Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Haben Design-Regeln Einfluss auf die Verständlichkeit von RESTful APIs? Ein kontrolliertes Experiment

Timo Pfaff

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dr. Justus Bogner, Sebastian Kotstein
Beginn am:	10. Mai 2021
Beendet am:	09. November 2021

Kurzfassung

REST wird in der Praxis häufig verwendet, um Programmierschnittstellen zu erstellen. Da keine einheitliche Spezifikation für REST existiert, wurden in verschiedenen wissenschaftlichen Arbeiten Best Practices und Design-Regeln definiert, um die Entwickler bei der Erstellung von REST APIs zu unterstützen. Aufgrund der Vielzahl an Regeln und Best Practices stehen die Entwickler vor dem Problem auszuwählen, welche von diesen sie umsetzen sollen. Die Auswahl wird durch das Fehlen einer empirischen Grundlage für den Effekt der Regeln zusätzlich erschwert.
Aus den genannten Gründen wurde in dieser Arbeit untersucht, ob Design-Regeln einen Einfluss auf die Verständlichkeit von REST APIs haben. Dies wurde anhand eines kontrollierten Experiments überprüft, welches online über das Werkzeug Limesurvey durchgeführt wurde. Die Aufgabe der 105 Probanden war es, Verständnisaufgaben zu verschiedenen REST API Schnipseln zu beantworten. Zusätzlich bewerteten die Probanden die Schwierigkeit, das Schnipsel zu verstehen. Die APIs existierten dabei in zwei Versionen. Eine Version für die Einhaltung der Regeln und eine weitere Version, bei der gegen die Regeln verstoßen wurde.
Ziel der Studie war es zu analysieren, welche der untersuchten Regeln einen signifikanten Einfluss auf die Verständlichkeit haben. Gemessen wurde die Verständlichkeit unter Verwendung einer geeigneten Metrik, welche sich aus der benötigten Zeit und der Korrektheit zusammensetzt. Außerdem wurde der Einfluss auf die wahrgenommenen Schwierigkeit untersucht. Zuletzt wurde überprüft, ob die Ergebnisse durch die unterschiedlichen Erfahrungen der Teilnehmer beeinflusst wurden.
Mit einer Ausnahme wurde in Bezug auf die Verständlichkeit für alle untersuchten Regeln ein signifikant besseres Ergebnis bei Einhaltung der Regeln gemessen. Ein ähnliches Resultat ist auch für die wahrgenommene Schwierigkeit zu beobachten. Abgesehen von der genannten Ausnahme wurde die Version mit Regeleinhaltung von den Probanden signifikant leichter bewertet. Für den Einfluss der Erfahrung konnten signifikante Wechselwirkungen mit der Anzahl an Jahren, der Kenntnis über das Reifegradmodell und der Perspektive bei der Arbeit mit REST APIs beobachtet werden. Die gemessenen Effektstärken für diese Korrelationen werden als schwach eingestuft. Für die unterschiedlichen Berufsgruppen konnte keine signifikante Korrelation gefunden werden.
Bei der Bewertung der Ergebnisse ist zu berücksichtigen, dass lediglich ein Teil der vorhandenen Regeln untersucht wurde. Weiterhin wurden Regeln ausgewählt, für die ein großer Effekt erwartet wurde.
Insgesamt zeigen die Ergebnisse, dass die Verständlichkeit von REST APIs verbessert werden kann, wenn die dafür aufgestellten Regeln beachtet werden. Dies gilt unabhängig der bisherigen Erfahrungen mit REST APIs. Um die mit der Verständlichkeit verbundene Nutzbarkeit und Wartbarkeit zu steigern, wird auf Basis der Ergebnisse empfohlen, bei der Erstellung von REST APIs auf die Einhaltung der positiv evaluierten Regeln zu achten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In diesem Kapitel wird die Motivation für diese Arbeit beschrieben. Außerdem wird die Zielsetzung dargelegt und der Aufbau der Arbeit erläutert.

1.1 Motivation

REST ist kurz für *Representational State Transfer* und wurde im Jahre 2000 von Roy Thomas Fielding eingeführt [17]. Es handelt sich dabei um einen Architekturstil, der dazu verwendet wird, um Programmierschnittstellen (kurz *APIs*) für webbasierte Services zu modellieren. Es ist ebenfalls möglich REST für Webapplikationen anzuwenden, die auf den Prinzipen und Technologien des World Wide Web basieren. REST stützt sich dabei vor allem auf Standards wie *Hypertext Transfer Protocol* (kurz *HTTP*) und *Uniform Resource Identifiers* (kurz *URIs*) [63].
Inzwischen sind REST APIs weit verbreitet und werden in vielen gängigen Plattformen, wie beispielsweise *Netflix*[1] und *Twitter*[2] eingesetzt. Die Webseite *ProgrammableWeb*[3], welche über ein Register aus rund 24000 APIs verfügt, hat 2017 eine Statistik [53] veröffentlicht. In dieser wird dargelegt, dass 81,53 % der APIs im Register den Architekturstil REST verwenden. Zur Einordnung und Erstellung dieser Statistik wurde überprüft, ob eine Programmierschnittstelle HTTP Methoden für die Kommunikation verwendet. Eine Überprüfung auf die bestehenden REST Prinzipien wurde nicht angestellt. Daher besteht die Möglichkeit, dass diese Statistik auch APIs enthält, die nicht wirklich RESTful sind. Dennoch gibt sie einen Hinweis auf die Verbreitung von REST.
Fielding [17] definiert für REST verschiedene Prinzipien, welche als Empfehlungen für das Verhalten einer modernen Webapplikation fungieren. Zusätzlich zu den Prinzipien legt Fielding in seiner Arbeit dar, welchen Effekt diese auf die Qualitätsmerkmale von Software haben. Es werden jedoch keine expliziten Implementierungsdetails beschrieben. Weiterhin existiert auch kein offizieller Standard oder eine Spezifikation. Dies führt dazu, dass Entwickler die Prinzipien von REST in der Praxis unterschiedlich interpretieren und folglich unterschiedliche Lösungsansätze für die Implementierung der APIs anwenden. Besonders bei der Integration mehrerer unterschiedlicher APIs kann dies zu Problemen führen [51].
Um den unterschiedlichen Lösungen für die Implementierung entgegenzuwirken, wurden seit der Einführung verschiedene Publikationen veröffentlicht, die Richtlinien und Design-Regeln zum Erstellen von REST APIs enthalten. Ein Beispiel hierfür ist das *REST API Design Rulebook* von Mark Masse [36]. In diesem Buch werden über 80 Regeln für REST APIs aufgestellt und beschrieben. Häufig existiert keine empirische Grundlage für den positiven Effekt der Regeln und Best Practices auf die Qualitätsattribute. Dieser Aspekt gepaart mit der großen Anzahl an Regeln führt dazu, dass

[1] https://www.netflix.com/
[2] https://twitter.com/
[3] https://www.programmableweb.com/

Entwickler nur schwer nachvollziehen können, ob die Anwendung einer bestimmten Regel einen realen Nutzen hat. Folglich wird die Auswahl, welche der Regeln in der Praxis umgesetzt werden, erschwert [32].
In einer weiteren wissenschaftlichen Arbeit [32] wurden die Design-Regeln von Masse mittels einer Delphi-Studie untersucht. Dazu befragten die Autoren Experten aus der Industrie bezüglich ihrer Einschätzung der Wichtigkeit der genannten Regeln. Außerdem wurden die Probanden gefragt, welche Software-Qualitätsattribute ihrer Meinung nach von den Regeln beeinflusst werden. Das Ergebnis zeigt, dass Benutzbarkeit und Wartbarkeit die am häufigsten gewählten Qualitätsattribute darstellen. Beide genannten Qualitätsattribute werden wiederum von Verständlichkeit beeinflusst. Diese Ergebnisse beruhen auf der Meinung der Experten und wurden nicht durch weitere Daten belegt.
Um den genannten Problemen entgegenzuwirken, soll im Rahmen dieser Arbeit für ein Set von Regeln der Einfluss auf die Verständlichkeit von REST APIs empirisch untersucht werden.

1.2 Zielsetzung

Der Fokus dieser Arbeit liegt darauf, zu untersuchen, ob und vor allem welche Design-Regeln für REST APIs einen Einfluss auf die Verständlichkeit haben. Die im vorherigen Abschnitt beschriebene Delphi-Studie [32] dient dabei als Grundlage. Auf Basis der Studienergebnisse sollen Regeln ausgewählt werden, die von den Industrieexperten als wichtig eingeschätzt wurden und einen Einfluss auf die Verständlichkeit haben. Anschließend soll im Rahmen eines kontrollierten Experiments überprüft werden, ob ein positiver oder unter Umständen negativer Einfluss für diese Regeln bestätigt werden kann.
Neben der messbaren Verständlichkeit wird auch der Effekt auf die wahrgenommene Schwierigkeit eine REST API zu verstehen beleuchtet. Hier wird zusätzlich untersucht, ob eine signifikante Korrelation zwischen den beiden genannten Variablen gemessen werden kann. Zuletzt soll analysiert werden, auf welche Art und Weise die individuellen Erfahrungen mit REST APIs die Ergebnisse beeinflussen. Dazu gehört neben der Anzahl an Jahren auch die Sichtweise und Berufsgruppe, von welcher aus Erfahrungen mit REST APIs gesammelt wurden. Dadurch soll ein tieferes Verständnis und eine empirische Grundlage dafür geschaffen werden, welche der Regeln zum Zweck eines besseren Verständnisses in der Praxis umgesetzt werden sollten.

1.3 Gliederung

Diese Arbeit ist in sieben Kapitel gegliedert:
Kapitel 2 – Grundlagen legt die Grundlagen für diese Arbeit dar. Dazu werden Informationen über REST und dem Software-Qualitätsattribut Verständlichkeit gegeben.
Kapitel 3 – Verwandte Arbeiten stellt verwandte Arbeiten vor und erläutert, wie sich diese Arbeit davon abgrenzt.
Kapitel 4 – Methodik beschreibt den Aufbau und die Methodik des Experiments. Dazu gehört neben dem Forschungsdesign auch eine Beschreibung der Teilnehmer, Materialien, Aufgaben, Variablen und den zu klärenden Forschungsfragen.
Kapitel 5 – Ergebnisse beschreibt die Ergebnisse des Experiments und geht dabei auf die Forschungsfragen ein.

Kapitel 6 – Diskussion interpretiert die Ergebnisse des Experiments und weißt auf die Limitationen hin, die bei der Bewertung der Ergebnisse zu berücksichtigen sind.
Kapitel 7 – Zusammenfassung und Ausblick fasst die Arbeit zusammen und gibt einen Ausblick.

2 Grundlagen

In dieser Arbeit soll der Einfluss von Design-Regeln auf die Verständlichkeit von REST APIs untersucht werden. Auf Basis der Zielsetzung werden in diesem Kapitel zunächst Konzepte zu REST thematisiert und anschließend eine Erläuterung des Qualitätsattributes Verständlichkeit gegeben. Diese Grundlagen werden zum Verständnis der Arbeit benötigt.

2.1 REST

Der Architekturstil REST, kurz für *Representational State Transfer*, wurde im Jahre 2000 von Roy Thomas Fielding definiert [17]. Im Folgenden werden die Prinzipien von REST sowie die Grundlagen einer REST API erläutert. Anschließend wird auf Best Practices und Design-Regeln für REST APIs eingegangen. Im letzten Teil dieses Abschnitts wird das Richardson-Reifegradmodell beschrieben und Informationen über die Dokumentation von REST APIs gegeben.

2.1.1 REST Prinzipien

Wie in Abschnitt 1.1 beschrieben, existiert für REST keine offizielle Spezifikation. Fielding definiert jedoch für REST die folgenden sechs Prinzipien mit ihren jeweiligen Vor- und Nachteilen [17]:

Client-Server Client und Server existieren getrennt voneinander als einzelne Anwendungen, die jeweils ihre eigene Rolle einnehmen. Dadurch können beide Anwendungen auf unterschiedlichen Technologien basieren und unabhängig voneinander skaliert werden. Des Weiteren wirkt sich die Trennung positiv auf die Portabilität aus.

Zustandslos Es wird von Zustandslosigkeit gesprochen, wenn weder der Server, noch der Client, Informationen für die Kommunikation untereinander zwischenspeichern müssen. Das bedeutet, in einer Nachricht vom Client müssen sämtliche Informationen enthalten sein, die der Server benötigt, um die Anfrage bearbeiten zu können. Falls beispielsweise eine Authentifizierung notwendig ist, müssen diese Daten bei jeder Anfrage mitgesendet werden. Der positive Effekt hiervon ist die Steigerung der Skalierbarkeit, Zuverlässigkeit und Sichtbarkeit. Dieses Prinzip besitzt jedoch auch Nachteile. Die größere Menge an Informationen in den Nachrichten wirkt sich beispielsweise negativ auf die Latenz aus.

Cache Um den Nachteilen für die Zustandslosigkeit entgegenzuwirken wird als weiteres Prinzip der Cache definiert. Durch die Verwendung von Caching kann beispielsweise der Austausch von Nachrichten verringert werden, wodurch wiederum ein positiver Effekt auf die Latenz entsteht. Hierfür wird vorausgesetzt, dass eine Nachricht eindeutig als cachefähig markiert wird.

Einheitliche Schnittstelle Dieses Prinzip fördert die Trennung von Client und Server und die unabhängige Entwicklung der einzelnen Komponenten. Grund hierfür ist, dass die Realisierung der Komponenten, also beispielsweise Architektur oder Programmiersprache, von der angebotenen Schnittstelle entkoppelt sind. Diese standardisierte Kommunikation besitzt jedoch einen negativen Einfluss auf die Effizienz, da sie nicht explizit auf die jeweilige Anwendung zugeschnitten ist.
Fielding [17] definiert für die einheitliche Schnittstelle zusätzlich vier weitere Schnittstellenprinzipien. Zu diesen zählt die Identifikation einer Ressource mittels URIs und die Manipulation der Ressourcen durch die Verwendung von Repräsentationen. Beide Prinzipien werden in Abschnitt 2.1.2 näher erläutert. Das dritte Prinzip ist die Verwendung von selbsterklärenden Nachrichten. Wie der Name andeutet, ist damit gemeint, dass jede in REST ausgetauschte Nachricht alle notwendigen Informationen darüber enthält, wie diese zu verarbeiten ist. HATEOAS (Hypermedia as the engine of Application State) stellt das letzte der vier Schnittstellenprinzipien dar und wird in Abschnitt 2.1.4 beschrieben.

Schichtensystem Eine REST Architektur besteht aus mehreren Schichten, die unterschiedliche Funktionalitäten abdecken. Dadurch wird die Komplexität des Gesamtsystems reduziert. Die verschiedenen Schichten sind isoliert voneinander, können jedoch untereinander kommunizieren. Zusammen bilden die Schichten eine Hierarchie. Der Nachteil von einem Schichtensystem ist eine erhöhte Latenz bei der Verarbeitung von Daten, was wiederum einen negativen Effekt auf die Performanz hat.

Code on Demand Dieses Prinzip ist im Vergleich zu den bisherigen fünf optional. Die Idee von Code on Demand ist es, Code als Applets oder Skripte mittels der API zu übertragen, sodass dieser vom Client ausgeführt werden kann. Dies hat einen positiven Effekt auf die Erweiterbarkeit des Systems, reduziert jedoch die Sichtbarkeit.

2.1.2 REST API Grundlagen

Eine Programmierschnittstelle (API), die den im vorherigen Abschnitt beschriebenen Prinzipien folgt, wird *REST API* genannt. Diese wird dazu verwendet, um Nachrichten zwischen einem Client und einem webbasierten Service (Server) auszutauschen [36]. Im Folgenden werden die wichtigsten Bestandteile dieser Kommunikation beschrieben.
Bei REST handelt es sich um eine ressourcenorientierte Architektur [43]. Eine Ressource kann jeder beliebige Gegenstand oder Sache sein. Sie kann beispielsweise ein Datenbankeintrag, ein Dokument, aber auch ein physikalisches oder abstraktes Objekt sein. Typischerweise ist eine Ressource auf einem Computer abgelegt [50].
Um eine Ressource zu identifizieren und lokalisieren, werden bei REST sogenannte *Uniform Resource Identifiers* eingesetzt [43]. Abbildung 2.1 beschreibt die Syntax einer URI anhand eines Beispiels. Die Komponente *Schema* wird immer benötigt. *Autorität* oder *Pfad* muss ebenfalls

Abbildung 2.1: URI Syntax anhand eines Beispiels, angelehnt an den RFC 3986 Standard [6]

gegeben sein. Die Komponente *Abfrage* dagegen ist optional und hängt von der zugrunde liegenden API ab [6].
Wenn ein Client auf eine Ressource vom Server zugreifen möchte oder die Absicht hat, diese zu

HTTP Methode	**Beschreibung**
GET	Wird verwendet, um eine oder mehrere Ressourcen vom Server abzufragen und eine Repräsentation der Ressource zu erhalten.
HEAD	Ähnlich wie GET, es wird jedoch keine Repräsentation der Ressource vom Server geschickt, sondern lediglich die dazugehörigen Metadaten. So kann beispielsweise überprüft werden, ob eine Ressource existiert.
DELETE	Diese Methode wird verwendet, um eine Ressource vom Server zu löschen.
PUT	Diese HTTP Methode hat zwei verschiedene Anwendungsfälle. Einerseits wird sie genutzt, um eine Ressource, welche in einer Sammlung (englisch: *Collection*) abgelegt ist, zu verändern. Der Client schickt dazu die manipulierte Repräsentation der Ressource an den Server, welcher die alte Ressource mit der neuen ersetzt. In diesem Fall wird die URI vom Server bestimmt. Der zweite Anwendungsfall ist das Anlegen einer neuen Ressource innerhalb eines Speichers (englisch: *Store*). Im Vergleich zur Sammlung legt hier nicht der Server, sondern der Client die URI der Ressource fest. Sollte bereits eine Ressource unter der angegeben URI existieren, wird diese mit der neuen überschrieben.
POST	Diese Methode wird angewendet, wenn eine neue Ressource für eine Sammlung auf dem Server erstellt werden soll. POST wird außerdem verwendet, um einen Controller auszulösen.
PATCH	Ähnlich wie PUT wird auch PATCH zum Manipulieren von Ressourcen verwendet. Der Unterschied besteht darin, dass bei PATCH lediglich ein Teil der Ressource verändert wird.

Tabelle 2.1: Beschreibung der HTTP Methoden [62]

manipulieren, muss neben der URI die Art der Operation angegeben werden. REST verwendet dazu kein eigenes Format, sondern stützt sich auf die vorhandenen HTTP Methoden [43]. Tabelle 2.1 zeigt eine Übersicht der am häufigsten verwendeten HTTP Methoden. Wenn Nachrichten zwischen Client und Server ausgetauscht werden, wird nicht die Ressource selbst, sondern immer eine Repräsentation dieser in der Nachricht verschickt. Typischerweise ist das Format für die

Repräsentationen JSON[1] [51]. Es können jedoch auch andere Formate wie beispielsweise XML, HTML oder JPEG verwendet werden [62].
Zu einer Anfrage vom Client an den Server gehören neben der URI und der semantisch korrekten HTTP Methode noch weitere optionale Informationen:

- **Header:** Im Header einer Anfrage befinden sich weitere Metainformationen, die für den Server von Interesse sind. Dazu gehört beispielsweise das Format der Repräsentation, das der Client vom Server in dessen Antwort erwartet. Des Weiteren können im Header Informationen über die Autorisierung des Nutzers, wie etwa ein Token, enthalten sein. Dies wird vom Server benötigt, um den Zugriff auf die Ressource zu gewähren oder abzulehnen [23].
- **Parameter:** Hier unterscheidet man zwischen mehreren Arten. Abfrageparameter werden wie in Abbildung 2.1 zu sehen am Ende der URI mittels eines *"?"* angehangen. Bei der Verwendung mehrere Abfrageparameter wird das Symbol *"&"* genutzt, um diese voneinander zu trennen [36]. Pfadvariablen dagegen werden vor der Durchführung der Anfrage in der URI ersetzt. Dazu wird beim Erstellen der URI ein Platzhalter an die entsprechende Stelle gesetzt. Folgendes Beispiel zeigt eine URI mit einem Platzhalter:

  http://test.com/employee/{employeeId}

  Die *employeeId* stellt den Platzhalter dar, der vor der Durchführung der Anfrage mit dem richtigen Identifier ersetzt wird [36]. Eine weitere Art der Parameter ist der *Request Body*. Dadurch können ganze Objekte im Body der Nachricht verschickt werden. Dieser wird hauptsächlich bei POST oder PUT Anfragen genutzt. Bei Nachrichten, die die HTTP Methode GET verwenden, ist die Verwendung von Request Bodys verboten [20].

Wenn der Server die Anfrage eines Clients erhält, bearbeitet er diese und sendet eine Antwort an den Client zurück. Je nach Art der Operation kann die Antwort unterschiedliche Komponenten beinhalten. Bei einer GET Anfrage befindet sich beispielsweise die Repräsentation der angefragten Ressource im Body der Nachricht. Die Antwort beinhaltet jedoch immer einen sogenannten *HTTP Status Code*. Dabei handelt es sich um einen dreistelligen Code, mithilfe dessen der Server ausdrücken kann, zu welchem Ergebnis eine bearbeitete Anfrage geführt hat. Die erste Zahl des Codes bestimmt, zu welcher der folgenden fünf Kategorien der Status Code gehört [62]:

- **Codes zur Information (1XX):** Die Codes aus dieser Kategorie informieren darüber, dass eine Anfrage vom Server entgegengenommen wurde und noch bearbeitet wird.
- **Codes zur erfolgreichen Durchführung (2XX):** Diese Codes geben an, dass eine Anfrage erfolgreich vom Server bearbeitet wurde. Zu den bekanntesten zählen hier 200 (OK - Die Anfrage war erfolgreich), 201 (Created - Die Ressource wurde erfolgreich angelegt) und 202 (Accepted - Die Anfrage war erfolgreich und befindet sich noch in der Verarbeitung).
- **Codes für die Umleitung (3XX):** Diese deuten typischerweise darauf hin, dass die angefragte Ressource an eine andere Stelle hinverschoben wurde. Ein Beispiel hierfür ist 301 (Moved Permanently). Im Falle eines solchen Status Codes ist notwendig, dass der Client weitere Schritte durchführt, um die Anfrage erfolgreich abzuschließen.

[1] JSON: JavaScript Object Notation, https://www.json.org/json-en.html

- **Codes für Fehler auf der Client-Seite (4XX):** Codes aus dieser Kategorie signalisieren, dass die Anfrage aufgrund eines Fehlers auf der Client-Seite fehlgeschlagen ist. 400 (Bad Request - Die Anfrage ist fehlerhaft), 401 (Unauthorized - Der Client hat keine oder fehlerhafte Zugangsdaten gesendet) und 404 (Not Found - Die angefragte Ressource konnte nicht gefunden werden, da sie nicht existiert) gehören bei dieser Kategorie zu den am häufigsten genutzten.
- **Codes für Fehler auf der Server-Seite (5XX):** Ähnlich wie bei der vorherigen Kategorie, der Grund für die fehlgeschlagene Anfrage liegt hier jedoch auf der Server-Seite. Die verbreitetsten Codes dieser Kategorie sind 500 (Internal Server Error - Bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten) und 503 (Service Unavailable - Der Server befindet sich in Wartung oder ist überlastet).

2.1.3 REST Design-Regeln und Best Practices

Die in 2.1.1 erläuterten REST Prinzipien können von Entwicklern unterschiedlich interpretiert werden. Dies führt wiederum zu unterschiedlich designten REST APIs, was mitunter problematisch für die Integration mehrerer APIs ist [51]. Daher wurden in unterschiedlichen Publikationen Design-Regeln und Best Practices für REST APIs aufgestellt, um die Entwickler bei der Erstellung REST konformer APIs zu unterstützen. Auf die Best Practices fokussieren sich die Bücher von Richardson und Ruby [50], Webber et al. [63], Allamaraju [1], Patni [43] und Subramanian et al. [59]. Neben den genannten Büchern beschäftigten sich auch die Publikationen von Fredrich [18] und Giessler et al. [19] mit den Best Practices für REST APIs. Li und Chou [34] stellen in ihrer Arbeit einen Ansatz vor, der dazu genutzt werden kann, mittels REST Diagrammen APIs in Form von Petri-Netzen zu entwerfen, ohne die REST Prinzipien zu verletzen. 2014 wurde von Zhou et al. [66] außerdem ein Framework veröffentlicht, um anhand von Design-Pattern eine RESTful API zu erstellen. Der Fokus liegt dabei vor allem auf der Verwendung von Hypertext.
Design-Regeln für REST APIs wurden in den Büchern von Masse [36] und Subramanian et al. [59] veröffentlicht. Masse stellt einen Katalog aus über 80 Regeln. Diese sind in verschiedene Kategorien eingeteilt, welche die unterschiedlichen Komponenten von REST APIs abdecken. Es existieren daher Regeln für das Design der URIs, sowie für die korrekte Verwendung von HTTP Methoden und Status Codes. Regeln aus diesen drei Kategorien werden in der für diese Arbeit durchgeführten Studie untersucht. Neben den genannten Kategorien sind auch Regeln zu Metadaten, Repräsentationsdesign und Anliegen aus Client-Sicht in Masses Buch [36] zu finden. Aufgrund der Vielzahl an Regeln und Best Practices, wird in diesem Abschnitt keine detaillierte Beschreibung aller Regeln gegeben. Stattdessen wird die finale Auswahl der in dieser Arbeit untersuchten Regeln mit Beschreibungen in Abschnitt 4.4 dargelegt.

2.1.4 Richardson-Reifegradmodell

Das Richardson-Reifegradmodell wurde von Leonard Richardson 2008 vorgestellt [49] und kann dazu verwendet werden, um zu überprüfen, in welchem Grad eine API konform zu den REST Prinzipien ist. Wie in Abbildung 2.2 zu sehen besteht das Modell aus vier aufeinander aufbauenden Stufen. Richardson [49] beschreibt diese wie folgt:

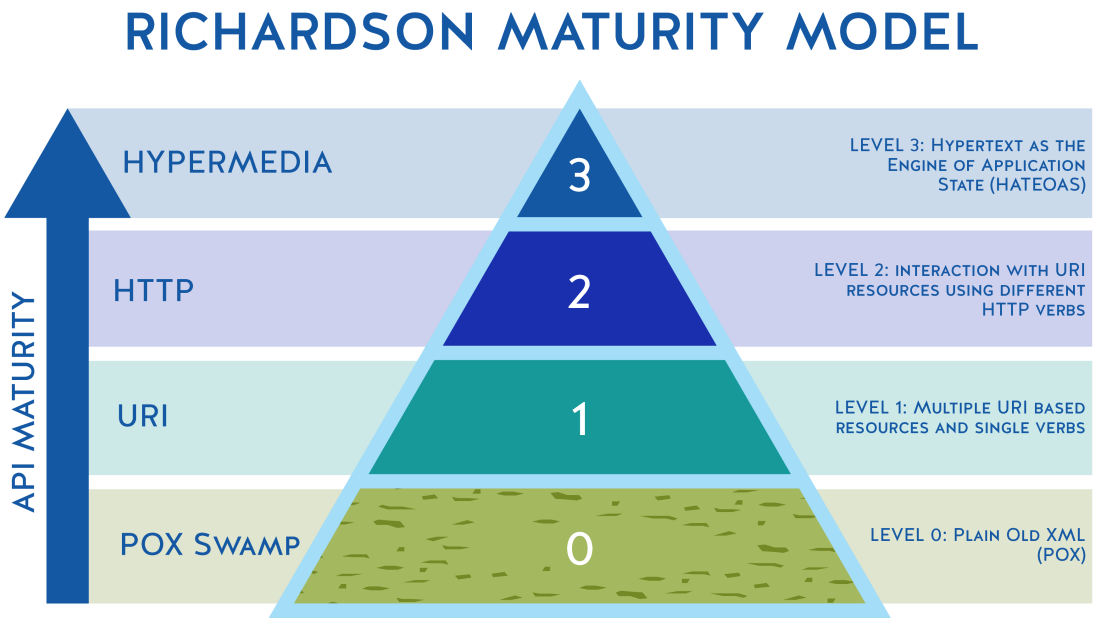

Abbildung 2.2: Abbildung des Richardson Maturity Models [52]

- Die unterste Stufe (Level 0) ist typisch für XML-RPC und SOAP. Auf dieser Stufe wird eine einzelne URI verwendet, an die alle Anfragen gesendet werden. Des Weiteren wird eine einzige HTTP Methode, wie beispielsweise POST verwendet, unabhängig davon welche Art der Operation durchgeführt werden soll.
- Die nächste Stufe (Level 1) wird erreicht, indem nicht mehr nur eine URI verwendet wird, sondern für jede Ressource eine eigene. Wie auch schon bei der vorherigen Stufe wird lediglich eine HTTP Methode für sämtliche Anfragen verwendet.
- Auf Level 2 kommt zusätzlich zu individuellen URIs für jede Ressource die Verwendung von unterschiedlichen HTTP Methoden dazu. Abhängig von der Art der Operation wird die semantisch korrekte HTTP Methode gewählt.
- Auf der höchsten Stufe (Level 3) wird darüber hinaus Hypermedia verwendet, um auf Basis der Antwort in Form eines Dokuments innerhalb der Anwendung zu navigieren. Dies folgt dem Prinzip von *HATEAOS* (Hypertext as the engine of Application State). Die Navigation, also die Lokalisierung von Ressourcen, erfolgt über Hyperlinks, die im Dokument enthalten sind. Dies hat den Vorteil, dass serverseitige Änderungen vereinfacht werden. Neben den Hyperlinks können zusätzlich sogenannte Hypermedia *Forms* genutzt werden, um dem Client Informationen über die Verwendung der Ressource zu geben.

2.1.5 Dokumentation von REST APIs

Während die Entwickler der APIs das notwendige Wissen über die Funktionalität und Nutzungsweise haben, ist dies für einen potenziellen Konsumenten der API nicht der Fall. Um dem entgegenzuwirken und die Nutzung einer API zu vereinfachen, werden diese dokumentiert [13].

Für die Dokumentation von REST APIs stehen mehrere Werkzeuge zur Verfügung, die wiederum auf verschiedenen Spezifikationen basieren. Zu den drei am meisten genutzten Spezifikationen gehören OpenAPI[2] (ehemals Swagger[3]), RESTful API Modeling Language (kurz RAML)[4] und API Blueprint[5] [13][61]. Diese unterscheiden sich unter anderem in den verwendeten Formaten. OpenAPI verwendet JSON, RAML Markdown und API Blueprint nutzt das YAML Format [13]. Da für die Dokumentation der REST APIs in dieser Arbeit OpenAPI beziehungsweise Swagger verwendet wird, werden die beiden anderen genannten Spezifikation nicht näher erläutert.
Swagger bietet mit dem Swagger-Editor[6] ein Werkzeug zur Modellierung von REST APIs an, welches sowohl lokal, als auch online über die verlinkte Webseite genutzt werden kann. Abbildung 2.3 zeigt die online verfügbare Version des Editors. Auf der linken Seite des Editors wird die API mit allen ihren Endpunkten über Code im JSON-Format modelliert. Auf der rechten Seite wird anschließend die vom Editor erstellte Dokumentation angezeigt. Dabei werden die verschiedenen HTTP Methoden farblich hervorgehoben. Durch einen Klick auf die einzelnen Endpunkte werden außerdem zusätzliche Details, wie beispielsweise die Parameter, angezeigt.

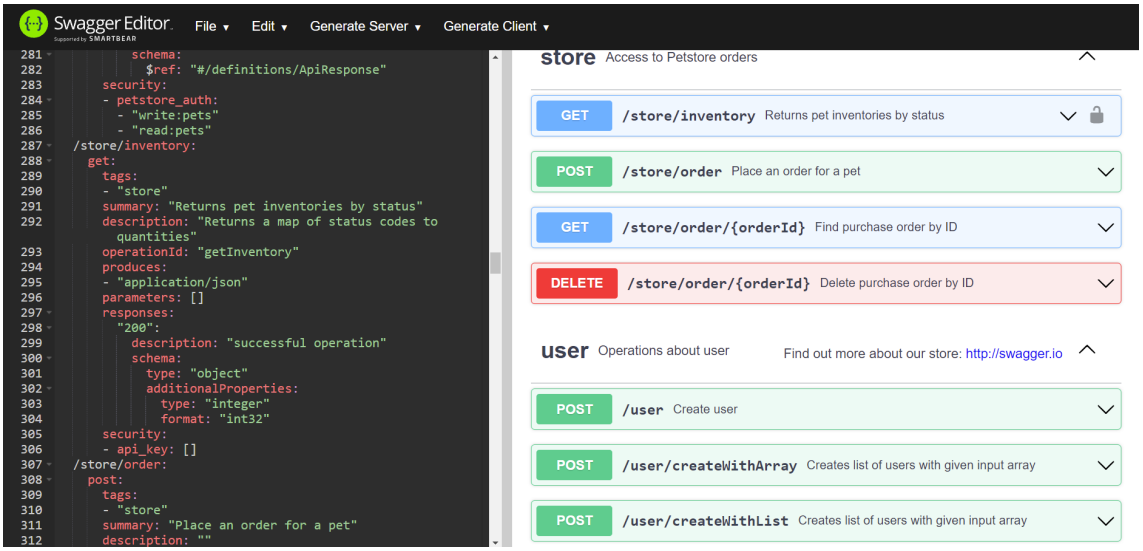

Abbildung 2.3: Online-Variante des Swagger-Editors (Quelle: `https://editor.swagger.io/`)

2.2 Verständlichkeit

Verständlichkeit gehört nach dem ISO 9126 Standard [25] zu den Software-Qualitätsattributen und wird als Aufwand, der nötig ist, um eine Anwendung oder System zu verstehen, definiert. Wie in Abbildung 2.4 zu sehen, ist sie eine der Untermetriken und beeinflusst die Benutzbarkeit. Dieser Standard wurde 2011 überarbeitet und durch die ISO 25010 [24] ersetzt. In dieser

[2] `https://www.openapis.org/`
[3] `https://swagger.io/`
[4] `https://raml.org/`
[5] `https://apiblueprint.org/`
[6] `https://editor.swagger.io/`

ist Verständlichkeit keine explizite Untermetrik von Benutzbarkeit mehr, sondern wurde durch Erkennbarkeit der Angemessenheit ersetzt. Diese wird nach der ISO 25010 [24] als Grad, in dem Nutzer erkennen können, ob ein Produkt oder System für ihre Bedürfnisse geeignet ist, definiert. Da es zur Erkennung dessen notwendig ist, das Produkt oder System zu verstehen, kann dieses Attribut mit der Verständlichkeit aus ISO 9126 assoziiert werden.
Boehm et al. [7] geben außerdem an, dass Verständlichkeit das Qualitätsattribut Wartbarkeit beeinflusst. Die Benutzbarkeit beziehungsweise Wartbarkeit decken dabei unterschiedliche Sichtweisen in Bezug auf REST APIs ab. Während mit der Benutzbarkeit die Verständlichkeit aus Nutzersicht adressiert wird, zielt Wartbarkeit auf die Verständlichkeit aus der Entwicklerperspektive ab.

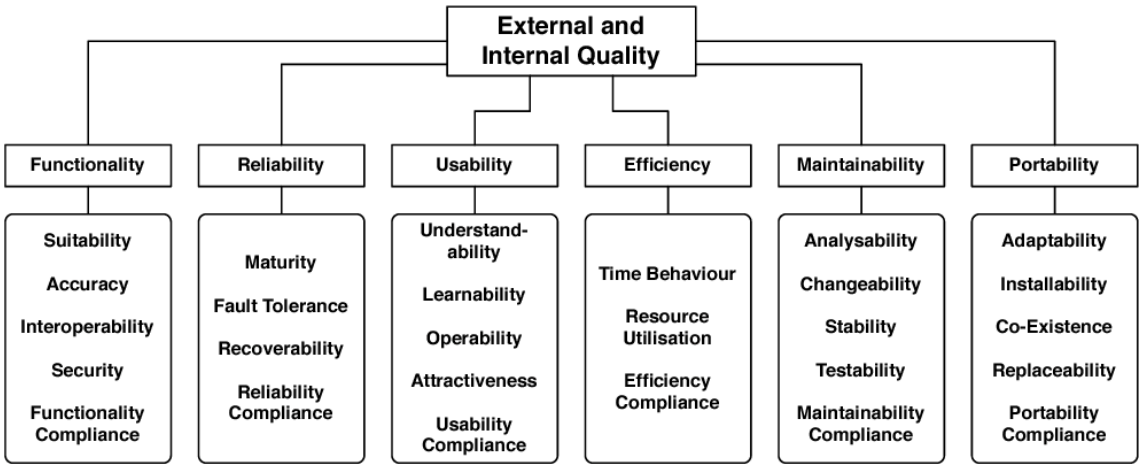

Abbildung 2.4: ISO 9126 Modell der Software-Qualitätsattribute [65]

Ein wichtiger Aspekt bei der Untersuchung von Verständlichkeit ist die Wahl geeigneter Metriken. Rajlich und Cowan [46] schlagen für Experimente mit Verständnisfragen drei abhängige Variablen vor. Die Korrektheit der Antwort eines Teilnehmers, die benötigte Zeit für eine korrekte Antwort sowie die benötigte Zeit für eine falsche Antwort. Feitelson [15] untersuchte ebenfalls Experimente zum Verständnis von Code. Der Autor gibt an, dass die am häufigsten verwendete abhängige Variable die benötigte Zeit für eine korrekte Antwort ist. Die Zeit für eine falsche Antwort dagegen wird meist nicht genutzt.
Des Weiteren legt er verschiedene Überlegungen für die Messung von Verständlichkeit dar. Zum einen muss bei der Bewertung der Korrektheit einer Antwort berücksichtigt werden, dass es überhaupt möglich ist festzulegen, ob eine Antwort korrekt ist oder nicht. Wird nach dem Rückgabewert einer Methode gefragt, kann dies ohne großen Aufwand bestimmt werden. Wird eine Frage dagegen offener formuliert, müssen weitere Maßnahmen getroffen werden, um die Korrektheit zu bestimmen. Die Frage nach dem Zweck eines Codefragments ist ein Beispiel für eine solche Frage.
Zum anderen muss im Falle einer Bewertung der Korrektheit und Zeit entschieden werden, ob beide Metriken für das Ergebnis kombiniert werden oder nicht. Für die Kombination beider Metriken existieren verschiedene Ansätze. Bergersen et al. [5] kombinieren die beiden Metriken mittels einer Einteilung in verschiedene Kategorien. Eine Kategorie ist beispielsweise ein korrektes Ergebnis mit langsamer Zeit oder ein korrektes Ergebnis mit schneller Zeit. Ein weiterer Ansatz wird von Beniamini et al. [4] gegeben. Die Autoren berechnen zunächst die Erfolgsrate eines Teilnehmers und dividieren diese durch die benötigte Zeit.

Eine dritte Variante zur Kombination von Korrektheit und Zeit wurde von Scalabrino et al. [55] vorgeschlagen. Sie verwenden für die Berechnung des Codeverständnisses *Timed Actual Understandability (TAU)*. TAU berechnet sich dabei wie folgt:

$$TAU = AU \times (1 - \frac{TNPU}{maxTNPU})$$

Bei AU (Actual Understandability) handelt es sich um die tatsächliche Verständlichkeit, also die Korrektheit. TNPU (Time Needed for Perceived Understandability) ist die Zeit, die ein Proband benötigt, um eine Aussage darüber treffen zu können, ob er den Code verstanden hat oder nicht. Bei der Berechnung von TAU wird nicht die absolute Zeit, sondern die relative Zeit verwendet, um den Einfluss von Korrektheit und Zeit auf das Gesamtergebnis anzugleichen.

3 Verwandte Arbeiten

In folgendem Abschnitt werden verwandte Arbeiten vorgestellt. Hierbei wird auf Arbeiten eingegangen, die die Einhaltung von Best Practices und die Konformität zu den REST Prinzipien evaluieren. Außerdem werden Arbeiten, die einen Bezug zwischen REST und Qualitätsattributen herstellen, untersucht. Im letzten Abschnitt dieses Kapitels wird eine Zusammenfassung gegeben und erläutert, wie sich diese Arbeit von den bisherigen unterscheidet.

3.1 Evaluation der Best Practices und Prinzipien von REST

2010 wurden 222 webbasierte APIs von Maleshkova et al. [35] analysiert. In Bezug auf REST kamen sie zu dem Ergebnis, dass die Entwickler der APIs die Prinzipien von REST missachten. Sie begründen dies damit, dass APIs basierend auf Operationen und Methoden entworfen werden, anstatt wie von REST nahegelegt anhand von Ressourcen.
Renzel et al. [48] untersuchten 2012 die 20 beliebtesten RESTful APIs, welche der Webseite *ProgrammableWeb* entnommen wurden. Die APIs wurden anschließend auf 17 verschiedene Kriterien für RESTful APIs untersucht. Zu den Kriterien gehört beispielsweise die Verwendung aussagekräftiger HTTP Status Codes oder die korrekte Verwendung der HTTP Methoden. Das Ergebnis zeigt, dass der Großteil der APIs nicht RESTful sind, obwohl dies von den Entwicklern der APIs beworben wird.
Auch Bülthoff und Maleshkova [9] betrachteten APIs der *ProgrammableWeb* Webseite. Insgesamt 45 APIs wurden auf 17 verschiedene Eigenschaften, wie beispielsweise der verwendete HTTP Status Code im Fehlerfall, getestet. Sie bemerkten dabei, dass die APIs deutliche Unterschiede bei der Einhaltung der REST Prinzipien aufweisen. Die Autoren führen das Ergebnis auf einen fehlenden Standard, wie es unter anderem bei SOAP der Fall ist, zurück.
Palma et al. [40] stellten 2014 *SODA-R* vor, ein heuristikbasierter Ansatz, um Pattern und Antipattern von REST zu erkennen. Dabei wird die Erkennung von fünf Pattern und acht Antipattern unterstützt. Zusätzlich wurde SODA-R auf 12 verschiedene bekannte APIs angewendet. Das Ergebnis zeigt, dass eine hohe Präzision bei der Erkennung der Pattern gewährleistet ist. Außerdem wurde festgestellt, dass die APIs von Facebook und Best Buy weniger Antipattern und mehr Pattern enthalten und somit gut designt sind. Bei den APIs von Twitter und Dropbox ist das Gegenteil der Fall.
Dieser Ansatz wurde 2015 von Palma et al. [42] weiter verfolgt. *DOLAR (Detection of Linguistic Antipatterns in REST)* dient ebenfalls zur Erkennung von Pattern und Antipattern von RESTful APIs. Mittels diesem Verfahren können jeweils fünf Pattern und Antipattern erfasst werden. Zur Evaluierung von DOLAR wurden 15 bekannte APIs untersucht, wobei mit einer Präzision von ungefähr 80 % ein Pattern erkannt wurde. Zudem wurde festgestellt, dass die unterstützten Pattern in den untersuchten APIs zu finden waren.
DOLAR wurde von Palma et al. [41] ein weiteres Mal betrachtet, verbessert und unter dem neuen Namen *SARA (Semantic Analysis of RESTful APIs)* veröffentlicht. Neben der gestiegenen Zahl

der Pattern und Antipattern auf jeweils sechs besitzt SARA im Vergleich zu DOLAR eine höhere Präzision in Bezug auf die Erkennung der Pattern.
2016 betrachteten Perillo et al. [45] 73 Best Practices aus der Literatur für die Steigerung der Verständlichkeit und Wiederverwendbarkeit von REST APIs. Diese wurden in die fünf Kategorien *URI*, *Anfragemethoden*, *Fehlerbehandlung*, *HTTP Header* und *Andere* gegliedert. Zusätzlich wurden die APIs von drei Cloud Anbietern auf die Einhaltung der 73 Best Practices untersucht. Dabei wurde festgestellt, dass die Konformität der drei APIs mit den Best Practices zwischen 56 % und 66 % liegt. Die Autoren merken an, dass einer der Gründe, warum keine höheren Ergebnisse erzielt werden konnten, die strikte und präzise Spezifikation der Best Practices ist. Weiterhin geben sie an, dass das Ergebnis dennoch ein annehmbares Niveau in Bezug auf den Reifegrad der REST APIs suggeriert.
Auch Rodriguez et al. [51] untersuchten die Konformität von REST APIs mit den bestehenden Prinzipien und Best Practices. Zu diesem Zweck analysierten sie 78 GB an Daten des Netzwerkverkehrs eines Mobilfunkanbieters. Unter anderem wurde dokumentiert, welche HTTP Methoden und HTTP Status Codes verwendet und in welchem Format die Daten verschickt wurden. Die Konformität mit den Best Practices wurde mittels Heuristiken getestet. Zusätzlich wurde so der Reifegrad nach dem Richardson-Reifegradmodell für die API einer Domain bestimmt. Das Ergebnis zeigt, dass die wenigsten APIs das höchste Level des Reifegradmodells erreichen. Die Autoren merken außerdem an, dass ein deutlich erkennbarer Unterschied zwischen Theorie und Praxis in Bezug auf das Design von REST APIs besteht.
Haupt et al. [21] nutzten ein kanonisches Metamodell, um die Struktur von REST APIs zu durchleuchten. Zur Validierung wurden 268 API Beschreibungen verwendet. Dabei wurde herausgefunden, dass eine ungleichmäßige Verteilung bei der Größe der APIs besteht. Der Mittelwert liegt bei neun Ressourcen, wohingegen jedoch auch Ausreißer mit mehr als 250 Ressourcen gegeben waren. Zusätzlich wurde erkannt, dass viele der APIs nur einen Lesezugriff auf die Ressourcen gaben und keine andere Art der Manipulation möglich war.
In der Arbeit von Neumann et al. [39] 2018 betrachteten die Autoren 500 REST APIs und prüften, ob diese die Prinzipien und Best Practices beachten. Sie kamen zu dem Ergebnis, dass lediglich 0,8 % der APIs alle REST Richtlinien befolgten. In Bezug auf die Best Practices war ein ähnliches Ergebnis zu beobachten. Zwar wurden die Hälfte dieser von mehr als 50 % der APIs eingehalten, jedoch erfüllte keiner der APIs sämtliche Best Practices.
Koschel et al. [31] überprüften 2019 zehn REST APIs auf ihre Eingliederung in das Reifegradmodell von Richardson. Neun der analysierten APIs erreichten dabei mindestens Level 2. Vier davon implementieren HATEOAS und erreichten mit Level 3 die höchste Stufe des Reifegradmodells.

3.2 Evaluation von Qualitätsattributen in Bezug auf REST

Auch die Evaluation von Qualitätsattributen in Bezug auf REST wurde in verschiedenen Arbeiten beleuchtet.
2013 untersuchten Fernandes et al. [16] die Performanz von RESTful Webservices und dem *Advanced Message Queuing Protocol (AMQP)*. Bei Letzterem handelt es sich um ein Protokoll für Message-orienterte Middlewares, um Nachrichten zu empfangen und zu versenden. Um die Performanz zu vergleichen wurden für jeweils 30 Minuten Nachrichten entweder mittels REST oder dem AMQP von einem Client an einen Server geschickt. Die Autoren kamen zu dem Ergebnis, dass für große Mengen an Nachrichten AMPQ eine bessere Performanz bietet und daher für diesen Fall

zu bevorzugen ist.
Eine ähnliche Untersuchung stellten auch Hong et al. [22] an, die die Performanz zwischen RESTful APIs und dem Nachrichtenbroker RabbitMQ[1] analysierten. RabbitMQ verwendet ebenfalls das in der Arbeit von Fernandes et al. behandelte AMQP. Als Basis wurde eine Microservice-basierte Webanwendung genutzt, welche von einer sich ändernden Anzahl an Nutzern gleichzeitig Anfragen gesendet bekommt. Die Autoren fanden heraus, dass die Performanz von REST mit steigender Anzahl an Anfragen schlechter wird, wohingegen bei RabbitMQ eine gleichbleibende Performanz festgestellt werden konnte. Bei einer geringen Anzahl gleichzeitiger Anfragen war jedoch eine höhere Performanz für den Ansatz mit REST zu messen. Das Ergebnis ist dementsprechend deckungsgleich mit der im vorherigen Absatz beschriebenen Arbeit.
Costa et al. [12] stellten 2014 Richtlinien auf, die genutzt werden können, um die Evaluation einer REST Architektur in Bezug auf Qualitätsattribute durchzuführen. Zu diesem Zweck wurden einerseits Interviews mit zwei Experten der Evaluierung von Architekturen geführt und andererseits eine Literaturrecherche angestellt. Die Autoren stellten anschließend Szenarien für zehn verschiedene Qualitätsattribute auf. Zusätzlich wurden Designfragen für die fünf Kategorien *Design der Ressourcen*, *Repräsentation und Identifikation*, *Dokumentation und Test* sowie *Verhalten* und *Sicherheit* formuliert. Diese sollten basierend auf dem zutreffenden Szenario bei der Evaluierung gestellt werden. Abschließend validierten die Autoren ihren Ansatz unter der Verwendung der *Architecture Tradeoff Analysis Method (ATAM)* [28] und einem webbasierten System.
Khan und Abbasi [30] verglichen 2015 SOAP und REST basierend auf sechs verschiedenen Parametern. Zu diesen gehören unter anderem auch Sicherheit und Netzwerklatenz. Zusätzlich wurden REST und SOAP jeweils für ein Middleware Framework implementiert, um die Performanz auf Basis der Skalierbarkeit und Effizienz zu überprüfen. Das Ergebnis der Arbeit zeigt, dass REST besser für mobile Anwendungen geeignet ist, da es im Vergleich zu SOAP einfacher zu entwickeln und leichtgewichtiger ist. Im Gegenzug ist SOAP für Anwendungen, in denen die Sicherheit eine große Rolle spielt, zu bevorzugen.
Auch Tihomirovs und Grabis [60] wogen REST und SOAP gegeneinander auf. Sie verwendeten dafür Software Evaluationsmetriken, die einen Einfluss auf die *Funktionalität*, *Qualität*, *Komplexität*, *Effizienz*, *Verlässlichkeit* und *Wartbarkeit* haben. Sie kamen sie zu dem Ergebnis, dass beide Vor- und Nachteile besitzen und je nach Anforderungen an die Anwendung entschieden werden sollte, welches der beiden verwendet wird. REST besitzt Vorteile bei der Performanz, Skalierbarkeit und Kompatibilität, wovon vor allem mobile Anwendungen profitieren. SOAP dagegen ist für komplexere Anwendungen mit Fokus auf Sicherheit die bessere Wahl. Dies deckt sich mit den Ergebnissen von Khan und Abbasi [30]. Einen Vorteil von REST gegenüber SOAP in Bezug auf die Performanz und Skalierbarkeit zeigt auch die Arbeit von Rathod [47].
Bogner et al. [8] stellten 2020 *RESTful API Metric Analyzer (RAMA)* vor. Dabei handelt es sich um einen Ansatz, der basierend auf der Schnittstellenbeschreibung von RESTful Services zehn Metriken für die Wartbarkeit auswertet. Sieben der zehn Metriken fokussieren sich auf die Komplexität, zwei adressieren die Kohäsion und die letzte Metrik beschäftigt sich mit der Größe. Die drei Formate *OpenAPI*, *RAML* und *WADL* werden für die Schnittstellenbeschreibungen unterstützt. Zusätzlich wurde eine Benchmark auf Basis von 1737 REST APIs erstellt.
In der Arbeit von Kotstein und Bogner [32] wurden Industrie Experten bezüglich ihrer Einschätzung zur Wichtigkeit der Design-Regeln für REST APIs von Masse [36] befragt. 45 % der Regeln wurden mit niedriger, 21 % mit mittlerer und die restlichen 34 % mit hoher Wichtigkeit bewertet. Außerdem

[1] https://www.rabbitmq.com/

sollten die Experten eine Einschätzung abgeben, welche Qualitätsattribute von der jeweiligen Regel beeinflusst werden. Nutzbarkeit, Wartbarkeit und Kompatibilität stellen dabei, für die mit mittlerer oder hoher Bedeutung bewerteten Regeln, die drei am häufigsten gewählten dar. Es konnte außerdem beobachtet werden, dass die Teilnehmer die zum Erreichen des zweiten Levels vom Richardson-Reifegradmodell notwendigen Regeln als sehr wichtig einschätzten. Für die Regeln zum Erreichen von Level 3 war dies nicht der Fall.

3.3 Zusammenfassung und Abgrenzung

In diesem Kapitel wurden Arbeiten vorgestellt, die sich mit der Evaluation von REST beschäftigten. Der größte Teil untersuchte, in welchem Ausmaß sich Entwickler von REST APIs an die vorgegebenen Prinzipien und Best Practices aus der Literatur halten. In Bezug auf die Software Qualitätsattribute wurden überwiegend Vergleiche zwischen REST und anderen Architekturstilen oder Technologien angestellt. Eine Untersuchung des direkten Einflusses von Best Practices oder Design-Regeln für REST auf die Qualitätsattribute ist dagegen selten anzutreffen. Eine Ausnahme ist die Arbeit von Kotstein und Bogner [32]. Vergleichbare Arbeiten sind im Kontext mit REST nicht bekannt. Daher stellt die durchgeführte Untersuchung auf Basis eines Experiments ein Novum in diesem Kontext dar. Die Erkenntnisse der Autoren sollen in dieser Arbeit aufgegriffen und weiter untersucht werden. Daher wird in dieser Arbeit der Einfluss der Design-Regeln auf die Verständlichkeit von REST APIs mittels eines kontrollierten Experiments empirisch untersucht. Dabei soll der Fokus vor allem auf Regeln liegen, die von den Industrieexperten in [32] mit hoher Wichtigkeit eingeschätzt wurden und nach deren Bewertung entweder einen Einfluss auf die Benutzbarkeit oder Wartbarkeit besitzen. Diese beiden Qualitätsattribute werden gewählt, da diese, wie in 2.2 beschrieben, von Verständlichkeit beeinflusst werden.

4 Methodik

In diesem Kapitel werden die verschiedene Aspekte der Methodik dargelegt. Der Aufbau dieses Kapitels orientiert sich an der Richtlinie zur Dokumentation von Experimenten im Bereich Softwaretechnik, welche von Jedlitschka et al. [27] vorgeschlagen wurde. Zunächst werden die zu beantwortenden Forschungsfragen definiert, sowie Teilnehmer, Material und Aufgaben beschrieben. Außerdem werden die untersuchten Regeln aufgelistet. Anschließend werden Treatments, abhängige und unabhängige Variablen, Hypothesen und das Forschungsdesign erläutert. Zuletzt wird die Durchführung des Experiments und der Analyse dargelegt. Die genutzten Artefakte werden auf der Webseite *Zenodo*[1] bereitgestellt. Dadurch soll die Reproduzierbarkeit des Experiments gewährleistet werden.

4.1 Forschungsfragen

Die folgenden Forschungsfragen werden für diese Arbeit aufgestellt:

RQ1 Welche Design-Regeln haben einen Einfluss auf die Verständlichkeit von RESTful APIs?

Dies stellt die zentrale Forschungsfrage dieser Arbeit dar. Es soll geklärt werden, ob und welche Design-Regeln einen positiven oder negativen Einfluss auf die Verständlichkeit von REST APIs besitzen. Es wird jedoch davon ausgegangen, dass ein positiver Effekt zu beobachten ist. Für diese Forschungsfrage wird in einem bestätigendem Anteil zunächst jede untersuchte Regel auf Basis einer gewählten Metrik für das Verständnis analysiert. Zusätzlich werden in einem explorativen Anteil die benötigte Zeit und die Korrektheit für alle Regel separat betrachtet.

Feitelson [15] merkt in seiner Arbeit an, dass es mitunter schwierig sein kann festzulegen, ob ein Teilnehmer die Aufgabe tatsächlich verstanden hat. Weiterhin wirft er die Frage auf, ob einzelne Verständnisfragen ausreichen, um die Verständlichkeit zu bewerten. Auch Scalabrino et al. [55] geben an, dass die Verständlichkeit eine subjektive Eigenschaft darstellt. Daher wird zusätzlich zu RQ1 die folgende Forschungsfrage untersucht:

RQ2 Korreliert die wahrgenommene Verständlichkeit von REST API Designs mit der tatsächlichen Verständlichkeit?

Diese Forschungsfrage wird explorativ untersucht. Die Teilnehmer des Experiments sollen die Verständlichkeit einer gezeigten REST API bewerten, nachdem sie eine Verständnisfrage zur selbigen beantwortet haben. Zum einen soll dadurch überprüft werden, welchen Einfluss die

[1] https://zenodo.org/record/5636853#.YYAnxp7MJPY

Design-Regeln auf die wahrgenommene Schwierigkeit haben. Zum anderen soll festgestellt werden, ob ein Zusammenhang zwischen der wahrgenommenen Schwierigkeit eine REST API zu verstehen und der gemessenen Performanz der Teilnehmer besteht. Hierbei wird sowohl die Gesamtheit der Regeln, als auch jede Regel individuell betrachtet.

RQ3 Auf welche Weise beeinflusst die Erfahrung mit REST die Ergebnisse von RQ1 und RQ2?

Auch diese Forschungsfrage wird explorativ untersucht. Die Idee dahinter ist es, zu beleuchten, ob und wie die bisherigen Erfahrungen der Teilnehmer mit REST die Performanz aus RQ1 und die wahrgenommene Verständlichkeit aus RQ2 beeinflussen. Zur Erfahrung gehört neben der Anzahl an Jahren auch die Art und Weise der bisherigen Arbeit mit REST. Beispielsweise könnte ein Teilnehmer in seinen bisherigen Berührungspunkten mit REST hauptsächlich APIs entwickelt haben, während ein weiterer Teilnehmer weitestgehend als Nutzer von APIs tätig war.

4.2 Teilnehmer

Die Teilnehmer für die Studie werden via E-Mail und über Bekanntmachung auf sozialen Medien zu der Studie eingeladen. Ziel ist es eine Mischung aus Studenten, Personen aus der Forschung und Personen aus der Industrie zu erreichen. So soll ein breites Spektrum der Population abgebildet werden. Studenten werden über einen E-Mail-Verteiler der Universität auf die Studie aufmerksam gemacht. Personen aus Forschung und Industrie werden über persönliche Kontakte und soziale Medien gewonnen. Am Ende der Studie wird außerdem der Link zur selbigen mit der Bitte um Weiterleitung an weitere mögliche Teilnehmer bereitgestellt. Um ein aussagekräftiges Ergebnis zu erhalten, soll insgesamt mindestens eine Teilnehmerzahl von 100 erreicht werden. Die Probanden werden für die Bearbeitung der Fragen in zwei Gruppen aufgeteilt. Die Aufteilung erfolgt randomisiert ohne weitere Kriterien oder Eigenschaften der Teilnehmer zu berücksichtigen.
Die Teilnehmer benötigen Grundkenntnisse mit REST APIs, um die Fragen beantworten zu können. Zudem werden Englischkenntnisse benötigt, da die Studie in dieser Sprache modelliert wurde. Weitere Voraussetzungen werden nicht gestellt. Als Motivation wird für die ersten 100 Teilnehmer jeweils 1€ an eine gemeinnützige Organisation gespendet. Zusätzlich wurde darauf geachtet den zeitlichen Rahmen der Studie zwischen 10 und 15 Minuten zu halten. Dies soll nicht nur den Anreiz steigern an der Studie teilzunehmen, sondern diese auch vollständig abzuschließen. Der zeitliche Rahmen wurde mittels einer Pilotstudie evaluiert.
Die Probanden werden auf der Willkommensseite der Studie darauf hingewiesen, welche Daten erhoben und zu welchem Zweck diese verwendet werden. Sämtliche Daten werden anonymisiert erfasst. Es gibt keine Möglichkeit Rückschlüsse auf die jeweilige Person zu ziehen. Des Weiteren wird den Teilnehmern mitgeteilt, dass sie die Studie ohne Folgen zu jederzeit abbrechen können. Die bis zu diesem Zeitpunkt gespeicherten Antworten werden in diesem Fall gelöscht. Jeder Proband muss zu dem Genannten seine Einwilligung erteilen, bevor er an der Studie teilnehmen kann.

4.3 Material

Das Experiment wird mittels einer Online-Umfrage durchgeführt. Zu diesem Zweck wird das Umfragewerkzeug LimeSurvey[2] verwendet. Die Instanz wird auf einem Server der Universität in Version 3.22.14 gehostet. LimeSurvey bietet die Möglichkeit, Umfragen zu designen, zu strukturieren und einer beliebigen Anzahl an Teilnehmern bereitzustellen. Dafür kann aus einer Vielzahl von Fragetypen ausgewählt werden. Zusätzlich erlaubt das Tool die automatische Messung der verbrachten Zeit bei einer Aufgabe, was für die Auswertung der Ergebnisse und Beantwortung der Forschungsfragen benötigt wird. Weitere nützliche Aspekte sind die Möglichkeit der randomisierten Zuteilung von Teilnehmern auf Gruppen und das anonymisierte Erfassen von Daten.
Die Umfrage ist in zwei Abschnitte aufgeteilt. Insgesamt werden jedem Teilnehmer 32 Seiten zur

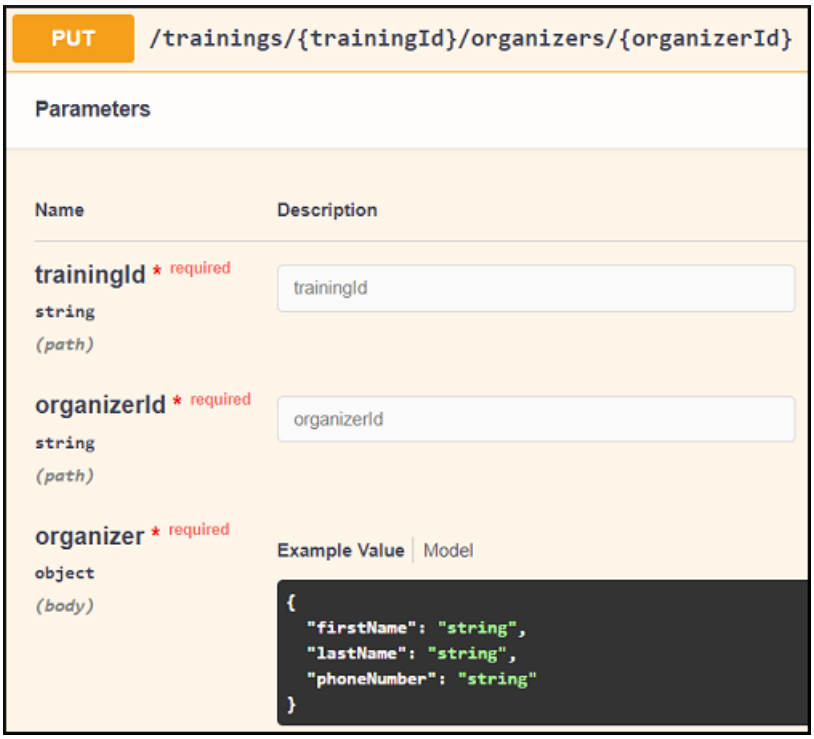

Abbildung 4.1: Beispiel eines REST API Schnipsels

Bearbeitung vorgelegt. Der erste Teil besteht aus Verständnis- und Bewertungsfragen. Hierfür werden jedem Teilnehmer 12 verschiedene REST API Schnipsel präsentiert. Wie in Abschnitt 2.1.5 erwähnt, wurden die Schnipsel mithilfe des Swagger-Editors erstellt. Abbildung 4.1 zeigt ein Beispiel eines solchen Schnipsels. Für die Bearbeitung der Fragen nicht benötigte Aspekte der Darstellung wurden herausgeschnitten. Die REST APIs wurden basierend auf Beispielen aus der realen Welt angefertigt. Hierfür wurde die Webseite API Guru[3] als Quelle verwendet. Die Beispiele wurden entsprechend der zu untersuchenden Design-Regel für REST APIs angepasst.
Die Schnipsel existieren in zwei Versionen. Für die erste Version werden die Regeln befolgt. Bei

[2] https://www.limesurvey.org/
[3] https://apis.guru/

Version zwei wird wiederum gegen die Regeln verstoßen. Die Schnipsel zur Untersuchung der Regeln für HTTP Status Codes stellen eine Ausnahme dar. Diese sind für beide Versionen identisch. Grund dafür ist es, dass der Status Code im Schnipsel nicht zu sehen ist, sondern dynamisch je nach Anfrage zustande kommt. Abhängig der untersuchten Regel werden den Teilnehmern bei diesen Aufgaben neben der API zusätzlich ein Request Body oder ein Request Header präsentiert. Außerdem wird die Antwort des Servers in Form des Status Codes und einer dazugehörigen Nachricht gezeigt.
Zu jedem Schnipsel wird zunächst eine Verständnisfrage gestellt. Anschließend bewerten die Teilnehmer die Schwierigkeit, das Schnipsel zu verstehen. Dies geschieht anhand einer ordinalen Skala mit fünf Punkten. Um die Bewertung zu erleichtern wird den Teilnehmern das Schnipsel ein weiteres Mal präsentiert. Dieses Muster wird für die 12 verschiedenen REST API Schnipsel beibehalten und füllt somit die ersten 24 Seiten. Die restlichen Seiten der Studie werden für das Sammeln der demografischen Daten verwendet.

4.4 Untersuchte Regeln

In diesem Abschnitt werden die in dieser Arbeit untersuchten Regeln dargelegt und beschrieben. Ein Überblick der Regeln ist in Tabelle 4.1 zu finden. In der genannten Tabelle werden zusätzlich Kürzel für die explizit untersuchten Regel definiert, die für die restliche Arbeit als Referenz dienen. Die Auswahl der Regeln erfolgte unter Berücksichtigung der in [32] erlangten Erkenntnisse. Eine Ausnahme stellt die Untersuchung der Regel Hierarchy dar, welche dem Buch von Ruby und Richardson [50] entnommen wurde. Bis auf die bereits erwähnte Ausnahme wurden Regeln ausgewählt, die von den Experten mit einer hohen Wichtigkeit eingestuft wurden und entweder die Benutzbarkeit oder die Wartbarkeit beeinflussen. Die Regel von Richardson und Ruby wurde in [32] nicht untersucht. Sie kann jedoch mit den beiden Regeln *Variable path segments may be substituted with identity-based values* und *Forward slash separator (/) must be used to indicate a hierarchical relationship* von Masse [36] in Verbindung gebracht werden. Beide genannten Regeln wurden in [32] ebenfalls mit einer hohen Wichtigkeit bewertet. Zusätzlich wirkt sich nach Einschätzung der befragten Experten die zuerst genannte Regel auf die Benutzbarkeit und die zweite Regel auf die Wartbarkeit aus.
Durch die Gegebenheiten der Regeln werden neben den explizit ausgewählten Regeln noch zwei weitere abgedeckt. Diese sind in Tabelle 4.1 als nicht explizit untersucht markiert. Für die Untersuchung der Verwendung von Funktionsnamen bei CRUD-Operationen (Regel CRUD) wird zusätzlich abgedeckt, dass die HTTP Methode DELETE zum Entfernen einer Ressource verwendet werden soll. Die zweite nicht explizit untersuchte Regel adressiert die Verwendung von Abfrageparametern zum Filtern. Dies wird bei der Regel GET zusätzlich abgedeckt. Die Gründe hierfür, eine Schilderung aller Regeln, sowie der Unterschied bei Nichteinhaltung der Regeln wird im Folgenden dargelegt. Die jeweiligen Beschreibungen sind mit Ausnahme der Hierarchy Regel dem Buch von Masse [36] entnommen. Die Erläuterung der Hierarchy Regel stammt aus dem Buch von Richardson und Ruby [50].

PluralNoun Masse [36] definiert diese Regel sowohl für Sammlungen (englisch: *Collections*), als auch für Speicher (englisch: *Stores*). Da davon ausgegangen wird, dass der Unterschied nicht jedem der Teilnehmer bekannt ist und entsprechend dieselben Auswirkungen für beide Regeln zu erwarten

Kürzel	Design-Regel	Kategorie	Explizit untersucht?
PluralNoun	A plural noun should be used for collection and store names	URI Design	Ja
Verb	A verb or verb phrase should be used for controller names	URI Design	Ja
CRUD	CRUD function names should not be used in URIs	URI Design	Ja
-	The query component of a URI may be used to filter collections or stores	URI Design	Nein
Hierarchy	Use path variables to encode hierarchy	Hierarchie Design	Ja
Tunnel	GET and POST must not be used to tunnel other request methods	Anfragemethoden	Ja
GET	GET must be used to retrieve a representation of a resource	Anfragemethoden	Ja
POST	POST must be used to create a new resource in a collection	Anfragemethoden	Ja
-	DELETE must be used to remove a resource from its parent	Anfragemethoden	Nein
RC200	200 (OK) must not be used to communicate errors in the response body	HTTP Status Code	Ja
RC401	401 (Unauthorized) must be used when there is a problem with the client's credentials	HTTP Status Code	Ja
RC415	415 (Unsupported Media Type) must be used when the media type of a request's payload cannot be processed	HTTP Status Code	Ja

Tabelle 4.1: In dieser Arbeit untersuchten Design-Regeln für REST APIs [50] [36]

sind, wurden diese Regeln zu einer kombiniert. Die Idee hinter der Regel ist es, für den Namen der Sammlung oder des Speichers in der URI die Mehrzahl zu verwenden. Daher wird dies für die Einhaltung der Regel erfüllt, wohingegen bei Nichteinhaltung der Singular verwendet wird.

Verb Diese Regel besagt, dass für Controller-Ressourcen ein Verb verwendet werden soll, um die jeweilige Aktion dahinter zu beschreiben. Dies ist angelehnt an Methoden eines Computerprogramms. Der Unterschied bei Nichteinhaltung ist hierbei die Verwendung eines Nomens anstatt eines Verbs.

CRUD Die Einhaltung dieser Regel legt nahe, in der URI nicht anzugeben, welche der vier CRUD Funktionen verwendet wird. Dies sollte dagegen über die semantisch korrekte HTTP Methode geschehen. In diesem Experiment wird daher für die Einhaltung der Regel das Löschen eines

Elements über die HTTP Methode DELETE modelliert, in der der zugehörige Funktionsname nicht in der URI auftaucht. Für die Missachtung der Regel wird dagegen der Funktionsname in der URI angegeben und mit GET die semantisch falsche HTTP Methode verwendet. Letzteres stellt den Grund für die Abdeckung der weiteren Regel dar.

Hierarchy Diese Regel adressiert die Verwendung von Pfadvariablen, um die Hierarchie in der URI zu verdeutlichen. Da hier für die Missachtung der Regel unterschiedliche Lösungsansätze denkbar sind, wird diese in drei verschiedenen Versionen untersucht. Die entsprechenden Kürzel sind Hierarchy1, Hierarchy2 und Hierarchy3, wobei die Zahl am Ende die jeweilige Version darstellt. Für Version 1 werden für den Regelverstoß Abfrageparameter anstatt von Pfadvariablen verwendet. Version 2 ist so strukturiert, dass die Hierarchie bei Nichtanwendung der Regel umgedreht ist. Anstatt von */companies/{companyId}/employees* wird daher */employees/companies/{companyId}* für die Variante ohne Regel angezeigt. Bei Version 3 befinden sich die Pfadvariablen für den Regelverstoß in Form von Identifiern im Request Body anstatt in der URI der API.

Tunnel Der Grundgedanke hier ist die Verwendung der korrekten HTTP Methode für den jeweiligen Anwendungsfall. Dies deckt sich mit den Anforderungen zum Erreichen von Level 2 des Richardson-Reifegradmodells. Die Regel besagt, dass weder GET noch POST dazu verwendet werden sollte, um die Funktionalität anderer HTTP Methoden, wie beispielsweise PUT, zu imitieren. Die für diese Regel modellierte API hat die Intention, eine Ressource zu updaten. Bei Regelanwendung wird daher die HTTP Methode PUT verwendet. Für den Regelverstoß wird dagegen POST genutzt. Um den Teilnehmern einen Hinweis auf die Semantik der API zu geben und somit eine korrekte Beantwortung zu ermöglichen, wird für die API mit Regelmissachtung zusätzlich ein weiterer Anfrageparameter *operation* mit dem Default-Wert *update* hinzugefügt.

GET Auch diese Regel beschäftigt sich mit der korrekten Nutzung von HTTP Methoden. In diesem Fall wird adressiert, dass für die Anfrage der Repräsentation einer Ressource GET verwendet werden soll. Für den Regelverstoß wird daher im Gegensatz zur Regeleinhaltung POST verwendet. Zusätzlich wird anstatt von Anfrageparametern ein Body mit den jeweiligen Parametern genutzt. Aufgrund dessen wird hier außerdem die Regel abgedeckt, die besagt, dass Abfrageparameter zur Filterung verwendet werden sollen.

POST Diese Regel besagt, dass POST zum Erstellen neuer Ressourcen innerhalb einer Sammlung verwendet werden soll. Sie besitzt somit dieselbe Intention des korrekten HTTP Verbs wie die beiden vorherigen Regeln. Außer der Verwendung von PUT anstatt von POST sind die für diese Regel modellierten APIs identisch.

RC200 Dies stellt die erste der drei untersuchten Regeln für HTTP Status Codes dar. Die Idee dahinter ist es zu vermeiden, im Fehlerfall den Status Code 200 zurückzugeben und das entsprechende Problem im Body der Antwort zu kommunizieren. Stattdessen wird zur Verwendung des zum Fehler passenden Status Codes geraten. Wie in Abschnitt 4.3 beschrieben wird für diese und die beiden folgenden Regeln sowohl bei Anwendung der Regel, als auch bei Regelverstoß dieselbe API verwendet. Der Unterschied zwischen den beiden besteht daher in der Antwort des

Servers. Für Regeleinhaltung wird der Status Code 400 verwendet wohingegen bei Regelverstoß 200 zurückgegeben wird. Zusätzlich wird für beide in der Antwort des Servers eine Nachricht angezeigt, mittels derer über den aufgetretenen Fehler informiert wird.

RC401 Diese Regel legt nahe, dass bei falscher oder fehlender Angabe der Zugangsdaten, der Status Code 401 zu verwenden ist. Dies sollte nicht mit Code 403 verwechselt werden. Der Unterschied liegt darin, dass für 403 zwar korrekte Zugangsdaten verwendet wurden, die entsprechenden Rechte für den Zugriff jedoch fehlen. Da dieser Unterschied unter Umständen nicht jedem der Teilnehmer bekannt ist, wird darauf verzichtet bei Nichtanwendung den Status Code 403 anzuzeigen. Stattdessen wird hier 400 verwendet.

RC415 Die Intention hinter dieser Regel ist es, dass der Code 415 zurückgegeben wird, wenn der Client einen Medientyp (englisch: *Media Type*) verwendet, der vom Server nicht akzeptiert wird. Für Regelverstoß wird, wie bereits für die vorherige Regel, der Status Code 400 verwendet.

4.5 Aufgaben

Die Aufgabe der Teilnehmer dieser Studie ist es REST API Schnipsel zu betrachten und zu verstehen. Zu diesem Zweck werden den Probanden Fragen zu den gezeigten Schnipseln gestellt. Die Fragen variieren je nach Art der API. Im Allgemeinen können die Fragen in die folgenden 3 Kategorien eingeteilt werden:

- **Rückgabewert:** Diese Kategorie wird vor allem für die APIs genutzt, bei welchen eine Repräsentation der Ressource angefragt wird (GET-Anfragen). Die Teilnehmer sollen nicht den exakten Rückgabewert bestimmten, da hierfür Informationen über den gespeicherten Datensatz notwendig sind. Stattdessen müssen die Probanden einerseits bestimmen, welche Art von Ressource erwartet wird. Andererseits gilt es zu erkennen, ob eine Liste aus Objekten oder ein einzelnes Objekt zurückgegeben wird.
- **Zweck:** Ziel dieser Kategorie ist es zu überprüfen, ob die Probanden die Intention hinter der gezeigten API erkennen. Dieser Fragetyp wird hauptsächlich für APIs verwendet, bei denen kein Rückgabewert in Form einer Ressource zu erwarten ist (POST-,PUT- und DELETE-Anfragen).
- **Grund für den angezeigten Status Code:** Diese Fragekategorie ist speziell für die Aufgaben, die sich mit der Überprüfung der HTTP Status Code Regeln beschäftigen. Bei der Untersuchung von Status Codes der 400er Serie (Regel RC401 und RC415) wird beispielsweise nach dem Grund für die fehlgeschlagene Anfrage gefragt. Für die Regel RC200 müssen die Probanden dagegen eine Antwort auf die Frage nach dem Ausgang der durchgeführten Anfrage geben.

Für die Beantwortung der oben beschriebenen Fragen stehen den Probanden fünf verschiedene Antwortmöglichkeiten zur Verfügung. Eine Mehrfachauswahl ist nicht möglich. Lediglich eine der fünf Fragen stellt die korrekte Antwort dar. Unabhängig davon, ob die Variante mit oder ohne Regel gezeigt wird, bleiben die Antwortmöglichkeiten gleich. Vorgeschriebene Antwortoptionen wurden aus mehreren Gründen gewählt. Einerseits ist eine Bewertung der Korrektheit über Freitextantworten wie in Abschnitt 2.2 beschrieben nicht ohne weitere Maßnahmen realisierbar. Andererseits können

Freitextantworten die benötigte Zeit beeinflussen, indem Probanden unterschiedlich lange Antworten geben oder die Schreibgeschwindigkeit dieser variiert. Durch die Verwendung von Einzelauswahl als Fragetyp wird diesen Problemen entgegengewirkt. Dadurch, dass die richtige Antwort im Vorfeld bereits festgelegt wurde, wird außerdem die spätere automatische Auswertung der Korrektheit vereinfacht.
Nach jeder beantwortenden Verständnisfrage werden die Probanden nach der Schwierigkeit das Schnipsel zu verstehen gefragt. Die wahrgenommene Schwierigkeit wird über eine fünf Punkte Skala bewertet. Die Skala bewegt sich dementsprechend in einem Bereich von sehr einfach (1) bis sehr schwer (5). Anschließend wird den Teilnehmern die nächste Verständnisfrage zur Bearbeitung vorgelegt.
Für jede der vier Kategorien von Design-Regeln, welche in Abschnitt 4.4 dargelegt wurden, werden drei Aufgaben aufgestellt. Jede Aufgabe untersucht jeweils eine eigene Design-Regel der Kategorie. Eine Ausnahme hiervon stellt die Untersuchung des Hierarchiedesigns dar. Hier wird für alle drei Aufgaben dieselbe Regel (Hierarchy) analysiert. Wie bereits in Abschnitt 4.4 beschrieben wurden zu diesem Zweck Schnipsel in drei verschiedene Version dieser Regel erstellt.

4.6 Treatments, Variablen und Hypothesen

Für dieses Experiment wurden REST API Schnipsel in zwei verschiedenen Versionen erstellt. Diese unterscheiden sich ausschließlich im Design der API. Für Version 1 (Regeleinhaltung, kurz RE) wurden die jeweiligen Design-Regeln für REST APIs berücksichtigt und angewandt. Bei Version 2 (Regelverstoß, kurz RV) dagegen wurden die Design-Regeln ignoriert oder ein Antipattern der Regel generiert. Die beiden beschriebenen Versionen stellen die in dieser Studie verwendeten Treatments dar. Auf eine Kontrollgruppe wird verzichtet, da zwischen Einhaltung und Nichteinhaltung der Design-Regeln keine weitere sinnvolle Abstufung existiert.
Zu den unabhängigen Variablen gehört die gerade dargelegte Version der REST APIs und sämtliche Parameter der Teilnehmer, die bei der Abfrage der demografischen Daten innerhalb der Umfrage gesammelt werden. Dazu zählt das Herkunftsland (L), die momentane primäre Rolle (R) in Bezug auf den Arbeitsbereich, die Perspektive (P) aus welcher mit REST APIs gearbeitet wird, die Jahre an Erfahrung mit REST (ER) und das bevorzugte Level für REST APIs in Bezug auf das Reifegradmodell von Richardson (BLR).
Die gemessenen und somit abhängigen Variablen sind die Zeit (t) zum Bearbeiten der Aufgaben und die Korrektheit (K) der Antworten zu den gestellten Fragen. Zu der gemessenen Zeit ist anzumerken, dass lediglich die verbrachte Zeit bei den Verständnisfragen berücksichtigt wird. Die Zeit zum Bewerten der Schwierigkeit und zur Beantwortung der demografischen Fragen wird bei der Auswertung nicht mit einbezogen.
Eine weitere abhängige Variable ist die Verständlichkeit (TAU). Diese berechnet sich durch eine Kombination aus der gemessenen Zeit und der Korrektheit. Diese Variable ist an die *Timed actual Understandability* von Scalabrino et al. [55] angelehnt. Basierend auf den hier beschriebenen Parametern berechnet sich TAU wie folgt:

$$TAU_{\mathrm{A}} = K_{\mathrm{A}} \times (1 - \frac{t_{\mathrm{A}}}{t_{\mathrm{max}}})$$

A steht dabei für eine gegebene Antwort auf eine Verständnisfrage. Die maximale Zeit t_{max} ist die von allen Teilnehmern am längsten benötigte Zeit, um dieselbe Aufgabe zu bearbeiten. Dabei wird die Zeit von beiden Treatments, RE und RV, berücksichtigt. K kann die Werte 0 (falsche Antwort)

oder 1 (richtige Antwort) annehmen. Dementsprechend ist der TAU-Wert für falsche Antworten 0. Für eine richtige Antwort nimmt TAU einen Wert zwischen 0 und 1 an. Je schneller eine korrekte Antwort im Vergleich zu den anderen Teilnehmern gegeben wurde, desto näher an 1 liegt der für diese Antwort berechnete TAU-Wert. TAU wird für die Beantwortung aller drei Forschungsfragen benötigt.
Die letzte abhängige Variable ist die wahrgenommene Schwierigkeit (WS). Diese wird über die Bewertungen der Schnipsel durch die Teilnehmer gesammelt. Sie wird hauptsächlich zur Beantwortung der zweiten (RQ2) und dritten (RQ3) Forschungsfrage verwendet. Dies sind alle in dieser Arbeit vorkommenden abhängigen oder unabhängigen Variablen. Im Folgenden werden die Hypothesen für die in Abschnitt 4.1 definierten Forschungsfragen dargelegt:

Hypothesen für RQ1 Um diese Forschungsfrage zu beantworten wird die abhängige Variable TAU verwendet. Die folgenden Hypothesen werden für RQ1 aufgestellt:

Nullhypothese	**Alternativhypothese**
H_{010}: Für die Regel PluralNoun ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{110}:** Für die Regel PluralNoun ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{011}: Für die Regel Verb ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{111}:** Für die Regel Verb ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{012}: Für die Regel CRUD ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{112}:** Für die Regel CRUD ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{013}: Für die erste Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{113}:** Für die erste Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{014}: Für die zweite Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{114}:** Für die zweite Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{015}: Für die dritte Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{115}:** Für die dritte Version der Hierarchy Regel ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{016}: Für die Regel Tunnel ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{116}:** Für die Regel Tunnel ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{017}: Für die Regel GET ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{117}:** Für die Regel GET ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV

H_{018}: Für die Regel POST ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{118}:** Für die Regel POST ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{019}: Für die Regel RC200 ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{119}:** Für die Regel RC200 ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{0110}: Für die Regel RC401 ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{1110}:** Für die Regel RC401 ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV
H_{0111}: Für die Regel RC415 ist das Verständnis (TAU) von REST APIs für Treatment RE gleich oder schlechter als für Treatment RV	**H_{1111}:** Für die Regel RC415 ist das Verständnis (TAU) von REST APIs für Treatment RE signifikant besser als für Treatment RV

Tabelle 4.2: Gegenüberstellung der Null- und Alternativhypothesen für RQ1

Jede Hypothese deckt eine der untersuchten Regeln ab. Für alle drei Versionen der Regel Hierarchy wird außerdem eine eigene Hypothese aufgestellt. Für den explorativen Teil dieser Forschungsfrage werden keine Hypothesen formuliert. Hier werden die einzelnen Regeln nicht in Bezug auf TAU untersucht. Stattdessen wird die Korrektheit und die benötigte Zeit für alle Regeln separat betrachtet.

Hypothesen für RQ2 und RQ3 Da RQ2 und RQ3 explorativ untersucht werden, werden für diese Forschungsfragen keine Hypothesen aufgestellt.
Für RQ2 wird die wahrgenommene Schwierigkeit und der TAU-Wert benötigt. Mit diesen beiden Werten wird eine Korrelationsanalyse durchgeführt, um eine Antwort auf die Forschungsfrage geben zu können. Dabei wird sowohl die Gesamtheit der Regeln, als auch jede Regel individuell untersucht.
Für die Beantwortung der dritten und letzten Forschungsfrage werden neben den TAU-Werten und der wahrgenommenen Schwierigkeit auch die unabhängigen Variablen der demografischen Datenabfrage berücksichtigt. Dadurch kann untersucht werden, ob die individuelle Erfahrung der Probanden mit REST einen Einfluss auf die Ergebnisse von RQ1 und RQ2 haben.

4.7 Forschungsdesign

Für diese Studie wird das sogenannte *within-subject* Design verwendet. Das bedeutet jeder Teilnehmer bearbeitet unabhängig der randomisiert zugeordneten Gruppe Aufgaben zu beiden Treatments. Die Hälfte der Aufgaben jedes Probanden besteht dementsprechend aus REST APIs, die die zugrunde liegende Regel befolgen. Die andere Hälfte besteht aus Aufgaben, die die Regeln ignorieren. Die beiden Gruppen unterscheiden sich darin, welche der untersuchten Regel in Treatment RE oder Treatment RV vorliegt. Beispielsweise bekommen Teilnehmer der ersten Gruppe die Regel CRUD in der Version mit Regeleinhaltung und die Probanden, welche der zweiten Gruppe zugeordnet wurden, die Version mit Regelverstoß. Mit der Wahl des *within-subject* Designs soll verhindert werden,

dass durch die randomisierte Zuteilung der Teilnehmer auf die Gruppen ein Ungleichgewicht in Bezug auf die Erfahrung mit REST APIs entsteht und so das Ergebnis beeinflusst wird. Im Falle eines *between-subject* Designs, also der Aufteilung der Probanden in zwei Gruppen, in der eine Gruppe nur APIs mit Anwendung der Regel und eine Gruppe nur Aufgaben mit Regelverstößen bearbeitet, müsste im Vorfeld der Studie sichergestellt werden, dass die durchschnittlichen Jahre an Erfahrung mit REST für beide Gruppen gleich verteilt ist. Anderenfalls könnte das Ergebnis nicht aussagekräftig sein oder verfälscht werden [10].
Die Reihenfolge, in der die Aufgaben angezeigt werden, ist nicht randomisiert, sondern festgeschrieben. Es wurde darauf geachtet, dass weder alle Aufgaben desselben Treatments, noch Fragen derselben Kategorie nacheinander auftreten. In den folgenden Abschnitten werden die Vorkehrungen beschrieben, die getroffen wurden, um die Validität der Studie zu gewährleisten. Entsprechend werden hier alle Bedrohungen adressiert, durch die keine Gefahr für die Validität des Experiments erwartet wird. Die Bedrohungen sind dem Buch von Wohlin et al. [64] entnommen. Sie werden daher in die vier Kategorien valide Schlussfolgerung, interne Validität, Konstruktvalidität und externe Validität eingeteilt.

4.7.1 Valide Schlussfolgerung

Diese Kategorie von Bedrohungen beschäftigt sich mit Problemen, die eine korrekte Schlussfolgerung zwischen dem verwendeten Treatment und dem Ergebnis der Studie verhindern können [64]. Da mehrere Hypothesen untersucht werden, wird das Signifikanzniveau entsprechend der Bonferroni-Korrektur [56] angepasst. Dies verringert die Wahrscheinlichkeit einen sogenannten *Typ I-Fehler* zu begehen. Um diesen Fehler handelt es sich, wenn fälschlicherweise eine Null-Hypothese verworfen wird, obwohl diese in Wahrheit korrekt ist [33]. Diese Maßnahme wird für die Bedrohung der expliziten Suche nach einem Ergebnis in den Daten und der Verwendung eines unangemessenen Signifikanzniveaus getroffen (vgl. *fishing and error rate* [64]). Weitere Maßnahmen in Bezug auf eine valide Auswertung der Daten und der Bedrohung durch eine geringe statistische Aussagekraft beziehungsweise der Verletzung von Annahmen der statistischen Tests (vgl. *Low statistical power* und *Violatet assumptions of statistical tests* [64]) werden im Abschnitt 4.9 dargelegt.
Die korrekte Messung der benötigten Zeit durch das Umfragewerkzeug wurde im Zuge einer durchgeführten Pilotstudie überprüft und für zuverlässig deklariert. Da die korrekte Antwort auf die in der Studie gestellten Fragen im Vorfeld festgelegt wurden, ist auch bei der Auswertung der Korrektheit keine Verletzung der Validität zu erwarten. Es werden keine subjektiven Messungen durchgeführt. Diese Maßnahmen wirken der Bedrohung einer unzuverlässigen Messung der Variablen (vgl. *Reliability of Measures* [64]) entgegen.
Die Gefahr einer unzuverlässigen Treatmentdurchführung (vgl. *Reliability of treatment implementation* [64]) wird außerdem durch die Verwendung des Umfragewerkzeugs Limesurvey gemindert. Dadurch ist sichergestellt, dass jeder Teilnehmer dieselbe Implementierung der Studie erhält. Zuletzt wird, wie im vorherigen Abschnitt beschrieben, ein *within-subject* Design verwendet. Die Bedrohung durch eine starke Heterogenität der Teilnehmer und die damit verbundene Gefahr, dass ein Unterschied bei den Ergebnissen der verwendeten Treatments durch diese entsteht (vgl. *Random heterogeneity of subjects* [64]), ist somit vernachlässigbar, da jeder Proband Aufgaben zu beiden Treatments bearbeitet. Dadurch wird der Einfluss der individuellen Charakteristiken auf das Gesamtergebnis verringert.

4.7.2 Interne Validität

Für diese Kategorie werden Bedrohungen analysiert, die, ohne das Wissen des Forschers, einen Einfluss auf die unabhängigen Variablen haben können [64]. Durch die kurze Dauer des Experiments von 10 bis 15 Minuten wird erwartet, dass keine Müdigkeit oder Langeweile bei den Probanden auftritt. Zusätzlich wird während der gesamten Umfrage ein Fortschrittsbalken eingeblendet, durch welchen die Teilnehmer jederzeit sehen können, an welcher Stelle der Umfrage sie sich befinden. Beim Erstellen der REST API Schnipsel wurde außerdem darauf geachtet, dass unterschiedliche Arten von Objekten verwendet werden. Das heißt während ein Schnipsel sich mit Universitäten und Studenten beschäftigt, wird bei einem anderen Schnipsel der Organisator eines Trainings bearbeitet. Durch diese Heterogenität der APIs und die kurze Dauer der Umfrage soll der Lerneffekt vermindert werden. Da ein auftretender Lerneffekt jedoch nicht auszuschließen ist, wurde als weitere Maßnahme die Reihenfolge der Aufgaben für beide Gruppen beibehalten. Dadurch sollte ein möglicher Lerneffekt bei beiden Gruppen für dieselben Aufgaben eintreten, was wiederum den Einfluss auf das Gesamtergebnis verringert. Zusätzlich ist die Reihenfolge der Antwortmöglichkeiten randomisiert. Alle genannten Maßnahmen wurden für die Gefahr eines möglichen Reifeprozesses der Teilnehmer während des Experiments (vgl. *Maturation* [64]) getroffen.
Da die Antworten der Teilnehmer und die benötigte Zeit nicht manuell durch eine Person erfasst werden, sondern durch ein Werkzeug, welches speziell für Umfragen entwickelt wurde, werden keine Fehler in Bezug auf die Erfassung der Antworten erwartet. Des Weiteren werden die Probanden randomisiert auf die beiden Gruppen verteilt. Es findet daher keine Aufteilung basierend auf vorherigen Studien oder bestimmten Charakteristiken der Teilnehmer statt. Dementsprechend wird nicht von einer Bedrohung durch die Gefahren der falschen Instrumentierung oder statistische Regression (vgl. *Instrumentation* und *Statistical regression* [64]) ausgegangen.
Die Probanden können durch die Verwendung eines Online-Umfragewerkzeugs das Experiment durchführen, ohne dabei von sozialen Interaktionen mit anderen Teilnehmern beeinflusst zu werden. Die Teilnehmer werden am Ende der Umfrage darauf hingewiesen, den Link zur Studie mit anderen potenziellen Interessenten zu teilen. Dadurch soll eine höhere Zahl an Teilnehmern generiert werden. Es kann nicht ausgeschlossen werden, dass dadurch weitere Informationen über die Studie an die dritte Person weitergegeben werden. Durch die Verwendung des *within-subject* Designs, was den Teilnehmern der Studie nicht mitgeteilt wird, würde sich eine Anpassung des Verhaltens der Probanden (vgl. *Diffusion or imitation of treatment* [64]) jedoch auf beide Treatments auswirken und den Effekt somit ausgleichen. Aufgrund dessen wird davon ausgegangen, dass eine Bedrohung durch die genannte Gefahr nicht zum Tragen kommt.

4.7.3 Konstruktvalidität

Die Bedrohungen der Konstruktvalidität richten sich hauptsächlich auf das Design und den Aufbau der Studie [64]. Für dieses Experiment werden zwei Treatments und 12 verschiedene REST API Schnipsel in jeweils zwei Versionen verwendet. Außerdem existiert für diese Studie, wie in Abschnitt 4.6 beschrieben, mehr als eine unabhängige Variable. Des Weiteren werden durch die Messung der Zeit, Korrektheit und wahrgenommenen Schwierigkeit mehrere Parameter für die Auswertung der Ergebnisse betrachtet. Aus den genannten Gründen kann eine Bedrohung durch die beiden Gefahren der Verwendung von zu wenigen Variablen oder Messungen (vgl. *Mono-operation bias* und *Mono-method bias* [64]) ausgeschlossen werden.
Die Gefahr der *Confounding constructs and levels of constructs* [64] wird durch die Verwendung

des *within-subject* Designs entgegengewirkt. Wie bereits erwähnt wird durch die Bearbeitung beider Treatments durch denselben Teilnehmer versucht, den Einfluss individueller Erfahrung mit REST APIs auf das Ergebnis der ersten beiden Forschungsfragen zu vermindern. In RQ3 wird zusätzlich untersucht, ob ein Unterschied in Bezug auf die Erfahrung zu erkennen ist.
Den Teilnehmern wird mitgeteilt, dass das Ziel der Studie ist, die Verständlichkeit unterschiedlicher REST API Designs zu analysieren. Durch die Angabe des Ziels soll verhindert werden, dass die Probanden versuchen herauszufinden, was die zugrunde liegende Hypothese hinter der Studie ist (vgl. *Hypothesis guessing* [64]). Fälle, in denen es trotz aller dem dazu kommt, dass Teilnehmer die Intention hinter der Studie zu erraten versuchen und ihr Verhalten auf eine positive oder negative Weise anpassen, werden wiederum durch das *within-subject* Design abgefangen. Das angepasste Verhalten würde sich dementsprechend auf die Ergebnisse beider Treatments auswirken.
Dadurch, dass das Experiment online stattfindet, führen die Teilnehmer die Studie an einem privaten Endgerät durch. Neben einem Rechner ist es außerdem möglich ein internetfähiges Smartphone oder Tablet zu verwenden. Eine Überwachung während der Teilnahme findet daher nicht statt. Zusätzlich werden alle Daten anonymisiert erfasst, was den Teilnehmern am Anfang der Studie mitgeteilt wird. Aufgrund dessen wird eine Verhaltensänderung der Probanden durch die Experimentsituation nicht erwartet (vgl. *Evaluation apprehension* [64]).
Zuletzt wird der Gefahr durch den Einfluss der Erwartungshaltung der Probanden (vgl. *Experimenter expectancies* [64]) entgegengewirkt, indem eine möglichst heterogene Menge an Teilnehmern erreicht werden soll. Wie bereits in 4.2 beschrieben sollen sowohl Studenten, als auch Personen aus der Forschung und Industrie für das Experiment gewonnen werden.

4.7.4 Externe Validität

Gefahren dieser Kategorie wirken sich negativ auf den Rückschluss der Ergebnisse auf die Praktiken der Industrie aus [64]. Der zuletzt beschriebene Punkt der heterogenen Teilnehmermenge im letzten Abschnitt kommt auch für die externe Validität zum Tragen. Da das Ziel der Studie die Untersuchung des Einflusses von Design-Regeln auf REST APIs ist, sollen möglichst viele unterschiedliche Teilnehmer aus Berufsbildern, die Berührungspunkte mit REST APIs haben, einbezogen werden. Daher werden für dieses Experiment neben Studenten, auch Personen aus der Industrie und Forschung, eingeladen. Dadurch soll ein valider Rückschluss auf die Population (vgl. *Interaction of selection and treatment* [64]) gewährleistet werden.
Wie in Abschnitt 1.1 erläutert ist REST ein weit verbreiteter Architekturstil und wird dementsprechend auch in der Industrie häufig verwendet. Für die Erstellung und Darstellung der REST API Schnipsel in diesem Experiment wurde außerdem der Swagger-Editor verwendet. Wie in Abschnitt 2.1.5 dargelegt, gehört Swagger neben RAML und API Blueprint zu den populärsten Werkzeugen für die Dokumentation von REST APIs. Die Hersteller geben auf ihrer Webseite[4] zudem an, dass Swagger von tausenden Teams, unter anderem auch Industriegrößen wie Microsoft[5], verwendet wird. Daher wird von einer angemessenen und zeitgemäßen Wahl für die Darstellung der REST APIs ausgegangen. Eine Bedrohung durch die Gefahr der Verwendung veralteter oder irrelevanter Praktiken in der Industrie (vgl. *Interaction of setting and treatment* [64]) wird dementsprechend nicht erwartet.
In diesem Experiment wird das Verständnis von REST APIs auf Basis gezeigter REST API Schnipsel

[4] https://swagger.io/
[5] https://www.microsoft.com/

untersucht. Die Probanden haben daher nicht die Möglichkeit, die APIs auszuprobieren und führen auch keine Änderungen an diesen durch. Je nach Beruf gehören diese und weitere Aspekte zur normalen Arbeitsweise mit REST APIs. Da für diese Tätigkeiten jedoch ein gewisses Maß an Verständnis notwendig ist und Entwickler nach Minelli et al. [37] rund 70 % ihrer Zeit mit dem Verstehen eines Programms verbringen, wird davon ausgegangen, dass sich die Ergebnisse dieses Experiments auf den Kontext der normalen Arbeitsweise übertragen lassen.

4.8 Durchführung des Experiments

Abbildung 4.2: REST API Schnipsel auf der Startseite der Umfrage

Die Teilnehmer gelangen über den in der Einladung bereitgestellten Link zunächst auf die Startseite der Umfrage. Auf dieser Seite wird den Teilnehmern erklärt, was das Ziel der Studie ist, wie viel Zeit diese in Anspruch nimmt und welche Voraussetzungen für die Teilnahme erforderlich sind. Anschließend folgt ein Abschnitt, indem die Aufgabenstellung beschrieben wird. Hier werden die Probanden außerdem darauf hingewiesen, dass für die Verständnisfragen eine Zeitmessung erfolgt, die korrekte Beantwortung der Fragen jedoch eine höhere Priorität besitzt.
Um den Teilnehmern den Aufbau der REST API Schnipsel zu verdeutlichen, befindet sich auf der Willkommensseite der Umfrage zusätzlich das auf Abbildung 4.2 zu sehende Beispiel. Die mithilfe eines roten Kastens markierten Bereiche werden unterhalb des Schnipsels erläutert.
Im letzten Abschnitt der Startseite befindet sich eine Beschreibung der Datenschutzrichtlinie. In dieser werden die Probanden darauf hingewiesen, welche Daten erhoben und zu welchem Zweck die Daten anonymisiert verwendet werden. Um fortzufahren müssen die Teilnehmer der Datenschutzrichtlinie zustimmen.
Wurde dies getan, wird die nächste Seite mit der ersten Aufgabe eingeblendet. Wie in Abschnitt 4.5 beschrieben folgt auf jede Verständnisfrage eine Seite, in der die Schwierigkeit bewertet wird.

Dieses Muster wird für alle zwölf zu bearbeitenden Aufgaben beibehalten. Nach Abschluss des Aufgabenteils wird mittels einer Informationsseite zu den demografischen Fragen übergeleitet. Sobald auch diese Fragen beantwortet wurden, folgt eine Seite, auf der sich ein Freitextfeld befindet. Dieses kann von den Teilnehmern dazu genutzt werden, weitere Anmerkungen oder Feedback zur Umfrage zu geben. Auf der letzten Seite der Umfrage wird den Probanden der Dank für die Teilnahme an der Studie ausgesprochen. Außerdem enthält die Abschlussseite den Link zur Umfrage mit der Bitte diesen an weitere potenziell Interessierte weiterzuleiten.

4.9 Durchführung der Analyse

Für die Analyse der Daten werden zunächst die Antworten der Teilnehmer exportiert. Limesurvey bietet zu diesem Zweck einen Export in Form einer CSV-Datei an. Um die Analyse zu vereinfachen werden folgende Anpassungen an der CSV-Datei vorgenommen:

- Der von Limesurvey angehängte String *[SQ001]* wird aus dem Spaltennamem für alle Verständnisfragen gelöscht.
- Die beiden Strings *(very easy)* und *(very hard)* werden aus den Antworten zu den Evaluationsfragen entfernt.
- Antworten in Bezug auf den Beruf oder die Perspektive, bei denen die Antwortoption *Andere* gewählt wurde, werden mit den vordefinierten Antwortmöglichkeiten harmonisiert, als neue Antwortmöglichkeit aufgenommen oder im Falle einer fehlenden Angabe im dazugehörigen Freitextfeld unverändert beibehalten.

Zusätzlich zu den genannten Änderungen werden die gegebenen Antworten auf Validität überprüft. Zunächst werden dazu die Angaben im Kommentarfeld überprüft. Beispielsweise gab ein Teilnehmer an, die Studie des Öfteren offen gehabt zu haben, ohne die Aufgaben aktiv zu bearbeiten. Da dadurch die gemessene Zeit nicht als valide angesehen werden kann, wurde diese Antwort aus dem Datensatz entfernt. Zur weiteren Prüfung der Validität wurden die gemessenen Zeiten für alle Verständnisfragen betrachtet und auf Basis folgender Kriterien bewertet:

- Ist die gemessene Zeit bei einer Aufgabe kürzer als fünf Sekunden oder länger als drei Minuten, wird diese Zeit als nicht valide angesehen. Eine Ausnahme stellen die drei Regeln für HTTP Status Codes dar. Für diese muss neben dem Schnipsel auch ein Request Body oder Request Header, sowie die Antwort des Servers betrachtet werden. Daher gelten für diese Regeln Zeiten als nicht valide, die kürzer als zehn Sekunden oder länger als vier Minuten sind.
- Sind drei oder mehr Zeiten eines Teilnehmers auf Basis der dargelegten Kriterien als nicht valide zu bewerten, wird die gesamte Antwort aus dem Datensatz entfernt. Für eine beziehungsweise zwei invalide Zeiten werden dagegen die einzelnen Antworten für diese Fragen, sowie die Bewertung der Schwierigkeit gelöscht.

Diese Filterung wird angewandt, da für die Berechnung des Verständnisses (TAU) relative Zeiten verwendet werden. Entsprechend beeinflussen invalide Zeiten das Ergebnis auf eine negative Weise. Zu kurze Zeiten werden entfernt, da diese Grund zur Annahme geben, dass der Teilnehmer die Fragen beantwortet hat, ohne die Aufgabenstellung durchzulesen. Zu lange Antworten geben dagegen einen Hinweis darauf, dass der Teilnehmer während der Durchführung der Studie abgelenkt

wurde oder, wie im bereits erwähnten Fall, die Studie offen hatte ohne diese aktiv zu bearbeiten. Weitere Manipulationen der CSV-Datei finden nicht statt.
Die CSV-Datei wird anschließend in ein vorher angelegtes Analyseskript, welches mit der Programmiersprache R[6] implementiert wurde, übergeben. Die Funktionalität des Skripts wurde mit den erhaltenen Antworten aus der Pilotstudie verifiziert. Im Folgenden werden die wichtigsten Berechnungen des Skripts genannt und beschrieben.
Nach dem Einlesen der CSV-Datei werden zunächst die demografischen Daten mittels beschreibender Statistik analysiert. Außerdem werden die Daten manipuliert, indem weitere Spalte hinzugefügt werden. Diese Spalten sagen mittels "1"(Wahr) beziehungsweise "0"(Falsch) aus, ob ein Proband REST APIs designt oder verwendet. Nach demselben Prinzip wird auch festgelegt, ob ein Proband aus dem Umfeld Student, Industrie oder Wissenschaft kommt, ob er das Richardson-Reifegradmodell kennt und ob er in Deutschland lebt. Dies ist für die spätere Beantwortung von RQ3 notwendig.
Anschließend werden die Antworten auf die Verständnisfragen ausgewertet. Für eine richtige Antwort wird eine 1 und für falsche Antworten eine 0 vergeben. Dies ist notwendig, um die Berechnung der TAU-Werte im nächsten Schritt zu ermöglichen. Basierend auf der Formel, die in Abschnitt 4.6 aufgestellt und beschrieben wurde, werden die TAU-Werte für jede Frage einzeln berechnet. Um zu entscheiden, welcher Hypothesentest verwendet werden kann, wird auf Normalverteilung der Daten überprüft. Zu diesem Zweck wird der Shapiro-Wilk-Test [57] für Normalverteilung angewendet. Für den Fall einer Normalverteilung (p-Wert > 0,05) würde zur Überprüfung der Hypothesen der t-Test [58] in Betracht gezogen. Im umgekehrten Fall, folglich keiner Normalverteilung, wird der Wilcoxon-Mann-Whitney-Test [38] angewendet. Das verwendete Signifikanzniveau wird aufgrund der Überprüfung von zwölf Hypothesen auf demselben Datensatz auf 0,00416 (0,05/12) über die Bonferroni-Korrektur [56] angepasst. Anschließend wird, im Falle eines signifikanten Ergebnisses, die Effektstärke mittels Cohens d [11] berechnet. Die Bewertung der Effektstärken erfolgt dabei unter Verwendung der von Sawilowsky [54] definierten Effektgrenzen, welche eine Anpassung der von Cohen [11] vorgeschlagenen Effektgrenzen darstellt. Durch diese Berechnungen können die für RQ1 aufgestellten Hypothesen überprüft werden. Für den explorativen Anteil von RQ1 werden die Korrektheit und die benötigte Zeit für jede Regel einzeln betrachtet. Auch für diese wird mittels Hypothesentests auf einen signifikanten Unterschied zwischen den beiden Treatments geprüft. Zur Visualisierung der Ergebnisse werden außerdem Boxplots für alle Regeln in beiden Treatments erstellt.
Zur Beantwortung der zweiten Forschungsfrage wird die wahrgenommene Schwierigkeit untersucht. Auch hier werden Boxplots und Balkendiagramme zum Zweck der Visualisierung erstellt. Anschließend werden, wie bereits für RQ1, Hypothesentests angewendet, um die Unterschiede zwischen den Treatments zu analysieren. Zusätzlich wird eine Korrelationsanalyse unter der Verwendung von Kendalls Tau [29] durchgeführt. Dies geschieht zum einen für die Gesamtheit aller Regeln und zum anderen individuell für jede Regel einzeln.
Für die letzte Forschungsfrage wird eine Korrelationsmatrix aufgestellt, in der sowohl die Ergebnisse von RQ1 und RQ2, als auch die demografischen Daten berücksichtigt werden. Dadurch sollen auffällige Datenpunkte gefunden werden, für welche weitere Analysen oder Berechnungen durchgeführt werden können. Da das hierfür verwendete Paket von R Kendalls Tau nicht unterstützt, wird Pearsons Korrelationskoeffizient [44] für die Korrelationsmatrix verwendet.

[6] https://www.r-project.org/

5 Ergebnisse

Dieses Kapitel beschreibt die Ergebnisse des Experiments. Dazu werden die demografischen Daten der Teilnehmer ausgewertet und berichtet. Anschließend wird anhand der Ergebnisse eine Antwort auf die drei Forschungsfragen gegeben.

5.1 Auswertung der Teilnehmer und demografischen Daten

Insgesamt haben 107 Probanden die Studie vollständig durchgeführt. Die kompletten Antworten von zwei Teilnehmern wurden aufgrund der in Abschnitt 4.9 beschriebenen Validitätsprüfung aus dem Datensatz entfernt. Zwölf weitere einzelne Antworten wurden zusätzlich durch selbige Prüfung für nicht valide deklariert und somit nicht für die Auswertung berücksichtigt. Entsprechend blieben 105 Antwortsätze für die Auswertung übrig, von denen wiederum 96 vollständig waren. Die neun restlichen Antwortsätze waren aufgrund der beschriebenen Entfernung einzelner Antworten nicht vollständig. Die Verteilung auf die beiden Gruppen ist mit 52 für Gruppe 1 und 53 für Gruppe 2 ausgeglichen. Die 105 Teilnehmer haben zwischen weniger als einem und 15 Jahren an Erfahrung mit REST APIs. Im Durchschnitt sind es 4,5 Jahre und der Median liegt bei 4 Jahren.
Abbildung 5.1 zeigt die Verteilung der Teilnehmer nach deren Beruf. Ein Teilnehmer gab keine Angabe über seinen Beruf und wurde hier somit nicht berücksichtigt. Wie anhand der Abbildung zu sehen ist, wurde das gewünschte heterogene Spektrum aus Studenten beziehungsweise Personen aus Forschung und Industrie erreicht. Mit 56 % ist der Anteil an Personen aus der Industrie am größten, gefolgt von Studenten mit 27 % und Forschung mit 17 %. Für den Bereich Forschung gaben 17 (94 %) Teilnehmer an, als Forscher zu arbeiten und ein weiterer Proband als Dozent tätig zu sein. Die weitere Verteilung der Berufsfelder für die Teilnehmer aus der Industrie ist in Abbildung 5.1 ebenfalls dargestellt. Zu *Andere* gehört jeweils ein Proband aus den Arbeitsbereichen Anforderungsingenieur, Softwaretester, Frontend-Webdesigner, Systemadministrator und Systemingenieur. Außerdem gaben je zwei weitere Teilnehmer den Beruf des Softwarearchitekten und Manager an. Für Studenten ist keine weitere Unterteilung gegeben.
In Bezug auf die Perspektive, von welcher aus mit REST APIs gearbeitet wird, gaben acht Teilnehmer an, REST APIs zu entwickeln beziehungsweise zu designen. 16 Probanden wählten die Perspektive des API-Nutzers oder Client-Entwicklers. Der größte Anteil liegt bei der Kombination aus den beiden genannten Perspektiven. Insgesamt 76 Teilnehmer entschieden sich für diese Option. Die übrigen fünf gaben keine Angabe über ihre Perspektive in der Arbeit mit REST APIs.
Mit 67 % kamen die meisten Antworten aus Deutschland, gefolgt von Portugal mit 11 % und USA mit 6 %. Weitere in dieser Studie vertretene Länder waren Schweiz (5 %), Spanien (3 %), Italien (2 %) und mit jeweils 1 % Niederlande, Dänemark, Österreich, Estland und die Antarktis.
Von 105 Teilnehmern gaben 28 (27 %) an, das Richardson-Reifegradmodell zu kennen. Die Frage nach dem Level, welches von REST APIs mindestens erreicht werden sollte, wurde daher ebenfalls von 28 Teilnehmern beantwortet. Von diesen 28 wählten 6 (21 %) Level 3 und 22 (79 %) Level 2

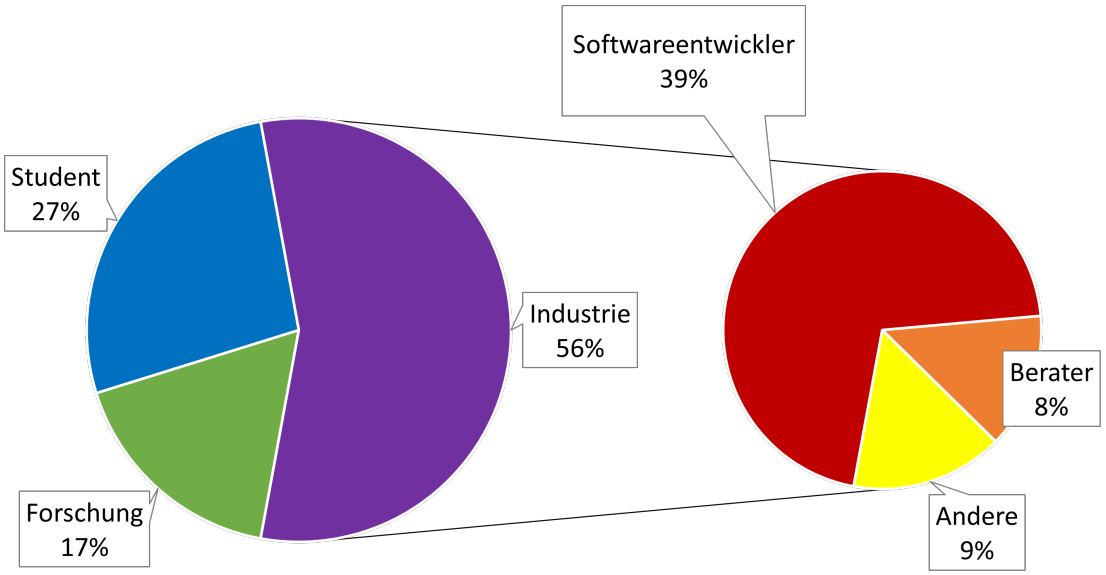

Abbildung 5.1: Übersicht über die Verteilung der Teilnehmer nach Beruf

als bevorzugtes Level. Die beiden Level 0 und 1 wurden von keinem der Teilnehmer als präferiertes Level angegeben. Im Durchschnitt wurde das zu erreichende Level mit 2,2 bewertet, während der Median bei 2 liegt.

5.2 Ergebnisse für RQ1

Für RQ1 wird untersucht, welche der untersuchten Regeln einen Einfluss auf die Verständlichkeit haben. Tabelle 5.1 zeigt das arithmetische Mittel für TAU und die benötigte Zeit, sowie die Anzahl korrekter Antworten für die einzelnen Regeln und Treatments.
Für elf der zwölf REST API Schnipsel ist unter Anwendung von Treatment RE eine durchschnittlich kürzere Zeit zur Beantwortung der Frage zu beobachten. Selbiges gilt auch für die Anzahl korrekter Antworten. Entsprechend sind für diese Regeln auch die TAU-Werte, in welchen Zeit und Korrektheit miteinander kombiniert sind, im Durchschnitt höher. Eine Ausnahme stellt die dritte Version der Hierarchy Regel dar. Hier fallen die Ergebnisse mit durchschnittlich 5 Sekunden weniger Zeit und 14 % mehr korrekter Antworten für das Treatment, in der die Regel nicht angewandt wurde, besser aus.
Weiterhin ist der Unterschied der TAU-Werte zwischen den Treatments für die Regel *CRUD* am größten. Zwar ist der Zeitunterschied mit knapp 13 Sekunden nicht extrem, die Anzahl korrekter Antworten geht dagegen stark auseinander. Während bei Anwendung der Regel 96 % der Teilnehmer auf die richtige Lösung gekommen sind, waren es für den gegenteiligen Fall lediglich 36 %. Aufgrund dessen entsteht für diese Regel eine große Diskrepanz zwischen den durchschnittlichen

TAU-Werten für die beiden Treatments.
Um einen besseren Überblick über die Verteilungen der TAU-Werte zu gewinnen, wurden Boxplots

Regel	TAU		Zeit in Sekunden		Korrekte Antworten	
	RE	RV	RE	RV	RE	RV
PluralNoun	0,8369	0,7046	20,82	32,23	52 (100%)	49 (94%)
Verb	0,7617	0,5345	33,07	45,22	50 (94%)	38 (73%)
CRUD	0,7392	0,1982	27,10	41,55	49 (96%)	20 (38%)
Hierarchy1	0,8457	0,5678	22,85	44,19	52 (98%)	40 (80%)
Hierarchy2	0,7916	0,4913	23,62	46,55	52 (100%)	44 (83%)
Hierarchy3	0,6141	0,7505	42,69	37,51	43 (83%)	50 (96%)
Tunnel	0,4820	0,3396	44,80	52,60	33 (66%)	28 (54%)
GET	0,8095	0,4337	28,40	45,29	53 (100%)	31 (60%)
POST	0,8427	0,4201	24,99	43,88	52 (98%)	29 (57%)
RC200	0,7268	0,3776	41,37	58,35	52 (98%)	29 (58%)
RC401	0,6058	0,5170	43,42	67,21	38 (75%)	37 (70%)
RC415	0,7654	0,7044	50,05	57,97	51 (96%)	49 (94%)

Tabelle 5.1: Gegenüberstellung von TAU, der benötigten Zeit und der Korrektheit für die einzelnen Regeln und Treatments

für die einzelnen Regeln und Treatments erstellt. Abbildung 5.2 zeigt die Verteilung der TAU-Werte für die drei Regeln der Kategorie URI Design. Auch hier wird deutlich, dass die Unterschiede zwischen den beiden Treatments für CRUD am größten sind. Während für Treatment RE die meisten Werte um 0,75 liegen, bewegt sich TAU für Treatment RV im Bereich 0,4 - 0. Der geringste Unterschied in dieser Kategorie ist für die Regel PluralNoun zu beobachten.
Die Boxplots für die 3 Versionen der Hierarchy Regel sind in Abbildung 5.3 zu finden. Für die ersten beiden Versionen ist zu beobachten, dass sich die Boxplots der beiden Treatments nicht überschneiden. Dies deutet auf ein signifikant besseres Ergebnis für Treatment RE hin. Für die dritte Version ist dagegen ein anderes Ergebnis zu sehen. Wie bereits erwähnt, stellt Hierarchy3 die einzige Regel dar, für die ein durchschnittlich besseres Ergebnis bei Regelverstoß erzielt wurde. Dies wird anhand der Verteilung verdeutlicht. Während die meisten Werte für Treatment RV um 0,75 liegen, wurde für Treatment RE ein breiteres Spektrum mit Werten bis 0,6 gemessen.
Für die Regeln GET und POST aus der Kategorie Anfragemethoden sind, wie in Abbildung 5.4 zu sehen, bei Regelverstoß Werte zwischen knapp 0,8 und 0,0 gemessen worden. Bei Regelanwendung bewegen sich die Werte für diese Regeln weitestgehend über 0,75. Auch die Mediane sind für Treatment RE deutlich höher als für Treatment RV. Letzteres gilt auch für die Regel Tunnel. Hier ist der Median mit 0,705 für RE im Vergleich fast doppelt so hoch wie für RV mit 0,371. Im Gegensatz zu den anderen beiden Regeln dieser Kategorie sind die TAU-Werte jedoch für beide Treatments breit gefächert.
Die letzte untersuchte Kategorie sind die HTTP Status Codes. Die Verteilung der Regeln dieser Kategorie sind in Abbildung 5.5 dargestellt. Für RC200 gleicht die Verteilung den Regeln GET und POST aus der vorherigen Kategorie. Die Verteilung von RC401 zeigt ein ähnliches Bild wie Tunnel,

wobei hier die Mediane der beiden Treatments näher beieinander liegen. Zuletzt bewegen sich die

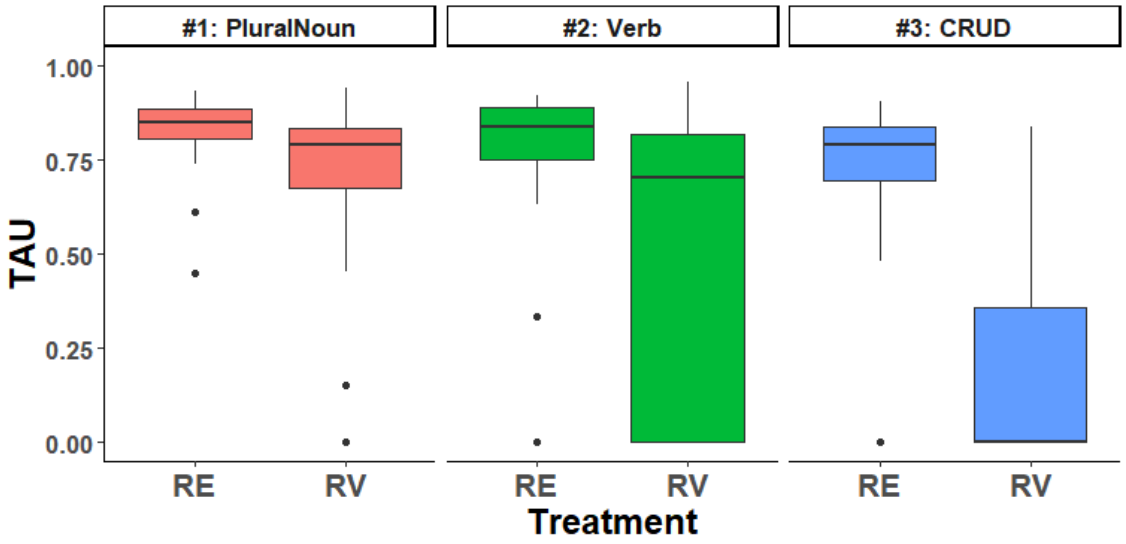

Abbildung 5.2: Boxplots der TAU-Werte für die Regeln der Kategorie URI Design. RE steht für Regeleinhaltung und RV für Regelverstoß.

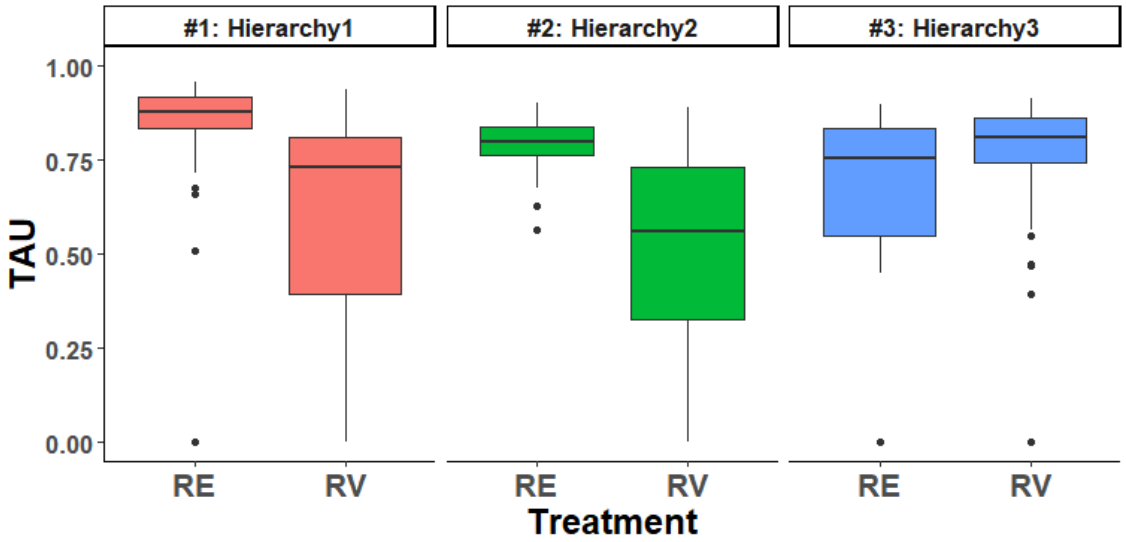

Abbildung 5.3: Boxplots der TAU-Werte für drei Versionen der Regel aus der Kategorie Hierarchy Design. RE steht für Regeleinhaltung und RV für Regelverstoß.

TAU Werte, mit ein paar Ausreißern, für RC415 im Durchschnitt knapp über 0,75. Dies gilt für beide Treatments, wobei der Median für RE mit 0,829 höher liegt als der Median von RV mit 0,774. Um die in Abschnitt 4.6 aufgestellten Hypothesen zu überprüfen, wird zunächst eine Prüfung auf Normalverteilung der Daten mittels des Shapiro-Wilk-Tests durchgeführt. Da für keine der Datensätze eine Normalverteilung vorliegt (p-Werte < 0.05), kann der t-Test nicht verwendet werden. Stattdessen wird der Wilcoxon-Mann-Whitney-Test (U-Test) herangezogen. Die Ergebnisse dieses

Tests sind in Tabelle 5.2 dargelegt. Da mehrere Hypothesen für denselben Datensatz aufgestellt wurden, wird eine Anpassung des Signifikanzniveaus nach der Bonferroni-Korrektur angestellt.

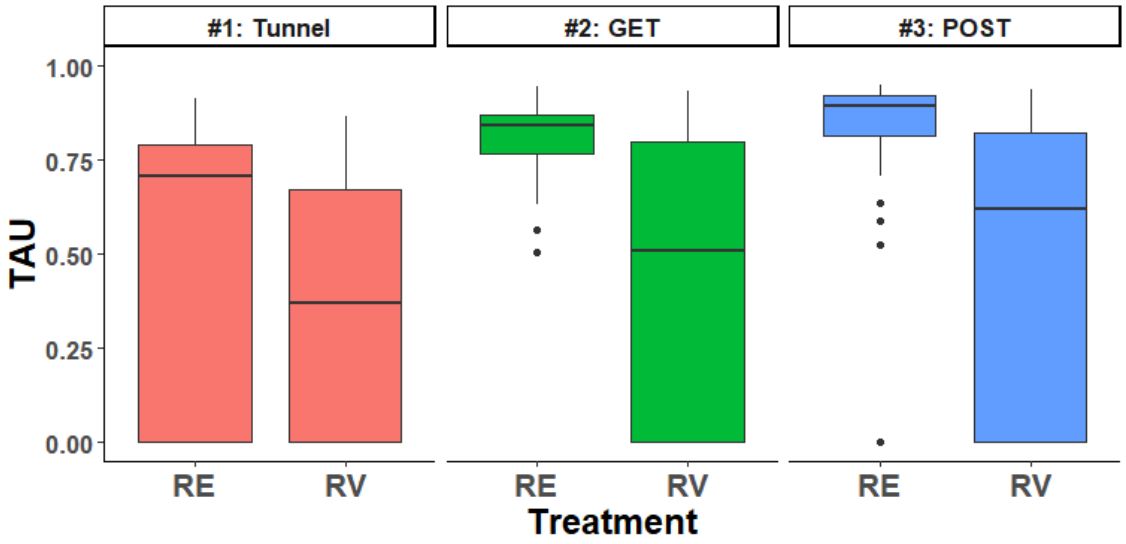

Abbildung 5.4: Boxplots der TAU-Werte für die Regeln der Kategorie Anfragemethoden. RE steht für Regeleinhaltung und RV für Regelverstoß.

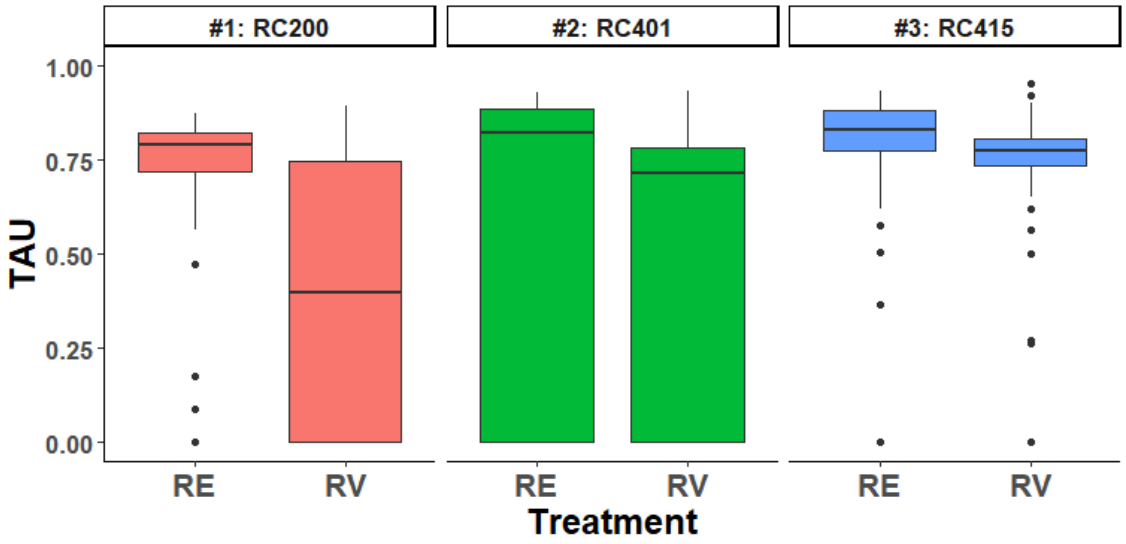

Abbildung 5.5: Boxplots der TAU-Werte für die Regeln der Kategorie HTTP Status Codes. RE steht für Regeleinhaltung und RV für Regelverstoß.

Das verwendete Signifikanzniveau beträgt daher $\alpha = 0{,}00416$. Für elf der zwölf Hypothesen wurde ein p-Wert unter dem definierten Signifikanzniveau erreicht. Diese Ergebnisse werden somit als statistisch signifikant angesehen. Für die Kategorien URI Design, Anfragemethoden und HTTP Status Codes ist das Ergebnis bei allen darin enthaltenen Regeln signifikant. In der Kategorie Hierarchy sind es zwei der drei Regeln (66 %).

Zusätzlich wurde die Effektstärke mittels Cohens d überprüft. Auch diese Ergebnisse sind in Tabelle 5.2 für die jeweilige Regel gelistet. Für Hierarchy3 wird Cohens d nicht berechnet, da hier kein statistisch signifikantes Ergebnis erzielt wurde. Nach Sawilowsky [54] können die Effektstärken wie folgt bewertet werden: Für die Regeln Tunnel, RC401 und RC415 liegt ein kleiner Effekt (<0,5) vor. Ein mittlerer Effekt (>0,5 und <0,8) ist für PluralNoun und Verb zu beobachten. Bei den übrigen Regeln ist der Effekt groß (>0,8) oder sehr groß (>1,2). Der stärkste Effekt mit 2,17 wurde für die Regel CRUD gemessen, was nach Sawilowsky [54] einen riesigen Effekt darstellt. Dies deckt sich mit der Diskrepanz in der Verteilung der TAU-Werte und der Anzahl korrekter Antworten.
Zusätzlich zu den U-Tests für die TAU-Werte wird der Unterschied zwischen den Treatments

Regel	**U-Test (p-Wert)**	**Cohens d**
CRUD	<0,001	2,17
Hierarchy2	<0,001	1,43
POST	<0,001	1,40
GET	<0,001	1,36
RC200	<0,001	1,25
Hierarchy1	<0,001	1,09
Verb	<0,001	0,75
PluralNoun	<0,001	0,72
Tunnel	0,003	0,40
RC415	<0,001	0,28
RC401	0,0039	0,24
Hierarchy3	0,996	-

Tabelle 5.2: Ergebnisse der Hypothesentests und die dazugehörigen Effektstärken für die zwölf Hypothesen

in Bezug auf die benötigte Zeit und die Korrektheit ebenfalls auf Signifikanz untersucht. Mit Ausnahme der benötigten Zeit für die zweite Version der Hierarchy Regel und dem Treatment RV (p-Wert = 0,052) konnte auch hier für keinen der Datensätze eine Normalverteilung mittels des Shapiro-Wilk-Tests nachgewiesen werden. Daher wird auch hier der U-Test verwendet. Da hierfür keine Hypothesen aufgestellt und getestet werden, wird von einem Signifikanzniveau α= 0,05 ausgegangen, wenn von einem signifikanten Ergebnis gesprochen wird.
In Bezug auf die benötigte Zeit ergibt der U-Test für Hierarchy3 einen p-Wert von 0,968. Wird die Hypothese umgedreht, also ein besseres Ergebnis für RV erwartet, liegt der p-Wert entsprechend bei 0,032. Außerdem liegen die p-Werte von CRUD (0,003), Tunnel (0,024), RC200 (0,001) und RC415 (0,002) knapp über 0,001, während für die restlichen Regeln ein Wert darunter festgestellt wurde.
Für die Korrektheit ist das Ergebnis des U-Tests im Vergleich zur benötigten Zeit für mehrere Regeln nicht signifikant. Neben Hierarchy3 (0,987, bei umgedrehter Hypothese 0,013) gehören auch Tunnel (0,107), RC401 (0,299) und RC415 (0,319) zu dieser Gruppe. PluralNoun (0,041), Verb (0,002), Hierarchy1 (0,002) und Hierarchy2 (0,001) stellen die Regeln dar, bei denen ein signifikantes Ergebnis zu beobachten ist und der p-Wert über 0,001 liegt. Für die übrigen Regeln ist der p-Wert kleiner als 0,001.

Antwort für RQ1: Mit Ausnahme von Hierarchy3 konnte für alle untersuchten Regeln ein signifikant besser Ergebnis in Bezug auf die Verständlichkeit für Treatment RE gemessen werden. Daher werden die jeweiligen Nullhypothesen für diese elf Regeln abgelehnt und die Alternativhypothesen angenommen. Für Hierarchy3 wird dagegen die Alternativhypothese für die Nullhypothese verworfen. Bei Betrachtung der Regeln in Bezug auf die benötigte Zeit wurde mit Ausnahme von Hierarchy3 ein signifikant besser Ergebnis für Treatment RE gemessen. Für die Korrektheit der Antworten war zusätzlich zu Hierarchy3 auch der Unterschied für die Regeln RC401, RC415 und Tunnel nicht signifikant.

5.3 Ergebnisse für RQ2

Für RQ2 wird die wahrgenommene Schwierigkeit der REST APIs explorativ untersucht. Abbildung 5.6 zeigt die Bewertungen der Probanden für die jeweilige Regel und die beiden Treatments. Dabei ist zu beobachten, dass die Schnipsel mit Anwendung der Regel im Durchschnitt mit einer geringen Schwierigkeit eingeschätzt wurden. Für Version 1 und 2 der Hierarchy Regel ist ein deutlicher Unterschied zu erkennen, wohingegen für Version 3 eine ähnliche Bewertung beobachtet werden kann. Im Vergleich zur gemessenen Verständlichkeit ist das Ergebnis für Treatment RE hier jedoch geringfügig besser. Für die Regeln RC415 und Verb ist der Unterschied in Bezug auf die Bewertung ebenfalls marginal.
Wird die Gesamtheit der Regeln betrachtet, liegt der Median für Treatment RE bei 2 und das arithmetische Mittel bei 1,762. Für Treatment RV ist der Median 3 und das arithmetische Mittel 2,867. Keiner der beiden Datensätze sind nach dem Shapiro-Wilk-Test normalverteilt. Der daher angewandte U-Test ergibt einen p-Wert kleiner als 0,001. Die Effektstärke nach Cohens d liegt bei 0,98, was nach Sawilowsky [54] einen großen Effekt darstellt. Mit Ausnahme der drei bereits erwähnten Regeln ergeben auch die U-Tests bei separater Betrachtung der Regeln einen p-Wert kleiner als 0,001. Für Verb liegt der p-Wert bei 0,023 und für RC415 bei 0,039, was bei einem typischen Signifikanzniveau von 0,05 statistisch signifikant wäre. Für Hierarchy3 liegt der p-Wert dagegen bei 0,405.
Zur Beantwortung der Forschungsfrage wurde außerdem untersucht, ob eine Korrelation zwischen der wahrgenommenen Schwierigkeit durch die Probanden und der gemessenen Performanz besteht. Dazu wird sowohl die Gesamtheit aller Regeln, als auch jede Regel individuell betrachtet. Tabelle 5.3 zeigt die Ergebnisse der Korrelationsanalyse unter Verwendung von Kendalls TAU für beide Treatments. Für den Großteil der Regeln ist eine schwach negative Wechselwirkung zwischen der Bewertung und der Performanz zu sehen. Lediglich für GET mit Treatment RV wurde eine mittelstarke negative Korrelation gemessen. Ein negativer Zusammenhang sagt in diesem Fall aus, dass je höher die Schwierigkeit bewertet wurde, desto niedriger ist der gemessene TAU Wert. Außerdem gilt: Je niedriger der TAU Wert, desto schlechter die Verständlichkeit.
Die Unterschiede zwischen den beiden Treatment sind mit zwei Ausnahmen ebenfalls gering. Für CRUD und Tunnel in der Version ohne die Anwendung der Regel wurde eine schwach positive Relation festgestellt, wobei der p-Wert für CRUD bei mehr als 0,05 liegt und der Zusammenhang somit statistisch nicht signifikant ist. Für die genannten Regeln wurde bei Regelanwendung eine negative Korrelation mit p-Werten <0,05 gemessen.
Die Korrelationsanalyse für die Gesamtheit der Regeln spiegelt die dargelegten Beobachtungen wider. Auch hier ist ein geringer Unterschied zwischen den Treatments zu sehen. Für beide liegt eine schwach negative Korrelation vor. Für die Gesamtheit der Regeln wurde zusätzlich die Korrelation

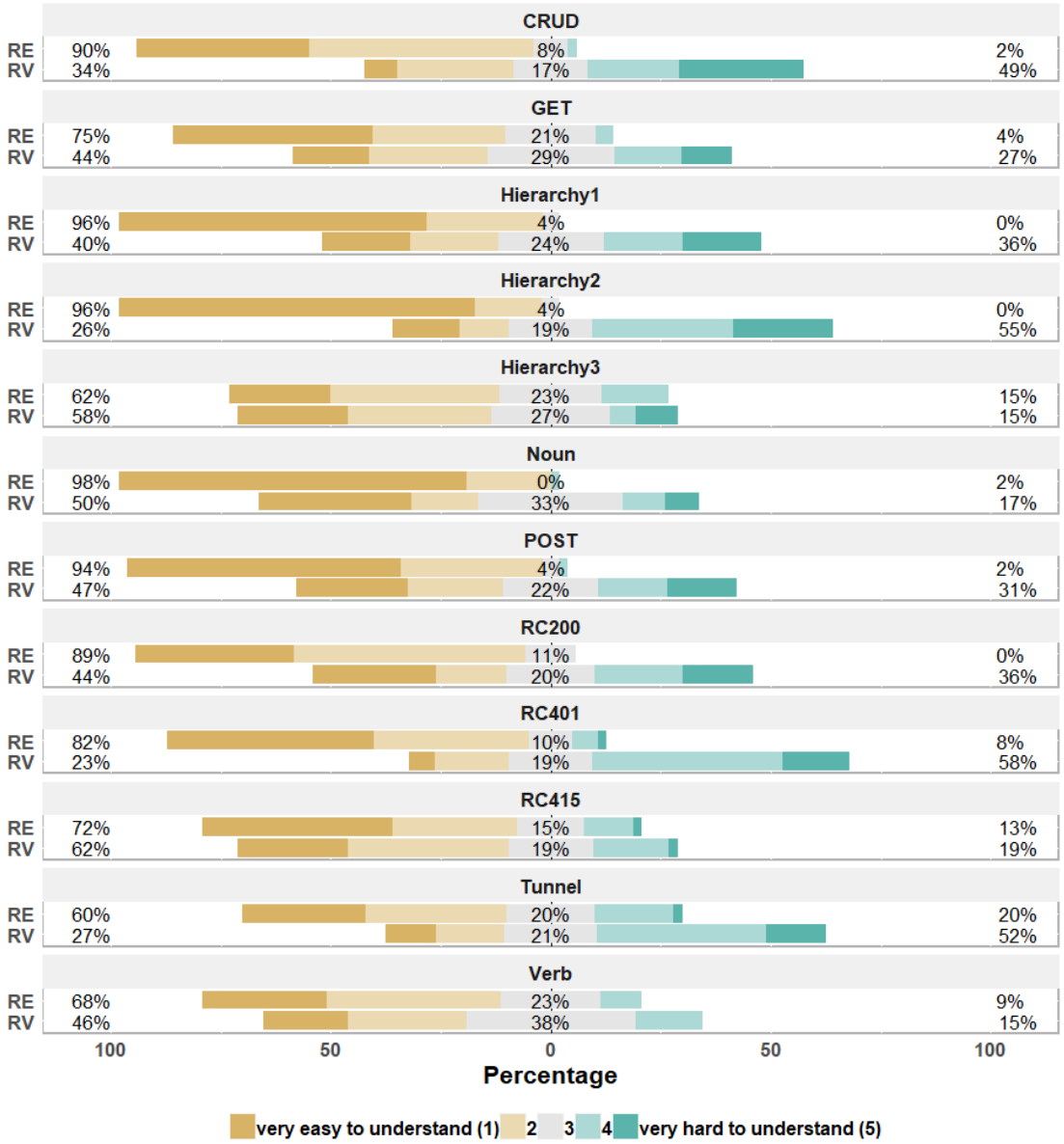

Abbildung 5.6: Vergleich der wahrgenommenen Schwierigkeit für die beiden Treatments aufgeteilt in die untersuchten Regeln

mit Messwiederholung (siehe *repeated measures correlation*[2]) überprüft. Das Ergebnis für RE liegt bei -0,350 und für RV bei -0,305. Für beide Treatments ist der p-Wert kleiner als 0,001. Auch hier ist kein eindeutiger Unterschied zwischen den beiden Treatments zu erkennen.

Regel	RE		RV	
	Kendalls TAU	p-Wert	Kendalls TAU	p-Wert
Alle	-0,287	<0,001	-0,228	<0,001
PluralNoun	-0,301	0,009	-0,256	0,014
Verb	-0,293	0,006	-0,354	0,001
CRUD	-0,324	0,004	0,212	0,065
Hierarchy1	-0,132	0,241	-0,057	0,594
Hierarchy2	-0,134	0,238	-0,258	0,014
Hierarchy3	-0,325	0,003	-0,344	0,001
Tunnel	-0,249	0,026	0,226	0,043
GET	-0,303	0,005	-0,508	<0,001
POST	-0,292	0,009	-0,063	0,570
200	-0,085	0,440	-0,155	0,167
401	-0,072	0,520	-0,252	0,020
415	-0,305	0,004	-0,095	0,369

Tabelle 5.3: Korrelationsanalyse zwischen wahrgenommener Schwierigkeit und gemessener Verständlichkeit (TAU). Signifikante Korrelationen (p-Wert < 0,05) sind farblich hervorgehoben.

Antwort für RQ2: Die Bewertungen der Schwierigkeit unterscheiden sich zwischen den beiden Treatments mit Ausnahme der Regel Hierarchy3 signifikant voneinander, wobei für RV höhere Bewertungen zu beobachten sind. In Bezug auf den Zusammenhang zwischen der wahrgenommenen Schwierigkeit und der Verständlichkeit konnten mit der Ausnahme von GET (Treatment RV) überwiegend schwach negative Korrelationen gemessen werden. Hervorzuheben ist eine positive Korrelation für Tunnel bei Regelverstoß. Der Unterschied zwischen den beiden Treatments ist ebenfalls gering.

5.4 Ergebnisse für RQ3

Für diese Forschungsfrage wurde der Einfluss von Erfahrungen mit REST APIs auf die unabhängigen Variablen von RQ1 und RQ2 untersucht. Zu diesem Zweck wurde für jeden Teilnehmer das arithmetische Mittel von TAU, der wahrgenommene Schwierigkeit, der benötigten Zeit und der Korrektheit aggregiert. Dies wurde jeweils für die sechs Aufgaben mit Treatment RE und Treatment RV durchgeführt. Anschließend wurden unter Verwendung von Kendalls TAU die Korrelationen zwischen den vier genannten Variablen in beiden Treatments und den demografischen Daten der Teilnehmer berechnet. Da für diese Forschungsfrage explorativ vorgegangen wird und keine Hypothesen untersucht werden, wird hier von einem signifikanten Ergebnis gesprochen, sofern das Signifikanzniveau von α= 0,05 unterschritten wird. Im Folgenden werden die Ergebnisse dieser Analyse beschrieben.

Tabelle 5.4 zeigt die signifikanten Korrelationen mit den Jahren an Erfahrung. Wie anhand dieser

Variable	RE		RV	
	Kendalls TAU	p-Wert	Kendalls TAU	p-Wert
TAU	0,196	0,005	0,136	0,052
t	-0,172	0,014	-0,010	0,882

Tabelle 5.4: Signifikante Korrelationen mit den Jahren an Erfahrung und das jeweilige Ergebnis für das zweite Treatment. TAU ist die gemessene Verständlichkeit und t die benötigte Zeit für die Beantwortung. Die signifikanten Korrelationen (p-Wert < 0,05) sind farblich hervorgehoben.

zu sehen, ist für Treatment RE sowohl für TAU, als auch die benötigte Zeit ein signifikantes Ergebnis gemessen worden. Die entsprechenden Korrelationen sind mit Werten <0,3 jedoch als schwach einzustufen. Für Treatment RV konnte für die benötigte Zeit kein signifikantes Ergebnis beobachtet werden. Selbiges gilt auch für TAU, wobei hier der p-Wert mit 0,052 nur minimal über dem definierten Signifikanzniveau liegt. In Bezug auf die Korrektheit und die wahrgenommene Schwierigkeit konnte weder für RE, noch für RV eine signifikante Korrelation mit den Jahren an Erfahrung festgestellt werden.
Neben den Jahren an Erfahrung beinhalten die demografischen Daten außerdem eine Auskunft

Variable	RE		RV	
	Kendalls TAU	p-Wert	Kendalls TAU	p-Wert
TAU	0,194	0,016	0,127	0,114
WS	-0,014	0,869	0,303	<0,001
K	0,229	0,016	0,150	0,086

Tabelle 5.5: Signifikante Korrelationen mit der Kenntnis des Richardson-Reifegradmodells und das jeweilige Ergebnis für das zweite Treatment. TAU ist die gemessene Verständlichkeit, WS die wahrgenommene Schwierigkeit und K die Korrektheit der Antworten. Die signifikanten Korrelationen (p-Wert < 0,05) sind farblich hervorgehoben.

darüber, ob ein Proband das Richardson-Reifegradmodell kennt oder nicht. Auch hierfür wurden, wie in Tabelle 5.5 zu sehen, signifikante Korrelationen gefunden. Bei Betrachtung von TAU und der Korrektheit der Antworten konnte jeweils ein schwach positiver Zusammenhang für Treatment RE festgestellt werden. Bei beiden genannten Variablen ist das Ergebnis für Treatment RV nicht signifikant. Für die wahrgenommene Schwierigkeit ist das Gegenteil der Fall. Hier wurde für RV mit einer positiven Korrelation von 0,303 und einem p-Wert kleiner als 0,001 ein signifikantes Ergebnis erzielt. Für Treatment RE trifft dies nicht zu.
Zuletzt wurde für die Perspektive des API-Designers und der wahrgenommenen Schwierigkeit in der Version RE eine signifikante negative Wechselwirkung (-0,169) bei einem p-Wert von 0,044 gemessen. Für Treatment RV ergibt selbige Kombination mit einem p-Wert von 0,610 keinen signifikanten Zusammenhang.
Für die Berufsgruppen Student, Industrie und Forschung, sowie für die Perspektive des API-Nutzers konnten keine signifikanten Korrelationen mit den vier unabhängigen Variablen festgestellt werden.

Da der Großteil der Antworten (67 %) aus Deutschland kam und sich die restlichen 33 % auf verschiedene Länder verteilen, wurde darauf verzichtet eine Analyse mittels Einteilung in die einzelnen Länder durchzuführen. Stattdessen wurden die Antworten in aus Deutschland bzw. nicht aus Deutschland gruppiert. Auch für diese Variable wurden keine signifikanten Wechselwirkungen mit den unabhängigen Variablen gemessen.

Antwort für RQ3: Für die Jahre an Erfahrung, das Wissen über das Richardson-Reifegradmodell und die Perspektive, aus welcher mit REST APIs gearbeitet wird, konnte ein Einfluss auf die verschiedenen unabhängigen Variablen von RQ1 und RQ2 beobachtet werden. Die Effektstärke der entsprechenden Korrelationen sind jedoch als schwach zu bewerten. Während bei Regelanwendung drei signifikante Korrelationen gefunden wurden, war es bei Regelverstoß lediglich eine. Das heißt, der positive Effekt der Erfahrung auf die Verständlichkeit ist bei Regelanwendung stärker als bei Regelverstoß. Zuletzt konnte für die unterschiedlichen Berufsgruppen und das Herkunftsland keine signifikante Wechselwirkung mit den unabhängigen Variablen gemessen werden.

6 Diskussion

Im ersten Abschnitt dieses Kapitels werden die im vorherigen Kapitel dargelegten Ergebnisse bewertet. Im zweiten Teil werden anschließend die Limitationen für das durchgeführte Experiment diskutiert.

6.1 Bewertung der Ergebnisse

In diesem Abschnitt werden die Ergebnisse aus dem vorherigen Kapitel aufgegriffen und interpretiert, sowie Auffälligkeiten diskutiert.

6.1.1 Bewertung der Ergebnisse zu RQ1

RQ1 stellt die zentrale Forschungsfrage dieser Arbeit dar. Die Intention dahinter war es herauszufinden, ob und vor allem welche Design-Regeln für REST APIs einen Einfluss auf die Verständlichkeit selbiger haben. Dabei wurde von einem positiven Effekt auf die Verständlichkeit ausgegangen, was anhand der aufgestellten Hypothesen für diese Forschungsfrage verdeutlicht wird. Die Ergebnisse zeigen, dass für elf der zwölf Aufgaben, die wiederum unterschiedliche Regeln abdecken, ein signifikant besseres Resultat bei Regeleinhaltung gemessen werden konnte. Die vorangestellte Hypothese wurde folglich für diese elf Regeln bestätigt. Dieses Ergebnis deckt sich mit den Aussagen der Industrieexperten aus [32] bezüglich der Wichtigkeit und des Einflusses auf die Qualitätsattribute Nutzbarkeit und Wartbarkeit der hier untersuchen Regeln.
Für die Regel CRUD war der gemessene Unterschied in Bezug auf TAU am größten. Bei einzelner Betrachtung der beiden Komponenten von TAU konnte für die benötigte Zeit, vor allem aber auch für die Korrektheit ein signifikanter Unterschied zwischen den beiden Treatments festgestellt werden. Daraus lässt sich schlussfolgern, dass die Verwendung des semantisch korrekten HTTP Verbs, anstatt des entsprechenden Namens der Methode in der URI, einen wichtigen Bestandteil darstellt, den Sinn hinter einer API zu verstehen.
Für die dritte Version der Hierarchy Regel konnte kein positiver Einfluss bei Anwendung der Regel gemessen werden. Hier waren die Ergebnisse für Treatment RV sowohl für die benötigte Zeit, als auch für die Korrektheit besser. Entsprechend sind auch die durchschnittlichen TAU-Werte im Vergleich zu Treatment RE höher. Die beiden Versionen RE und RV dieser Regel unterscheiden sich darin, dass bei Regelanwendung Pfadvariablen für die Identifier verwendet werden, wohingegen bei Regelverstoß die genannten IDs über den Body der Nachricht kommuniziert werden. Eine mögliche Erklärung für dieses Ergebnis könnte sein, dass die Länge der URI und eine damit möglicherweise einhergehende Komplexität, einen negativen Einfluss auf die Verständlichkeit der API hat. Auch wäre denkbar, dass die Zuordnung mittels IDs über Variablen des Objekts ein besseres Konzept darstellt, als die Verwendung von Pfadvariablen. Ohne weitere Untersuchung kann hierfür keine genauere Aussage getroffen werden.

Werden die Anzahl korrekter Antworten für das Treatment RE betrachtet, fällt auf, dass mit Ausnahme der Regel Tunnel über 75 % korrekte Antworten gegeben wurden. Weiterhin ist für diese Regel kein signifikanter Unterschied in Bezug auf die Korrektheit zwischen den beiden Treatments zu beobachten. Verschiedene Webseiten, wie beispielsweise [3], [26] und [14] legen nahe, dass es für Entwickler nicht immer klar ist, welches das korrekte HTTP Verb zum Anlegen von Ressourcen darstellt. Die Ergebnisse für die Regel Tunnel könnten daher auf die genannte Kontroverse zwischen POST und PUT zurückgeführt werden.
Zuletzt konnte für die beiden Regeln RC401 und RC415, welche die Rückgabe des semantisch korrekten HTTP Status Codes im Fehlerfall adressieren, kein signifikanter Unterschied der Korrektheit zwischen den beiden Treatment beobachtet werden. Die benötigten Zeiten für die genannten Regeln waren bei Regeleinhaltung jedoch signifikant kürzer als bei Regelverstoß. Das heißt, auch bei Nichtanwendung dieser Regeln wird die korrekte Beantwortung der Fragen nicht maßgeblich beeinflusst. Die Anwendung der Regeln führt jedoch dazu, dass schneller klar wird, welches Problem bei der jeweiligen Anfrage vorliegt.

6.1.2 Bewertung der Ergebnisse zu RQ2

Auch für die Bewertung der Schwierigkeit, ein REST API Schnipsel zu verstehen, konnte für elf der zwölf Aufgaben ein signifikant besseres Ergebnis für Treatment RE gemessen werden. Die einzige Ausnahme stellt, wie bereits für RQ1, die dritte Version der Hierarchy Regel dar. Auf Basis dieser Ergebnisse wird geschlussfolgert, dass die Anwendung der Design-Regeln nicht nur einen Einfluss auf die messbare Verständlichkeit, sondern auch auf die wahrgenommene Schwierigkeit hat.
In Bezug auf die Korrelationen zwischen TAU und der wahrgenommenen Schwierigkeit konnten für die Regeln überwiegend schwach negative Zusammenhänge gemessen werden. 17 der insgesamt 26 waren außerdem signifikant. Für die Regeln Hierarchy1 und RC200 konnte für keines der beiden Treatments eine signifikante Korrelation festgestellt werden. Dies legt nahe, dass diese, unabhängig des Treatments, schwer einzuschätzen waren.
Auch die Unterschiede zwischen den beiden Treatments waren mit Ausnahme von CRUD und Tunnel marginal. Für CRUD und Tunnel mit Treatment RV konnte in Kontrast zu RE eine schwach positive Korrelation beobachtet werden. Das Ergebnis für CRUD ist nicht signifikant und wird daher nicht weiter ausgeführt. Für Tunnel lässt sich die positive Korrelation ebenfalls mit der in RQ1 genannten Kontroverse für das korrekte HTTP Verb zum Anlegen von Ressourcen erklären. Probanden, die mit POST die Erstellung einer Ressource assoziieren, empfinden ihre Antwort als richtig und schätzen die entsprechende Schwierigkeit als leicht ein. Da in Wirklichkeit jedoch eine falsche Antwort gegeben wurde ist der TAU-Wert für diese Aufgabe entsprechend 0. Dadurch kommt eine positive Korrelation zwischen TAU und WS zustande. Für Treatment RE ist dieser Effekt nicht zu sehen, da die Anzahl korrekter Antworten höher ist als für RV.

6.1.3 Bewertung der Ergebnisse zu RQ3

Für die letzte Forschungsfrage wurde untersucht, ob die unterschiedlichen Erfahrungen der Probanden mit REST APIs einen Einfluss auf die Ergebnisse der vorherigen Forschungsfragen besitzen. Während die Anzahl der Jahre eine, wenn auch schwache, signifikante Korrelation mit TAU und der benötigten Zeit bei Regelanwendung hatte, konnte selbiges für RV nicht beobachtet werden. Hier gilt es jedoch anzumerken, dass mit einem p-Wert von 0,052 das Signifikanzniveau von α= 0,05 für Treatment

RV und der Korrelation mit TAU nur minimal überschritten wurde. Selbst bei einem signifikanten Ergebnis wäre die Stärke der Korrelation mit 0,136 als schwach einzustufen. Insgesamt lässt sich daraus schließen, dass die Jahre an Erfahrung einen positiven Effekt für das Verständnis bei Regelanwendung haben. Außerdem hat die Nichteinhaltung der Regeln einen negativen Effekt auf das Verständnis, unabhängig davon, wie lange ein Proband bereits mit REST APIs arbeitet.
Für die Kenntnis über das Richardson-Reifegradmodell und TAU konnte ein ähnliches Ergebnis wie für die Jahre an Erfahrung gemessen werden. Während hier weder für RE, noch für RV, eine signifikante Wechselwirkung mit der benötigten Zeit gefunden werden konnte, wurde für Treatment RE eine schwach positive Korrelation mit der Korrektheit beobachtet. Da selbiges für RV nicht zutrifft, kann daraus geschlossen werden, dass die Kenntnis über das Reifegradmodell und das damit potenziell einhergehende breitere Wissen über REST APIs, einen positiven Einfluss auf das Verständnis bei Regelanwendung hat, bei Regelverstoß dagegen eine vernachlässigbare Rolle spielt. Weiterhin wurde für dieses Attribut eine positive Korrelation mit der wahrgenommenen Schwierigkeit bei Treatment RV gefunden. Das heißt, die Probanden, die das Reifegradmodell kennen, schätzen die Schwierigkeit der API mit Nichteinhaltung der Regel tendenziell höher ein, als Probanden, denen das Reifegradmodell kein Begriff ist. Daraus kann geschlossen werden, dass das bereits erwähnte tiefere Wissen über REST APIs dazu führt, den erhöhten Schwierigkeitsgrad für RV besser zu erkennen.
Zuletzt wurde eine signifikante Korrelation zwischen der wahrgenommenen Schwierigkeit für Treatment RE und der Perspektive des API-Designers gefunden. Mit einem Wert von -0,169 ist diese als schwach zu beurteilen. Für Treatment RV konnte dies nicht beobachtet werden. Eine mögliche Begründung für diesen Effekt könnte sein, dass Designer von APIs sich häufiger mit Best Practices und Design-Regeln beschäftigen und die Schwierigkeit von Treatment RE daher geringer bewerten. Gegen diese Vermutung spricht, dass keine signifikante Korrelation zwischen dieser Perspektive und den Variablen TAU, der benötigten Zeit oder der Korrektheit für Treatment RE gefunden wurde. Auch hier sind weitere Untersuchungen notwendig, um eine Aussage treffen zu können.
Insgesamt konnte kein starker Einfluss der Erfahrung mit REST auf die Ergebnisse gemessen werden. Die Aufgaben wurden bewusst so entworfen, dass rudimentäre Erfahrung mit REST genügt, um am Experiment teilzunehmen. Dadurch wird eine höhere Zugänglichkeit gewährleistet. Gleichzeitig sinkt dadurch der allgemeine Schwierigkeitsgrad der Aufgaben. Dies könnte ein Grund dafür sein, dass kein größerer Einfluss der Erfahrung gemessen werden konnte.
Für Treatment RV wurde eine signifikante Korrelation mit der Kenntnis über das Reifegradmodell festgestellt. Für Treatment RE wurden dagegen mehrere signifikante Wechselwirkungen beobachtet. Die Erfahrung mit REST hatte also insgesamt einen größeren Einfluss auf RE als auf RV. Dies legt nahe, dass REST Erfahrung förderlich ist, um den Regeln folgende APIs zu verstehen. Für APIs mit Regelverstoß ist der positive Effekt durch die Erfahrung dagegen vergleichsweise schwach. Eine mögliche Begründung für diese Beobachtung könnte sein, dass APIs, bei denen die Regeln befolgt werden, in der Praxis häufiger anzutreffen sind. Diese These würde jedoch gegen die Beobachtungen der beschriebenen Arbeiten aus Abschnitt 3.1 sprechen. Der allgemeine Konsens dieser Arbeiten besagt, dass Best Practices und Prinzipien in der Praxis häufig nur teilweise beachtet werden. Daher ist es wahrscheinlicher, dass die für Treatment RV erstellten Schnipsel Extrembeispiele darstellen und in der Praxis Regelanwendung und Regelverstoß miteinander vermischt werden.
Unabhängig davon, ob eine Antwort aus Deutschland kam oder nicht, wurde kein signifikanter Einfluss auf die unabhängigen Variablen dieser Studie gefunden. Dies stellt ein Indiz dafür dar, dass sich die Ergebnisse auch auf andere Länder übertragen lassen. Da mit 67 % der Großteil der Antworten aus Deutschland kam und viele Länder spärlich oder gar nicht vertreten waren, wird jedoch auf eine Generalisierung zu anderen Ländern verzichtet.

6.1.4 Implikation der Ergebnisse

In Bezug auf die Verständlichkeit von REST APIs wurde für die Einhaltung der Regel ein signifikant besseres Ergebnis bei elf der zwölf Schnipsel erzielt als für den Verstoß gegen die Regeln. Da für die unterschiedlichen Berufsgruppen keine signifikanten Korrelationen in Bezug auf die Verständlichkeit gefunden werden konnten, wird davon ausgegangen, dass die Anwendung der Regeln den gleichen Effekt für all diese aufweist. Auch in Bezug auf die Perspektive, die Kenntnis über das Reifegradmodell und die Jahre an Erfahrung konnten hauptsächlich schwache wechselseitige Beziehungen mit der Verständlichkeit gemessen werden. All diese Beobachtungen implizieren, dass es, unabhängig der bisherigen Erfahrungen mit REST APIs, einen positiven Einfluss auf die Verständlichkeit hat, die in dieser Arbeit positiv evaluierten Design-Regeln für REST APIs zu beachten. Dies wird durch die durchschnittlich leichtere Bewertung der Schwierigkeit für Treatment RE aus den Beobachtungen der zweiten Forschungsfrage zusätzlich unterstützt. Daher wird empfohlen, bei der Erstellung von REST APIs Rücksicht auf die vorhandenen Design-Regeln zu nehmen und diese einzuhalten. Dadurch wird eine höhere Verständlichkeit gewährleistet, was sich wiederum positiv auf die Wartbarkeit und Nutzbarkeit der REST APIs auswirkt.

6.2 Limitationen

Im Folgenden wird auf die Limitationen für dieses Experiment eingegangen, die bei der Bewertung der Ergebnisse berücksichtigt werden müssen. In Abschnitt 4.7 wurde dargelegt, durch welche Maßnahmen die Validität des Experiments und der Ergebnisse bewahrt wird. Daher werden hier die Bedrohungen diskutiert, die nicht oder unzureichend adressiert wurden und entsprechend einen Einfluss auf die Ergebnisse haben können.

6.2.1 Durchführung der Studie als Online-Experiment

Die Studie für dieses Experiment wurde online über das Tool Limesurvey durchgeführt. Wie in Abschnitt 4.7 beschrieben bringt dies gewisse Vorteile mit sich. Es gibt jedoch auch Nachteile, die einen Einfluss auf die Validität haben. Die Probanden haben das Experiment von einem beliebigen Gerät ihrer Wahl mit Internetzugang durchgeführt. Dadurch war es nicht möglich sicherzustellen, dass für jeden Teilnehmer die gleichen äußeren Bedingungen gelten. Es konnte beispielsweise kein Einfluss auf die Geräuschkulisse während der Durchführung des Experiments genommen werden. Auch könnten die Probanden je nach Standort von Mitarbeitern, Familienmitgliedern oder anderen Personen/Geräten in ihrer Konzentration gestört worden sein (vgl. *Random irrelevancies in experimental setting* [64]).
Die Teilnahme an der Studie war anonym. Zusätzlich zum Online-Experiment führt dies dazu, dass es nicht möglich ist zu verhindern, dass Personen mehr als einmal an der Studie teilnehmen und somit mit ihrem eventuellen Vorwissen die Ergebnisse beeinflussen. Auch ist es dadurch nicht möglich falsche Angaben der Person zu verifizieren. Zu diesem Zweck wurden die Antworten vor der Auswertung der Ergebnisse auf Validität überprüft (siehe Abschnitt 4.9). Trotz dieser Maßnahme kann eine Mehrfachteilnahme aufgrund der Anonymität nicht ausgeschlossen werden.

6.2.2 Verzicht auf eine Kontrollgruppe

Wohlin et al. [64] merken an, dass die Verwendung einer Kontrollgruppe einem Großteil der Bedrohungen auf die interne Validität entgegenwirkt. Für dieses Experiment wurde jedoch auf eine Kontrollgruppe verzichtet. Grund hierfür ist das Fehlen einer weiteren sinnvollen Abstufung zwischen Einhaltung und Verstoß der Design-Regeln. Eine teilweise Einhaltung der Regeln als Kontrollgruppe wäre denkbar, ist jedoch aufgrund der Beschaffenheit der Regeln schwer umsetzbar. Eine andere Möglichkeit wäre für jede Aufgabe zwei oder mehr Regeln zu kombinieren und innerhalb der Aufgabe eine Regel zu befolgen und eine weitere zu missachten. Da für dieses Experiment nicht nur untersucht werden sollte, ob die Regeln einen Einfluss auf die Verständlichkeit haben, sondern zusätzlich für welche Regeln dies zutrifft, wurde versucht die Regeln möglichst getrennt voneinander zu betrachten. Das vorgeschlagene Design würde diese Analyse erschweren.

6.2.3 Stichprobe

Wie in Abschnitt 5.1 beschrieben bestand die Stichprobe sowohl aus Studenten (27 %), als auch aus Personen der Industrie (56 %) und Forschung (17 %). Die gewünschte Heterogenität wurde entsprechend erreicht. Daher wird davon ausgegangen, dass es sich um eine repräsentative Stichprobe handelt und Rückschlüsse auf die Gesamtheit der Entwickler valide sind. Dennoch gilt es hier anzumerken, dass mit knapp 70 % der Personen aus der Industrie der Großteil angab, als Softwareentwickler zu arbeiten. Den zweitgrößten Anteil stellen Personen mit dem Beruf des Beraters dar. Die weiteren Berufsfelder, wie beispielsweise Softwaretester oder Softwarearchitekt, waren spärlich oder gar nicht vertreten. Es wäre denkbar, dass die Ergebnisse unterschiedlich ausfallen, wenn mehr Personen aus diesen Berufen an der Studie teilgenommen hätten. Letzteres gilt auch für das Herkunftsland. Der größte Anteil (67 %) der Antworten kam aus Deutschland. Eine Veränderung der Ergebnisse durch eine größere Verteilung der Antworten auf mehrere Länder ist daher nicht auszuschließen. Aus diesem Grund wird, wie in Abschnitt 6.1.3 dargelegt, auf eine Generalisierung zu anderen Ländern verzichtet. Diese Aspekte sollten bei der Bewertung der Ergebnisse berücksichtigt werden.

6.2.4 TAU als Metrik für Verständnis

Für die Messung des Verständnisses von REST APIs wurde in diesem Experiment der von Scalabrino et al. [55] vorgeschlagene TAU zur Kombination der Korrektheit und benötigten Zeit gewählt. Diese Metrik wurde ursprünglich für die Messung des Verständnisses von Code aufgestellt. Die Verwendung der Metrik für dieses Experiment zeigt, dass diese auch für das Verständnis von APIs anwendbar ist. Da in dieser Arbeit für jedes Schnipsel eine einzelne Frage gestellt wurde, ergibt TAU für falsche Antworten entsprechend den Wert 0. Die Korrektheit der Antwort besitzt daher einen größeren Einfluss, als für Fälle, in denen mehrere Fragen für dasselbe Schnipsel gestellt werden. Aus diesem Grund sollte angemerkt werden, dass für Studien, bei denen einzelne Fragen zu Schnipseln gestellt werden, die Wahl einer anderen Metrik potenziell besser geeignet sein könnte. Dies gilt entsprechend auch für das in dieser Arbeit durchgeführte Experiment.
Weiterhin wurden bei der Berechnung von TAU für t_{max} die Zeiten beider Treatments berücksichtigt. Dies hat den Vorteil, dass Ausreißer die Ergebnisse beider Treatments gleichermaßen beeinflussen. Andererseits sind die TAU-Werte, vor allem für Fälle, in denen keine extremen Ausreißer vorhanden

sind, für die durchschnittlich schnellere Gruppe dadurch entsprechend höher. Daher wäre es auch denkbar gewesen für t_{max} nur die jeweiligen Zeiten der einzelnen Treatments zu verwenden. Scalabrino et al. [55] beschreiben t_{max} als die längste gemessene Zeit für ein Schnipsel. In dieser Arbeit existiert für jede untersuchte Regel ein Schnipsel in zwei Versionen. Entsprechend können entweder beide Versionen als dasselbe Schnipsel oder jede Version als ein eigenes Schnipsel angesehen werden. Je nach Interpretation wäre die Verwendung beider beschriebenen Varianten legitim. Zusammenfassend gilt bei der Bewertung der Ergebnisse zu berücksichtigen, welche Variante für t_{max} verwendet wurde und dass die Ergebnisse unter Verwendung der zweiten Variante leicht abweichen können.

6.2.5 REST API Schnipsel

Wie in Abschnitt 4.4 beschrieben, decken zwei Schnipsel mehr als eine Regel ab. Dazu gehört auch das Schnipsel für die Regel CRUD, für welche der größte Unterschied zwischen den beiden Treatments festgestellt wurde. Masse [36] legt bei der Beschreibung dieser Regel dar, dass die verwendete CRUD Methode über das semantisch korrekte HTTP Verb kommuniziert werden soll. Er gibt außerdem drei Beispiele für Antipattern dieser Regel an, für die ein semantisch falsches HTTP Verb verwendet wird. Daher wurde bei der Erstellung des Schnipsels für diese Arbeit von einem validen Antipattern ausgegangen. Es ist dennoch vorstellbar, dass der Effekt bei dieser Aufgabe geringer ausgefallen wäre, wenn nur eine der beiden Regeln im Schnipsel vorkommt. Letzteres gilt auch für das Schnipsel der Regel GET, bei dem ebenfalls zwei Regeln abgedeckt werden.
Weiterhin wurden für die Erstellung der Schnipsel Beispiele aus der realen Welt als Vorlage genommen und auf Basis der untersuchten Regeln angepasst. Aus diesem Grund wird davon ausgegangen, dass die Schnipsel einen validen Praxisbezug besitzen. Es sollte dennoch angemerkt werden, dass, wie in Abschnitt 6.1.3 diskutiert, die Ergebnisse für RQ3 Anlass dazu geben, dass die für den Regelverstoß erstellen Schnipsel Extrembeispiele darstellen könnten.

6.2.6 Darstellung der APIs

Für die Erstellung und Darstellung der REST API Schnipsel wurde Swagger verwendet. Wie bereits in Abschnitt 4.7 erläutert wird dabei von einer geeigneten und zeitgemäßen Wahl ausgegangen, da es sich bei Swagger um eine bekannte, in der Industrie gängige Art der Visualisierung von REST APIs handelt. Es ist jedoch denkbar, dass zumindest ein Teil der Probanden bisher keine Berührungspunkte mit Swagger hatten und sich daher zunächst an die Darstellung gewöhnen mussten. Um dem entgegenzuwirken wurde wie in Abschnitt 4.8 beschrieben auf der Startseite der Studie anhand eines Beispiels erläutert, an welcher Stelle im Schnipsel die einzelnen Komponenten der REST API zu finden sind. Trotz alledem sollte hier erwähnt werden, dass eine andere Art und Weise der Darstellung die Ergebnisse des Experiments beeinflussen könnte.

6.2.7 Auswahl der Regeln

Um auf bestehender Forschung aufzubauen, wurden für diese Studie bewusst Regeln ausgewählt, die nach Meinung der Experten aus [32] eine hohe Wichtigkeit besitzen. Entsprechend wurde auch von einem signifikanten Einfluss auf die Verständlichkeit ausgegangen. Auch wurden, um die

Zugänglichkeit zur Teilnahme zu erhöhen, keine Regeln ausgewählt, die sich mit den Aspekten von *HATEOAS* beschäftigen, da für diese unter Umständen ein tieferes Wissen über REST APIs benötigt wird. Außerdem wäre hierfür ein dynamisches Experiment, in denen die Probanden die APIs benutzen oder Änderungen durchführen besser geeignet. Zusätzlich wurde mit diesem Experiment lediglich ein Teil der über 80 Regeln von Masse [36] abgedeckt. Grund dafür ist es, dass für den zeitlichen Rahmen des Experiments 10-15 Minuten gewählt wurde, um eine höhere Motivation für die Teilnahme zu schaffen. Es ist daher denkbar, dass das Ergebnis unter Verwendung anderer Regeln unterschiedlich ausfällt. Die Auswahl der Regeln sollte entsprechend bei der Bewertung der hier vorgestellten Ergebnisse berücksichtigt werden.

7 Zusammenfassung und Ausblick

Der Architekturstil REST wird in der heutigen Zeit häufig eingesetzt, um Programmierschnittstellen zu modellieren. Da für REST keine offizielle Spezifikation existiert, wurden Best Practices und Design-Regeln in verschiedenen Literaturwerken vorgeschlagen. Die bisherigen Untersuchungen zu REST APIs zeigen, dass diese von Entwicklern lediglich zu Teilen eingehalten werden. Ob und inwiefern die Design-Regeln einen Einfluss auf die verschiedenen Qualitätsattribute haben, ist in der Literatur bisher spärlich zu finden. Das Fehlen einer empirischen Grundlage und die große Anzahl an Regeln und Best Practices erschweren den Entwicklern die Auswahl selbiger. Daher wurde in dieser Arbeit empirisch untersucht, ob Design-Regeln einen Einfluss auf die Verständlichkeit von REST APIs haben. Zu diesem Zweck wurde basierend auf den Meinungen der Industrieexperten aus [32] Design-Regeln ausgesucht, die nach deren Bewertung einen Einfluss auf die Nutzbarkeit oder Wartbarkeit besitzen. Beide Qualitätsattribute werden wiederum von Verständlichkeit beeinflusst. Für das durchgeführte Experiment wurden anschließend basierend auf den gewählten Regeln REST APIs in zwei Versionen entworfen. Bei der ersten Version wurde darauf geachtet, die Regeln einzuhalten. Für Version 2 wurde bewusst gegen die Regeln verstoßen. Das Experiment wurde online über das Werkzeug Limesurvey durchgeführt. Insgesamt haben 105 Probanden aus verschiedenen Berufsgruppen an der Studie teilgenommen und valide Antworten abgegeben.
Die Ergebnisse des Experiments zeigen, dass für elf der zwölf REST API Schnipsel mit unterschiedlichen Regeln ein signifikant besseres Ergebnis bei Einhaltung der Regeln in Bezug auf die Verständlichkeit gemessen werden konnte. Die Meinung der Industrieexperten aus [32] konnte entsprechend durch dieses Experiment bestätigt werden. Für die Verwendung des semantisch korrekten HTTP Verbs anstelle des Namens der entsprechenden Methode in der URI (Regel CRUD) wurde dabei der größte Unterschied beobachtet. Zusätzlich zur gemessenen Verständlichkeit wurde auch die Schwierigkeit eine REST API zu verstehen bei Einhaltung der Regeln für elf der zwölf Schnipsel leichter eingeschätzt. Weiterhin wurde in dieser Arbeit untersucht, ob die bisherigen Erfahrungen mit REST APIs einen Einfluss auf die Ergebnisse besitzen. Hierfür wurden Zusammenhänge mit den Jahren an Erfahrung, mit dem Wissen über das Richardson-Reifegradmodell und der Perspektive, aus welche mit REST APIs gearbeitet wird, gefunden. Diese Zusammenhänge waren hauptsächlich für die Version mit Regelanwendung zu beobachten. Da auch die gemessenen Effektstärken als schwach zu bewerten sind, wird davon ausgegangen, dass die Regeleinhaltung, unabhängig davon, ob eine Person Anfänger oder Experte auf diesem Gebiet ist, einen positiven Effekt auf die Verständlichkeit hat. Insgesamt kann auf Basis der dargelegten Ergebnisse eine Empfehlung für die Einhaltung der untersuchten Regeln abgegeben werden. Ausnahme hiervon ist die Regel Hierarchy3. Da für diese ein schlechteres Ergebnis bei Regeleinhaltung erzielt wurde, ist eine Anwendung der Hierarchy Regel in dieser Form nicht zu empfehlen.

Ausblick

Für zukünftige Arbeiten auf diesem Gebiet sollte untersucht werden, was der Grund für die schlechteren Ergebnisse bei Regeleinhaltung und der dritten Version der Hierarchy Regel ist. Mögliche Gründe wurden in Abschnitt 6.1.1 diskutiert.
Auch wäre es interessant zu untersuchen, wie die Ergebnisse für weitere Regeln aussehen. Hier wäre es denkbar zusätzlich Regeln, die mit mittlerer oder niedriger Wichtigkeit bewertet wurden, zu betrachten. Weiterhin könnte erwägt werden, Regeln, die die Aspekte von HATEOAS abdecken, zu untersuchen. Für letzteres sollte berücksichtigt werden, dass ein dynamisches Experiment, in welcher die Probanden die APIs benutzen oder Implementierungsaufgaben lösen, unter Umständen besser geeignet sein könnte. Auch sollte bei der Auswahl der Metrik bedacht werden, dass TAU für einzelne Fragen pro Aufgabe der Korrektheit einen großen Stellenwert zuschreibt. Daher sollten entweder mehr Fragen pro Aufgabe gestellt werden oder eine andere Metrik gewählt werden.
Zuletzt wäre es auch interessant neben der Verständlichkeit den Einfluss von Design-Regeln auf weitere Qualitätsattribute zu betrachten. So könnte ein breiteres Wissen über die Auswirkungen der Regeln gewonnen werden und damit verbunden den Entwicklern mehr Anlass dazu geben, die Design-Regeln zu berücksichtigten.

Literaturverzeichnis

[1] S. Allamaraju. *Restful web services cookbook: solutions for improving scalability and simplicity*. Ö'Reilly Media, Inc.", 2010 (zitiert auf S. 19).

[2] J. Z. Bakdash, L. R. Marusich. „Repeated measures correlation". In: *Frontiers in psychology* 8 (2017), S. 456 (zitiert auf S. 52).

[3] Bealdung. *HTTP PUT vs. POST in REST API | Baeldung*. https://www.baeldung.com/rest-http-put-vs-post (zitiert auf S. 58).

[4] G. Beniamini, S. Gingichashvili, A. K. Orbach, D. G. Feitelson. „Meaningful identifier names: the case of single-letter variables". In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE. 2017, S. 45–54 (zitiert auf S. 22).

[5] G. R. Bergersen, J. E. Hannay, D. I. Sjoberg, T. Dyba, A. Karahasanovic. „Inferring skill from tests of programming performance: Combining time and quality". In: *2011 international symposium on empirical software engineering and measurement*. IEEE. 2011, S. 305–314 (zitiert auf S. 22).

[6] T. Berners-Lee, R. Fielding, L. Masinter. *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. 2005. URL: http://rfc.net/rfc3986.html (zitiert auf S. 17).

[7] B. W. Boehm, J. R. Brown, M. Lipow. „Quantitative evaluation of software quality". In: *Proceedings of the 2nd international conference on Software engineering*. 1976, S. 592–605 (zitiert auf S. 22).

[8] J. Bogner, S. Wagner, A. Zimmermann. „Collecting service-based maintainability metrics from RESTful API descriptions: static analysis and threshold derivation". In: *European Conference on Software Architecture*. Springer. 2020, S. 215–227 (zitiert auf S. 27).

[9] F. Bülthoff, M. Maleshkova. „RESTful or RESTless–current state of today's top web APIs". In: *European Semantic Web Conference*. Springer. 2014, S. 64–74 (zitiert auf S. 25).

[10] G. Charness, U. Gneezy, M. A. Kuhn. „Experimental methods: Between-subject and within-subject design". In: *Journal of economic behavior & organization* 81.1 (2012), S. 1–8 (zitiert auf S. 39).

[11] J. Cohen. „Statistical power analysis for the behavioral sciences New York". In: *NY: Academic* (1988), S. 54 (zitiert auf S. 44).

[12] B. Costa, P. F. Pires, F. C. Delicato, P. Merson. „Evaluating a Representational State Transfer (REST) architecture: What is the impact of REST in my architecture?" In: *2014 IEEE/IFIP Conference on Software Architecture*. IEEE. 2014, S. 105–114 (zitiert auf S. 27).

[13] B. De. „API documentation". In: *API Management*. Springer, 2017, S. 59–80 (zitiert auf S. 20, 21).

[14] *Difference between PUT and POST in REST APIs*. https://restfulapi.net/rest-put-vs-post/ (zitiert auf S. 58).

[15] D. G. Feitelson. „Considerations and Pitfalls in Controlled Experiments on Code Comprehension“. In: *arXiv preprint arXiv:2103.08769* (2021) (zitiert auf S. 22, 29).

[16] J. L. Fernandes, I. C. Lopes, J. J. Rodrigues, S. Ullah. „Performance evaluation of RESTful web services and AMQP protocol“. In: *2013 fifth international conference on ubiquitous and future networks (ICUFN)*. IEEE. 2013, S. 810–815 (zitiert auf S. 26).

[17] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000 (zitiert auf S. 11, 15, 16).

[18] T. Fredrich. „Restful service best practices“. In: *Recommendations for Creating Web Services* (2012) (zitiert auf S. 19).

[19] P. Giessler, M. Gebhart, D. Sarancin, R. Steinegger, S. Abeck. „Best practices for the design of restful web services“. In: *International Conferences of Software Advances (ICSEA)*. 2015, S. 392–397 (zitiert auf S. 19).

[20] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, S. Reddy. *HTTP: the definitive guide*. Ö'Reilly Media, Inc.", 2002 (zitiert auf S. 18).

[21] F. Haupt, F. Leymann, A. Scherer, K. Vukojevic-Haupt. „A framework for the structural analysis of REST APIs“. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, S. 55–58 (zitiert auf S. 26).

[22] X. J. Hong, H. S. Yang, Y. H. Kim. „Performance analysis of RESTful API and RabbitMQ for microservice web application“. In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE. 2018, S. 257–259 (zitiert auf S. 27).

[23] L. L. Iacono, H. V. Nguyen. „Authentication scheme for REST“. In: *International Conference on Future Network Systems and Security*. Springer. 2015, S. 113–128 (zitiert auf S. 18).

[24] ISO/IEC. *ISO/IEC 25010-2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO/IEC, 2011 (zitiert auf S. 21, 22).

[25] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001 (zitiert auf S. 21).

[26] P. Javin. *When to use PUT or POST in a RESTful Web Service? Answer | Java67*. https://www.java67.com/2016/09/when-to-use-put-or-post-in-restful-web-services.html (zitiert auf S. 58).

[27] A. Jedlitschka, M. Ciolkowski, D. Pfahl. „Reporting experiments in software engineering“. In: *Guide to advanced empirical software engineering*. Springer, 2008, S. 201–228 (zitiert auf S. 29).

[28] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere. „The architecture tradeoff analysis method“. In: *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 98EX193)*. IEEE. 1998, S. 68–78 (zitiert auf S. 27).

[29] M. G. Kendall. „Rank correlation methods.“ In: (1948) (zitiert auf S. 44).

[30] M. W. Khan, E. Abbasi. „Differentiating Parameters for Selecting Simple Object Access Protocol (SOAP) vs. Representational State Transfer (REST) Based Architecture“. In: *Journal of Advances in Computer Networks* 3.1 (2015), S. 63–6 (zitiert auf S. 27).

[31] A. Koschel, M. Blankschyn, K. Schulze, D. Schöner, I. Astrova, I. Astrov. „RESTfulness of APIs in the Wild“. In: *2019 IEEE World Congress on Services (SERVICES)*. Bd. 2642. IEEE. 2019, S. 382–383 (zitiert auf S. 26).

[32] S. Kotstein, J. Bogner. „Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts“. In: *arXiv preprint arXiv:2108.00033* (2021) (zitiert auf S. 12, 27, 28, 32, 57, 62, 65).

[33] E. LANN. *Testing statistical hypotheses*. Wiley, New York, 1959 (zitiert auf S. 39).

[34] L. Li, W. Chou. „Design and describe REST API without violating REST: A Petri net based approach“. In: *2011 IEEE International Conference on Web Services*. IEEE. 2011, S. 508–515 (zitiert auf S. 19).

[35] M. Maleshkova, C. Pedrinaci, J. Domingue. „Investigating web apis on the world wide web“. In: *2010 eighth ieee european conference on web services*. IEEE. 2010, S. 107–114 (zitiert auf S. 25).

[36] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Ö'Reilly Media, Inc.", 2011 (zitiert auf S. 11, 16, 18, 19, 27, 32, 33, 62, 63).

[37] R. Minelli, A. Mocci, M. Lanza. „I know what you did last summer-an investigation of how developers spend their time“. In: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, S. 25–35 (zitiert auf S. 42).

[38] M. Neuhäuser. „Wilcoxon–Mann–Whitney Test“. In: *International Encyclopedia of Statistical Science*. Hrsg. von M. Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 1656–1658. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_615 (zitiert auf S. 44).

[39] A. Neumann, N. Laranjeiro, J. Bernardino. „An analysis of public REST web service APIs“. In: *IEEE Transactions on Services Computing* (2018) (zitiert auf S. 26).

[40] F. Palma, J. Dubois, N. Moha, Y.-G. Guéhéneuc. „Detection of REST patterns and antipatterns: a heuristics-based approach“. In: *International Conference on Service-Oriented Computing*. Springer. 2014, S. 230–244 (zitiert auf S. 25).

[41] F. Palma, J. Gonzalez-Huerta, M. Founi, N. Moha, G. Tremblay, Y.-G. Guéhéneuc. „Semantic analysis of restful apis for the detection of linguistic patterns and antipatterns“. In: *International Journal of Cooperative Information Systems* 26.02 (2017), S. 1742001 (zitiert auf S. 25).

[42] F. Palma, J. Gonzalez-Huerta, N. Moha, Y.-G. Guéhéneuc, G. Tremblay. „Are restful apis well-designed? detection of their linguistic (anti) patterns“. In: *International Conference on Service-Oriented Computing*. Springer. 2015, S. 171–187 (zitiert auf S. 25).

[43] S. Patni. *Pro RESTful APIs*. Springer, 2017 (zitiert auf S. 16, 17, 19).

[44] K. Pearson. „VII. Mathematical contributions to the theory of evolution.—III. Regression, heredity, and panmixia“. In: *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 187 (1896), S. 253–318 (zitiert auf S. 44).

[45] F. Petrillo, P. Merle, N. Moha, Y.-G. Guéhéneuc. „Are REST APIs for cloud computing well-designed? An exploratory study“. In: *International Conference on Service-Oriented Computing*. Springer. 2016, S. 157–170 (zitiert auf S. 26).

[46] V. Rajlich, G. S. Cowan. „Towards standard for experiments in program comprehension“. In: *Proceedings Fifth International Workshop on Program Comprehension. IWPC'97*. IEEE. 1997, S. 160–161 (zitiert auf S. 22).

[47] D. Rathod. „Performance evaluation of restful web services and soap/wsdl web services“. In: *International Journal of Advanced Research in Computer Science* 8.7 (2017), S. 415–420 (zitiert auf S. 27).

[48] D. Renzel, P. Schlebusch, R. Klamma. „Today's top "RESTful" services and why they are not RESTful“. In: *International Conference on Web Information Systems Engineering*. Springer. 2012, S. 354–367 (zitiert auf S. 25).

[49] L. Richardson. *The Maturity Heuristic*. https://www.crummy.com/writing/speaking/2008-QCon/act3.html. 2020 (zitiert auf S. 19).

[50] L. Richardson, S. Ruby. *RESTful web services*. Ö'Reilly Media, Inc.", 2008 (zitiert auf S. 16, 19, 32, 33).

[51] C. Rodrıguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, G. Percannella. „REST APIs: a large-scale analysis of compliance with principles and best practices“. In: *International conference on web engineering*. Springer. 2016, S. 21–39 (zitiert auf S. 11, 18, 19, 26).

[52] K. Sandoval. *What is the Richardson Maturity Model?* https://nordicapis.com/what-is-the-richardson-maturity-model/. Apr. 2018 (zitiert auf S. 20).

[53] W. Santos. *Which API Types and Architectural Styles are Most Used?* https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26. Nov. 2017 (zitiert auf S. 11).

[54] S. S. Sawilowsky. „New effect size rules of thumb“. In: *Journal of modern applied statistical methods* 8.2 (2009), S. 26 (zitiert auf S. 44, 50, 51).

[55] S. Scalabrino, G. Bavota, C. Vendome, D. Poshyvanyk, R. Oliveto et al. „Automatically assessing code understandability“. In: *IEEE Transactions on Software Engineering* (2019) (zitiert auf S. 23, 29, 36, 61, 62).

[56] J. P. Shaffer. „Multiple hypothesis testing“. In: *Annual review of psychology* 46.1 (1995), S. 561–584 (zitiert auf S. 39, 44).

[57] S. S. Shapiro, M. B. Wilk. „An analysis of variance test for normality (complete samples)“. In: *Biometrika* 52.3/4 (1965), S. 591–611 (zitiert auf S. 44).

[58] Student. „The probable error of a mean“. In: *Biometrika* (1908), S. 1–25 (zitiert auf S. 44).

[59] H. Subramanian, P. Raj. *Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*. Packt Publishing Ltd, 2019 (zitiert auf S. 19).

[60] J. Tihomirovs, J. Grabis. „Comparison of soap and rest based web services using software evaluation metrics.“ In: *Information Technology & Management Science (Sciendo)* 19.1 (2016) (zitiert auf S. 27).

[61] R. Tsouroplis, M. Petychakis, I. Alvertis, E. Biliri, D. Askounis. „Community-based API Builder to manage APIs and their connections with Cloud-based Services.“ In: *CAiSE Forum*. 2015, S. 17–23 (zitiert auf S. 21).

[62] B. Varanasi, S. Belida. „Introduction to REST“. In: *Spring REST*. Springer, 2015 (zitiert auf S. 17, 18).

[63] J. Webber, S. Parastatidis, I. Robinson. *REST in practice: Hypermedia and systems architecture*. Ö'Reilly Media, Inc.", 2010 (zitiert auf S. 11, 19).

[64] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (zitiert auf S. 39–41, 60, 61).

[65] B. Zeiss, D. Vega, I. Schieferdecker, H. Neukirchen, J. Grabowski. „Applying the ISO 9126 quality model to test specifications–exemplified for TTCN-3 test specifications“. In: *Software Engineering 2007–Fachtagung des GI-Fachbereichs Softwaretechnik* (2007) (zitiert auf S. 22).

[66] W. Zhou, L. Li, M. Luo, W. Chou. „REST API design patterns for SDN northbound API“. In: *2014 28th international conference on advanced information networking and applications workshops*. IEEE. 2014, S. 358–365 (zitiert auf S. 19).

Alle URLs wurden zuletzt am 09. 11. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift