TECHNISCHE UNIVERSITÄT
CHEMNITZ

Einfluss der Sprache von Code Identifiers auf das Codeverständnis

Bachelorarbeit

zur Erlangung
des akademischen Grades
B.Sc.

Fakultät für Informatik
Professur Softwaretechnik

Eingereicht von: Florian Grabs

Einreichungsdatum: 18.11.2022

Betreuerin: Prof. Dr. Janet Siegmund
Elisa Madeleine Hartmann

Danksagung

Ich danke meiner Betreuerin Elisa Madeleine Hartmann für die gute Zusammenarbeit und vor allem für die Hilfe beim Suchen einer geeigneten Teilnehmergruppe.

Außerdem bedanke ich mich bei allen Studenten, die an meiner Studie teilgenommen haben.

Abstract

Hintergrund: Es gibt eine Vielzahl an Studien, welche sich mit Codeverständnis beschäftigen. Es wichtiger Themenbereich ist der Einfluss von Codeidentifiers (Variablennamen). Diese folgen bestimmten Namenskonventionen, können jedoch vom Programmierer frei gewählt werden.
Forschungsziel: Die vorliegende Arbeit befasst sich mit Codeidentifiers in verschiedenen natürlichen Sprachen. Dabei soll die Auswirkung dieser auf das Programmverständnis untersucht werden.
Forschungsmethode: Es wurde eine online Studie durchgeführt, bei der Teilnehmende 6 unterschiedlich komplexe Algorithmen lösen mussten. Dabei waren die Codeidentifier entweder auf Deutsch, Englisch oder Arabisch. Gemessen wurde jeweils die Bearbeitungszeit und Korrektheit der Antworten. Es nahmen insgesamt 71 Studenten an der Studie teil.
Ergebnisse: Am besten schnitten Codesnippets mit englischen Identifiers ab. Sie wurden durchschnittlich am schnellsten und korrektesten beantwortet. Je nach Spracherfahrung des jeweiligen Teilnehmers schnitten deutsche und arabische Codeidentifier mindestens genauso gut ab. Codeidentifier in fremden Schriftsystem wurden selten korrekt beantwortet.
Schlussfolgerung: Codeidentifier in verschiedenen natürlichen Sprachen haben keinen signifikanten Vorteil gegenüber den Englischen. Diese sind universeller und zuverlässiger.
Zukünftige Arbeiten: Zur Bestätigung der Ergebnisse sind weitere Studien notwendig. Diese können aus einer anderen Teilnehmergruppe, anderen Programmiersprachen oder weiteren Sprachen der Codeidentifier bestehen. Außerdem können unter Anderem Eye-Tracker oder EEG zu anderen Ergebnissen führen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CI Codeidentifier

TUC Technische Universität Chemnitz

PA Programmieranfänger

AA ArrayAverage

BD BinaryToDecimal

CS CrossSum

FA Fatorial

GD GreatestCommonDivisor

IP IsPrime

PO Power

SU SumUp

1 Einleitung

Codeverständnis ist ein wichtiges Themengebiet des Software Engineering. Es beschreibt wie Entwickler Quellcode verstehen. Studien zeigen das Entwickler den Großteil ihrer Zeit damit verbringen Programmcode zu verstehen. Zudem bilden Wartungskosten den größten Anteil bei der Entwicklung von Software [1]. Eine bessere Codeverständlichkeit spart folglich Ressourcen und Kosten [5].
Neben Namenskonventionen wie camelCase gibt es auch weitere Methoden um Quellcode verständlich zu machen. Ein sauber und nachvollziehbar kommentierter Code ist ein wesentlicher Bestandteil der Code-Qualität. Kommentare beeinflussen zwar die Funktionsweise des Programms nicht, tragen jedoch zur Verständlichkeit bei [10].
Weitere beispielhafte Codekonventionen in Java sind unter Anderem Einrückungen [2], Dateinamen [3] oder Deklarationen [4].

1.1 Forschungslücke und Ziel

Neben den verschiedenen Namenskonventionen für Variablen können diese vom Programmierer stets frei gewählt werden. Standardmäßig werden diese nach englischen Begriffen benannt, jedoch ist es möglich anderssprachige Variablennamen zu verwenden um die Codeverständlichkeit für einige Betrachter zu verbessern. Laut einer Umfrage haben ca. 35% der Deutschen ab 14 Jahren geringe oder keine Englischkenntnisse [5]. Dazu kommen noch einige Kinder unter 14 Jahren, welche lediglich Schulenglisch beherrschen. Laut einer weiteren Umfrage haben ca. 12% [6] geringe bis keine Englischkenntnisse.
Es gibt bereits Konzepte wie man Programmieren für Anfänger einfacher und verständlicher gestalten kann. An einigen deutschen Schulen wird das Programm Robot Karol genutzt, welches eine eigene Programmierumgebung in Deutsch mit sich bringt [8].
Als internationales Beispiel ist Hedy zu nennen, eine Programmiersprache, welche sich auf Python basiert, jedoch zu Beginn nur einen Bruchteil der Funktionalität bietet. Dabei sind Befehle in verschiedenen natürlichen Sprachen vorhanden [4].
Die vorliegende Arbeit soll zeigen ob Variablennamen (auch Identifier oder Codeidentifier genannt) in verschiedenen natürlichen Sprachen einen Einfluss auf das

[1] https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html
[2] https://www.oracle.com/java/technologies/javase/codeconventions-indentation.html
[3] https://www.oracle.com/java/technologies/javase/codeconventions-filenames.html
[4] https://www.oracle.com/java/technologies/javase/codeconventions-declarations.html
[5] https://de.statista.com/statistik/daten/studie/170896/umfrage/einschaetzung-zu-eigenen-englischkenntnissen/
[6] https://www.presseportal.de/pm/110144/3898005

Codeverständnis haben. Dazu wurde ein Experiment mit 71 Teilnehmern durchgeführt, welche Codesnippets mit Codeidentifier (CI) in verschiedenen Sprachen lösen mussten. Als Bewertungskriterien werden Korrektheit und Beantwortungszeit genommen. Dabei sollen die folgenden Forschungsfragen beantwortet werden:

Forschungsfrage 1 Welchen Effekt haben CI in verschiedenen natürlichen Sprachen auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

Forschungsfrage 2 Hat das Schriftsystem der CI einen Effekt auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

1.2 Struktur

Die vorliegende Arbeit ist folgendermaßen gegliedert:

Kapitel 2 Im zweiten Kapitel soll der theoretische Hintergrund vorgestellt werden. Dazu zählen ausgewählte Methoden zur Messung des Programmverständnisses wie das Top-Down und Bottom-Up Modell. Außerdem wird die optimale Länge einer online Studie behandelt.

Kapitel 3 Im Kapitel 3 wird der aktuelle Forschungsstand über den Einfluss von Variablennamen dargestellt.

Kapitel 4 Im nächsten Kapitel wird die Methodik der Arbeit beschrieben. Dazu zählt die Auswahl der Studienteilnehmer und der eigentliche Aufbau der Studie. Dazu werden alle verwendeten Materialien präsentiert.

Kapitel 5 Aufbauend auf das vorherigen Kapitel wird im fünften Kapitel die Durchführung des Experimentes beschrieben.

Kapitel 6 Im Kapitel 6 werden die Ergebnisse des Versuches präsentiert.

Kapitel 7 Im folgenden Kapitel werden die Ergebnisse aus Kapitel 6 diskutiert und die Forschungsfragen beantwortet.

Kapitel 8 Im letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst und weiterführende Forschungsmöglichkeiten erläutert.

Alle verwendeten Bilder, Diagramme und Datensätze sind in der TUC Cloud abgelegt[7].

[7] https://tuc.cloud/index.php/s/XnpGzZQ2ZEWCWJ4

2 Theoretischer Rahmen

Im folgenden Kapitel werden gewissen Methoden zum Messen des Programmverständnis vorgestellt. Außerdem wird die optimale Länge einer online Studie diskutiert.

2.1 Programmverständnis

Im Gebiet Programmverständnis wird bereits seit über 30 Jahren an verschiedenen Methoden geforscht. Neben alten, etablierten Ansätzen gibt es auch moderne Methoden um das Programmverständnis zu messen [19]. Außerdem gibt es 3 verschiedene Verständlichkeitsmodelle: Top-Down, Bottom-Up und Integrated.

2.1.1 Gängige Testmethoden

Eine gängige Methode sind Think-Aloud Protokolle. Teilnehmer einer Studie beschreiben während sie eine Aufgabe bearbeiten ihren aktuellen Denkprozess. Die Gespräche werden entweder aufgezeichnet oder es werden Notizen erstellt. Daraus entstehen später Schlussfolgerungen oder Vergleiche mit anderen Teilnehmern [19]. Ein Beispiel dafür ist eine Studie von Shaft und Vessey. Programmieren mussten verschiedenen Quellcode verstehen und dabei ihren Denkprozess erläutern [19] [16]. Dabei verwendeten Programmierer eher Top-Down Modelle (Abschnitt 2.1.2) wenn sie mit dem Anwendungsbereich vertraut waren. Wenn dieser jedoch für sie unbekannt ist wurden eher Bottom-Up Modelle verwendet (Abschnitt 2.1.3) [16].

Eine weitere etablierte Methode ist das Auswendiglernen und Wiedergeben von Quellcode. Dabei müssen sich Teilnehmer Quellcode so gut wie möglich einprägen und anschließend replizieren. Das Einprägen ist dabei stark abhängig von dem Programmverständnis. Je besser ein Teilnehmer den Quellcode verstanden hat, desto eher kann er ihn fehlerfrei wiedergeben [19].

Eine weitere Methode sind Verständlichkeitsaufgaben. In einer Studie von Soloway und Ehrlich müssen zum Beispiel fehlende Zeilen im Quellcode ausgefüllt werden [19] [21]. In einer anderen Studie von Boysen sollen Teilnehmer einfache Codeaufgaben lösen. Dabei werden Korrektheit und Beantwortungszeit gemessen [19] [6].

2.1.2 Top-Down Modell

Wie bereits in Abschnitt 2.1.1 beschrieben nutzen Programmierer das Top-Down Modell meistens wenn sie mit dem Anwendungsreich, bzw. der Problemstellung, vertraut sind [16]. Dabei werden initiale Hypothesen über die Funktionalität und Ziel gestellt. Dafür benötigt man gewisses Hintergrundwissen über die Problemstellung um das aktuelle Problem mit ähnliche Aufgabenstellungen zu vergleichen und anschließend einen geeigneten Lösungsweg zu finden [19]. Während der Analyse stoßen Programmierer auf sogenannte beacons - "sets of features that typically indicate the occurence of certain structures or operations in the code" [2]. Diese können zum Beispiel Schleifen oder Verzweigungen sein. Diese beacons helfen die Hypothese zu bestätigen [2].

2.1.3 Bottom-Up Modell

Das Bottom-Up Modell wird genutzt wenn Programmierer wenig bzw. kein Fachwissen über die Problemstellung haben [16]. Deswegen können auch keine beacons ausfindig gemacht werden [19]. Folglich müssen Programmierer den Quellcode zuerst analysieren um Hypothesen aufzustellen. Dabei wird jede Zeile betrachtet, verstanden und mit anderen Zeilen verknüpft. Den Prozess des Verknüpfens nennt man chunking. Wenn genügend chunks gesammelt sind, werden diese zusammengesetzt um schlussendlich die Bedeutung des gesamten Quellcodes zu verstehen [19].

2.1.4 Integrated Modell

Das Integrated Modell ist die Kombination des Top-Down (Abschnitt 2.1.2) und Bottom-Up (Abschnitt 2.1.3) Modells. Beispielsweise kann ein Programmierer gutes Hintergrundwissen über die Problemstellung haben und Hypothesen formulieren. Bei der Codeanalyse finden sich jedoch Codeabschnitte die unbekannt sind. Dann werden diese Abschnitte zeilenweise Analysiert um ihre Funktion im gesamten Programm zu verstehen. Da die Top-Down Methode wesentlich effizienter ist, wird diese der Bottom-Up Methode vorgezogen [19].

2.1.5 Moderne Methoden

Moderne Testmethoden beziehen sich hauptsächlich auf neuronale Techniken.
Eine Studie von Frau Siegmund untersuchte das Verhalten von Programmieren mit Hilfe von functional magnetic resonance imaging (fMRI) [20]. Die 17 Teilnehmer mussten in einem fMRI Scanner Syntaxfehler in kurzen Codesnippets finden. Dabei wurden eindeutige Aktivieren in 5 verschiedenen Hirnarealen festgestellt, dazu zählen das Arbeitsgedächtnis, Aufmerksamkeits- und Sprachverarbeitungszentrum [20].
Eine weitere Studie befasste sich mit dem zerebralen Blutfluss im Gehirn. Dazu wurden ein Near Infra-red Spectroscopy (NIRS) genutzt um den Blutfluss zu messen.

8 der 10 Testpersonen zeigten einen erhöhten Blutfluss im Gehirn beim Verstehen von schwierigem Quellcode [11].
Kluthe nutzte in seiner Studie ein Elektroenzephalogramm (EEG) um das Programmverständnis zu messen [9] [19]. Dabei mussten Teilnehmer die Ausgabe von Codesnippets bestimmen. Man fand heraus das weniger erfahrene Programmierer eine höhere kognitive Belastung beim lösen der Aufgaben zeigten, was auf dem EEG Signal zu sehen war [9] [19].
Eine weitere Möglichkeit der Messung ist das Eye-Tracking. Dabei werden Bewegungen der Augen beim Lesen von Quellcode aufgezeichnet und anschließend Muster analysiert [17]. In einer Studie von Sharif wurde ein Eyetracker verwendet um den Einfluss von Namenskonvention wie camelCase oder under_score zu untersuchen. Man hat herausgefunden das der under_score Stil eher erkannt wird. Auf die Korrektheit der Antworten hat der Schreibstil jedoch keine Auswirkungen gehabt [18].

2.2 Länge und Umfang einer Online Studie

Der zeitliche Umfang einer Umfrage hat einen großen Einfluss auf die Qualität der Antworten. So haben längere Befragungen die Tendenz "don't know" Antworten oder komplette Befragungsabbrüche zu erzeugen. Außerdem werden Fragen zum Ende der Umfrage oft nur noch kurz oder gar nicht beantwortet [13, S. 2]. Eine Studie aus Deutschland fand heraus das die optimale Länge für eine Umfrage 10 bis 15 Minuten ist. Dabei sollte sie aber maximal 28,7 Minuten lang sein [13, S. 5]. Die optimale Befragungsdauer ist jedoch von vielen Faktoren, wie zum Beispiel die Art der Befragung. So sind einige Teilnehmer eher gewillt eine längere online Studie als eine längere offline Studie durchzuführen [13, S. 7]. Eine weitere Studie aus Mexiko fand heraus, das eine online Umfrage maximal 20 Minuten in Anspruch nehmen sollte [14].

3 Related Work

Im folgenden Kapitel werden verwandte Studien diskutiert. Im Abschnitt 3.1 werden Studien über die Länge von CI erläutert. Danach wird im Kapitel 3.2 die Programmiersprache Hedy vorgestellt, welche Programmieranfängern den Einsteig erleichtern soll.

3.1 Länge von Codeidentifier

In einer Programmiersprache sind Befehle wie Funktionen, Schleifen, Bedingungen syntaktisch festgelegt. Sie können nicht umbenannt werden. Jedoch kann jeder Programmieren Variablen nach dem eigenen Belieben benennen. Natürlich können bereits festgelegte Anweisungen (z.B. while in Java) nicht als Variablenname verwendet werden.
Mit zunehmender Programmlänge, oder wenn Dritte den Quellcode betrachten, kann es schnell zu Unverständlichkeiten kommen.
Dawn Lawrie befasste sich 2007 in ihrer Studie mit den Auswirkungen verschiedener Identifier auf das Programmverständnis und das Kurzzeitgedächtnis der Teilnehmer [3]. Über 100 Programmierer sollten jeweils 12 Funktionen verstehen und beschreiben und anschließend die verwendeten Identifier wiedergeben. Dabei wurden 3 verschiedene Arten von Codeidentifier genutzt: einzelne Buchstaben, Abkürzungen und ganze Wörter. Das beste Codeverständnis wurde mit ganzen Wörtern erreicht, obwohl der statistische Unterschied zu Abkürzungen sehr gering ist. Im Bezug auf das Kurzzeitgedächtnis des Menschen sind gut gewählte Abkürzungen die beste Wahl [3].
Johannes Hofmeister, Janet Siegmund und Daniel V. Holt haben eine ähnliche Studie durchgeführt [7]. Die 72 Teilnehmer mussten Fehler in Codeabschnitten finden. Dabei waren die Identifier entweder einzelne Buchstaben, Abkürzungen oder ganze Wörter. Zwischen einzelnen Buchstaben und Abkürzungen konnte kein signifikanter Unterschied in der Lösungszeit festgestellt werden. Ganze Wörter führten jedoch zu einer 19% schnelleren Antwortzeit. Folglich sind Identifier in Form von ganzen Wörtern hilfreich für das Finden von Codefehlern [7].
Eine Studie von Gal Beniamini beschäftigt sich ausschließlich mit single-letter Identifiers [1]. Diese können in einigen Situation aussagekräftiger sein als längere Codeidentifier. Ein Beispiel ist i, j oder k als Schleifenparameter zu nutzen. Anderenfalls wird s von vielen sofort als string gedeutet oder t als time [1].
Andrea Schankin und weitere befassten sich 2018 mit der Auswirkung längerer, aber dafür beschreibender Identifier [15]. 88 Programmierer mussten semantische Fehler

in Java Code finden. Mit langen, beschreibenden Identifiers wurden semantische Fehler 14% schneller gefunden als mit kurzen Codeidentifiers. Außerdem gab es beim Lesen des Codes weniger Zeilensprünge auf Grund von Unverständlichkeit. Zudem war der Einfluss der Identifier bei erfahrenen Programmieren größer als bei Unerfahrenen [15].

3.2 Hedy

Hedy ist eine "mitwachsende" Programmiersprache, welche Programmieranfängern das Lernen einer Programmiersprache erleichtern soll [4]. Dabei soll es primär darum gehen wie die Syntax der Sprache dem Lernenden beigebracht wird. Hedy besteht aus verschiedenen Leveln, welche stetig neue Sytax in Form von Befehlen und Elementen hinzufügt. Schlussendlich ist Hedy eine Variante von Python [4]. Die Befehle und Funktionen sind dabei in verschiedenen natürlichen Sprachen verfügbar. Beispielsweise heißt die "if" Bedingung im Deutschen "falls".
Hedy wird im eigenen Texteditor auf der Hedy Homepage ausgeführt [1]. Die Benutzeroberfläche ist sehr intuitiv und besteht aus einem Texteditor, einer Ausgabekonsole und den Aufgaben der einzelnen Level.

```
print Hello!
print Welcome to your first programming lesson
```

Abbildung 3.1: Beispiel Hedy Level 1 [4]

In Level 1 (Abbildung 3.1) werden lediglich 3 einfache Befehle verwendet: print, aks und echo. Print gibt eine Nachricht auf der Konsole aus, Ask öffnet ein Eingabefeld und Echo gibt diese Eingabe aus.

```
print name is Jason
print Welcome to your first programming lesson name
```

Abbildung 3.2: Beispiel Hedy Level 2 [4]

In Level 2 wird das Konzept von Variablen integriert (Abbildung 3.2). Der Befehl is weist einer Variable, in diesem Fall "name", einen Wert zu (Jason). Dieser kann dann im Verlauf des Programms über den Variablennamen "name" abgerufen werden.
Erreicht der Teilnehmer Level 7, so sind die wichtigsten Grundbefehle von Python bereits vorhanden, darunter Variablen, Schleifen und Bedingungen. Auf diesem Level sind sogar bereits Sortier- und Filteralgorithmen programmierbar [4].

[1] https://www.hedycode.com/hedy/

```
Preise = ['1 Million Euro', 'ein Apfelkuchen', 'nichts']
dein_Preis = Preise[random]
print 'Du gewinnst ' dein_Preis
if dein_Preis == '1 Million Euro' :
    print 'Juhuu! Du bist reich!'
elif dein_Preis == 'ein Apfelkuchen' :
    print 'Wunderbar, ein Apfelkuchen!'
else:
    print 'Viel Glück beim nächsten Mal...'
```

Abbildung 3.3: Beispiel Hedy Level 17 Englisch

```
Preise = ['1 Million Euro', 'ein Apfelkuchen', 'nichts']
dein_Preis = Preise[zufällig]
drucke 'Du gewinnst ' dein_Preis
falls dein_Preis == '1 Million Euro' :
    drucke 'Juhuu! Du bist reich!'
sofalls dein_Preis == 'ein Apfelkuchen' :
    drucke 'Wunderbar, ein Apfelkuchen!'
sonst:
    drucke 'Viel Glück beim nächsten Mal...'
```

Abbildung 3.4: Beispiel Hedy Level 17 Deutsch

In Abbildung 3.3 sieht man eines der letzten Level in Hedy. Dieses erinnert schon stark an Python. Es gibt Arrays (Zeile 1), if, elseif und else Verzweigungen. Die Sprache der Befehle ist jedoch trotzdem noch einstellbar (Abbildung 3.4). Es gibt noch viele weitere Befehle, welche schrittweise dem Lernenden beigebracht werden.

Ergebnisse von Hedy

In der Studie der Entwicklerin von Hedy, Felienne Hermans, wurden insgesamt 9714 geschrieben Programme der Level 1 bis 7 analysiert [4]. Die allgemeine Fehlerrate betrug 12,2%. 20% davon sind Fehler die durch die Lernweise entstehen. Dies entspricht 2,6% aller Programme. Da Hedy in Level aufgeteilt ist und nicht jedes Level die gleichen Befehle hat, kann es vorkommen das Lernende Befehle aus einem vorherigen Level nutzen, diese aber im aktuellen Level nicht unterstützt werden. Alle weiteren Fehler sind Programmierfehler, welche bei jeder anderen Lernmethode auch vorkommen können. Da nur ein Bruchteil der Fehler auf die Lernweise zurückzuführen sind und die gesamte Fehlerrate vergleichsweise gering ist, wir Hedy als Erfolg betrachtet [4].

4 Methodik

Im folgenden Kapitel werden die Forschungsfragen im Abschnitt 4.1.1 und Hypothesen (Abschnitt 4.1.2) beschrieben. Anschließend wird im Abschnitt 4.2 die Teilnehmergruppe dargestellt. Im letzten Teil 4.3 werden die Materialien der Studie beschrieben.

4.1 Forschungsziel

Die Studie befasst sich mit den Auswirkungen von CI in verschiedenen natürlichen Sprachen auf die Verständlichkeit des Codes für verschiedene Personengruppen. Primär sollen jedoch Programmieranfänger, bzw. Programmierneulinge im Fokus stehen, da diese eher von muttersprachlichen CI profitieren als fortgeschrittene Programmierer, welche bereits jahrelange Erfahrung mit englischen CI haben.

4.1.1 Forschungsfragen

Basierend auf den verwandten Studien aus Kapitel 3 entstehen zwei Forschungsfragen. Diese lauten wie folgt:

Forschungsfrage 1 Welchen Effekt haben CI in verschiedenen natürlichen Sprachen auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

Forschungsfrage 2 Hat das Schriftsystem der CI einen Effekt auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

4.1.2 Hypothesen

Hinsichtlich der ersten Forschungsfrage siehe Abschnitt 4.1.1 ergeben sich folgende Hypothesen:

H_1 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese schneller beantwortet als Codeprobleme mit CI in einer anderen Sprache.

H_2 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese eher korrekt beantwortet als Codeprobleme mit CI in einer anderen Sprache.

H_3 Teilnehmer werden Codeprobleme mit englischen CI schneller beantworten als Codeprobleme mit fremdsprachigen CI.

H_4 Teilnehmer werden Codeprobleme mit englischen CI eher korrekt beantworten als Codeprobleme mit fremdsprachigen CI.

Bezüglich der zweiten Forschungsfrage siehe Abschnitt 4.1.1 wird folgende Hypothese definiert:

H_5 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem schneller beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

H_6 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem eher korrekt beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

4.2 Studienteilnehmer und Sprache der Identifier

Diese Studie richtet sich an Informatikstudenten der TUC, welche zumindest grundlegende Programmiererfahrung in Java haben sollten. Außerdem sollte ein guter Mix aus deutschsprachigen und internationalen Studienteilnehmern entstehen. An dem Experiment durften sowohl Bachelor- als auch Masterstudent-innen teilnehmen. Die Befragung war freiwillig und konnte jederzeit abgebrochen werden. Da an der TUC im Zeitraum der Studie kein Präsenzunterricht stattfand, musste die Teilnehmersuche online durchgeführt werden. Da viele internationale Studenten nicht vor Ort sind wird das Experiment remote von jedem Teilnehmer selbst bearbeitet werden (siehe Abschnitt 4.3).

Die Studie soll mit CI in 3 verschiedenen Sprachen durchgeführt werden. Zum Einen wird Englisch verwendet, da dies die standardmäßige Programmiersprache ist. Neben Englisch ist eine Sprache im gleichen und eine in einem fremden Schreibsystem notwendig. Laut einer Statistik des Prüfungsamtes der TUC (siehe Anhang 9.1) ist 1/3 und damit der größte Anteil der Informatikstudenten deutscher Herkunft. Alle weiteren Herkunftsländer mit lateinischen Schriftsystem sind nur kleine Minderheiten. Folglich wird neben Englisch Deutsch als zweite Sprache der CI verwendet. Als Sprache mit einem anderen Schriftsystem stehen Arabisch (ca. 10% der Studenten), eine chinesische Sprache (ca. 2% der Studenten) oder eine indische Sprache (a. 25% der Studenten) zur Auswahl. Da es verschiedene chinesische Sprachen, z.B. Mandarin oder Cantonese und verschiedene Indische Sprachen, z.B. Urdu, Hindi oder Bengali gibt, fallen beide Sprachgruppen für die weitere Betrachtung raus. Schlussendlich wird Arabisch als dritte Sprache für CI verwendet, da diese eine allgemeine Standartsprache besitzt, welche jeder arabisch sprechende Teilnehmer verstehen kann.

Die zu lösenden Algorithmen lagen mit CI in englischer Sprache vor. Die Übersetzung

ins Deutsche wurde eigenständig durchgeführt. Die arabischen CI wurde mit Hilfe eines Familienmitgliedes syrischer Herkunft übersetzt.
Da das arabische Schriftsystem, im Gegensatz zum Lateinischen, von rechts nach links gelesen wird, mussten die arabischen Codesnippets per Hand angepasst werden.

```
int num1 = 0;
```

Abbildung 4.1: Beispiel Englisch

```
int 0 = 1رقم;
```

Abbildung 4.2: Beispiel Arabisch

Wenn man beispielsweise den englischen CI im Code in Abbildung 4.1 durch einen arabischen CI ersetzt erhält man die Zeile in Abbildung 4.2. Jedes Mal wenn ein Wort im lateinischen Schriftsystem gegen eines im arabischen Schriftsystem getauscht wird, wird die dazugehörige Operation automatisch von rechts nach links geschrieben. Das kann schnell zu Verwirrung führen, da zum Beispiel Variablentypen trotzdem am Zeilenanfang stehen (siehe Abbildung 4.2). Um dies zu Umgehen wurden alle arabischen CI in einem Grafikprogramm über die englischen CI eingefügt. Das Resultat ist in Abbildung 4.3 zu sehen.

```
int 1رقم = 0;
```

Abbildung 4.3: Beispiel Arabisch angepasst

4.3 Materialien und Aufgaben

Die Studie besteht aus 6 Programmverständnisaufgaben, welche mit CI in Deutsch, Englisch und Arabisch eingesetzt werden. Neben einer Beispielaufgabe und der Abfrage demographischer Daten mit Hilfe eines Fragebogens können Teilnehmer zum Ende des Experimentes die allgemeine Schwierigkeit bewerten und Feedback geben. Die Studie wurde mit Hilfe des Onlinebefragungsservices SoSci durchgeführt. [1]

4.3.1 Fragebogen

Nach einer einfachen Beispielaufgabe in Java mit englischen Identifiers siehe Abbildung 9.1, wurde im ersten Teil der Studie ein Fragebogen verwendet, um demographische Daten und Programmiererfahrung aufzunehmen. Dieser besteht aus 6 Fragen. Die ersten beiden Fragen befassen sich mit der Programmiererfahrung. Zunächst mussten die Teilnehmer angeben, wie lange sie bereits in Java programmieren können siehe Abbildung 9.2. Sie können auf einer fünfteiligen Skala (0, 0-1, 1-3, 3-5, 5+ Jahre) genau einen Zeitraum auswählen. Anschließend mussten die Teilnehmer auf der gleichen Skala angeben, wie lange sie generell programmieren können siehe Abbildung 9.3. Die dritte und vierte Frage beschäftigt sich mit der Herkunft des Teilnehmers. Das Herkunftsland wurde in ein einfaches Textfeld eingetragen siehe Abbildung 9.4. Danach mussten die Teilnehmer ihre Sprachkenntnisse in Deutsch, Englisch und Arabisch einschätzen siehe Abbildung 9.5. Dazu diente eine Multiple Choice Abfrage mit jeweils fünf möglichen Antworten. Diese sind Novice (Anfänger), Intermediate (Mittelmäßig), Advanced (Fortgeschritten), Fluently (Fließend) und Mother tongue (Muttersprache). Anschließend konnten in einem Textfeld weitere Sprachen genannt und bewertet werden. Die fünfte Frage ist ebenfalls eine Multiple Choice Frage, welche das Alter des Teilnehmers abfragt siehe Abbildung 9.6. Es gab die Wahl zwischen jünger als 20, 20-25, 25-30, 30-40 oder 40+ Jahre alt. Als Letztes wurde der akademische Abschluss der Teilnehmer abgefragt siehe Abbildung 9.7. Es besteht die Auswahl zwischen Bachelor, Master, kein Abschluss und anderer Abschluss, wobei für letzteres ein zusätzliches Textfeld erscheint, wo der Teilnehmer den alternativen Abschluss eintragen kann.

4.3.2 Codesnippets

Die verwendeten Codesnippets liegen in der Snippetdatenbank der Professur Softwaretechnik an der TUC bereit [2]. Die Algorithmen sind in Java und haben englische CI. Als Programmiersprache wurde Java gewählt, da diese laut einer Umfrage (siehe Abschnitt 5.1) am meisten verstanden wird. Die Codesnippets bestehen immer aus einer einzelnen Funktion, welche mindestens eine Eingangsvariable besitzt

[1] https://www.soscisurvey.de/

[2] Snippetdatenbank der Professur Softwaretechnik (TUC) https://www.tu-chemnitz.de/informatik/ST/CodeWebsite/index.php

und eine Variable zurück gibt. Die teilnehmenden Studenten bekommen zu jedem Codesnippet die Eingabe und müssen die Ausgabe bestimmen. Die Snippets haben verschiedene Schwierigkeitsgrade, welche durch eine Studie bestimmt wurden.[12] Die Ergebnisse sind in Abbildung 4.4 erkennbar.

TABLE I: Code snippets used in the study with four selected complexity metric scores and experimental results: behavioral data (correctness, time) and subjective complexity. A higher intensity of a cell's background color indicates a higher complexity. The histogram plots show the distribution of subjective complexity; skewness to the right represents higher subjective complexity.

	Snippet	Complexity Metrics				Experiment Results		
		Code Size (LOC)[1]	Vocabulary (Halstead)[1]	Control Flow (McCabe)[1]	Data Flow (DepDegree)[2]	Correctness (in %)	Time (in sec.)	Subjective Complexity Low → Medium → High
Loop	Average of array	17	12.64	3	17	47%	49.0	
	Contains substring	26	25.50	7	29	32%	50.2	
	Count vowels in string	19	13.00	5	22	89%	30.4	
	Greatest common divisor	24	26.63	5	33	47%	50.0	
	h index	21	16.25	4	20	53%	46.0	
	Length of last word	22	18.58	8	17	58%	43.3	
	Palindrome check	17	16.72	4	17	79%	38.4	
	Square root of array	23	39.83	5	27	68%	40.1	
Recursion	Binary to decimal	17	16.75	4	10	68%	42.3	
	Cross sum	12	14.66	3	4	84%	27.6	
	Factorial	12	16.50	3	4	95%	22.3	
	Fibonacci variation	12	10.88	3	4	84%	35.6	
	Power	17	15.38	4	8	79%	33.3	
If/Else	Contains yes or no	22	6.13	6	7	95%	23.4	
	Hurricane check	17	9.00	7	13	100%	21.5	
	Sort four elements	17	30.30	7	61	79%	41.4	

[1]https://github.com/BasLeijdekkers/MetricsReloaded/ [2]https://www.sosy-lab.org/~dbeyer/DepDigger/

Abbildung 4.4: Komplexität ausgewählter Codesnippets der Snippetdatenbank der Professur Softwaretechnik (TUC)[12]

Im linken Teil der Tabelle sind vier verschiedene Komplexitätsmatrix Bewertungen berechnet und farblich abgestuft. Je intensiver die Farbe einer Zelle, desto komplexer ist das Codesnippet bewertet. Im rechten Teil der Tabelle sieht man die Ergebnisse des Experimentes. Darunter ganz rechts ein Balkendiagramm der subjektiven Komplexität, sprich wie Studienteilnehmer jedes Codesnippet individuell bewertet haben. Aus den vorliegenden Codesnippets wurden verschieden komplexe ausgewählt, welche für die hier beschriebene Studie verwendet werden. Als Bewertung der Schwierigkeit wurde die subjektive Komplexität, siehe Tabelle 4.4 letzte Spalte, genutzt. Wenn diese durchschnittlich eher als einfach eingeschätzt wurde, so wird das Snippet in Kategorie 1 (einfaches Snippet) eingeordnet. Ist die Komplexität gleichmäßig verteilt eingeschätzt wurden, so ist das Codesnippet in Kategorie 2 (mittelschweres Snippet). Die schweren Algorithmen kommen in Kategorie 3 (komplexes Snippet).

Kategorie 1 - einfache Snippets

```
public int blue(int input) {
    int result = 0;

    while (input != 0) {
        result += input % 10;
        input /= 10;
    }

    return result;
}
```

Abbildung 4.5: Snippet CrossSum

Das erste einfache Codesnippet ist CrossSum (CS) (siehe Abbildung 4.5). Dieses berechnet die Quersumme eines Integer und gibt diese zurück.

```
public int pink (int input) {
    int result = 1;

    while (input > 1) {
        result = result * input;
        input--;
    }

    return result;
}
```

Abbildung 4.6: Snippet Factorial

Der rekursive Algorithmus Fatorial (FA) (siehe Abbildung 4.6) gibt die Fakultät eines Integer zurück.

Kategorie 2 - mittelschwere Snippets

```java
public float green(int[] input) {
    int number1 = 0;
    int number2 = 0;

    while (number1 < input.length) {
        number2 = number2 + input[number1];
        number1 = number1 + 1;
    }

    float result = number2 / (float) number1;
    return result;
}
```

Abbildung 4.7: Snippet ArrayAverage

Der Algorithmus ArrayAverage (AA) bestimmt den Mittelwert eines Integer Arrays. Dieser wird als float-Wert zurück gegeben.

```java
public int yellow(int num1, int num2) {
    int result = num1;
    if (num2 == 0)
        return 1;

    for (int i = 1; i < num2; i++) {
        result = result * num1;
    }

    return result;
}
```

Abbildung 4.8: Snippet Power

Das Codesnippet Power (PO) bekommt zwei Integer als Eingabewert. Dabei ist der erste die Basis und der zweite der Exponent. Anschließend wird die Potenz der beiden Zahlen berechnet und zurück gegeben.

Kategorie 3 - komplexe Snippets

```
public int grey(int number) {
    if (number < 0) {
        return -1;
    }

    int result = 0;
    int tempNumber = number;
    int variable = 0;

    for (int i = 0; tempNumber > 0; i++) {
        variable = tempNumber % 10;
        result = result + variable * (int) Math.pow(2, i);
        tempNumber = tempNumber / 10;
    }

    return result;
}
```

Abbildung 4.9: Snippet BinaryToDecimal

Der Algorithmus BinaryToDecimal (BD) wandelt eine eingegeben Binärzahl in eine Dezimalzahl um und gibt diese anschließend zurück.

```
public int orange (int number1, int number2) {
    int temp;
    do {
        if (number1 < number2) {
            temp = number1;
            number1 = number2;
            number2 = temp;
        }
        temp = number1 % number2;
        if (temp != 0) {
            number1 = number2;
            number2 = temp;
        }
    } while (temp != 0);
    return number2;
}
```

Abbildung 4.10: Snippet GreatestCommonDivisor

Das Codesnippet GreatestCommonDivisor (GD) bestimmt den größten gemeinsamen Teiler der beiden Eingansvariablen und gibt diesen zurück.

weitere Snippets

Zusätzlich zu den Snippets der 3 Kategorien wurden zwei weitere Algorithmen für die weitere Betrachtung verwendet. Diese sind nicht in Tabelle 4.4 aufgeführt, da sie nicht in der Studie betrachtet wurden. Daher bekommen die beiden Codesnippets auch keine Komplexitätsbewertung.

```
public boolean green(int input) {
    boolean result = true;

    for(int i = 2; i < input; i++) {
        if (input %i == 0) {
            result = false;
            break
        }
    }

    return result;
}
```

Abbildung 4.11: Snippet IsPrime

Zum Einen wird der Algorithmus IsPrime (IP) betrachtet, da dieser im Gegensatz zu den restlichen Codesnippets einen Boolean zurück gibt. Dieser ist True wenn die Eingabe eine Primzahl ist, andernfalls ist die Ausgabe False.

```
public int blue (int input) {
    int result = 0;

    for (int i = 1; i <= input; i++) {
        result += i;
    }
    return result;
}
```

Abbildung 4.12: Snippet SumUp

Das Snippet SumUp (SU) summiert alle ganzen Zahlen (beginnend bei 1) bis zur Eingabezahl auf und gibt diese zurück.

Für die eigentliche Studie wurden nach der Pilotierung (siehe Kapitel 5.2) jedoch nur 6 der 8 vorgestellten Snippets verwendet.

5 Ablauf

Im folgenden Kapitel wird die Durchführung der Studie beschrieben. Im ersten Abschnitt 5.1 wird eine online Umfrage zur Teilnehmersuche dargestellt. Danach folgt im Abschnitt 5.2 die Pilotierung der eigentlichen Studie, gefolgt von der Studienbeschreibung im Abschnitt 5.3.

5.1 Umfrage Google

Am 29.04.2022 startete der erste Versuch Teilnehmer für die Studie zu finden. Dazu wurde ein Google Forms [1] Dokument genutzt, welches erste demographische Daten von Interessenten abfragt. Außerdem konnte eine Email Adresse hinterlegt werden, welche als Kontaktmöglichkeit zur Teilnahme der eigentlichen Studie dienen soll. Die Umfrage wurde über den Fachschaftsrat und in den Kursen von Frau Hartmann verbreitet. Es haben insgesamt 8 Studenten teilgenommen.
Neben Alter (Abbildung 9.12), Geschlecht (Abbildung 9.11), Studiengang (Abbildung 9.8), aktuelles Semester (Abbildung 9.9) und Programmiererfahrung (Abbildung 9.10) mussten die Teilnehmer angeben welche Programmiersprachen sie beherrschen (Abbildung 5.1). Auf Grund der geringen Teilnehmerzahl können keine allgemeinen Schlussfolgerungen gezogen werden, jedoch ist eine klare Tendenz sichtbar. Da 100% der Teilnehmer mit Java vertraut sind wurde Java als Programmiersprache der Codesnippets in Abschnitt 4.3.2 gewählt.

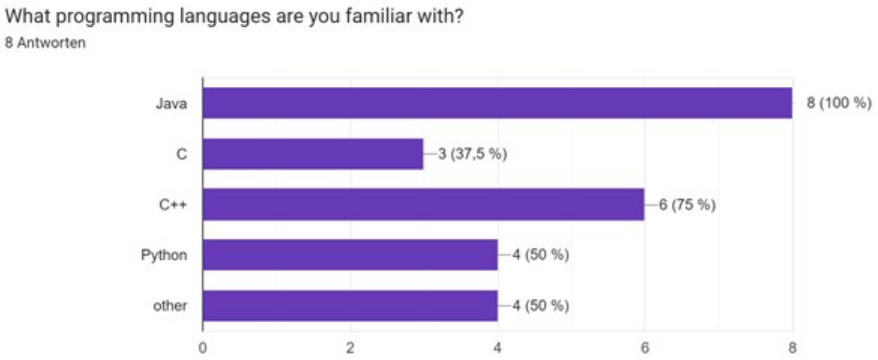


Abbildung 5.1: Google Umfrage, Programmiersprachen

[1] https://www.google.de/intl/de/forms/about/

5.2 Pilotierung

Am 20.07.2022 wurde eine Pilotierung der Studie (Abschnitt 5.3) durchgeführt. Diese sollte noch vorhandene Fehler in den Codesnippets feststellen. Außerdem soll ein geeignetes Zeitlimit gesucht bzw. die Anzahl der Snippets angepasst werden. An der Pilotierung nahmen zwei deutschsprachige Bachelorstudenten aus dem 6. Semester teil. Sie sollten versuchen die Codeaufgaben so gut wie möglich zu lösen und am Besten die gesamte Zeit ausnutzen.
Als Bearbeitungszeit wurde der empfohlene Maximalwert aus Abschnitt 2.2, also 30 Minuten verwendet. Dies entspricht bei 8 verschiedenen Snippets durchschnittlich 3 Minuten und 45 Sekunden pro Snippet. Einer der beiden Teilnehmer konnte alle 8 Snippets in der vorgegebenen Zeit lösen, der Zweite nur 7. Nach der Umfrage sollten beide ein Feedback abgeben. Teilnehmer 1 meinte das die Zeit zum Ende hin knapp wurde, da unter den ersten 5 Snippets zwei schwierige (BD und GD) waren und deswegen wenig Zeit für die letzten 3 Codesnippets blieb. Teilnehmer 2 hatte eine ähnliche Meinung, jedoch hat dieser 4 arabische Snippets lösen müssen, welche deutlich mehr Zeit für eine korrekte Bearbeitung in Anspruch nehmen.
Schlussendlich wurden für die finale Studie 2 Snippets aus der Befragung entfernt. Neben den 5 Codesnippets der Kategorien 1,2 und 3 (Abschnitt 4.3.2) wurde der Algorithmus IP verwendet, da dieser, anders als die restlichen Snippets, einen Boolean als Output hat und deswegen etwas Abwechslung in die Studie bringt. Der Timer von 30 Minuten wurde beibehalten.

5.3 Online Studie

Nach der Pilotierung fand die eigentliche Studie im Zeitraum vom 23.07.2022 bis 23.10.2022 statt. Die SoSci Studie [2] beginnt mit einem Fragebogen (Abschnitt 4.3.1), welcher wichtige demographische Daten abfragt und anhand einer Beispielaufgabe die Funktionsweise der Codeaufgaben erklärt. Anschließend haben die Teilnehmer 30 Minuten Zeit 6 verschiedene Algorithmen (Abschnitt 4.3.2) mit CI in 3 verschiedenen Sprachen zu lösen. Die Algorithmen sind im SoSci alphabetisch sortiert und sind auch in dieser Reihenfolge in der Umfrage erschienen. Dabei erhält jeder Teilnehmer jedes Codesnippet mit CI in zufällig genau einer Sprache. Daher ist es auch möglich das eine individuelle Umfrage aus 6 Snippets mit CI in einer einzigen Sprache besteht. Die Aufgabenform ist in Abbildung 9.1 zu sehen. Oben befindet sich der Timer, welcher ab Beginn der Snippets von 30 Minuten herunter zählt. Danach folgt die Aufgabenstellung in Satzform. Es ist immer ein Input gegeben und der Teilnehmer soll den Output des beistehenden Codesnippets bestimmen. Dieser wird in ein Textfeld am unteren Ende eingegeben. Mit einem Klick auf "Next" gelangt man zum nächsten Snippet. Ein Zurückgehen ist nicht möglich.
Der Input ist je Snippet stets gleich, so das jeder Teilnehmer zu einem bestimmten Snippet den gleichen Input bekommt. So ist die Bearbeitungszeit und Korrektheit

[2] https://www.soscisurvey.de/

zwischen den Teilnehmern vergleichbar, da jeder den gleichen Lösungsweg haben sollte. Die Eingaben und erwarteten Ausgaben sind wie folgt.

Snippet	Input	Output
AA	[1,7,13,9,20]	10
BD	1011010	90
CS	59451	24
FA	10	3628800 (= 10!)
GD	(12, 15)	3
IP	37	True

Table 5.1: Input und Output je Snippet

Zu Beginn des Befragungszeitraumes begann an der TUC die Prüfungszeit, gefolgt von den Semesterferien bis Ende September. Folglich haben nicht viele Studenten an der Studie teilgenommen. Neben den Interessenten aus der Google Umfrage (Abschnitt 5.1) haben eine Handvoll Studenten aus meinem Freundeskreis teilgenommen. Da die geringe Datenmenge nicht aussagekräftig war, wurde der Umfragelink Zu Beginn des Semesters erneut verteilt. Darunter in einem Masterkurs von Frau Hartmann, welcher am 20.10.2022 ca. 200 neue Datensätze lieferte.
Am 23.10.2022 wurde die Umfrage beendet.

6 Ergebnisse

Im folgenden Kapitel werden die Ergebnisse der Umfrage präsentiert. Die Interpretation dieser folgt dann im Kapitel 7.

6.1 Datenvorbereitung

Die erhobenen Daten wurden im xlsx-Format herunterladen und anschließend in Microsoft Excel bearbeitet. Insgesamt waren 155 Datensätze vorhanden. Diese enthielten jedoch unzählige Testversuche, sofort abgebrochene Versuche oder Mehrfachversuche der gleichen Person. Diese wurden aussortiert und es blieben 89 Datensätze übrig.
Anschließend wurde ein zweites Outlier Removal durchgeführt, welches sich auf unangemessene Antworten und absichtliches Überspringen von Fragen spezialisiert. Dabei wurden 18 Datensätze entfernt. Einer davon hat unangemessene Begriffe geschrieben als ein arabischen Snippet präsentiert wurde, ein weiterer Teilnehmer hat absichtlich falsche Ergebnisse angegeben, zum Beispiel eine Zahl im Snippet IP, welches einen Boolean zurück gibt. Alle weiteren Outlier haben nur 1-2 Snippets beantwortet und danach abgebrochen. Es blieben 71 verwertbare Datensätze übrig.
Schlussendlich mussten die individuellen Antworten der Teilnehmer normiert werden. Da die Codeverständnisaufgaben mit einem Textfeld beantwortet wurden, gab es viele verschiedene Antworten. Alle korrekten Antworten werden mit der Zahl 1 ersetzt, alle Falschen mit 0.
Leider ist im Codesnippet GD mit CI in deutscher Sprache ein Syntaxfehler im Code, welche in der Pilotierung nicht aufgefallen ist. Im return Statement ist der Variablenname auf Englisch. Deswegen werden die Antworten dieser Frage nicht gewertet.
Für die weitere Betrachtung wurden die Teilnehmer zusätzlich in 3 Gruppen geteilt. Je nach Spracherfahrung, welche im Fragebogen (Abschnitt 4.3.1) angegeben wurde, bekam jeder Teilnehmer eine Gruppe zugeteilt.

L1 Arabischkenntnisse mindestens 2/5

L2 Deutschkenntnisse mindestens 2/5

L3 Englischkenntnisse mindestens 2/5

L4 Rest

Die Gruppe L1 besteht aus 2 Teilnehmern, welche beide Arabisch als Muttersprache gelernt haben. Gruppe L2 besteht aus 28 Teilnehmern, wovon 5 Deutsch als Muttersprache haben. Die Gruppe L3 setzt sich aus 41 Teilnehmern zusammen. Von ihnen hat niemand Englisch als Muttersprache. Die Gruppe L4 ist leer, da jeder Teilnehmer mindestens Grundkenntnisse in Englisch hat.

6.2 Deskriptive Statistik

Im diesem Unterabschnitt werden zuerst die demographischen Daten der Teilnehmer ausgewertet. Anschließend folgt die Auswertung der Codesnippets.

6.2.1 Beschreibung der Teilnehmer

In Tabelle 6.1 ist die Testgruppe dargestellt. Der obere Teil zeigt die gesamte Teilnehmerzahl und den Anteil der Outlier. Danach ist die Verteilung der Studenten zu sehen. Diese bezieht sich auf die Datensätze, welche das Outlier Removal überstanden haben. Neben 2 Bachelorstudenten und 67 Masterstudenten gab es 2 Datensätze ohne Angabe.

Demographische Daten		
Teilnehmer gesamt	89	
davon Outlier	18	
Teilnehmer Rest	71	
Anzahl Bachelorstudenten	2	
Anzahl Masterstudenten	67	
keine Angabe	2	
	Mittelwert	Median
Alter in Jahren	20-25	25-30
Erfahrung mit Softwareentwicklung in Jahren	1-3	1-3
Erfahrung mit Java in Jahren	0-1	0-1
Englisch Kenntnisse	Advanced - Fluently	Fluently
Deutsch Kenntnisse	Novice - Advanced	Novice
Arabisch Kenntnisse	Novice	Novice

Table 6.1: Demographische Daten der Probanden

Im unteren Teil der Tabelle sind Mittelwert und Median von Alter, Erfahrung mit Softwareentwicklung, Erfahrung mit Java, sowie die Einschätzung der Sprachkenntnisse in Englisch, Deutsch und Arabisch zu finden. Die Teilnehmer waren durchschnittlich 20-25 Jahre alt. Der Median lag bei 25-30 Jahren.
Der Mittelwert der Programmiererfahrung lag bei 1-3 Jahren, wobei die Teilnehmer davon durchschnittlich 0-1 Jahre Erfahrung mit Java haben.

Auf einer Skala von 1-5 haben die Teilnehmer ihre Englischkenntnisse durchschnittlich mit 3,5 (entspricht Advanced - Fluently, siehe Abbildung 9.5) bewertet. Der Median lag bei 4 (Fluently).
Im Gegensatz dazu wurden die Deutschkenntnisse durchschnittlich mit 1-2 (Novice - Advanced) bewertet. Der Median lag bei 1 (Novice).
Ähnlich dazu war der Mittelwert und Median der Arabischkenntnisse 1 (Novice). Dies liegt jedoch daran, das es nur zwei Teilnehmer gab, welche Arabisch verstehen. In beiden Fällen war es die Muttersprache.

6.2.2 Codesnippets

Im folgenden Teil werden die allgemeinen Ergebnisse der Codeaufgaben präsentiert. In der Tabelle 6.2 ist die Gesamtzahl der Antworten und durchschnittliche Korrektheit jeder Aufgabe zu sehen.

Korrektheit und Anzahl der Antworten

Snippet	Anzahl der Antworten	Korrektheit
AA Englisch	26	53,84%
AA Deutsch	26	42,31%
AA Arabisch	16	25%
BD Englisch	24	37,5%
BD Deutsch	17	47,06%
BD Arabisch	13	23,08%
CS Englisch	23	43,48%
CS Deutsch	25	44%
CS Arabisch	11	45,45%
FA Englisch	14	64,29%
FA Deutsch	22	40,91%
FA Arabisch	18	22,22%
GD Englisch	18	50%
GD Deutsch	0	0%
GD Arabisch	9	44,44%
IP Englisch	11	54,55%
IP Deutsch	14	78,57%
IP Arabisch	19	36,84%

Table 6.2: Anzahl der Antworten und Korrektheit je Snippet

Es ist deutlich erkennbar das trotz der zufälligen Verteilung der Snippets (siehe Abschnitt 5.3) ausreichend Datensätze zu jeder Aufgabe entstanden sind. Um die Ergebnisse besser zu visualisieren wurden zwei Diagramme erstellt.
Zum Einen wurden die Snippets in Abbildung 6.1 nach Algorithmus zusammenfasst.

Außerdem sind die Snippets so sortiert, wie sie in der Studie bearbeitet wurden. Zum Anderen ist in Abbildung 6.2 nach Sprache zusammengefasst. Wichtig für beide Diagramme ist das GD in der Sprache Deutsch nicht beachtet wurde (siehe Abschnitt 6.1).

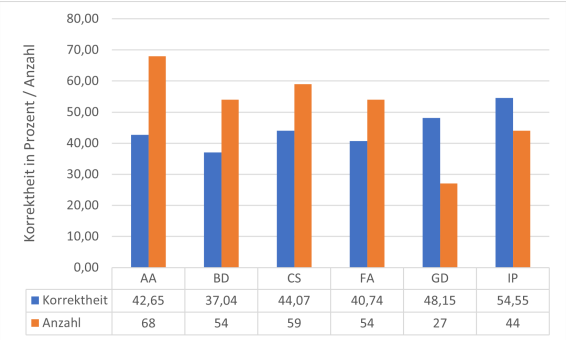


Abbildung 6.1: Korrektheit und Anzahl der Teilnehmer pro Snippet

In Abbildung 6.1 ist deutlich erkennbar das die Anzahl an Antworten mit zunehmenden Fortschritt der Umfrage abnimmt. Dafür steigt die Korrektheit, besonders bei den letzten beiden Codesnippets, deutlich an. Außerdem wurde kein Algorithmus von jedem Teilnehmer gelöst. Am meisten wurde AA bearbeitet (68 von 71 Teilnehmer), da dies das erste Snippet war und fast jeder noch zielstrebig an der Umfrage teilgenommen hat. Am wenigsten wurde das letzte Snippet IP gelöst. Dieses wurde nur von 44 der 71 Teilnehmer bearbeitet.

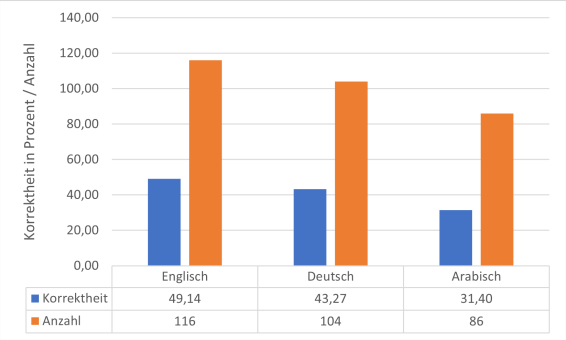


Abbildung 6.2: Korrektheit und Anzahl der Teilnehmer je Sprache

In Abbildung 6.2 erkennt man das die Sprachen trotz der Randomisierung einigermaßen gleich verteilt sind. Es wurden 116 Englische, 104 Deutsche und 86 Arabische Snippets bearbeitet. Die Zahl letzterer ist bedeutend geringer, da viele Teilnehmer laut ihrem eigenen Feedback mit diesen nicht klar gekommen sind und daher auch keine Antwort angegeben haben. Außerdem sieht man das die englischen Snippets mit durchschnittlich 49,14% am korrektesten beantwortet wurden. Die deutschen

Snippets wurden 43,27% und die arabischen Snippets 31,4% korrekt beantwortet.

Beantwortungszeit

In der folgenden Tabelle ist die durchschnittliche Beantwortungszeit und der Median der Beantwortungszeit je Aufgabe dargestellt.

Snippet	durchschnittliche Beantwortungszeit	Median der Beantwortungszeit
AA Englisch	302s	233s
AA Deutsch	349s	301s
AA Arabisch	190s	142s
BD Englisch	336s	205s
BD Deutsch	394s	262s
BD Arabisch	396s	265s
CS Englisch	201s	131s
CS Deutsch	171s	154s
CS Arabisch	200s	167s
FA Englisch	118s	107s
FA Deutsch	104s	83s
FA Arabisch	71s	65s
GD Englisch	179s	147s
GD Deutsch	0	0
GD Arabisch	91s	9s
IP Englisch	60s	44s
IP Deutsch	94s	75s
IP Arabisch	83s	50s
gesamt	1184s	1097s

Table 6.3: Anzahl der Antworten und Korrektheit je Snippet

Die letzte Zeile der Tabelle zeigt den Mittelwert und Median der Gesamtdauer der Umfrage. Diese beziehen sich jedoch auf die gesamte Befragung, inklusive demographischer Fragen und Beispielaufgabe. Durchschnittlich wurden 1184 Sekunden für die Umfrage benötigt. Dies entspricht 19 Minuten und 44 Sekunden.
Die Zeiten wurden ebenfalls in zwei Diagramme zusammengefasst.

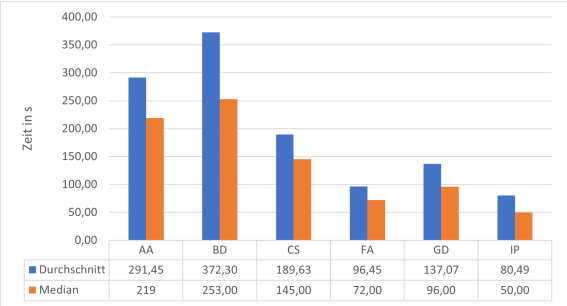


Abbildung 6.3: Durchschnitt und Median der Zeit pro Snippet

In Abbildung 6.3 sind der Mittelwert und Median der Bearbeitungszeit nach Algorithmus zusammengefasst. Es ist deutlich erkennbar, das die Beantwortungszeit mit zunehmendem Fortschritt der Bearbeitung abnimmt. Ausnahmen dafür sind die Snippets BD und GD, welche Snippets der Kategorie 3 sind (siehe Abschnitt 4.3.2) und deswegen durchschnittlich mehr Zeit benötigen als andere Codesnippets. Grund dafür ist der gleiche wie bei Korrektheit und Anzahl. Die Teilnehmer lösen Algorithmen mit zunehmender Bearbeitungsdauer nicht mehr gewissenhaft und tendieren zm schnellen Überspringen bzw. Aufgeben.

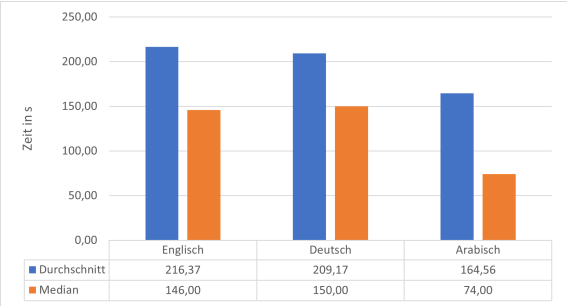


Abbildung 6.4: Durchschnitt und Median der Zeit je Sprache

In Abbildung 6.4 ist der Mittelwert und Median der Bearbeitungszeit je Sprache zusammengefasst. Englische und Deutsche Codesnippets wurden durchschnittlich ähnlich schnell beantwortet. Arabische Snippets wurden hingegen deutlich schneller beantwortet. Dafür haben Teilnehmer diese deutlich weniger korrekt beantwortet (vgl. Abbildung 6.2). Da nur 2 der 71 Teilnehmer Arabisch verstehen (siehe Abschnitt 6.2.1) ist naheliegend, das arabische Codesnippets tendenziell mit weniger Engagement beantwortet wurden.

6.2.3 Codesnippets Gruppe L1 - Arabisch

Im folgenden Abschnitt werden Diagramme der Gruppe L1 dargestellt.

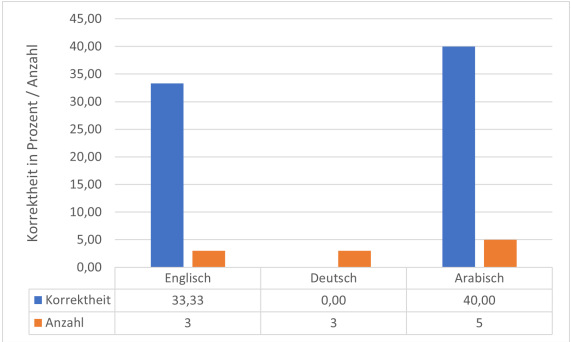


Abbildung 6.5: Korrektheit und Anzahl der Teilnehmer je Sprache der Gruppe L1 - Arabisch

Abbildung 6.5 zeigt die durchschnittliche Korrektheit und Anzahl der Antworten je Sprache der Gruppe L1. Analog dazu ist in Abbildung 9.13 im Anhang die Anzahl und Korrektheit je Snippet zu sehen. Die Einteilung nach Snippet ist für die weitere Betrachtung jedoch irrelevant.

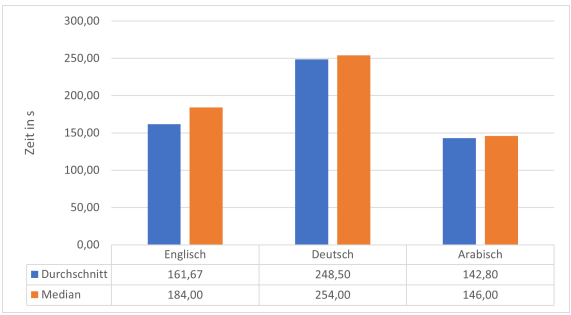


Abbildung 6.6: Durchschnitt und Median der Zeit je Sprache der Gruppe L1 - Arabisch

In Abbildung 6.6 ist der Mittelwert und Median der Beantwortungszeit je Sprache für die Gruppe L1 dargestellt. Die Durchschnittszeit und Median der Beantwortungszeit je Snippet sind im Anhang auf Abbindung 9.14 zu sehen.

6.2.4 Codesnippets Gruppe L2 - Deutsch

Im folgenden Abschnitt werden Diagramme der Gruppe L2 dargestellt.

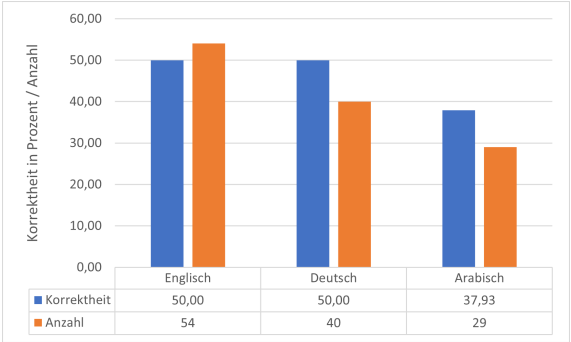


Abbildung 6.7: Korrektheit und Anzahl der Teilnehmer je Sprache der Gruppe L2 - Deutsch

Abbildung 6.7 zeigt die durchschnittliche Korrektheit und Anzahl der Antworten je Sprache der Gruppe L2. Analog dazu ist in Abbildung 9.15 im Anhang die Anzahl und Korrektheit je Snippet zu sehen. Die Einteilung nach Snippet ist für die weitere Betrachtung jedoch irrelevant.

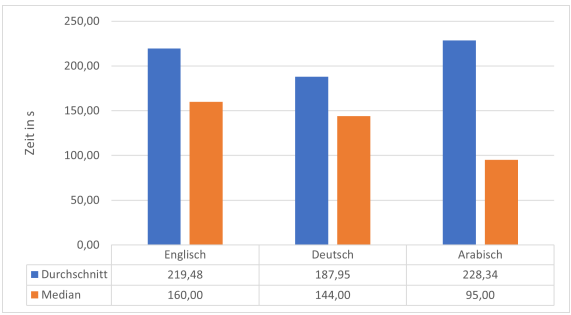


Abbildung 6.8: Durchschnitt und Median der Zeit je Sprache der Gruppe L2 - Deutsch

In Abbildung 6.8 ist der Mittelwert und Median der Beantwortungszeit je Sprache für die Gruppe L2 dargestellt. Die Durchschnittszeit und Median der Beantwortungszeit je Snippet sind im Anhang auf Abbindung 9.16 zu sehen.

6.2.5 Codesnippets Gruppe L3 - Englisch

Im folgenden Abschnitt werden Diagramme der Gruppe L3 dargestellt.

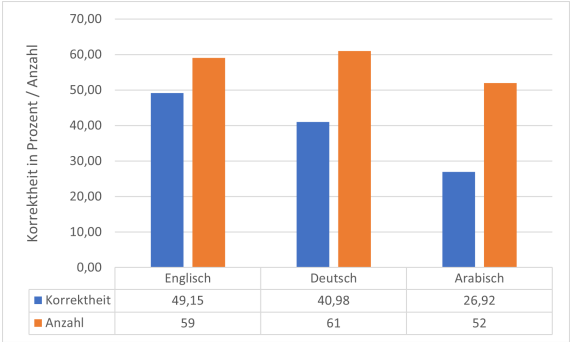


Abbildung 6.9: Korrektheit und Anzahl der Teilnehmer je Sprache der Gruppe L3 - Englisch

Abbildung 6.9 zeigt die durchschnittliche Korrektheit und Anzahl der Antworten je Sprache der Gruppe L3. Analog dazu ist in Abbildung 9.17 im Anhang die Anzahl und Korrektheit je Snippet zu sehen. Die Einteilung nach Snippet ist für die weitere Betrachtung jedoch irrelevant.

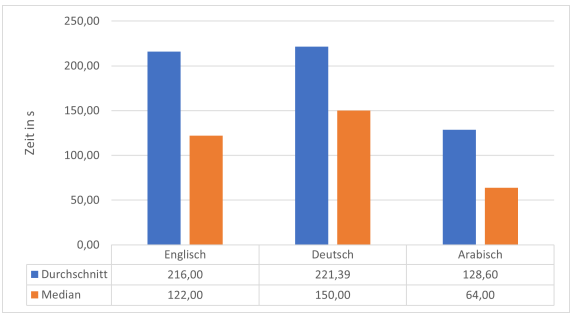


Abbildung 6.10: Durchschnitt und Median der Zeit je Sprache der Gruppe L3 - Englisch

In Abbildung 6.10 ist der Mittelwert und Median der Beantwortungszeit je Sprache für die Gruppe L3 dargestellt. Die Durchschnittszeit und Median der Beantwortungszeit je Snippet sind im Anhang auf Abbindung 9.18 zu sehen.

Da die Gruppe L1 nur 2 Teilnehmer umfasst, sind die dazugehörigen Diagramme nicht allgemeingültig und stellen nur eine Tendenz dar. Die anderen beiden Gruppen L2 und L3 hingegen haben ausreichend Mitglieder um aussagekräftige Schlussfolgerungen zu ziehen.

7 Diskussion

Im folgenden Kapitel werden die Ergebnisse aus Kapitel 6 interpretiert.

7.1 Prüfung der Hypothesen

Zunächst werden die Hypothesen geprüft.

Beantwortungszeit

Die erste Hypothese lauten wie folgt:

H_1 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese schneller beantwortet als Codeprobleme mit CI in einer anderen Sprache.

In der Gruppe L1 - Arabisch (Abschnitt 6.2.3) wurden Codesnippets mit arabischen CI durchschnittlich ca. 20 Sekunden schneller als englische CI und ca. 100 Sekunden schneller als deutsche CI beantwortet (Abbildung 6.6). Da jedoch die Datenmenge sehr gering war (Abbildung 6.5) sind diese Ergebnisse mit Vorsicht zu betrachten.
Die deutschsprachige Gruppe L2 (Abschnitt 6.2.4) hat ebenfalls Snippets mit ihrer vertrautesten Sprache (Deutsch) am schnellsten beantworten können (Abbildung 6.8).
Teilnehmer der Gruppe L3 (Abschnitt 6.2.5) haben arabische Snippets mit Abstand am schnellsten beantwortet. Dies kann daran liegen das viele von den arabischen Schriftzeichen überrumpelt wurden und die Frage schnell übersprungen haben. Das würde auch den deutlich geringeren Medianwert von 64 Sekunden erklären. Zudem ist die Korrektheit der arabischen Snippets nur bei ca. 27% gewesen (Abbildung 6.9). Im individuellen Feedback, welches jeder Teilnehmer geben konnte, haben auch sehr viele angegeben, das die arabischen Codesnippets sehr schwierig waren. Englische Snippets wurden in der Gruppe L3 durchschnittlich 5 Sekunden schneller beantwortet als Deutsche. Außerdem ist der Median bei den englischen Snippets um 28 Sekunden geringer.
Da in den ersten beiden Gruppen Codesnippets mit der vertrautesten Sprache am schnellsten beantwortet wurden und in Gruppe L3 die englischen Snippets augenscheinlich konsequenter gelöst wurden im Vergleich zu den Arabischen, gilt die Hypothese 1 als bestätigt. Jedoch sollten die Ergebnisse durch weitere Versuche mit einer geeigneten, neuen Testgruppe geprüft werden.

Die zweite Hypothese lautet:

H_3 Teilnehmer werden Codeprobleme mit englischen CI schneller beantworten als Codeprobleme mit fremdsprachigen CI.

In der Gruppe L1 ist Deutsch die Fremdsprache. Codesnippets mit englischen CI wurden durchschnittlich 87 Sekunden schneller beantwortet als Snippets mit deutschen CI. Auch der Median ist geringer (Abbildung 6.6).
Die Gruppe L2 hat Arabisch als Fremdsprache. Dort wurden englische Snippets durchschnittlich etwas schneller beantwortet als Arabische. Dafür ist der Median der arabischen Codesnippets geringer (Abbildung 6.8). Grund dafür könnte analog zur vorherigen Hypothese die fehlende Bereitschaft sein Codesnippets mit einem fremden Schriftsystem zu lösen.
Die Ergebnisse der Gruppe L3 sind identisch zu denen der vorherigen Hypothese, da sowohl Deutsch als auch arabisch als Fremdsprache zählen.
Folglich ist die Hypothese H_3 nur für die Gruppe L1 bestätigt. Da diese jedoch nur eine geringe Datenmenge hat, wird die Hypothese insgesamt nicht bestätigt.

Die dritte Hypothese lautet:

H_5 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem schneller beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

In Gruppe L1 sind keine unbekannten Schreibsysteme vorhanden, da jeder Teilnehmer mindestens grundlegende Englischkenntnisse besitzt (vgl. Abschnitt 6.1).
In Gruppe L2 und L3 ist das fremde Schriftsystem jeweils Arabisch. Die bekannten Schriftsysteme sind Deutsch und Englisch. In L2 wurden diese durchschnittlich schneller beantwortet als die arabischen Codesnippets. In L3 hingegen haben sie durchschnittlich länger gebraucht.
Die Datenlage ist insgesamt zu unsicher um die Hypothese zu bestätigen oder abzulehnen. Man bräuchte eine vierte Sprache, welche ein fremdes Schriftsystem für Gruppe L1 darstellt, zum Beispiel Hindi. Im Zuge dessen müssten für die erste Gruppe mehr Teilnehmer gesucht werden.

Korrektheit

Die erste Hypothese lautet:

H_2 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese eher korrekt beantwortet als Codeprobleme mit CI in einer anderen Sprache.

In Gruppe L1 ist Arabisch die vertrauteste Sprache. In Abbildung 6.5 ist zu sehen, das Codesnippets in dieser auch durchschnittlich am korrektesten beantwortet wurden. Jedoch ist die Anzahl an Datensätzen sehr gering. Interessant ist jedoch das kein einziges deutsches Snippet korrekt beantwortet wurde.
In Gruppe L2 wurden deutsche und englische Codesnippets beide zu 50% korrekt

beantwortet (Abbildung 6.7). Quellcode mit arabischen CI wurde nur mit 37,93% Korrektheit beantwortet.
In Gruppe L3 wurden englische Snippets deutlich korrekter beantwortet als Anderssprachige (Abbildung 6.9).
Folglich ist die Hypothese nur für englischsprachige Teilnehmer bestätigt. Deutschsprachige Probanden können womöglich mit englischen und deutschen CI ähnlich gut umgehen.

Die zweite Hypothese lautet:

H_4 Teilnehmer werden Codeprobleme mit englischen CI eher korrekt beantworten als Codeprobleme mit fremdsprachigen CI.

Laut Abbildung 6.2 ist diese Annahme korrekt, da 49,14% der englischen Snippets korrekt beantwortet wurden. Im Vergleich dazu wurden Deutsche nur 43,27% und Arabische 31,4% korrekt beantwortet. Sieht man sich die einzelnen Gruppen konkret an, wird diese Annahme ebenfalls bestätigt.
In Gruppe L1 (Abbildung 6.5) wurden englische Snippets durchschnittlich korrekter beantwortet als deutsche Codesnippets. Das Gleiche gilt für L3. In Abbildung 6.9 ist zu sehen das Englisch mit Abstand am korrektesten beantwortet wurde.
Die einzige Ausnahme ist L2 (Abbildung 6.7). Dort wurde Quellcode mit englischen und deutschen CI jeweils zu 50% korrekt beantwortet. Dies widerlegt jedoch die Hypothese nicht.
Folglich wird die Hypothese H_4 als bestätigt angesehen.

Die dritte Hypothese lautet:

H_6 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem eher korrekt beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

In Gruppe L1 gibt es, wie bereits bei H_5 erwähnt, keine Sprache mit einem fremden Schriftsystem.
In Gruppe L2 und L3 ist Arabisch das fremde Schriftsystem. In beiden Fällen wurden sowohl englische als auch deutsch Codesnippets korrekter beantwortet.
Folglich ist die Hypothese bestätigt.

Zusammenfassung

Folgende Hypothesen wurden bestätigt:

H_1 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese schneller beantwortet als Codeprobleme mit CI in einer anderen Sprache.

H_4 Teilnehmer werden Codeprobleme mit englischen CI eher korrekt beantworten als Codeprobleme mit fremdsprachigen CI.

H_6 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem eher korrekt beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

Folgende Hypothesen konnten nur teilweise bestätigt werden:

H_3 Teilnehmer werden Codeprobleme mit englischen CI schneller beantworten als Codeprobleme mit fremdsprachigen CI.

H_2 Wenn der Teilnehmer Codeprobleme mit CI in seiner vertrautesten Sprache löst, dann werden diese eher korrekt beantwortet als Codeprobleme mit CI in einer anderen Sprache.

Folgende Hypothese konnte nicht bestätigt werden:

H_5 Teilnehmer werden Codeprobleme mit CI in einem bekannten Schriftsystem schneller beantworten als Codeprobleme mit CI in einem fremden Schriftsystem.

7.2 Beantwortung der Forschungsfragen

Im folgenden Abschnitt sollen die Forschungsfragen beantwortet werden.

Forschungsfrage 1 Welchen Effekt haben CI in verschiedenen natürlichen Sprachen auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

Generell konnte kein allgemeiner Effekt festgestellt werden. Je nach Spracherfahrung des Teilnehmers kam es zu unterschiedlichen Ergebnissen. Während bei deutschsprachigen Probanden kein wesentlicher Unterschied in der Korrektheit und Beantwortungszeit zwischen englischen und deutschen CI festgestellt werden konnte (Abbildungen 6.7 und 6.8), wurden allgemein englische Snippets korrekter und schneller beantwortet als Deutsche (Abbildungen 6.2 und 6.4). Die Arabische Testgruppe hatte zu wenig Teilnehmer um bewiesene Schlussfolgerungen zu ziehen.
Über die Aussagekraft der Ergebnisse lässt sich streiten. Wie in Abbildung 6.3 zu sehen ist, wurden mit zunehmender Studienlänge die Aufgaben immer schneller beantwortet. Außerdem nahm die Gesamtzahl an Antworten mit der Zeit ab (Abbildung 6.1). Folglich war die Studie für einige Teilnehmer zu lang, was auch mehrmals im individuellen Feedback erwähnt wurde.

Forschungsfrage 2 Hat das Schriftsystem der CI einen Effekt auf die Korrektheit und Beantwortungszeit von Programmieranfängern in Codeaufgaben?

Das Schriftsystem hat einen bedeutenden Einfluss auf die Codeverständlichkeit. CI in einem fremden Schriftsystem werden deutlich inkorrekter und teilweise auch

langsamer beantwortet. Diese sind vergleichbar mit kontraintuitiven Variablennamen. Folglich müssen Programmieren den Code mit der Bottom-Up Methode bearbeiten, was wesentlich zeitintensiver ist (2.1.3). Außerdem führen CI in fremden Schriftsystemen häufig zu "don't know" Antworten oder Abbrüchen, ähnlich wie bei zu langen Studiendesigns (2.2).

7.3 Refelektion

Nach der Auswertung der Daten sind einige positive und negative Dinge zu erwähnen.

7.3.1 Studienteilnehmer

Die Auswahl an Studienteilnehmer war für das entworfene Studiendesign eher Nachteilhaft. Da das Experiment online durchgeführt werden musste und keine direkte Zielgruppe erstellt wurden konnte, musste das Studiendesign erstellt werden ohne die genaue Teilnehmergruppe zu kennen. Folglich war Arabisch als 3. Sprache eher suboptimal. Es haben sich leider nur 2 arabische Studenten gefunden, dafür unzählige mit indischer Abstammung. Diese sprachen zwar viele unterschiedliche Sprachen wie Hindi, Urdu, Tamil oder Bengali, jedoch hätte man so mindestens 10 Teilnehmer in der Gruppe L1 gehabt.

7.3.2 Studiendesign

Das Studiendesign war allgemein gut. Es gab jedoch zwei negative Punkte: die Studienlänge und Durchführung.
Das Experiment war mit 25-30 Minuten Zeitaufwand und 6 verschiedenen Codesnippets bedeutend zu lang. Wie man in Abbildung 6.3 sehen kann, wurde die durchschnittliche Beantwortungszeit nach dem dritten Snippet sehr kurz. Dies liegt daran das viele Teilnehmer danach keine Lust mehr hatten und die folgenden Snippets einfach übersprungen haben. In Abbildung 6.1 kann man auch die fallende Beteiligung mit steigender Studiendauer sehen. Einige Teilnehmer haben auch im Feedback erwähnt das die Studie zeitlich zu aufwendig war.
Im Nachhinein und anhand der beiden oben erwähnten Abbildungen wäre eine Studiendauer von 10-15 Minuten mit 2-3 Snippets optimal gewesen. Dies korreliert auch mit der optimalen Studienlänge aus Kapitel 2.2. Dann müssten ebenfalls die Snippets auf 3 vermindert werden oder eine weitere Randomisierung stattfinden.
Die Randomisierung nach Sprache hat gut funktioniert. In Abbildung 6.2 sieht man das die Gesamtanzahl an Antworten je Sprache einigermaßen gleich ist. Dabei ist der Balken der arabischen Snippets niedriger als der eigentliche Wert. Das liegt daran das nicht beantwortete arabische Snippets nicht in die Statistik aufgenommen wurden.
Nicht so gut funktioniert hat die Durchführung der Studie. Das Experiment hat während der Prüfungszeit gestartet und fand zum Großteil in den Semesterferien

statt. Deswegen haben bis Anfang Oktober nur sehr wenige Studenten teilgenommen. Erst Ende Oktober lagen genügend brauchbare Datensätze vor, wodurch die Auswertung zeitlich knapp wurde. In Zukunft wäre es von Vorteil das Experiment während der Vorlesungszeit durchzuführen, eventuell sogar als Bestandteil eines Kurses ähnlich wie Dominik Gorgosch in seiner Masterarbeit [1].

7.3.3 Studienauswertung

Wie bereits im vorherigen Kapitel erwähnt, blieb zum Ende dieser Arbeit nur wenig Zeit für die Auswertung der Daten. Folglich konnte nur eine einfache, deskriptive Statistik erstellt werden. Für die Zukunft ist eine bessere Planung und strikterer Zeitplan notwendig, damit nach dem Experiment noch genügend Zeit für die Auswertung bleibt.

[1] https://www.tu-chemnitz.de/informatik/ST/lectures/Masters%20Thesis%20Pdfs/Masterarbeit_Dominik_Gorgosch.pdf

8 Fazit und Ausblick

Im folgenden Kapitel wird ein Fazit der vorliegenden Arbeit gezogen. Anschließend wird ein Ausblick für weiterführende Studien gegeben.

8.1 Fazit

In der vorliegenden Arbeit wurde ersichtlich das CI in verschiedenen natürlichen Sprachen einen gewissen positiven Einfluss auf das Codeverständnis haben können. Dabei wurde die Korrektheit und Bearbeitungszeit von 6 verschiedenen Java Snippets mit CI in Deutsch, Englisch und Arabisch betrachtet. Eine Studie mit 71 Teilnehmern zeigte das CI in der Muttersprache des Programmierers gleich oder etwas besser verstanden werden als CI in Englisch. Eindeutig ist jedoch, dass CI in fremden Sprachen sehr oft zu Unklarheiten, schlechteren und langsameren Ergebnissen oder sogar dem kompletten Überspringen von Aufgaben führen. Folglich gibt es keinen signifikanten Vorteil CI in einer anderen Sprache als Englisch zu verwenden.

8.2 Ausblick

Weitere Studien können an die Ergebnisse dieser Arbeit anknüpfen. Zum Einen ist eine kontrollierte und ausgewählte Testgruppe notwendig. Dadurch können Codesnippetts und Sprachen vor dem Experiment auf die Zielgruppe angepasst werden. Außerdem sollte die Studie mit verschiedenen Programmiersprachen wiederholt werden.
Weiterhin wäre es interessant wie komplette Programmieranfänger auf einfachere Codesnippets reagieren. Analog dazu kann die Studie auch mit erfahrenen Programmierern mit jahrelanger Programmiererfahrung durchgeführt werden.
Als weitere Bewertungskriterien können ein Eye-Tracker oder EEG hinzugezogen werden. Dafür muss die Studie jedoch in Präsenz stattfinden. Im Zuge dessen kann die Think-Aloud Methode (Kapitel 2.1.1) genutzt werden um den Denkprozess jedes Teilnehmers nachvollziehen zu können.

9 Anhang

9.1 Überblick Informatikstudenten der TUC

Staat	Anzahl
Afghanistan	7
Ägypten	20
Albanien	4
Arabische Republ.Syrien	10
Aserbaidschan	7
Bahrain	1
Bangladesch	128
Brasilien	3
China (VR)	29
Deutschland	375
Frankreich	1
Georgien	1
Ghana	2
Griechenland	1
Indien	301
Indonesien	1
Irak	3
Iran	43
Israel	1
Italien	1
Jemen	3
Jordanien	2
Kamerun	1
Kolumbien	4
Korea, Republik	5
Libanon	4
Marokko	5
Mongolei	2
Nepal	11
Nigeria	11
Nordmazedonien	1
Österreich	1
Pakistan, Islamische Rep.	103
Palästinens. Gebiete	3
Polen	2
Republik Moldau	1
Russische Föderation	12
Schweiz	1
Sri Lanka	3
Staatenlos	3
Taiwan	4
Thailand	2
Tschechische Republik	2
Tunesien	4
Türkei	10
Ukraine	2
Ungeklärt	2
Ver. Staaten von Amerika	2
Vereinigtes Königreich	1
Vietnam	3
Gesamtergebnis	1149

Table 9.1: Statistik des Prüfungsamtes der TUC, Herkunftsländer der Informatikstudenten, Stand 15.03.2022

9.2 SoSci: demographische Daten und Beispielaufgabe

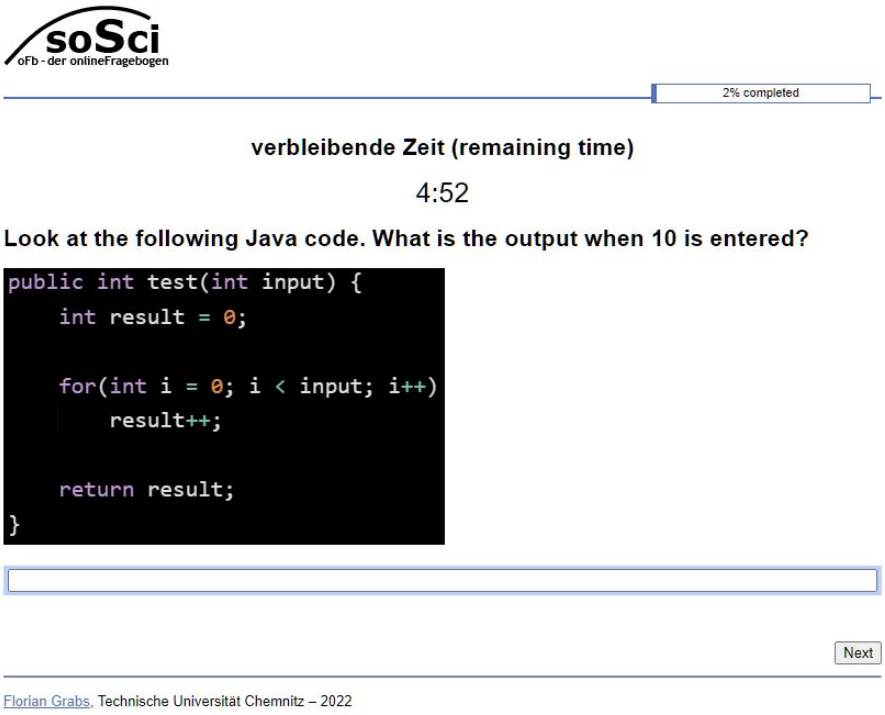


Abbildung 9.1: SoSci, Beispielaufgabe

Abbildung 9.2: SoSci demographische Frage, Java Verständnis

Abbildung 9.3: SoSci demographische Frage, Programmiererfahrung

Abbildung 9.4: SoSci demographische Frage, Herkunft

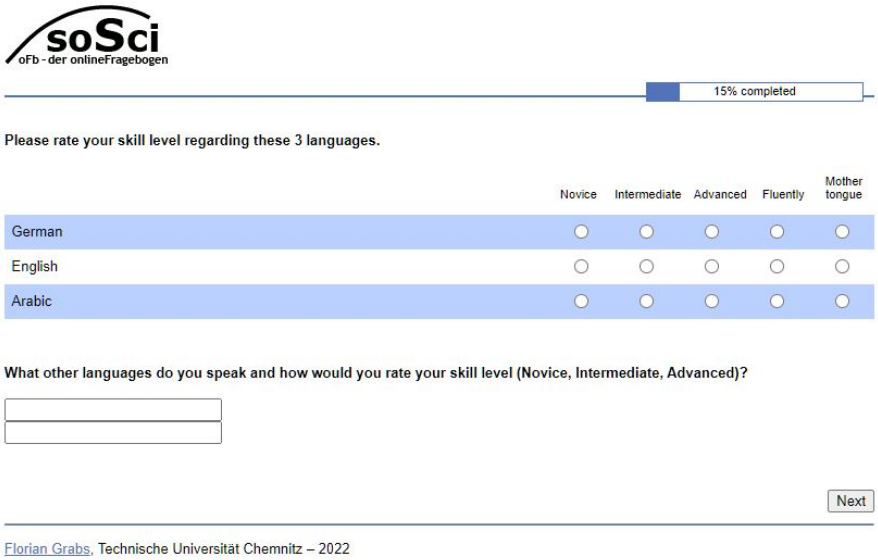


Abbildung 9.5: SoSci demographische Frage, Sprachverständnis Deutsch, Englisch, Arabisch

Abbildung 9.6: SoSci demographische Frage, Alter

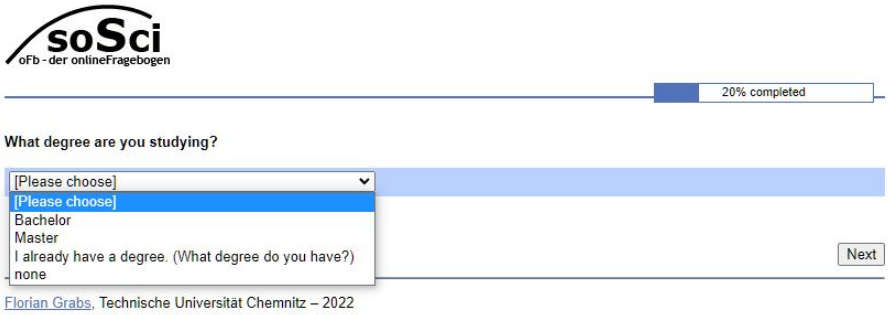


Abbildung 9.7: SoSci demographische Frage, Abschluss

9.3 Ergebnisse Google Umfrage

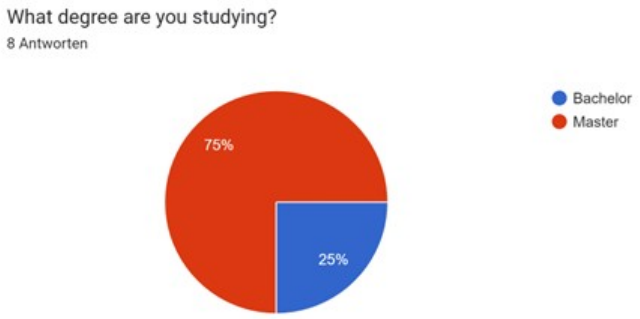


Abbildung 9.8: Google Umfrage, Studiengang

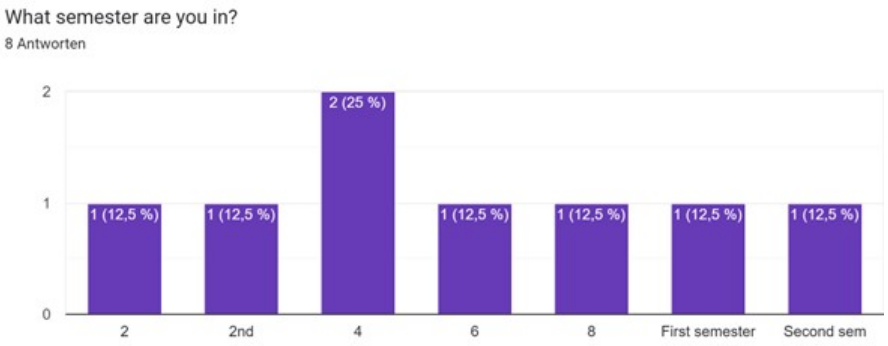


Abbildung 9.9: Google Umfrage, Semester

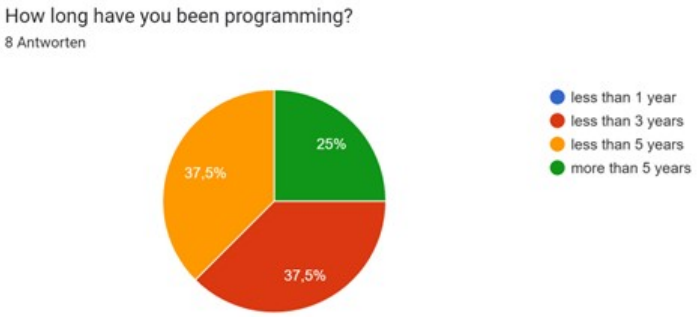


Abbildung 9.10: Google Umfrage, Programmiererfahrung

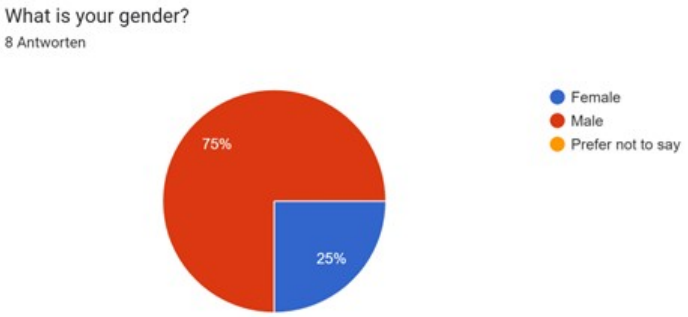


Abbildung 9.11: Google Umfrage, Geschlecht

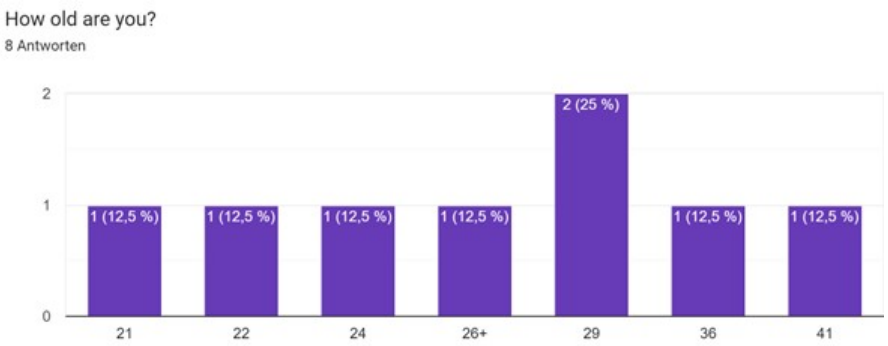


Abbildung 9.12: Google Umfrage, Alter

9.4 Diagramme L1 - Arabisch

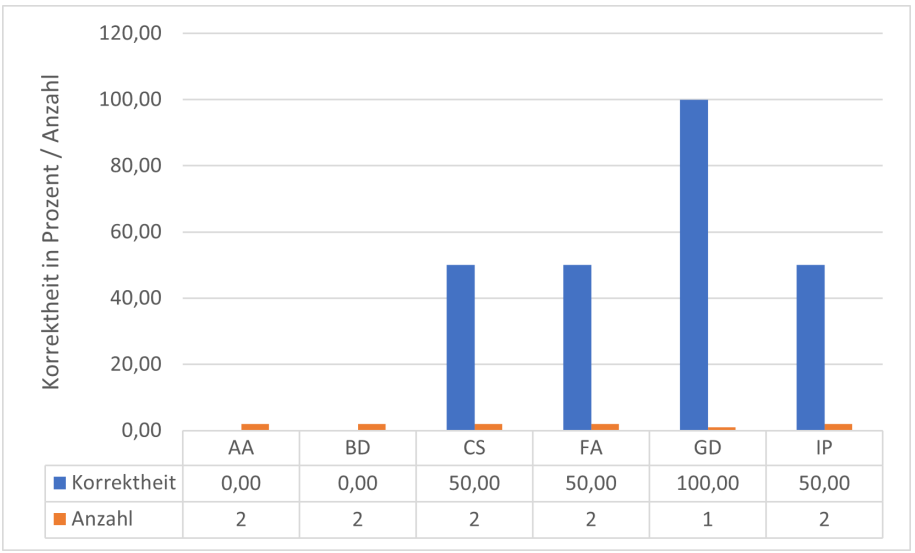
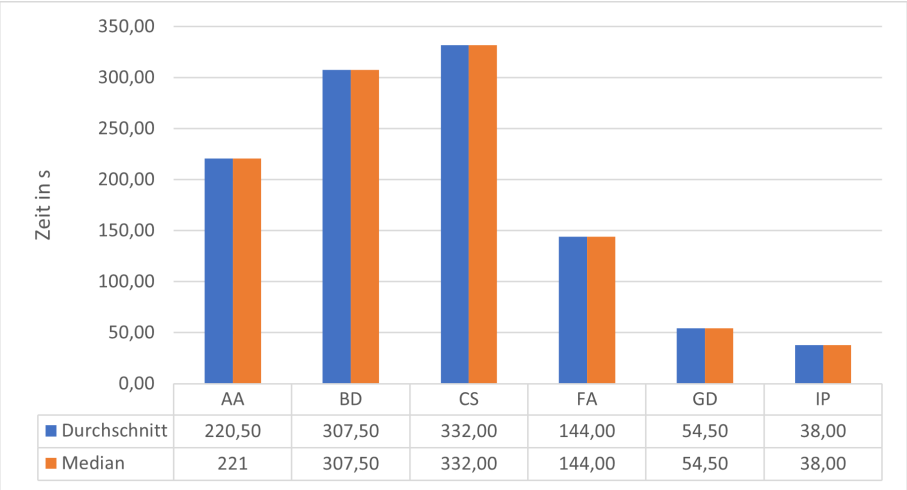


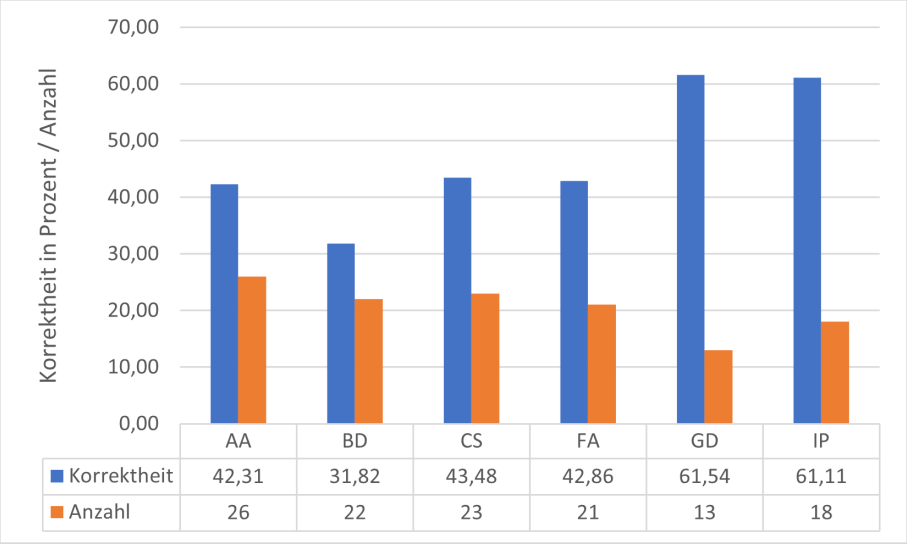
Abbildung 9.13: Anzahl und Korrektheit pro Snippet der Gruppe L1 - Arabisch



	AA	BD	CS	FA	GD	IP
Durchschnitt	220,50	307,50	332,00	144,00	54,50	38,00
Median	221	307,50	332,00	144,00	54,50	38,00

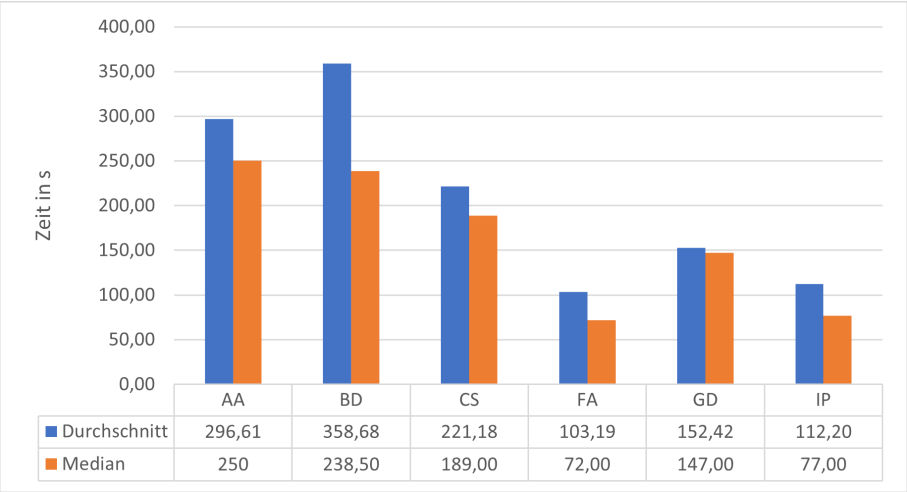
Abbildung 9.14: Durchschnitt und Median der Zeit pro Snippet der Gruppe L1 - Arabisch

9.5 Diagramme L2 - Deutsch



	AA	BD	CS	FA	GD	IP
Korrektheit	42,31	31,82	43,48	42,86	61,54	61,11
Anzahl	26	22	23	21	13	18

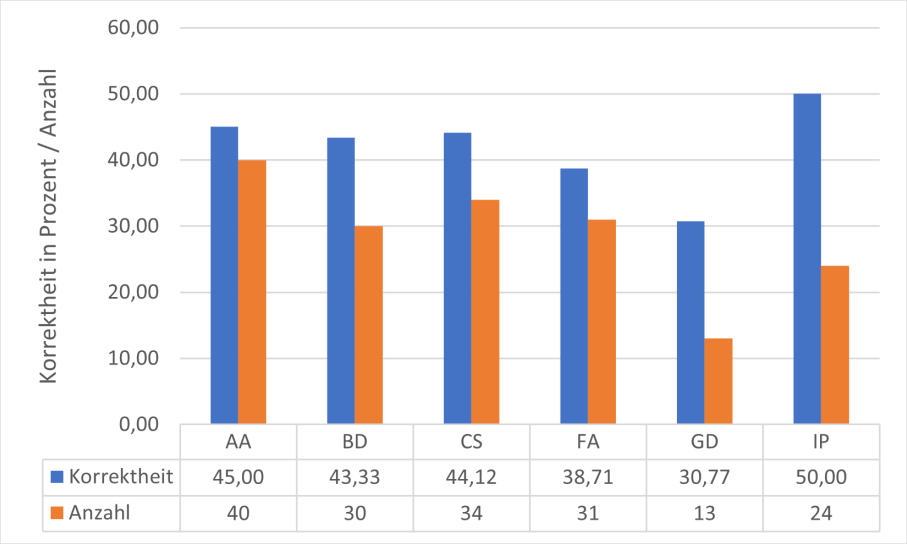
Abbildung 9.15: Anzahl und Korrektheit pro Snippet der Gruppe L2 - Deutsch



	AA	BD	CS	FA	GD	IP
Durchschnitt	296,61	358,68	221,18	103,19	152,42	112,20
Median	250	238,50	189,00	72,00	147,00	77,00

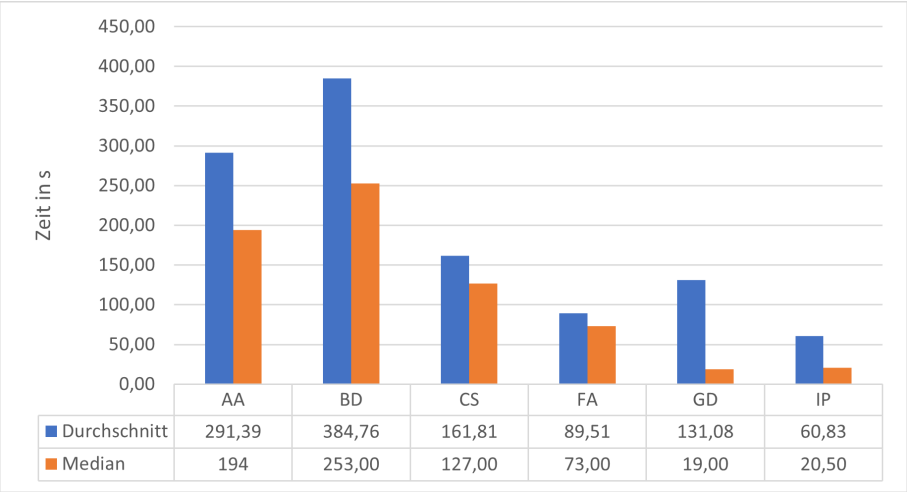
Abbildung 9.16: Durchschnitt und Median der Zeit pro Snippet der Gruppe L2 - Deutsch

9.6 Diagramme L3 - Englisch



	AA	BD	CS	FA	GD	IP
Korrektheit	45,00	43,33	44,12	38,71	30,77	50,00
Anzahl	40	30	34	31	13	24

Abbildung 9.17: Anzahl und Korrektheit pro Snippet der Gruppe L3 - Englisch



	AA	BD	CS	FA	GD	IP
Durchschnitt	291,39	384,76	161,81	89,51	131,08	60,83
Median	194	253,00	127,00	73,00	19,00	20,50

Abbildung 9.18: Durchschnitt und Median der Zeit pro Snippet der Gruppe L3 - Englisch

Literaturverzeichnis

[1] Beniamini, G., Gingichashvili, S., Orbach, A.K., Feitelson, D.G.: Meaningful identifier names: The case of single-letter variables pp. 45–54 (2017)

[2] Brooks, R.: Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies 18(6), 543–554 (1983), https://www.sciencedirect.com/science/article/pii/S0020737383800315

[3] Dawn, L.: Effective identifier names for comprehension and memory (10 2007), https://link.springer.com/article/10.1007/s11334-007-0031-2

[4] Hermans, F.: Hedy: A gradual language for programming education. In: Proceedings of the 2020 ACM Conference on International Computing Education Research. p. 259–270. ICER '20, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3372782.3406262

[5] Janet Siegmund, J.S.: Confounding Parameters on Program Comprehension: A Literature Survey, https://www.se.cs.uni-saarland.de/publications/docs/SiSchu14.pdf

[6] J.Boysen: Factors affecting computer program comprehension (1977)

[7] Johannes Hofmeister, Janet Siegmund, D.V.H.: Shorter Identifier Names Take Longer to Comprehend https://www.se.cs.uni-saarland.de/publications/docs/HoSeHo17.pdf

[8] Katja Wundermann, M.R.: Arbeitsgemeinschaft robot karol, https://docplayer.org/24226453-Arbeitsgemeinschaft-robot-karol.html

[9] Kluthe, T.: A measurement of programming language comprehension using p-bci: An empirical study on phasic changes in alpha and theta brain waves. Master's thesis, Southern Illinois University Edwardsville, Edwardsville, IL, USA (2014)

[10] MSCoder: Komplexität und qualität von software, https://www.verifysoft.com/de_cmtpp_mscoder.pdf

[11] Nakagawa, T., Kamei, Y., Uwano, H., Monden, A., Matsumoto, K., German, D.M.: Quantifying programmers' mental workload during program comprehension based on cerebral blood flow measurement: A controlled experiment. In:

Companion Proceedings of the 36th International Conference on Software Engineering. p. 448–451. ICSE Companion 2014, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2591062.2591098

[12] Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, Janet Siegmund: Program Comprehension and Code Complexity Metrics: An fMRI Study, https://www.se.cs.uni-saarland.de/publications/docs/PAP+21.pdf

[13] Revilla, M., Höhne, J.K.: How long do respondents think online surveys should be? new evidence from two online panels in germany. International Journal of Market Research 62(5), 538–545 (2020), https://doi.org/10.1177/1470785320943049

[14] Revilla, M., Ochoa, C.: Ideal and maximum length for a web survey. International Journal of Market Research 59(5), 557–565 (2017), https://doi.org/10.2501/IJMR-2017-039

[15] Schankin, A., Berger, A., Holt, D.V., Hofmeister, J.C., Riedel, T., Beigl, M.: Descriptive compound identifier names improve source code comprehension p. 31–40 (2018), https://doi.org/10.1145/3196321.3196332

[16] Shaft, T.M., Vessey, I.: Research report—the relevance of application domain knowledge: The case of computer program comprehension, https://pubsonline.informs.org/doi/10.1287/isre.6.3.286

[17] Sharafi, Z.: A practical guide on conducting eye tracking studies in software engineering (6 2020), https://link.springer.com/article/10.1007/s10664-020-09829-4

[18] Sharif, B., Maletic, J.I.: An eye tracking study on camelcase and under_score identifier styles. In: 2010 IEEE 18th International Conference on Program Comprehension. pp. 196–205 (2010)

[19] Siegmund, J.: Program comprehension: Past, present, and future. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). vol. 5, pp. 13–20 (2016)

[20] Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., Brechmann, A.: Understanding understanding source code with functional magnetic resonance imaging. In: Proceedings of the 36th International Conference on Software Engineering. p. 378–389. ICSE 2014, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2568225.2568252

[21] Soloway, E., Ehrlich, K.: Empirical studies of programming knowledge. IEEE Transactions on Software Engineering SE-10(5), 595–609 (1984)