dies öffentlich in einem Team tun würde, so bestünde das gleiche Problem immer noch, da die Schnittstelle nur alle Nachrichten aus diesem Team zur Verfügung stellen könnte, aber nicht aus allen anderen Teams, in denen der Nutzer ein Mitglied ist. Somit muss der Bot Entwickler selber über eine HTTPS Anfrage an den Microsoft Graph API Service stellen, um für jedes Team alle Nachrichten zu erhalten. Die vielen Anfragen für jedes Team sind vermutlich der Grund dafür, dass die Verarbeitungszeit von Microsoft Teams auffällig länger ist als die von Slack. Zwar muss auch bei Slack für jeden Kanal die Nachrichtenhistorie angefragt werden, dies wird jedoch vom Framework unterstützt.

Zusammengeasst ist zu erkennen, dass Slack ein besseres Zeitverhalten besitzt als Microsoft Teams und somit für die Implementierung eines Bots, der die Anwendungsfälle UC-01 bis UC-03 verarbeiten kann, besser geeignet, wenn das Zeitverhalten ein Auswahlkriterium ist.

Ressourcennutzung

Bei der Ressourcennutzung wurde untersucht, wie viel Prozent der CPU und wie viele MegaByte Arbeitsspeicher verwendet wurden, um jeden einzelnen Anwendungsfall (UC-01 bis UC-03) zu verarbeiten. In den Experimentergebnissen ist zu erkennen, dass beide Bots wenig CPU und Arbeitsspeicher nutzen. Beide liegen bei der Ressourcennutzung weit unter der Obergrenze, die in Kapitel 5.2.1 genannt wurde. Es fällt jedoch auf, dass Slack in den meisten Fällen weniger als die Hälfe der CPU- und Arebitssepicherressourcen als Microsoft Teams benötigt. Somit ergibt sich, dass Slack im Punkt der Ressourcennutzung die bessere Wahl ist, um einen Bot zu implementieren, der die Anwendungsfälle UC-01 bis UC-03 verarbeiten kann. Microsoft Teams ist jedoch eine gute Alternative, da dieses nur geringfügig mehr Ressourcen verwendet als Slack.

Eine Auffälligkeit bei den Ergebnissen ist, dass die CPU- und Arbeitsspeicher-Nutzung immer sehr ähnlich ist. Sie unterscheiden sich kaum je Anwendungsfall. Dies wäre jedoch zu erwarten gewesen, da zum einen die Verarbeitungszeiten variieren und zum anderen die Anwendungsfälle unterschiedliche Daten verarbeiten. Während UC-01 Daten von Meisterplan verarbeitet, muss UC-02 Daten von Outlook verwerten und UC-03 analysiert alle Nachrichten, die auf Slack / Microsoft Teams gefunden wurden. Hier entsteht der Verdacht, dass nicht die Ressourcennutzung des gesamten Prozesses gemessen wurde, sondern nur der Verbrauch, der nach Abschluss der Verarbeitung der Daten besteht. Dies würde auf fehlerhafte Nutzung der vefügbaren Messmethoden schließen. Sollte dies der

Fall sein, so ist die Ressourcennutzung nicht für jeden Anwendungsfall zu betrachten, sondern allgemein im Betriebszustand. Die andere Option, dass bei jedem Anwendungsfall immer die fast gleiche CPU- und Arbeitsspeicher-Nutzung entsteht, ist als unwahrscheinlicher zu betrachten, da sowohl die Verarbeitungszeit als auch die Daten, auf denen operiert wird, sich unterscheiden.

Kapazität

In Bezug auf die Kapazität wurden in den Experimenten untersucht, wie viele Nachrichten maximal gesendet werden dürfen und wo die Grenzen der Nachrichtengrößen liegen. Je höher die Werte sind, die ermittelt wurden, desto besser ist das Ergebnis zu bewerten.

Bei der Obergrenze der Nachrichtenzahl, die in einem vorgegeben Intervall versendet werden darf, hat Microsoft Teams die besseren Werte erzielt, da hier in der Sekunde 50 Mal mehr Nachrichten versendet werden dürfen. Zwar reduziert sich die erlaubte Obergrenze bei Microsoft Teams, wenn Nachrichten innerhalb von 30 Sekunden versendet werden, aber selbst dann dürfen bei Microsoft Teams im Schnitt immer noch zwei Nachrichten pro Sekunde versendet werden, während Slack nur eine pro Sekunde erlaubt. Im Punkt der Nachrichtenzahl, die maximal in einem Zeitraum versendet werden darf, ist Microsoft Teams die bessere Wahl.

Bei der Nachrichtengröße erlaubt Microsoft Teams mit 28 KB den Nutzern größere Nachrichten zu senden, als es Slack tut, da dort ein Nutzer höchsten eine Nachricht von 16 KB Größe versenden darf. Diese 16 KB gehen allerdings davon aus, dass jedes Zeichen in der Nachricht 4 Bytes benötigt. Benutzt der Nutzer in der Nachricht jedoch nur Zeichen, die 1 Byte zur Darstellung benötigen, so würde die Nachricht maximal die Größe von 4 KB erreichen. Da bei Microsoft Teams keine Angabe zur Begrenzung der Zeichen gemacht wurde, sondern nur eine Größe von 28 KB genannt wird, wird davon ausgeganen, dass die Anzahl der Zeichen irrelevant ist und nur die Gesamtgröße berücksichtigt wird. Aus diesem Grund ist Microsoft Teams bei der Nachrichtengröße von Nutzer eindeutig die bessere Wahl, wenn dies ein entscheidender Faktor bei der Auswahl einer Kollaborationssoftware ist, auf welcher ein Bot implementiert werden soll, der die in dieser Ausarbeitung genannten Anwendungsfälle bedienen soll. Sowohl Slack als auch Microsoft Teams erlauben Bots größere Nachrichten zu versenden als ein menschlicher Nutzer es darf. Hier ähneln sich die Ergebnisse beider Plattformen. Während bei Slack die Nachricht maximal zwischen 40 und 160 KB groß sein darf, so ist die Grenze bei Microsoft Teams auf

40 KB gesetzt. Da auch hier angenommen wird, dass die Angaben von Microsoft Teams sich nicht auf die Zeichenzahl bezieht, sondern schlicht die Größe von 40 KB nicht überschritten werden darf, ermöglicht Slack den Bots größere Nachrichten zu senden als es Microsoft Teams tut. Hier ist zu bedenken, dass die Zeichen des *American Standard Code for Information Interchange* (ASCII) nur ein Byte benötigen [10] und somit vermutlich die meisten Zeichen in einer Nachricht nicht mehr als 1 Byte verbrauchen. Daraus ist zu schließen, dass die meisten Nachrichten, die der Slack sendet, kaum größer als 40 KB sein werden. Nichts desto trotz ermöglicht Slack die Verwendung größerer Zeichen und stellt somit im Punkt der Nachrichtengröße gegenüber Microsoft Teams eine bessere Wahl dar.

Zusammenfassung
Im Hinblick auf die Diskussionsergebnisse, die aus dem Experiment zur Effizienz gezogen wurden, steht fest, dass überwiegend Slack bessere Werte erzielt hat als Microsoft Teams. Sowohl im Zeitverhalten als auch bei der Ressourcennutzung eignet Slack sich besser, um einen Bot mit den Anwendungsfällen, die in dieser Ausarbeitung genannt wurden, zu implementieren. Lediglich bei der Kapazität erzielte Microsoft Teams bei der Nachrichtenanzahl, die pro Sekunde versendet werden darf, ein besseres Ergebnis als Slack. Und die Grenzen der Nachrichtengröße ähneln sich ebenfalls, allerdings ist Slack wieder leicht führend. Insgesamtbetrachtet kann gesagt werden, dass ein Bot, der die Anwendungsfälle verarbeiten kann, die in dieser Ausarbeitung genannt wurden, effizienter ist, wenn er mit Slack implementiert wurd. Dennoch hat auch Microsoft Teams niedrige Werte in den Experimenten erzielt und kommt den Ergebnisse von Slack nahe. Aus diesem Grund ist Microsoft Teams eine gute Alternative zu Slack. Einschränkend ist dazu zu sagen, dass die Effizienz sich auf die Antwortzeit, Verarbeitungszeit, CPU-Nutzung, Arbeitsspeichernutzung und Nachrichtengröße bezieht. Bei der Nachrichtenmenge ist Microsoft Teams führend und andere Effizienzeigenschaften wurden nicht getestet.

7.2 Kosten

Um den Anwendungsfall UC-02 zu erfüllen, müssen die Kalenderdaten der Mitarbeiter ausgelesen werden. Wie in Kapitel 4.1.2 bereits erläutert, ist Microsoft Outlook als Email Programm vorgegeben. Die vom Entwickler empfohlene REST API ist Microsoft Graph [33] und wird aus diesem Grund verwendet. Um Microsoft Graph nutzen zu können, ist eine Registrierung bei der Plattform Microsoft Azure nötig [24]. Des Weiteren ist

für jeden Mitarbeiter, dessen Kalenderdaten gelesen werden sollen, eine Microsoft 365 Lizenz Pflicht, da Microsoft Graph nur die Kalenderdaten solcher Nutzer lesen kann [34]. Folglich müssten zur Erfüllung des Anwendungsfalls UC-02 immer eine Microsoft Teams Lizenz gekauft werden, die für den Nutzer eine Email beinhaltet. Der günstigste Tarif bei Microsoft Teams, der diese Anforderung erfüllt, liegt bei 5,60 € im Monat pro Nutzer [41]. Soll der Bot in Slack betrieben werden, so würden zusätzlich noch Kosten in Höhe von 8,25 € im Monat pro Nutzer [54] anfallen. Das bedeutet, dass für Microsoft Teams Kosten monatliche Kosten in Höhe von 5,60 € im Monat pro Nutzer anfallen würden und bei Slack würden 13,85 € im Monat pro Nutzer anfallen. Somit ist Slack bei den monatlichen Lizenzkosten teurer als Microsoft Teams.

Ein weiterer Kostenfaktor ist der Betrieb des Bots. Wie bereits erwähnt, muss der Microsoft Teams Bot im Azure Portal betrieben werden. Azure bietet ein Werkzeug zur Preiskalkulation [2]. Mit diesem wird berechnet, dass der Betrieb einer Applikation auf einem Linux Betriebssystem mit niedrigsten Hardwarekonfiguration 17,10 $ (ungefähr 15,74 €) im Monat kosten würde. Zusätzlich entstehen im Jahr 81,98 $ (ungefähr 75,46 €) im Jahr, um eine eigene Domäne mit einem Standard SSL Zertifikat zu verwenden. Wird diese Summe monatlich berechnet, so entstehen daraus ungefähr 6,83 $ (ungefähr 6,29 €) zusätzliche Kosten. Somit belaufen sich die geschätzten Kosten für den Betrieb eines Bots im Azure Portal auf 22,03 € pro Monat. Da das Unternehmen CAS AG zurzeit 180 Mitarbeiter beschäftigt [6] (Stand Januar 2024), würden monatliche Lizenzkosten in Höhe von 1.008 € entstehen. Zusammen mit den Betriebskosten des Bots entstünden insgesamt Kosten in Höhe von 1.030,03 €.

Der Slack Bot kann auf privat gemieteten oder betriebenen Server betrieben werden. Als Rechenbeispiel wird hier nun der Anbieter strato.de verwendet, welcher Server vermietet. Es gibt weitere Anbieter wie Hetzner.de oder webtropia.com, die das gleiche tun, da aber die Preise je Anbieter nicht stark abweichen, wird strato.de nun als Preisgrundlage verwendet, um die Kosten für den Betrieb eines Slack Bots auf einen gemieteten Server zu berechnen. Strato bietet einen Linux Server an, der weitaus mehr Hardware Ressourcen zur Verfügung hat als der, der im Rechenbeispiel für Microsoft Teams benutzt wurde. Allerdings ist dieser Strato Server bereits der günstigste und hat somit im Vergleich zu den anderen Angeboten bereits die wenigsten Hardware Ressourcen. Dieser Server kostet monatlich 6 € [68]. Um eine eigene Domäne und ein SSL Zertifikat zusätzlich zu mieten, fallen zusätzliche Kosten in Höhe von 0,55 € im Monat [67] [69] an. Somit fallen für den Betrieb eines Slack Bots auf einem gemieteten Server von strato.de Kosten in Höhe von 6,55 € im Monat an. Werden nun die monatlichen Lizenzkosten für alle Nutzer und die

Betriebskosten des Bots zusammengerechnet, dann ergibt sich eine Summe von 2.499,55 €, die monatlich gezahlt werden müsste.

Es ist zu erkennen, dass der größte Kostenfaktor die Lizenzkosten sind und die Betriebskosten nicht stark ins Gewicht fallen. Somit hat sich ergeben, dass Microsoft Teams in Bezug auf die monatlichen Kosten die bessere Wahl ist, um einen Bot zu implementieren, der die Anforderungen erfüllt, die in dieser Ausarbeitung beschrieben wurden.

7.3 Programmiersprachen

In diesem Abschnitt wird untersucht, welche Frameworks oder SKDs Slack und Microsoft Teams anbieten, um Bots zu implementieren. Dies ist von Interesse, da die Unterstützung von bekannten und beliebten Programmiersprachen in Framework oder SDK das Entwickeln erleichtert. Je beliebter und bekannter eine Programmiersprache ist, desto höher ist die Wahrscheinlichkeit, dass die Entwickler, die mit der Implementierung oder Wartung des Bots beauftragt sind, diese Sprache kennen. Dies hat zur Folge, dass die Entwickler keine Zeit zum Erlernen der Sprache benötigen und es passieren weniger Fehler aufgrund von Unerfahrenheit.

Slack bewirbt vor allem drei Bolt Frameworks und SDKs [53] mit denen ein Bot implementiert werden kann. Diese sind in folgenden Programmiersprachen verfügbar:

- JavaScript
- Python
- Java

Microsoft Teams hingegen empfiehlt zur Erstellung eines Bots in einer App JavaScript [38]. Hierfür hat die Plattform ihr eigenes *Teams Toolkit* [42]

	Slack	**Microsoft Teams**
Framework Sprachen	JavaScript, Python, Java	JavaScript
SDK Sprachen	JavaScript, Python, Java	-

Tabelle 19: Vergleich von unterstützen Programmiersprachen in den Framworks und SDKs von Microsoft Teams und Slack

Folglich bieten beide Plattformen ein Framework in JavaScript an. Slack verfügt zusätzlich noch über Frameworks in Python und Java, sowie über SDKs für alle drei genannten Programmiersprachen.
Aus einer Umfrage von Statista [63] aus dem Jahr 2022 geht hervor, dass JavaScript mit 65,36 % die meist genutzte Programmiersprache weltweit ist. Python befindet sich auf Platz vier mit 48,07 % und Java mit 33,27 % auf Platz sechs.
Somit haben beide Plattformen mit JavaScript eine Programmiersprache für ihre Frameworks gewählt, die weltweit am meisten genutzt wird und die Wahrscheinlichkeit ist hoch, dass Porgrammierer, die damit arbeiten werden, mit dieser Sprache vertraut sind. Slack hat hier allerdings noch den Vorteil, dass es zwei weitere Sprachen (Python und Java) anbietet, die ebenfalls weltweit bekannt sind. Außerdem bietet es nicht nur drei Frameworks an, sondern auch drei SDKs in den drei Sprachen, wodurch die Wahrscheinlichkeit weiter erhöht wird, dass die Bot Entwickler mit einer Sprache arbeiten können, die sie gut kennen. Des Weiteren ist durch das Angebot des SDKs eine Flexibilität gegeben, mit welcher der Entwickler die Möglichkeit hat, sein eigenes Framework zu implementieren.

8 Zusammenfassung

In diesem Abschnitt werden die Ergebnisse der Diskussion aufgegriffen, um die Forschungsfrage dieser Ausarbeitung zu beantworten. Das Ziel dieser Ausarbeitung war es, zu untersuchen, ob es besser ist, Slack oder Microsoft Teams als Plattform zu verwenden, um einen Bot zu implementieren, der drei Anwendungsfälle, die von der CAS AG gestellt wurden, zu verarbeiten. Die Untersuchung der Bots wurde auf die zwei Qualitätsmerkmale *Wartbarkeit* und *Effizienz* reduziert. Des Weiteren wurden auch die Kosten sowie die Programmiersprache als Bewertungskriterium hinzugezogen. Anschließend werden die Experimente in Bezug auf ihre möglichen Schwachstellen reflektiert. Zum Schluss wird ein Ausblick gegeben, welche weiteren Gegenstände untersucht werden könnten, wenn es für ein Unternehmen um die Wahl einer Kollaborationssoftware geht.

8.1 Fazit

Die Ergebnisse dieser Ausarbeitung zeigen, dass Slack in den meisten Vergleichen besser bewertet wurde als Microsoft Teams. Jedoch sind die Unterschiede zwischen Slack und Microsoft Teams in Bezug auf die Wartbarkeit und Effizienz eher gering. Im Punkt der Kosten ist Microsoft Teams allerdings weitaus günstiger, wodurch es aus wirtschaftlicher Sicht attraktiver ist. Bei der Auswahl der nutzbaren Programmiersprachen, die zur Botimplementierung verwendet werden können, bietet Slack zwar eine größere Auswahl, Microsoft Teams bietet mit JavaScript jedoch mindestens eine gute Option. Würden die Kosten in der Bewertung außer Acht gelassen werden, dann ist Slack eindeutig Microsoft Teams vorzuziehen, um einen Bot zu implementieren, der die Anwendungsfälle, die in dieser Ausarbeitung beschrieben wurden. Da der Kostenunterschied jedoch recht hoch ist, Microsoft Teams in den Punkten Wartbarkeit und Effizienz nur im geringen Maße nachsteht und Microsoft Teams mindestens eine weit verbreitete Programmiersprache zur Botimplementierung anbietet, wird für das Unternehmen CAS AG die Verwendung von Microsoft Teams empfohlen.

8.2 Relexionen

In diesem Abschnitt werden Punkte angesprochen, die in dieser Ausarbeitung Schwachstellen aufweisen und folglich Auswirkungen auf die Ergebisse gehabt haben könnten.

Zum einen sind die Metriken *duplizierter Code* und *zyklomatische Komplexität*, die verwendet wurden, um die Modularität und Wiederverwendbarkeit zu bewerten, nicht ideal. Zwar deutet ein Code mit wenig Codeduplikaten und mit einer niedrigen Komplexität allgemein darauf hin, dass dies ein gut wartbarer Code ist, jedoch ist dies keine direkte Abbildung auf die Modularität oder Wiederverwendbarkeit eines Programms. In dieser Ausarbeitung wurden Annahmen getroffen, die implizit darauf schließen, dass weniger Codeduplikate eine bessere Modularität und eine geringere zyklomatische Komplexität eine bessere Wiederverwendbarkeit bedeuten. Somit ist fraglich, wie aussagekräftig die Bewertung der Modularität und Wiederverwendbarkeit ist. Das dies jedoch beides Metriken sind, die zur Bewertung der Wartbarkeit dienen, ist das Endergebnis, das besagen soll, ob der getestete Code wartbar ist oder nicht, nicht in eine falsche Richtung gelenkt worden. Es besteht nur die Frage, ob Modularität und Wiederverwendbarkeit in diesem Ergebnis akkurat widergespiegelt sind.

Ein weiterer kritisch zu betrachtender Punkt ist das Effizienz Experiment zum Anwendungsfall UC-03. In diesem Experiment sollte der Bot alle Nachrichten aus allen Kanälen finden, die ein bestimmtes Thema beinhalten. Hier hätten mehr Experimente durchgeführt werden können, die unterschiedlich viele Nachrichten finden und unterschiedlich viele Nachrichtenkanäle durchsuchen. Somit hätten detailliertere Rückschlüsse gezogen werden können, die Effizienz zur Bearbeitung des Anwendungsfalls zu bewerten. Beispielsweise hätte sich zeigen können, ob entweder Slack oder Microsoft Teams bessere Werte liefert, wenn die Zahl der Kanäle oder die Zahl der Nachrichten steigt oder sinkt. Das Ergebnis, das in diesem Experiment erhalten wurde, bildet jedoch nur einen Zustand ab.

Zuletzt ist zu bemerken, dass bei den Experimenten zur Effizienz ein Mittelwert aus der Summe aller Ergebnisse gebildet wurde. Dies birgt die Gefahr, dass Störungen während der Messungen zu verfäschten Werten führen, die den Mittelwert beeinflussen. Durch die fehlende Dokumentation der Einzelwerte können solche ungewöhnlichen Werte im Nachhinein nicht identifiziert werden. Eine besser Alternative zur Berechnung des Mittelwerts wäre die Angabe eines Konfidenzintervalls gewesen, um dieses Problem zu umgehen. Da die Ergebnisse jedoch meist ungefähr 50 % auseinander liegen, hätte ein Konfidenzintervall mit einem Konfidenzniveau von 95 % oder 90 % jedoch keinen Einfluss auf das Ergebnis gehabt.

8.3 Ausblick

In diesem Abschnitt werden weitere Untersuchungsmöglichkeiten genannt, die nicht Gegenstand dieser Ausarbeitung waren und dennoch relevant sein können, wenn ein Unternehmen vor der Wahl steht, Slack oder Microsoft Teams als Kollaborationssoftware einzuführen.

In dieser Ausarbeitung wurden bezüglich der Softwarequalittät nur die zwei Merkmale *Wartbarkeit* und *Effizienz* untersucht. Es existieren aber noch die Kriterien Funktionalität, Kompability, Benutzbarkeit, Zuverlässigkeit, Sicherheit und Portabilität [61] S. 10. Vor allem in den Punkten Benutzbarkeit und Sicherheit würden sich weitere Untersuchungen anbieten. Denn da die Bots von Menschen genutzt werden, ist es zielführend, die Nutzung möglichst benutzerfreundlich zu gestalten. Und wenn die Bots in einem Unternehmensumfeld genutzt werden, in welchem sensible Daten ausgetauscht werden, dann ist es von großer Wichtigkeit, dass eine Sicherheit gewährleistet ist.

Ein weiterer Punkt, der in Zukunft untersucht werden könnte, sind andere Schnittstellenfunktionen. In dieser Ausarbeitung wurde ausschließlich die Funktion verwendet, die ausgelöst wird, wenn der Bot eine Nachricht erhält. Die Bot Frameworks können jedoch zusätzlich auf zahlreiche andere Ereignisse reagieren. Beispielsweise gibt es die Möglichkeit Automatisierungen zu implementieren, die ausgelöst werden, wenn ein Nutzer einen Kanal betritt, auf Nachrichten mit einem Emoji reagiert, eine Datei hochlädt, einem Sprachanruf beitritt und vieles mehr. Damit können neue Anwendungsfälle entstehen und auf die gleichen Eigenschaften wie in dieser Ausarbeitung getestet werden. Dort wäre die Frage zu klären, ob sich die Effizienz und Wartbarkeit ändern würde und ob dies dazu führen würde, dass eine Plattform der anderen weitaus überlegen ist.

Ein weiterer Aspekt, der für ein Unternehmen bei der Entscheidung, ob es Slack oder Microsoft Teams als Kollaborationssoftware benutzt soll, sind die weiteren Möglichkeiten, die die Plattform bietet. Die Implementierung eines Bots ist nur eine davon. Beispielsweise könnten die Grenzen der Webhooks untersucht werden, mit denen Nachrichten an Nachrichtenkanäle geschickt werden können. Microsoft Teams bietet viele weitere Applikationen und Dienste an, die beim Kauf einer Lizenz enthalten sind und könnte dadurch attraktiver als Slack sein, da hier mehr Funktionalität geboten wird.

Ein Punkt, der in dieser Ausarbeitung bezüglich der Effizienz in der Untersuchung nicht einbezogen wurde, sind Menschen als Ressourcen. Denn neben den Hardware Ressourcen zählen auch Menschen, die den Bot Entwickeln ebenfalls als Ressourcen. Hier könnte

geprüft werden, wie effizient ein Entwicklerteam einen Slack oder Microsoft Teams entwickeln könnte.

Literatur

[1] Anguswamy, Reghu ; Frakes, William: A Study of reusability, complexity, and reuse design principles, 09 2012

[2] Azure: *Pricing calculator*. – URL https://azure.microsoft.com/en-us/pricing/calculator/. – Zugriffsdatum: 28.01.2024

[3] Broy, Manfred ; Kuhrmann, Marco: *Einführung in die Softwaretechnik*. 1. Heidelberg : Springer Vieweg, 2021. – ISBN 978-3-662-50262-4

[4] CAS AG: *Homepage*. – URL https://www.c-a-s.de/. – Zugriffsdatum: 19.07.2023

[5] CAS AG: *Integriertes Stammdatenmanagement mit CAS CaseFamily*. – URL https://www.c-a-s.de/portfolio-items/integriertes-stammdatenmanagement/. – Zugriffsdatum: 19.07.2023

[6] CAS AG: *Über uns - Daten und Fakten*. – URL https://www.c-a-s.de/ueber-uns-cas-ag/#fakten. – Zugriffsdatum: 28.01.2024

[7] Duden: *Bot, der*. 2023. – URL https://www.duden.de/rechtschreibung/Bot. – Zugriffsdatum: 25.04.2023

[8] Fischer, Peter ; Hofer, Peter: *Lexikon der Informatik*. 15. Heidelberg : Springer Berlin, 2011. – ISBN 978-3-642-15126-2

[9] Furrer, Frank J.: *Future-Proof Software-Systems. A Sustainable Evolution Strategy*. 1. Wiesbaden : Springer Vieweg Wiesbaden, 2019. – ISBN 978-3-658-19937-1

[10] Gorn, S. ; Bemer, R. W. ; Green, J.: American standard code for information interchange. In: *Commun. ACM* 6 (1963), aug, Nr. 8, S. 422–426. – URL https://doi.org/10.1145/366707.367524. – ISSN 0001-0782

[11] Hoffmann, Dirk W.: *Software-Qualität*. 2. Heidelberg : Springer Vieweg Berlin, 2013. – ISBN 978-3-642-35699-5

[12] Huynh, Sunny ; Cai, Yuanfang ; Song, Yuanyuan ; Sullivan, Kevin: Automatic Modularity Conformance Checking. In: *Proceedings of the 30th International Conference on Software Engineering*. New York, NY, USA : Association for Computing Machinery, 2008 (ICSE '08), S. 411–420. – URL https://doi.org/10.1145/1368088.1368144. – ISBN 9781605580791

[13] International Organization for Standardization: *Homepage*. – URL https://www.iso.org/home.html. – Zugriffsdatum: 21.07.2023

[14] International Organization for Standardization: *IEC*. – URL https://www.iso.org/organization/70.html. – Zugriffsdatum: 21.07.2023

[15] International Organization for Standardization: *What we do*. – URL https://www.iso.org/what-we-do.html. – Zugriffsdatum: 21.07.2023

[16] Johnson-Lenz, Peter ; Johnson-Lenz, Trudy: Post-mechanistic groupware primitives: rhythms, boundaries and containers. In: *International Journal of Man-Machine Studies* 34 (1991), Nr. 3, S. 395–417. – URL https://www.sciencedirect.com/science/article/pii/0020737391900275. – Computer-supported Cooperative Work and Groupware. Part 2. – ISSN 0020-7373

[17] Lebeuf, Carlene ; Zagalsky, Alexey ; Foucault, Matthieu ; Storey, Margaret-Anne: Defining and Classifying Software Bots: A Faceted Taxonomy. In: *Proceedings of the 1st International Workshop on Bots in Software Engineering*, IEEE Press, 2019 (BotSE '19), S. 1–6. – URL https://doi.org/10.1109/BotSE.2019.00008

[18] Lenz, Manuel: *Server mieten - 12 Anbieter und 400+ Tarife im Vergleich*. – URL https://www.experte.de/server/mieten. – Zugriffsdatum: 16.01.2024

[19] Maus, Gregory: A Typology of Socialbots (Abbrev.). In: *Proceedings of the 2017 ACM on Web Science Conference*. New York, NY, USA : Association for Computing Machinery, 2017 (WebSci '17), S. 399–400. – URL https://doi.org/10.1145/3091478.3098860. – ISBN 9781450348966

[20] Meisterplan: *Meistern Sie Ihre projektübergreifende Ressourcenplanung*. – URL https://meisterplan.com/de/. – Zugriffsdatum: 18.07.2023

[21] Meisterplan: *Meisterplan API*. – URL https://api.eu.meisterplan.com/docs/api.html. – Zugriffsdatum: 18.07.2023

[22] Meisterplan: *Meisterplan Reporting API*. – URL https://api-reporting.eu.meisterplan.com/docs/api.html. – Zugriffsdatum: 18.07.2023

[23] Meisterplan: *Meisterplan Reporting API*. – URL https://api-reporting.eu.meisterplan.com/docs/api.html#section/Get-Started/Rate-Limiting. – Zugriffsdatum: 12.12.2023

[24] Microsoft: *Authentication and authorization basics*. – URL https://learn.microsoft.com/en-us/graph/auth/auth-concepts. – Zugriffsdatum: 20.07.2023

[25] Microsoft: *Format your bot messages*. – URL https://learn.microsoft.com/en-us/microsoftteams/platform/bots/how-to/format-your-bot-messages. – Zugriffsdatum: 22.12.2023

[26] Microsoft: *Get calendar*. – URL https://learn.microsoft.com/en-us/graph/api/calendar-get?view=graph-rest-1.0&tabs=http. – Zugriffsdatum: 18.07.2023

[27] Microsoft: *Limits and specifications for Microsoft Teams*. – URL https://learn.microsoft.com/en-us/microsoftteams/limits-specifications-teams. – Zugriffsdatum: 31.12.2023

[28] Microsoft: *Microsoft Graph REST API v1.0 endpoint reference*. – URL https://learn.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0. – Zugriffsdatum: 18.07.2023

[29] Microsoft: *Microsoft Graph service-specific throttling limits - Outlook service limits*. – URL https://learn.microsoft.com/en-us/graph/throttling-limits#outlook-service-limits. – Zugriffsdatum: 22.12.2023

[30] Microsoft: *Microsoft Outlook*. – URL https://www.microsoft.com/de-de/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook. – Zugriffsdatum: 18.07.2023

[31] Microsoft: *Optimize your bot with rate limiting in Teams*. – URL https://learn.microsoft.com/en-us/microsoftteams/platform/bots/how-to/rate-limit. – Zugriffsdatum: 22.12.2023

[32] Microsoft: *Payment models and licensing requirements for Microsoft Teams APIs*. – URL https://learn.microsoft.com/en-us/graph/teams-licenses. – Zugriffsdatum: 22.12.2023

[33] Microsoft: *Use the Outlook mail REST API*. – URL https://learn.microsoft.com/en-us/graph/api/resources/mail-api-overview?view=graph-rest-1.0. – Zugriffsdatum: 20.07.2023

[34] Microsoft: *Users you can reach with Microsoft Graph*. – URL https://learn.microsoft.com/en-us/graph/users-you-can-reach. – Zugriffsdatum: 20.07.2023

[35] Microsoft: *What is Azure?*. – URL https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/. – Zugriffsdatum: 13.08.2023

[36] Microsoft: *Übersicht über Microsoft Graph*. – URL https://learn.microsoft.com/de-de/graph/overview. – Zugriffsdatum: 13.08.2023

[37] Microsoft: *Microsoft Azure-Portal*. 2023. – URL https://azure.microsoft.com/de-de/get-started/azure-portal. – Zugriffsdatum: 10.01.2024

[38] Microsoft Teams: *Erstellen von Bots für Teams*. – URL https://learn.microsoft.com/de-de/microsoftteams/platform/bots/what-are-bots. – Zugriffsdatum: 25.04.2023

[39] Microsoft Teams: *channel: getAllMessages*. 2023. – URL https://learn.microsoft.com/en-us/graph/api/channel-getallmessages?view=graph-rest-1.0&tabs=http. – Zugriffsdatum: 10.01.2024

[40] Microsoft Teams: *List events*. 2023. – URL https://learn.microsoft.com/en-us/graph/api/calendar-list-events?view=graph-rest-1.0&tabs=http. – Zugriffsdatum: 10.01.2024

[41] Microsoft Teams: *Die passende Microsoft Teams-Version für Ihre Anforderungen*. 2023. – URL https://www.microsoft.com/de-de/microsoft-teams/compare-microsoft-teams-options?market=de. – Zugriffsdatum: 28.01.2024

[42] Microsoft Teams: *Voraussetzungen*. 2023. – URL https://learn.microsoft.com/de-de/microsoftteams/platform/sbs-gs-bot?tabs=vscode%2Cviscode&tutorial-step=1. – Zugriffsdatum: 25.04.2023

[43] Nielsen, Jakob: *Response Times: The 3 Important Limits*. – URL https://www.nngroup.com/articles/response-times-3-important-limits/. – Zugriffsdatum: 12.01.2023

[44] nodejs: *Node.js v21.2.0 documentation*. – URL https://nodejs.org/api/https.html. – Zugriffsdatum: 29.11.2023

[45] NODEJS: *Node.js v21.2.0 documentation - process.cpuUsage([previousValue])*. – URL https://nodejs.org/api/process.html#processcpuusagepreviousvalue. – Zugriffsdatum: 24.11.2023

[46] NODEJS: *Node.js v21.2.0 documentation - process.memoryUsage()*. – URL https://nodejs.org/api/process.html#process_process_memoryusage. – Zugriffsdatum: 30.11.2023

[47] NODEJS: *Startseite*. – URL https://nodejs.org/en. – Zugriffsdatum: 21.11.2023

[48] NODEJS: *Über Node.js*. – URL https://nodejs.org/de/about. – Zugriffsdatum: 22.08.2023

[49] NPMJS: *es6-plato*. – URL https://www.npmjs.com/package/es6-plato. – Zugriffsdatum: 21.11.2023

[50] NPMJS: *jscpd*. – URL https://www.npmjs.com/package/jscpd. – Zugriffsdatum: 31.12.2023

[51] NPMJS: *node-os-utils*. – URL https://www.npmjs.com/package/node-os-utils. – Zugriffsdatum: 26.11.2023

[52] NURDAYENTI, Lettisia ; AMIRUDDIN, Amiruddin: Comparative Analysis of Usability, Performance, and Security of Open-Source, Windows-Based Password Manager Applications Based on ISO/IEC 25010. In: *2022 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, 2022, S. 178–183

[53] SLACK: *Building a Slack app*. 2023. – URL https://api.slack.com/start/building. – Zugriffsdatum: 25.04.2023

[54] SLACK: *Team-Arbeit produktiver machen*. 2023. – URL https://slack.com/intl/de-de/pricing. – Zugriffsdatum: 23.04.2023

[55] SLACK API: *chat.postMessage*. – URL https://api.slack.com/methods/chat.postMessage. – Zugriffsdatum: 31.12.2023

[56] SLACK API: *Rate Limits*. – URL https://api.slack.com/apis/rate-limits. – Zugriffsdatum: 16.12.2023

[57] SLACK API: *Reference: blocks*. – URL https://api.slack.com/reference/block-kit/blocks. – Zugriffsdatum: 16.12.2023

[58] Slack API: *Reference: Secondary message attachments*. – URL https://api.slack.com/reference/messaging/attachments. – Zugriffsdatum: 31.12.2023

[59] Slack API: *An introduction to the Slack platform*. 2023. – URL https://api.slack.com/start. – Zugriffsdatum: 10.01.2024

[60] Slack API: *Using the Slack Web API*. 2024. – URL https://api.slack.com/web. – Zugriffsdatum: 30.01.2024

[61] Standardization, International O. for: *ISO / IEC 25010 : 2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. First edition. ISO, 2011. – URL https://www.iso.org/standard/35733.html. – ISO standard

[62] Statista: *Collaboration Software - Worldwide*. – URL https://www.statista.com/outlook/tmo/software/productivity-software/collaboration-software/worldwide. – Zugriffsdatum: 07.08.2023

[63] Statista: *Most used programming languages among developers worldwide as of 2022*. – URL https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. – Zugriffsdatum: 25.04.2023

[64] Steen, Maarten v. ; Tanenbaum, Andrew: *Distributed systems*. Third edition (Version 3.03 (2020)). Pearson Education, 2020. – ISBN 978-90-815406-2-9

[65] Steidl, Daniela ; Deissenboeck, Florian ; Poehlmann, Martin ; Heinke, Robert ; Uhink-Mergenthaler, Bärbel: Continuous Software Quality Control in Practice. In: *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, S. 561–564

[66] Storey, Margaret-Anne ; Zagalsky, Alexey: Disrupting Developer Productivity One Bot at a Time. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA : Association for Computing Machinery, 2016 (FSE 2016), S. 928–931. – URL https://doi.org/10.1145/2950290.2983989. – ISBN 9781450342186

[67] strato.de: *Domain - Günstige Domain sichern – Gedacht. Gemacht.*. – URL https://www.strato.de/domains/. – Zugriffsdatum: 28.01.2024

[68] strato.de: *Linux V-Server - Günstig Linux V-Server mieten*. – URL https://www.strato.de/server/linux-vserver/. – Zugriffsdatum: 28.01.2024

[69] STRATO.DE: *STRATO SSL - Schützen Sie Ihre Website und gehen Sie auf Nummer sicher.* – URL https://www.strato.de/ssl-zertifikat/. – Zugriffsdatum: 28.01.2024

[70] THOMSON, Edward: *About the public npm registry.* – URL https://docs.npmjs.com/about-the-public-npm-registry. – Zugriffsdatum: 21.11.2023

[71] THOMSON, Edward ; KARRYS, Luke: *Downloading and installing Node.js and npm.* – URL https://docs.npmjs.com/downloading-and-installing-node-js-and-npm. – Zugriffsdatum: 21.11.2023

[72] THOMSON, Edward ; KARRYS, Luke: *Downloading and installing packages locally.* – URL https://docs.npmjs.com/downloading-and-installing-packages-locally. – Zugriffsdatum: 21.11.2023

[73] VISUAL STUDIO CODE: *Code editing. Redefined.*. – URL https://code.visualstudio.com/. – Zugriffsdatum: 21.11.2023

[74] ZANTAL-WIENER, Amanda: *Where Do Bots Come From? A Brief History.* – URL https://blog.hubspot.com/marketing/where-do-bots-come-from. – Zugriffsdatum: 25.04.2023

Glossar

Antwortzeit "Bei vernetzten Systemen die Zeit zwischen dem Absetzen einer Anfrage bzw. eines Auftrags und dem Eintreffen der Antwort bzw. der Bestätigung" [8] S. 43.

ASCII American Standard Code for Information Interchange. Dies ist eine codierter Zeichensatz, die den Austausch von Informationen zwischen Informationsverarbeitungssystemen, Kommunikationssystemen und zugehörigen Geräten zu erleichtern soll [10] S. 422.

CAS CaseFamily System zum Stammdatenmanagement, welches von der CAS AG entwickelt wurde [5].

Client Benutzeroberfläche, die es dem Nutzer ermöglicht, mit entfernten Servern zu interagieren [64] S. 124.

Durchsatz Maß für den quantitativ ermittelten Datentransfer auf Bussen, in Kanälen oder in der Datenfernübertragung. Die Angaben sind meist in Bits pro Sekunde (bps) bzw. Zehnerpotenzen davon [8] S. 212.

IEC International Electrotechnical Commission [14].

ISO International Organization for Standardization [13].

Kohäsion Grad in dem Komponenten, Klassen und Methoden eines Systems zusammenhängen. Bei einer starken Kohäsion erfüllt jede Programmeinheit im System genau eine wohldefiniert Aufgabe [3] S. 331.

Kollaborationssoftware Anwendungen, deren Zweck es ist, Menschen bei der Arbeit an gemeinsamen Aufgaben und dem Erreichen von Zielen zu unterstützen [16].

Meisterplan Porgramm zum Koordinieren von Mitarbeitern über Abteilungen und Projekte hinweg [20].

Microsoft Azure Cloud Plattform mit welcher Anwendungen über mehrere Clouds gebaut, betrieben und verwaltet werden können [35].

Microsoft Graph "Gateway zu Daten und Informationen in Microsoft 365. Es bietet ein vereinheitlichtes Programmierbarkeitsmodell, mit dem [...] auf die enormen Datenmengen in Microsoft 365, Windows und Enterprise Mobility + Security" zugegriffen werden können [36].

NodeJs Asynchrone und Event-basierte JavaScript Laufzeitumgebung, die für die Entwicklung von skalierbaren Netzwerkanwendungen entworfen wurde [47] [48].

REST API Representational State Transfer Application Progrmming Interface.

SDK Software Development Kit.

Stakeholder "Ein Stakeholder ist eine Person, eine Gruppe von Personen oder eine Organisation, die ein direktes oder indirektes Interesse (Stake) am zu entwickelnden System hat oder seine Entwicklung beeinflusst" [3] S. 207.

Veroderung Dies ist eine Aussage, die durch die Verknüpfung zweiter Aussagen mit dem logischen Operator ODER entsteht. Beispiel: Das Auto ist grün ODER (das Auto ist) rot..

Anhang

In Tabelle 20 werden alle URLs aufgelistet, die in dieser Ausarbeitung mittels ihrer Tabellen-Position referenziert werden. Die Position, die in der ersten Spalte unter dem Symbol # eingetragen ist, ist eine fortlaufende Nummer, die in dieser Arbeit jeder einzelnen URLs zugewiesen wurde, um sie eindeutig und in kurzer Form zu identifizieren. In der URL Spalte ist lediglich die URL selber enthalten. In der Beschreibungsspalte werden weitere Informationen zur URL gegeben, um zu verdeutlichen, wofür diese steht.

In der Tabelle 21 werden alle Verzeichnisse aufgelistet, die in dieser Ausarbeitung mittels ihrer Tabellen-Position referenziert werden. Die Position, die in der ersten Spalte unter dem Symbol # eingetragen ist, ist eine fortlaufende Nummer, die in dieser Arbeit jedem einzelnen Verzeichnispfad zugewiesen wurde, um ihn eindeutig und in kurzer Form zu identifizieren. In der Verzeichnispfad Spalte ist lediglich der Pfad selber angegeben. In der Beschreibungsspalte wird erläutert, was das Verzeichnis beinhaltet und es werden Variablen beschrieben, falls welche in einem Pfad vorkommen. Der Name {PRJ} ist ein Platzhalter für den tatsächlichen Projektpfad, in welchem die Projekte liegen.

#	URL und Beschreibung
1	https://api.slack.com/apps Slack-Konfigurations-URL: Diese URL zeigt auf eine Webseite, die von Slack zur Verfügung gestellt wird, um dort Applicationen zu registrieren und zu verwalten [59].
2	https://portal.azure.com#view/Microsoft_AAD_RegisteredApps/ApplicationsListBlade Teams-Konfigurations-URL: Diese URL zeigt auf eine von Microsoft zur Verfügung gestellte Webseite, Applicationen zu erstellen, zu verwalten und zu überwachen [37].
3	https://graph.microsoft.com/v1.0/users/{id|userPrincipalName}/calendar/events Outlook-Kalender-Ereignisse-URL: Diese URL ermittelt alle Ereignisse, die im Outlook Kalender einer Person vorhanden sind [40]. Die Person wird durch die Email *id* oder *userPrincipalName* identifiziert.
4	https://graph.microsoft.com/v1.0/teams/{team-id}/channels/getAllMessages Teams-Alle-Nachrichten-URL: Diese URL ermittelt alle Nachrichten aus allen Kanälen, die sich innerhalb eines Teams befinden [39]. Das betroffene Team wird in der URL durch seine *team-id* identifiziert.
5	https://bai-bachelorarbeit.slack.com/api/chat.postMessage? Slack-PostMessage-URL: Diese URL wird von Slack gesendet, wenn es eine Antwortnachricht an den Client sendet, die durch einen der Anwendungsfälle UC-01 bis UC-03 erzeugt wurde.
6	https://ca.slack-edge.com/{GUID} Slack-Bot-Antwort-URL: {GUID} ist hier ein Platzhalter für eine Zeichenkette bestehend aus Buchstaben, Zahlen und Bindestrichen, die der Identifizierungsnummer des Bots entspricht.
7	https://emea.ng.msg.teams.microsoft.com/v1/users/ME/conversations/{GUID}unq.gbl.spaces/messages Teams-PostMessage-URL: Der Name {GUID} ist in diesem Fall ein Platzhalter und entspricht einer Folge von Zahlen und Buchstaben sowie Bindestrichen, die je Nutzer unterschiedlich ist.
8	datei:image/png;base64 Teams-Bot-Antwort-URL: Diese URL wird von Microsoft Teams gesendet, wenn es eine Antwortnachricht an den Client sendet, die durch einen der Anwendungsfälle UC-01 bis UC-03 erzeugt wurde.
9	https://gitlab.com/AkiNya/bachlorarbeit-microsoft-teams-und-slack-bots Bot-Repository: Dieser Link verweist auf ein GitLab Verzeichnis, welches den Code enthält, der für den Slack und Microsoft Teams Bot implementiert wurde.

Tabelle 20: Liste aller URLs, die in dieser Ausarbeitung referenziert wurden

#	**Verzeichnispfad**	**Beschreibung**
1	{PRJ}/slack_bot	Slack Bot Verzeichnis
2	{PRJ}/slack_bot/node_modules/@slack	Slack Bot Framework Verzeichnis
3	{PRJ}/MSTeamsBotlocal	Microsoft Teams Bot Verzeichnis
4	{PRJ}/MSTeamsBotlocal/ node_modules/botbuilder	Microsoft Teams Bot Framework Verzeichnis
5	{PRJ}/MSTeamsBotlocal/ node_modules/botbuilder-core	Microsoft Teams Bot Framework Verzeichnis
7	{PRJ}/MSTeamsBotlocal/ node_modules/botbuilder-dialogs-adaptive-runtime-core	Microsoft Teams Bot Framework Verzeichnis
8	{PRJ}/MSTeamsBotlocal/ node_modules/botbuilder-stdlib	Microsoft Teams Bot Framework Verzeichnis
9	{PRJ}/MSTeamsBotlocal/ node_modules/botframework-connector	Microsoft Teams Bot Framework Verzeichnis
10	{PRJ}/MSTeamsBotlocal/ node_modules/botframework-schema	Microsoft Teams Bot Framework Verzeichnis
11	{PRJ}/MSTeamsBotlocal/ node_modules/botframework-streaming	Microsoft Teams Bot Framework Verzeichnis
12	{PRJ}/MSTeamsBotlocal/ node_modules/@azure	Microsoft Teams Bot Framework Verzeichnis

Tabelle 21: Liste aller Verzeichnisse, die in dieser Ausarbeitung referenziert wurden

Abbildungsverzeichnis

1 Weltweiter Marktanteil von Produktivitäts- und Kollaborationssoftwares nach Statista [62] . . . 6
2 Fachlicher Kontext des Bots . . . 10
3 Technischer Kontext des Bots . . . 11
4 Anwendungsfälle des Bots . . . 12
5 Bausteindiagramm L0 des Bot Systems . . . 14
6 Bausteindiagramm L1 des Bot Systems . . . 16
7 Verteilungsdiagramm zur Betreibung des Bots . . . 18
8 Sequenzdiagramm für Anwednungsfall UC-01 . . . 22
9 Sequenzdiagramm für Anwednungsfall UC-02 . . . 25
10 Sequenzdiagramm für Anwednungsfall UC-03 . . . 27
11 Nutzung der es6-plato und jscpd Analysefunktionen in der Kommandozeile 45

Tabellenverzeichnis

1 Liste der Anwendungsfälle . . . 13
2 Beschreibung der Komponenten des Bausteindiagramms Level 0 . . . 15
3 Beschreibung der Komponenten des Bausteindiagramms Level 1 . . . 17
4 Komponenten im Verteilungsdiagramm . . . 18
5 Unterkateogiren der Wartbarkeit nach ISO/IEC 25010 [61] S. 14f . . . 30
6 Anwendbare Metriken, mit denen die Unterkategorien des Qualitätsmerkmals Wartbarkeit nach ISO/IEC 25010 in einem System zu bemessen werden können . . . 36
7 Unterkateogiren der Effizienz nach ISO/IEC 25010 [61] S. 11 . . . 37
8 Erläuterung der Befehlsparameter bei der Analyse durch die Programme *es6-plato* und *jscpd* . . . 46
9 Ergebnisse der Wartbarkeitsmessung für die gesamte Slack Bot Implementierung . . . 47
10 Ergebnisse der Wartbarkeitsmessung für die gesamte Implementierung . . . 47
11 Ergebnisse der Wartbarkeitsmessung für die gesamte Microsoft Teams Bot Implementierung . . . 48
12 Ergebnisse der Wartbarkeitsmessung für die gesamte Microsoft Teams Bot Framework Implementierung . . . 49

13 Zusammenfassung der Ergebnisse zur Wartbarkeitsmessung 50
14 Alle Ergebnisse für den Anwendungsfall UC-01 des Experiments zur Effizienz 58
15 Alle Ergebnisse für den Anwendungsfall UC-02 des Experiments zur Effizienz 59
16 Alle Ergebnisse für den Anwendungsfall UC-03 des Experiments zur Effizienz 59
17 Grenzen für das Senden von Nachrichten 60
18 Grenzen der Nachrichtenlängen 61
19 Vergleich von unterstützen Programmiersprachen in den Framworks und SDKs von Microsoft Teams und Slack 73
20 Liste aller URLs, die in dieser Ausarbeitung referenziert wurden XI
21 Liste aller Verzeichnisse, die in dieser Ausarbeitung referenziert wurden . XII

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original