HAW HAMBURG

Bachelorarbeit

David Hoeck

Evaluierung über den Einsatz von Kotlin in einem großen Softwareprojekt

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

David Hoeck

Evaluierung über den Einsatz von Kotlin in einem großen Softwareprojekt

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 29. April 2019

David Hoeck

Thema der Arbeit

Evaluierung über den Einsatz von Kotlin in einem großen Softwareprojekt

Stichworte

Kotlin, Evaluierung, Lesbarkeit, Metriken, Studie, Modell

Kurzzusammenfassung

Durch die Entwicklung einer Vielzahl von Metriken, lassen sich heute bereits viele Aspekte von Software objektiv beschreiben und bewerten. Im Rahmen einer Evaluation über den Einsatz von Kotlin bei der CoreMedia AG, beschäftigt sich diese Arbeit mit der Lesbarkeit von Quellcode, einem Aspekt für welchen eine solche objektive Beschreibung und Bewertung bisher nicht möglich ist. Die Grundlage der Untersuchung bildet eine vergleichende Studie zur Lesbarkeit von Kotlin und Java sowie Ansätze von Modellen und Metriken zur Lesbarkeit von Quellcode. Basierend auf diesen Ansätzen und den Ergebnissen der Studie wird ein Vorschlag für ein allgemeines Modell zur Lesbarkeit von Quellcode entwickelt. Anders als bisherige Modelle, ermöglicht dieses Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache.

David Hoeck

Title of Thesis

Evaluation about the use of Kotlin in a large software project

Keywords

Kotlin, Evaluation, Readability, Metrics, Study, Model

Abstract

Through the development of a multitude of metrics, many aspects of software can be objectively described and evaluated today. As part of an evaluation about the use of Kotlin at the CoreMedia AG, this thesis deals with the readability of source code, an aspect for

which such an objective description and evaluation is not yet possible. The examination is based on a comparative study, concerning the readability of Kotlin and Java as well as on existing approaches of models and metrics for the readability of source code. Based on these approaches and the results of the study, a proposal for a general model about the readability of source code was developed. Contrary to former models it allows conclusions to be drawn about the readability of a specific programming language.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die vorliegende Arbeit stellt eine Evaluierung über den Einsatz von Kotlin bei der CoreMedia AG dar. Sie beschränkt sich dabei hauptsächlich auf den Aspekt der Lesbarkeit des Quellcodes und beschäftigt sich mit der Frage wie sich diese Eigenschaft objektiv bewerten lässt. Ergänzend bietet sie zudem einen kurzen Überblick über grundlegende wichtige Eigenschaften und Funktionen von Kotlin, welche für ein grundsätzliches Verständnis unabdingbar sind und für eine vollständige Evaluation der Sprache in Betracht gezogen werden müssen.

Die Bewertung der Lesbarkeit erfolgt dabei auf Grundlage einer vergleichenden Studie zur Lesbarkeit von Kotlin und Java Quellcode sowie einiger Ansätze zu Modellen und Metriken, welche eine objektive Messung von Lesbarkeit ermöglichen sollen.

1.1 Motivation

Die CoreMedia AG möchte evaluieren, ob Kotlin, neben Java als weitere Programmiersprache in der Produktentwicklung eingesetzt werden kann. Die Entscheidung über den Einsatz soll dabei nicht auf Grundlage von subjektiven Eindrücken und Meinungen einiger Verantwortlicher getroffen werden, sondern auf einer objektiven Betrachtung der Sprache beruhen.

1.1.1 Motivation für die Einführung von Kotlin

Bei der CoreMedia AG wird derzeit überwiegend Java als Programmiersprache eingesetzt. Es stellt sich also die Frage worin die Motivation für die Einführung einer neuen Sprache in die bestehende Codebasis liegt. Eine weitere Sprache bedeutet für alle beteiligten Entwickler auch einen höheren (Lern-)Aufwand in der täglichen Arbeit. Syntax

und Konzepte der Sprache müssen zunächst verstanden werden, um diese auch produktiv nutzen zu können.

Die Sprache sollte demnach einen erheblichen Mehrwert bieten, um diesen Aufwand zu rechtfertigen.

Kotlin verfügt über einige Eigenschaften und Funktionen, welche einen solchen Mehrwert versprechen. Dazu zählen, unter Anderem, folgende Eigenschaften:

- Die Möglichkeit der funktionalen und objektorientierten Programmierung.
- Die Unterscheidung zwischen nullfähigen und nicht nullfähigen Referenzen im Typsystem und damit eine Vorbeugung von Nullreferenzen.
- Die automatische Umwandlung von Typen.
- Die Herleitung von Typen aus dem umgebenden Code.
- Die Deklaration von einfachen Klassen als Einzeiler.
- Die umfassende Standardbibliothek für das Arbeiten mit Collections.

Die Entscheidung über den Einsatz einer Sprache allein anhand ihrer Funktionalität zu treffen wird den Anforderungen der CoreMedia AG jedoch nicht gerecht. Ebenso wichtig ist die Einbeziehung des Software-Entwicklungsprozesses, also die Beantwortung der Frage, wie gut sich mit dieser Sprache entwickeln lässt.

Auch hier bieten sich mit Kotlin einige interessante Möglichkeiten. Dazu zählt etwa die Möglichkeit der Interoperabilität mit anderen Sprachen, wie z.B. Java oder JavaScript, die Integration in das Spring Framework oder die umfassende Tooling-Unterstützung. Alle diese Aspekte können die Integration von Kotlin in die bestehende Umgebung erheblich vereinfachen.

1.1.2 Motivation für die Messung der Lesbarkeit

Eine weitere beachtenswerte Eigenschaft von Kotlin ist die Knappheit und Ausdrucksstärke der Sprache und die damit vermeintlich gute Lesbarkeit des Quellcodes.

Bei der Betrachtung des Entwicklungsprozesses in der Softwareentwicklung wird die Wichtigkeit dieser Eigenschaft deutlich.

Es ist allgemein bekannt, dass Entwickler in ihrer täglichen Arbeit deutlich mehr Zeit mit dem Lesen als mit dem Schreiben von Quellcode verbringen [Scalabrino u. a., 2018] [Buse und Weimer, 2010]. Gerade in der Wartung sowie in der Weiterentwicklung von bestehendem Code ist die Lesbarkeit des Quellcodes ein wichtiges Kriterium, welches sich auch stark auf die Kosten der Entwicklung auswirken kann.

Nach Einschätzung von Boehm entfallen in einem typischen Softwareprojekt oft mehr als 70% der gesamten Entwicklungskosten allein auf die Produkt-Instandhaltung (Maintenance) [Boehm und Basili, 2001]. Emilio Collar Jr. und Ricardo Valerdi stellen ebenfalls einen direkten Zusammenhang zwischen schlecht lesbarem Code und hohen Kosten in der Instandhaltung fest.

> "[...] The high cost of software maintenance has been linked to the difficulty of reading and understanding programming code, particularly code written by someone else.“ [Collar und Valerdi, 2014, S. 1]

Gut lesbarer Quellcode erhöht also die Chance, dass dieser auch leichter zu verstehen und in der Folge auch leichter anzupassen bzw. weiter zu entwickeln ist. Zudem können bei einem leichteren Verständnis des Codes Programmierfehler reduziert werden. Dies erhöht Robustheit und Zuverlässigkeit des Codes.

Auf die letzten eineinhalb Jahre bezogen, haben die Entwickler bei der CoreMedia AG, nach firmeninternen Berechnungen, im Durchschnitt zwar lediglich 25.2% ihrer Arbeitszeit für die Produkt Instandhaltung verbucht. Dennoch muss davon ausgegangen werden, dass auch bei der Entwicklung von neuer Funktionalität, das Lesen und Verstehen von bestehendem Code stets Voraussetzung für die eigene Entwicklung ist.

Im Rahmen einer Evaluierung von Kotlin ist die objektive Betrachtung und Bewertung der Lesbarkeit der Sprache also unbedingt zu berücksichtigen.

1.2 Ziel der Arbeit

Wie einleitend bereits erwähnt, legt diese Arbeit den Schwerpunkt auf die Messung der Lesbarkeit von Kotlin Quellcode. Dabei werden zwei grundlegende Ziele verfolgt.

Das primäre Ziel ist es eine möglichst objektive Aussage darüber zu treffen, wie lesbar Kotlin Quellcode im Vergleich zu Java Quellcode ist. Auf Grundlage dieser Erkenntnisse wird dann eine Empfehlung über den Einsatz von Kotlin bei der CoreMedia AG gegeben.

Das zweite Ziel der Arbeit ergibt sich aus dem primären Ziel. Da es bisher keine bekannte Möglichkeit gibt die Lesbarkeit einer Programmiersprache objektiv zu bewerten, wird ein Vorschlag für ein Modell erarbeitet, mit welchem eine solche Bewertung ermöglicht werden soll. Die Grundlage für das Modell bieten einige bisherige Ansätze von Modellen und Metriken zur Beschreibung und Messung der Lesbarkeit von Quellcode.

1.3 Struktur der Arbeit

Nachdem einleitend die Motivation sowie die Zielsetzung der Arbeit beschrieben wurde, wird dem Leser in Kapitel 2 zunächst ein Überblick über die Philosophie und die grundlegenden Eigenschaften und Funktionen von Kotlin vermittelt. Zudem wird die Relevanz von Kotlin allgemein, sowie die Relevanz der Lesbarkeit der Sprache, für die CoreMedia AG herausgestellt.

In Kapitel 3 wird erläutert, wie die Lesbarkeit von Quellcode objektiv bewertet werden kann. Dafür werden zunächst Metriken als Werkzeug zur Vermessung von Software vorgestellt. Es wird beschrieben, wie sich diese klassifizieren lassen und welche Anforderungen an eine solche Metrik gestellt werden. Anschließend wird die Problematik bei der Messung der Lesbarkeit von Quellcode näher erläutert sowie einige Ansätze von Modellen und Metriken vorgestellt, welche trotz der schwierigen Voraussetzungen eine objektive Messung von Quellcode ermöglichen sollen.

Auf Grundlage dieser Ansätze, wird in Kapitel 4 ein vorläufiges Modell zur Messung der Lesbarkeit von Quellcode entwickelt, welches Rückschlüsse auf die Lesbarkeit von einer konkreten Programmiersprache, wie Kotlin oder Java ermöglichen soll.

Anhand einer begleitenden Studie zur Lesbarkeit von Kotlin und Java erfolgt zunächst eine allgemeine Bewertung der Lesbarkeit beider Sprachen. Anschließend wird die Auswirkung konkreter Aspekte des Quellcodes auf die Bewertung der Studienteilnehmer näher untersucht und das zuvor entwickelte Modell auf Grundlage der gewonnenen Erkenntnisse überarbeitet. Zudem wird anhand einiger kotlin-spezifischer Eigenschaften im

Code geprüft, ob sich Auswirkung einiger Sprachfeatures auf die Lesbarkeit der Sprache ermitteln lassen.

Abschließend wird eine Empfehlung für bzw. gegen den Einsatz von Kotlin bei der CoreMedia AG gegeben, sowie ein Vorschlag für ein allgemeines Modell zur Lesbarkeit von Quellcode unterbreitet, welches Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache zulässt.

2 Kotlin

Auch wenn sich diese Arbeit auf die Lesbarkeit von Kotlin beschränkt, werden im folgenden Kapitel zunächst einige grundlegende Eigenschaften und Charakteristika der Sprache erläutert. Dem Leser soll so ein grundlegendes Verständnis der Sprache vermittelt werden, welches für die Betrachtung der Lesbarkeit der Sprache unabdingbar ist.

Für die Coremedia AG können die beschriebenen Eigenschaften und Charakteristika auch als Ansatz für die weitere Evaluation der Sprache genutzt werden.

Kotlin ist eine objektorientierte und funktionale Sprache, welche seit 2010 von dem Software Unternehmen JetBrains entwickelt wird. JetBrains entwickelt hauptsächlich Entwicklungs-Tools für diverse Programmiersprachen. Unter Anderem für Java, C#, JavaScript, Python und Ruby. Die Entwicklung dieser Tools erfolgt dabei zum großen Teil in Java.

Nach Darstellung von JetBrains wurde die Entwicklung von Kotlin motiviert durch den Mangel an geeigneten Alternativen zu Java als Sprache für die Entwicklung der eigenen Produkte [Jemerov und Isakova, 2017]. Ziel war es die Produktivität in der eigenen Entwicklung zu erhöhen.

2.1 Grundlegende Eigenschaften und Philosophie

Im Folgenden werden einige grundlegende Eigenschaften von Kotlin kurz dargestellt. Für diese Arbeit besondere Relevanz hat hier zunächst die Knappheit (conciseness) der Sprache. Anschließend werden noch weitere wichtige Eigenschaften betrachtet. Dazu gehören die pragmatische Ausrichtung, die Sicherheit und Robustheit sowie die Möglichkeit der Interoperabilität der Sprache.

2.1.1 Knappheit

Nach Aussage von JetBrains spielt die Knappheit und Ausdrucksstärke von Kotlin eine zentrale Rolle im Design der Sprache. Entsprechend wird diese propagiert [KotlinLang].

Dmitry Jemerov und Swetlana Isakova beschreiben die Knappheit einer Programmiersprache wie folgt.

> „A language is concise if it's syntax clearly expresses the intent of the code you read and doesn't obscure it with boilerplate required to specify how the intent is accomplished."[Jemerov und Isakova, 2017, S. 12]

In einer groben Schätzung nimmt JetBrains an, dass ein Programm in Kotlin etwa 40% weniger Zeilen Code benötigt als ein vergleichbares Programm in Java [KotlinLangFaQ]. Gleichzeitig wird betont, dass es nicht darum geht den Quellcode auf eine minimale Anzahl an Zeichen herunterzubrechen.

Ließe sich diese Schätzung belegen, würde das bedeuten, dass der Informationsgehalt in der Syntax von Kotlin wesentlich höher als in Java ist bzw. Kotlin mit deutlich weniger sog. Boilerplate-Code[1] auskommt.

Jetbrains folgert daraus weiter, dass Kotlin dadurch auch sehr leicht zu lesen und in der Folge auch sehr leicht zu verstehen bzw. zu erlernen ist.

Das Bestreben einer Syntax, welche mit möglichst wenig Boilerplate-Code auskommt und einen sehr hohen Informationsgehalt besitzt zieht sich dementsprechend durch alle Bereiche der Sprache. Jemerov und Isakova schreiben dazu.

> „In Kotlin, we've tried hard to ensure that all the code you write carries meaning and isn't just there to satisfy code structure requirements."[Jemerov und Isakova, 2017, S. 11]

Um ein Verständnis dafür zu schaffen, wie sich dies konkret im Quellcode niederschlägt, folgen einige kurze Beispiele.

[1] häufig verwendete Codefragmente

DataClasses

Durch Data Klassen lassen sich einfache Klassen, welche einzig der Datenhaltung dienen, mit einer Zeile Code definieren. Alle nötigen Basisfunktionen, wie z.B. Getter, Setter, *equals()* und *hashCode()* werden vom Compiler generiert.

```
data class Customer(val name: String, val email: String)
```

Type Inference

Kotlin ist eine statisch typisierte Sprache. Das bedeutet in der Folge, dass der Typ im Quellcode stets angegeben werden muss. Häufig ist es jedoch möglich etwa den Rückgabetyp einer Funktion oder den Typ einer Variablen aus dem Kontext abzuleiten. Dies wird in Kotlin sehr konsequent umgesetzt. Beispiele dafür sind:

```
val string: String = "Ein String"
val string = "Ein String" // Typ hergeleitet

fun double(x: Int): Int = x * 2
fun double(x: Int) = x * 2  // Rückgabetyp hergeleitet
```

String Templates

Durch String Templates, ist es möglich ausführbaren Code in einen String einzubetten. Dies kann gerade bei komplexeren Ausdrücken oder längeren String Literalen zu einer besseren Übersichtlichkeit der entsprechenden Code Abschnitte führen. Beispiele dafür sind:

```
var a = 1
// Einfacher Name als Template:
val s1 = "a is $a"

a = 2
// Beliebiger Ausdruck als Template:
val s2 = "${s1.replace("is", "was")}, but now is $a"
```

Collections

In der Kotlin Standard Library wird explizit zwischen veränderbaren und unveränderbaren Collections unterschieden. Ebenso bietet die Standard Bibliothek hier ein breites Spektrum an Funktionen. Auch dies kann wieder zu einer klareren Struktur des Quellcodes führen. Ein Beispiel dafür ist:

```
val list = mutableListOf("some", "Strings", "for", "you")

list.filter { it.length % 2 == 0 }
        .sortedBy { it.length }
        .map { it.toUpperCase() }
```

2.1.2 Pragmatische Ausrichtung

Ziel von JetBrains ist es mit Kotlin eine möglichst pragmatische Sprache zu entwickeln mit welcher sich konkrete Probleme leicht lösen lassen [KotlinBlog]. Dabei soll Kotlin überall genutzt werden können, wo bisher Java zum Einsatz kommt.

Besonders komfortabel gestaltet sich der Einstieg in Kotlin, wenn bereits Erfahrungen in der Java Entwicklung vorliegen. So ist es in Kotlin gut möglich Programmiertechniken zu verwenden, welche sehr ähnlich zu Techniken in Java sind. Speziellere Kotlin Funktionen können dann mit zunehmendem Lernfortschritt angewendet werden.

Ein weiterer wichtiger Aspekt ist die Tooling-Unterstützung der Sprache, also die Schaffung einer 'smarten' Entwicklungsumgebung. Hier hat Jetbrains bereits parallel zum Design und der Entwicklung der Sprache, eine entsprechende Erweiterung für die eigene Entwicklungsumgebung IntelliJ IDEA entwickelt. Ähnliche Erweiterungen existieren inzwischen auch für weitere Entwicklungsumgebungen wie z.B. Eclipse.

Einige nützliche Funktionen der Tooling-Unterstützung sind z.B. eine Autovervollständigung, Verbesserungsvorschläge oder Alternativen zu bestimmten Code-Abschnitten oder die automatische Umwandlung von Java in Kotlin Quellcode.

Für CoreMedia AG ebenso relevant ist die Integration von Kotlin in das Spring Framework, welches in großen Teilen der Entwicklung eingesetzt wird.

2.1.3 Sicherheit

Eine weitere Eigenschaft von Kotlin, welche von Jetbrains besonders herausgestellt wird, ist die Sicherheit der Sprache. Mit Sicherheit ist in diesem Kontext gemeint, dass durch das Design der Sprache bestimmten Fehlern vorgebeugt wird [Jemerov und Isakova, 2017].

Ein hohes Level an Sicherheit wird in Kotlin schon dadurch erreicht, dass die Sprache, genau wie Java auf der JavaVirtualMachine läuft. Auch die statische Typisierung und die damit verbundenen Typprüfungen zur Übersetzungszeit führen zu einer hohen Zuverlässigkeit der Sprache.

Kotlin geht hier allerdings noch einen Schritt weiter und versucht einigen Fehlern bereits zur Übersetzungszeit vorzubeugen. Dazu zählen zum einen Fehler durch Nullreferenzen (*NullPointerException*) sowie Fehler welche durch das Umwandeln eines Objekt-Typs, durch sog. casten (*ClassCastException*) auftreten können.

Da das Verständnis dieser Prüfungen wichtig für das Verständnis von Kotlin Quellcode ist, werden sie hier kurz erläutert.

Nullsafety

Um Nullreferenzen und daraus resultierende *NullPointerExceptions* zu vermeiden, wird im Typsystem von Kotlin zwischen nullable- und non-nullable Typen unterschieden. Nullable Typen können *null* als Wert annehmen und werden durch ein Fragezeichen hinter der Typbezeichnung gekennzeichnet, bei non-nullable Typen wird dies verhindert.

```
val s1: String? = null   // nullable
val s2: String = ""      // non-nullable
```

Zugriffe auf Referenzen eines non-nullable Typs sind dabei unproblematisch, da solche Referenzen in keinem Fall *null* als Wert annehmen können. Dies wird durch den Compiler kontrolliert.

Um bei Zugriffen auf Referenzen eines nullable Typs eine *NullPointerException* zu vermeiden, wird vor dem Zugriff eine Prüfung auf *null* durch den Compiler erzwungen. Diese kann Explizit durch einen klassischen *If* Ausdruck erfolgen. Ein Beispiel ist:

```
val b: String? = null    // nullable
val l = if (b != null) b.length else -1
```

Kotlin bietet daneben jedoch noch weitere Möglichkeiten, welche syntaktisch deutlich simpler gestaltet sind. Eine davon wird im Anschluss kurz vorgestellt, da sie im Rahmen dieser Arbeit Relevant ist.

Safe Calls Ein Safe Call wird durch ein Fragezeichen vor dem Aufruf gekennzeichnet und nur ausgeführt, wenn der Receiver ungleich *null* ist. Andernfalls liefert der Aufruf *null* zurück.

```
var b: String? = "abc"
println(b?.length) // Ausgabe: "abc"

var c: String? = null
println(b?.length) // Ausgabe: null
```

Solche sicheren Aufrufe eignen sich besonders bei verketteten Aufrufen.

Intelligente Typumwandlung

In Kotlin kann oft darauf verzichtet werden Typen explizit umzuwandeln (Cast). Der Compiler realisiert Typprüfungen und führt Typumwandlungen durch, wenn sie benötigt werden.

```
fun demo(x: Any) {  // Any entspricht etwa Object in Java
    when (x) {
        is String -> println(x.length) //Umwandlung zu String
        is Int -> println(x.plus(3))   //Umwandlung zu Int
    }
}
```

Fokus auf Unveränderlichkeit

Ein weiterer Aspekt, welcher erheblich zur Sicherheit der Sprache beiträgt, ist die, im Vergleich zu Java, häufigere explizite Unterscheidung zwischen veränderbaren (mutable) und unveränderbaren (immutable) Objekten. Dies ist zwar in etwas anderer Form auch in Java möglich, durch das Design von Kotlin und die gute Integration in die Standardbibliotheken wird der Entwickler jedoch häufiger dazu aufgefordert, sich explizit Gedanken zur Veränderbarkeit der verwendeten Objekte zu machen.

Beispiele sind hier etwa die Unterscheidung zwischen den *val* (immutable) und *var* (mutable) Schlüsselwörtern bei der Deklaration von Variablen, oder die Unterscheidung zwischen veränderbaren und unveränderbaren Collections bereits in der Standardbibliothek.

Durch die wiederum sehr gute Tooling-Unterstützung bekommt der Entwickler zudem Hinweise, ob es im konkreten Fall sinnvoll sein kann ein unveränderbares Objekt zu nutzen.

Durch die Möglichkeit der funktionalen Programmierung profitiert die Sprache zudem von den Grundkonzepten welche diese mit sich bringt. Dazu gehört z.B. das Arbeiten auf unveränderbaren Objekten, die Vermeidung von Seiteneffekten durch Funktionen und in der Folge die sichere Nebenläufigkeit.

2.1.4 Interoperabilität

Das Institute of Electrical and Electronics Engineers (IEEE) definiert Interoperabilität wie folgt:

> „The ability of two or more systems or components to exchange information and to use the information that has been exchanged.“[Radatz u. a., 1991, S. 42]

Übertragen auf Programmiersprachen lässt sich dies frei in etwa übersetzen mit der Fähigkeit unterschiedlicher Sprachen, möglichst nahtlos zusammenzuarbeiten. Das bedeutet konkret, dass z.B. Bibliotheken einer Sprache in einer anderen Sprache genutzt werden können oder aber auch Funktionen einer Sprache direkt aus einer anderen Sprache aufgerufen werden können.

In Kotlin ist dies, bezogen auf Java, sehr umfassend umgesetzt. So ist es ohne weiteren Aufwand möglich sowohl Java Code von Kotlin als auch Kotlin Code von Java aus aufzurufen. Ebenso nutzt Kotlin Java Bibliotheken, wo immer dies Sinn ergibt. Die Collections in Kotlin bauen z.B. nahezu komplett auf den Java Collections auf und erweitern diese lediglich.

Um das Arbeiten sprachübergreifend so komfortabel wie möglich zu gestalten, setzt JetBrains hier auch wieder auf umfassende sprachübergreifende Tooling-Unterstützung. Dies gilt z.B. für die Navigation zwischen Kotlin und Java Dateien sowie das sprachübergreifende Debugging oder Refactoring. Ebenso wird ein Konverter bereitgestellt, mit welchem bestehender Java Code leicht zu Kotlin Code umgewandelt werden kann.

Diese umfassenden Möglichkeiten können die Integration von Kotlin in eine bestehende Java Codebase ungemein erleichtern.

2.2 Relevanz für die CoreMedia AG

In den vorherigen Abschnitten wurden einige der wichtigsten Charakteristiken und Eigenschaften von Kotlin aufbereitet.

Einige dieser Eigenschaften können für die CoreMedia AG ein weiteres Entscheidungskriterium, bezüglich des Einsatzes von Kotlin in der Produktentwicklung darstellen. Die genauere Betrachtung und Evaluierung dieser Eigenschaften liegt jedoch nicht im Fokus dieser Arbeit.

Ebenso interessant in diesem Zusammenhang, ist die beschriebene Motivation für die Entwicklung von Kotlin. Die Ausgangslage bei JetBrains, also eine bestehende große Java Codebasis, welche erhalten bleiben soll sowie ein starker Fokus auf Tooling-Unterstützung, lässt sich potentiell gut auf die CoreMedia AG übertragen.

Auch die umfassende Integration von Kotlin in das Spring Framework stellt einen weiteren Anreiz dar, sich auch über diese Arbeit hinaus mit der Sprache auseinanderzusetzen.

Beschränkung auf die Lesbarkeit

Im weiteren Verlauf, konzentriert sich diese Arbeit auf die Lesbarkeit von Kotlin, einer Eigenschaft, welche von JetBrains immer wieder hervorgehoben wird.

Diese Fokussierung auf die Lesbarkeit, als ein zentrales Merkmal der Sprache, beruht auch wieder auf ihrer praktischen Ausrichtung.

Grundlage ist die allgemein akzeptierte Annahme, dass Entwickler während des Entwicklungsprozesses deutlich mehr Zeit mit dem Lesen, als mit dem Schreiben von Code verbringen. Häufig von Code, welcher von anderen Entwicklern, über einen längeren Zeitraum hinweg geschrieben wurde.

Das Lesen und Verstehen von Code ist also sehr häufig Voraussetzung für die eigene (Weiter)Entwicklung des Codes. Eine schlechte Lesbarkeit kann das Verständnis deutlich erschweren [Posnett u. a., 2011].

Dmitry Jemerov und Svetlana Isakova, welche beide stark in die Entwicklung von Kotlin involviert sind, kommen zu folgendem Schluss.

> The simpler and more concise the code is, the faster you'll understand whats's going on"[Jemerov und Isakova, 2017, S. 11]

Als ein wichtiger Baustein für eine gute Lesbarkeit des Codes, wird hier auch wieder die Knappheit der Sprache genannt.

Quellcode sollte immer das Ziel haben, in einer knappen und dennoch verständlichen Art, möglichst viel Information zu transportieren. Kürzerer Code kann schneller geschrieben, und noch wichtiger, schneller und einfacher gelesen werden. Beides kann die Produktivität in der Softwareentwicklung deutlich erhöhen.

Dass JetBrains mit Kotlin das Ziel hat eine Sprache zu entwicklen, welche diese Kriterien erfüllt, ist deutlich. Ob Kotlin dieses Ziel erfüllt, lässt sich jedoch nur äußerst begrenzt bewerten.

Der Grund ist, dass Lesbarkeit, anders als viele andere Eigenschaft der Sprache, eine weitestgehend subjektive Eigenschaft ist und dadurch immer im Auge des Betrachters liegt. Entsprechend schwierig gestaltet sich eine daher eine objektive Bewertung dieser Eigenschaft.

Wie eine Bewertung dennoch möglich sein kann, wird im folgenden Kapitel erläutert.

3 Bewertung der Lesbarkeit von Quellcode

In den folgenden Abschnitten wird die Bewertung der Lesbarkeit von Quellcode betrachtet.

Dafür werden zunächst Metriken allgemein als Mittel zur Vermessung von Software vorgestellt. Es wird erläutert wie sich diese klassifizieren lassen, welche Anforderungen sie erfüllen müssen und wo die Grenzen solcher Metriken liegen.

Anschließend werden die Schwierigkeiten und Probleme bei der Messung von Lesbarkeit diskutiert sowie einige bestehende Ansätze von Modellen und Metriken zur Messung der Lesbarkeit von Quellcode vorgestellt.

3.1 Software-Metriken als Werkzeug

Das Begriff der Metrik stammt aus dem Griechischen und lässt sich in etwa mit "Kunst des Messens"übersetzen [Liggesmeyer, 2009].

Metriken kommen in vielen verschiedenen Bereichen wie etwa Mathematik, Physik, Linguistik oder auch der Informatik zum Einsatz. Jeder Bereich definiert den Begriff der Metrik etwas anders.

Für diese Arbeit sind neben den bisherigen Ansätzen von Metriken zur Lesbarkeit von Quellcode einige Software Metriken relevant.[1]

Eine Software Metrik ist eine Funktion, welche eine Eigenschaft von Software auf einen numerischen Werte abbildet. Ein Beispiel dafür ist die Größe einer Software. Sie lässt sich z.B. als die Anzahl der Zeilen, der Wörter oder Symbole im Quellcode abbilden. Solche Funktionen ermöglichen es, diese Eigenschaft besser und konkreter zu beschreiben.

[1] Zwar gibt es in der Linguistik verschiedene Metriken, welche das Messen der Lesbarkeit von natürlicher Sprache ermöglichen, sie werden in dieser Arbeit jedoch nicht explizit behandelt. Dennoch Begründen einige Modelle und Metriken zur Messung von Quellcode, auch auf solchen Metriken.

Der britische Physiker Lord Kelvin beschreibt dies wie folgt.

> „When you can measure what you are speaking about and express it in numbers, you know something about it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; [...]“ [Thomson, 1889–94, S. 73]

Übertragen auf Software bedeutet dies, dass erst die Quantifizierung der Eigenschaften einer Software, ein umfassendes Verständnis ermöglicht.

Eine weitere wichtige Eigenschaft, welche durch die Quantifizierung erreich wird, ist die Vergleichbarkeit der gemessenen Werte und somit die Vergleichbarkeit der Software, zu welcher diese Werte gemessen wurden. So können zum Beispiel konkrete Aussagen darüber getroffen werden, ob Software A komplexer, größer oder leistungsfähiger als Software B ist.

Sind solche Aussagen mit Zahlen hinterlegt, erleichtern sie zudem die zwischenmenschliche Kommunikation erheblich. Ohne konkrete Zahlen stößt die natürlich Sprache hier schnell an ihre Grenzen [Sneed u. a., 2010]. So können schwammige und ungenaue Aussagen entstehen.

Eine Aussage wie z.B. “Der Quellcode von Software A ist gut lesbar”, bietet erst einmal wenig Mehrwert. Es ist nicht klar, was “gut lesbar” hier bedeutet und in Relation zu was diese Aussage steht.

Durch die Abbildung der Eigenschaft Lesbarkeit auf einen Zahlenwert, also die Anwendung einer oder mehrer Metriken, ließe sich eine Maßskala definieren, welche “gut lesbar” mit einem Zahlenwert hinterlegt und zudem eine Relation zu anderen Messungen ermöglicht. So bekommt die Aussage eine andere Qualität und kann somit auch einen größeren Mehrwert darstellen.

3.1.1 Grundlegende Klassen

Software ist immer ein multidimensionales Konstrukt. Drei dieser Dimensionen sind von besonderem Interesse für die Messbarkeit von Software. Für diese Dimensionen definieren Harry M. Sneed et al. jeweils eine Klasse von Softwaremetriken [Sneed u. a., 2010].

Quantitätsmetriken: Zu dieser Klasse zählen Metriken, welche die Anzahl der Vorkommnisse eines Softwarebestandteiles messen. Beispiele sind etwa die Menge aller Zeilen des Quellcodes oder die Menge aller Anweisungen. Mit Quantitätsmetriken lässt sich also der Umfang einer Software beschreiben.

Komplexitätsmetriken: Mit Metriken dieser Klasse, lassen sich Aussagen über die Komplexität einer Software treffen. Wie Komplexität definiert wird, hängt dabei von der jeweiligen Metrik ab.

Ein populäres Beispiel sind die Halstead Metriken, benannt nach Maurice Howard Halstead. Diese beruhen auf der Annahme, dass der Quellcode eines Programmes aus einer Aneinanderreihung von Operatoren und Operanden besteht. Auf dieser Grundlage lassen sich Werte zur Größe, zum Implementierungsaufwand oder zur Komplexität einer Software berechnen.

Die Halstead Metriken werden in einem separaten Abschnitt ausführlicher erläutert, da sie in Teilen zur Messung der Lesbarkeit von Quellcode genutzt werden können.

Qualitätsmetriken: Für diese Klasse von Software Metriken existiert eine Definition nach IEEE Standard 1061.

> „Eine Software-Qualitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet, welcher als Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit interpretierbar ist.“ [Schneidewind, 1997, S. 3]

Mit solchen Metriken lässt sich also die Güte einer Software beschreiben. Voraussetzung für diese Beschreibung ist die Definition von Qualitätskriterien. Diese können je nach Kontext verschieden definiert werden.

Beispiele für Qualitätsmaße sind die Fehlerdichte, die Modularität oder die Konvertierbarkeit einer Software [Sneed u. a., 2010].

Sneed et al. beschreiben die Qualität, als eine der unkenntlichsten und am schwersten greifbaren Eigenschaften von Software [Sneed u. a., 2010]. Dementsprechend kann sich die Messung von Qualitätseigenschaften sehr aufwendig gestalten.

3.1.2 Anforderungen und Grenzen

Ob das Ergebnis einer Metrik aussagekräftig und belastbar ist, hängt mit an der Erfüllung einiger grundlegender Gütekriterien. Wird eines oder mehrere dieser Kriterien nicht erfüllt, kann dies die Relevanz der Ergebnisse schwächen oder sogar die Sinnhaftigkeit der Messung selbst in Frage stellen. Durch diese Kriterien, werden somit in gewisser Weise die Grenzen von Software Metriken abgesteckt.

Im Folgenden werden einige der wichtigsten Kriterien kurz erläutert. Die Ausführung stützt sich auf die Beschreibungen von Dirk W. Hoffmann [Hoffmann, 2008].

Objektivität: Das wohl grundlegendste Kriterium ist die Objektivität der Messung. Subjektive Einflüsse des Messenden müssen soweit wie möglich vermieden werden.

Robustheit: Die Eigenschaft der Robustheit oder Zuverlässigkeit beschreibt die Belastbarkeit der Ergebnisse einer Messung. Diese müssen bei wiederholter Messung reproduzierbar sein. Eine Voraussetzung dafür ist die Objektivität der Daten.

Vergleichbarkeit: Es muss möglich sein, die Ergebnisse unterschiedlicher Messungen, welche mit Hilfe der gleichen Metrik durchgeführt wurden, in Relation zueinander zu setzen.

Verwertbarkeit: Eine Metrik, sollte immer auch einen praktischen Nutzen haben. Andernfalls besteht die Gefahr, dass die Messung nur noch zum Selbstzweck durchgeführt wird. Dies kann die Akzeptanz von Metriken, z.B. in einem Unternehmen, allgemein schwächen.

Ökonomie: Eng mit dem Kriterium der Verwertbarkeit verwandt, ist die ökonomische Betrachtung einer Metrik. Nur wenn sich die Erhebung einer Metrik kostenökonomisch umsetzen lässt, wird diese letztendlich in der Praxis akzeptiert werden. Der Einsatz von Metriken stellt immer einen Mehraufwand dar. Dieser muss stets in Relation zum Mehrwert stehen, welcher durch die Ergebnisse der Metrik generiert werden kann.

3.1.3 Die Problematik bei der Messung von Lesbarkeit

Nach der Vorstellung von Metriken, als Mittel zur Vermessung von Software sowie der Betrachtung der grundlegenden Anforderungen und Grenzen einer solchen Metrik in 3.1, kann im Folgenden die Messung der Lesbarkeit von Quellcode differenziert betrachtet werden.

Wie in 2.2 beschrieben, handelt es sich bei der Lesbarkeit von Quellcode um eine überwiegend subjektive Eigenschaft, deren Bewertung immer auch im Auge des Betrachters liegt. Dennoch besteht häufig ein allgemeiner Konsens darüber, ob ein Stück Quellcode eher schlecht oder gut lesbar ist.

Diese Beschreibung von Lesbarkeit mit Aussagen wie “gut” oder “‘schlecht” suggeriert schon, dass es sich bei dieser Eigenschaft um ein Qualitätskriterium von Software handelt, welches sich jedoch zunächst nur undeutlich beschreiben lässt. Ein standardisiertes Maß zur Bewertung existiert nicht.

Eine Qualitätsmetrik zur Messung dieser Eigenschaft zu entwickeln gestaltet sich daher als äußerst schwierig. Durch die subjektiven Einflüsse bei der Bewertung, kann schon das erste, grundlegende Gütekriterium, die Objektivität der Messung, nicht, oder nur sehr begrenzt erfüllt werden. Dasselbe gilt in der Folge auch für weitere Kriterien, wie die Robustheit oder die Vergleichbarkeit der Ergebnisse.

Um eine aussagekräftige Metrik zur Bewertung der Lesbarkeit von Quellcode zu erhalten, müssen also diese subjektiven Einflüsse entfernt oder zumindest minimiert werden. Des Weiteren sollte ein Zahlenmaß definiert werden, welches eine differenziertere Bewertung ermöglicht.

3.2 Modelle und Metriken zur Lesbarkeit von Quellcode

Trotz der schwierigen Voraussetzungen gibt es einige Arbeiten, welche sich mit der Lesbarkeit von Quellcode beschäftigen. In diesen werden erste, fundierte Ansätze einer Metrik bzw. eines Modells zur Lesbarkeit von Software entwickelt.

Der Nutzen solcher Metriken kann dabei vielfältig sein. Sie können z.B.

- Entwicklern dabei helfen besser lesbaren Code zu schreiben.

- Das Projektmanagement bei der Überwachung der Lesbarkeit in einem Projekt unterstützen.
- Zur schnellen Identifikation von schlecht lesbarem Code beitragen.
- In der statischen Codeanalyse genutzt werden um entsprechende Warnungen in der Entwicklung zu generieren.

Im folgenden Abschnitt werden einige dieser Ansätze näher betrachtet.

Zunächst die viel beachtete Arbeit von Buse und Weimer [Buse und Weimer, 2010]. Sie bildet die Grundlage für die weiteren Arbeiten und beeinflusst diese auch stark in ihrer Vorgehensweise und Struktur.

Anschließend drei weitere Arbeiten, welche auf der Arbeit von Buse und Weimer aufbauen und diese um weitere Aspekte ergänzen bzw. verbessern.

Die Basis der entwickelten Modelle liegt jeweils in einer Studie, in welcher Probanden die Lesbarkeit von vorgelegten Code-Ausschnitten, auf einer Skala von 1 (sehr schlecht lesbar) bis 5 (sehr gut lesbar), bewerten. Aus den Ergebnissen der Studie und basierend auf den Hypothesen der Autoren, wird anschließend ein Modell zur Lesbarkeit entwickelt.

Die Validierung des Modells erfolg mithilfe eines binären Klassifikationsalgorithmus, welcher mit einem Teil der Daten aus der Studie trainiert wird. Der Algorithmus unterscheidet in der Bewertung lediglich zwischen den Kategorien “eher gut lesbar” und “eher schlecht lesbar.

Mithilfe des Klassifizierers wird gezeigt, dass sich das entwickelte Modell automatisieren lässt und zudem eine ähnlich hohe Vorhersagekraft, wie die Teilnehmer der Studie besitzt.

3.2.1 Learning a metric for code readability [Buse und Weimer, 2010]

Der im Rahmen dieser Arbeit relevanteste Ansatz für eine Metrik zur Lesbarkeit von Quellcode wurde 2008 unter dem Namen “Learning a metric for code readability” von Raymond P. L. Buse und Westley Weimer veröffentlicht.

Buse und Weimer definieren Lesbarkeit als die menschliche Bewertung davon, wie leicht ein Text zu verstehen ist.

Eine grundlegende Annahme ihrer Theorie ist dabei die klare Abgrenzung der Lesbarkeit von der Komplexität eines Codeabschnittes. Während Komplexität, nach der Auffassung der Autoren, stark abhängig von der Größe z.B. von Klassen oder Funktionen ist, sei die Lesbarkeit primär abhängig von lokalen, zeilenbasierten Eigenschaften des Codes.

Zur Überprüfung ihrer Theorie und als Grundlage für ein eigenes Modell zur Lesbarkeit führen die Autoren eine Studie mit 120 Teilnehmern in einigen Kursen der Universität Virginia durch. Für die Studie stellen sie eine Sammlung von 100 verschiedenen Java Code-Ausschnitten zusammen, welche automatisch mithilfe eines Tools aus verschiedenen open source Projekten extrahiert werden. Diese werden von den Teilnehmern der Studie nach ihrer Lesbarkeit, wie zuvor beschrieben, bewertet. Das Modell von Buse und Weimer baut also auf 12000 Bewertungen auf.

Ein Ausschnitt besteht dabei immer aus genau drei aufeinanderfolgenden, einfachen Anweisungen. Dazu gehören z.B. Zuweisungen, Funktionsaufrufe, *breaks, continues, throws* und *returns*. Zusätzlich kann ein Ausschnitt noch weitere Konstrukte wie z.B. *if-else, try-catch, while, switch* und *for* enthalten. Ein Ausschnitt ist dabei logisch in sich geschlossen und beschränkt sich auf einen Geltungsbereich (scope) im Code.

Um den Fokus der Teilnehmer auf die lokalen, zeilenbasierten Eigenschaften im Code zu lenken, sind die Ausschnitte zudem relativ kurz gehalten (im Schnitt nur 7,7 Zeilen). Aus demselben Grund wird bei der Auswahl darauf geachtet, dass für das Verständnis eines Ausschnitts, der Einsatzkontext nicht relevant ist.

Die Auswertung der Studie zeigt auf, dass eine große Übereinstimmung in der Bewertung der Code-Ausschnitte zwischen den einzelnen Teilnehmern existiert. Die Autoren analysieren diese Korrelation im Anschluss und bestimmen, welche lokalen Eigenschaften des Codes, welche Auswirkung auf die Bewertung haben.

Dabei beschränken sie sich auf statische, größenunabhängige Eigenschaften, welche sich einfach aus dem Code extrahieren lassen (siehe Tabelle 3.1).

Jede dieser Eigenschaften lässt sich auf einen Java Codeausschnitt beliebiger Größe anwenden. Beschrieben wird eine Eigenschaft dabei als Durchschnittswert pro Zeile oder maximaler Wert aller Zeilen. Ein Beispiel ist die Eigenschaft der Zeilenlänge. Für sie wird einmal die durchschnittliche Anzahl an Zeichen pro Zeile und einmal die Anzahl an Zeichen, der längsten Zeile im Codeausschnitt berücksichtigt.

Tabelle 3.1: Lokale Code-Eigenschaften nach Buse und Weimer [Scalabrino u. a., 2018]

AVG.	Max.	Eigenschaft
▽	▽	line length (# characters)
▽	▽	# identifiers
▽	▽	identifier length
▽	▽	indentation (preceding whitespace)
▽	▽	# keywords
▽	▽	# numbers
△		# comments
▽		# periods
▽		# spaces
▽		# parenthesis
△		# arithmetic operators
▽		# comparison operators
▽		# assignments (=)
▽		# branches (if)
▽		# loops (for, while)
△		# blank lines
	▽	# occurrences of any single character
	▽	# occurrences of any single identifier

Betrachtete Eigenschaften, sowie positive (△) bzw. negative (▽) Korrelation zur Bewertung der Lesbarkeit.
Die Auswirkung auf die Bewertung der Lesbarkeit wird farblich dargestellt:
hoch = ▽, mittel = ▽, gering = ▽
# wird als "Anzahl" gelesen.

Für jede Eigenschaft wird zudem bestimmt, wie stark sie sich positiv oder negativ auf die Bewertung der Lesbarkeit auswirkt.

So wird ein eigenes Modell zur Bewertung der Lesbarkeit von Quellcode hergeleitet.

Das entwickelte Modell wird im Anschluss mithilfe eines machine learning Algorithmus überprüft. Dieser klassifiziert die Code-Ausschnitte, wie bereits beschrieben, lediglich in zwei Kategorien. Die Überprüfung zeigt, dass sich das Modell eignet um die Lesbarkeit von Quellcode einzig auf der Grundlage von lokalen, zeilenbasierten Code-Eigenschaften zu bewerten.

Zudem lässt sich die Messung mit Hilfe eines solchen Klassifikations-Algorithmus auto-

matisieren. Dieser besitzt sogar eine leicht höhere Vorhersagekraft als die Teilnehmer der Studie.

Die Vorhersagekraft wird in diesem Kontext beschrieben, als die Übereinstimmung in der Bewertung der Lesbarkeit mit der durchschnittlichen Bewertung der Studienteilnehmer.

3.2.2 A simpler model of software readability [Posnett u. a., 2011]

Das Modell von Buse und Weimer wird 2011 durch die Professoren Daryl Posnett, Abram Hindle und Prem Devanbu in einem gemeinsamen Beitrag zur MSR (Mining Software Repositories) wieder aufgegriffen. Anders als andere Ansätze führen die Autoren keine eigene Studie durch. Ihre Arbeit basiert im Wesentlichen auf den Ergebnissen der Studie von Buse und Weimer.

Ihr Ziel ist es, das Modell von Buse und Weimer robuster zu gestalten, deutlich zu vereinfachen und um weitere Faktoren zu ergänzen. So soll ein theoretisch besser begründetes und praktisch einsetzbares Modell entstehen.

Die Autoren kritisieren zunächst die Nichtberücksichtigung der Größe der betrachteten Code-Ausschnitte. Sie zeigen, dass verschiedene Größenmaße, wie z.B. die Anzahl der Zeilen oder die Anzahl der Zeichen, durchaus eine positive Korrelation zur Bewertung der Lesbarkeit aufweisen. Zwar lässt sich durch solche Eigenschaften allein kein Modell zur Lesbarkeit beschreiben, dennoch sollte die Größe eines betrachteten Code-Ausschnittes immer in die Bewertung der Lesbarkeit mit einfließen.

Eine beobachtete, hohe Korrelation zwischen den verschiedenen Größenmetriken, legt zudem nahe, dass zur Vereinfachung des Modells nur eines dieser Maße berücksichtigt werden sollte.

Des Weiteren merken Posnett et al. an, dass Buse und Weimer in ihrer Arbeit bereits einige wenige der betrachteten Eigenschaften identifizieren, die einen signifikanten Einfluss auf die Lesbarkeit haben. Dennoch wurde das Modell nicht auf diese Eigenschaften beschränkt.

Sie stellen zudem fest, dass es sich bei vielen der betrachteten Metriken um einfache Zählungen von Tokens, wie z.B. Klammern, Schlüsselwörtern, Bezeichnern oder Kommas

handelt. Die Anzahl an Bezeichnern und Schlüsselwörtern weist dabei mit die höchste Korrelation zur Bewertung der Lesbarkeit auf.

Nach Ansicht der Autoren ließe sich durch eine Zusammenfassung solcher, relevanter Tokens ein deutlich einfacheres Modell und eine allgemeinere Theorie zur Lesbarkeit von Quellcode entwickeln.

Die Halstead Metriken ermöglichen eine solche, allgemeinere Betrachtung von Tokens.

Halstead Metriken Die Halstead Metriken sind eine Reihe aufeinander aufbauender Metriken, welche 1977 von Maurice Howard Halstead entwickelt wurden. Sie lassen sich unmittelbar aus der lexikalischen Struktur des untersuchten Quellcodes herleiten [Liggesmeyer, 2009].

Die Metriken beruhen dabei auf Halsteads Annahme, dass sich der Quellcode eines Programmes in eine Menge von Operatoren und Operanden aufteilen lässt.

Was dabei unter einem Operator und was unter einem Operand verstanden wird, ist in der Literatur nicht einheitlich definiert. Shen et al. schreiben dazu

> „Generally, any symbol or keyword in a program that specifies an algorithmic action is considered an operator, and a symbol used to represent data is considered an operand.“ [Shen u. a., 1983, S. 3]

Hoffmann definiert Operatoren als alle von einer Programmiersprache bereitgestellten Konstrukte wie z.B. Schlüsselwörter oder mathematische und vergleichende Operatoren. Als Operanden bezeichnet er die Bezeichner von Funktionen und Variablen sowie numerische und textuelle Konstanten. [Hoffmann, 2008]

Bei der Anwendung der Halstead Metriken auf eine Programmiersprache muss also noch konkretisiert werden, welche Konstrukte als Operator und welche als Operand gezählt werden.

Halstead berechnet aus den Operatoren und Operanden zunächst die folgenden Basismaße.

Anzahl unterschiedlicher Operatoren n_1 und Operanden n_2

Das Vokabular des Programms $n = n_1 + n_2$

Gesamtzahl aller Operatoren N_1 und Operanden N_2

Die Länge des Programmes $N = N_1 + N_2$

Auf Grundlage dieser Maße, entwickelt er eine Reihe verschiedener Metriken [Liggesmeyer, 2009].

In diesem Kontext ist zunächst nur das Volumen (V) eines Programmes relevant. Dieses beschreibt je nach Definition, die Anzahl Bits, die zur Darstellung des Programmes, auf binärer Ebene benötigt werden oder die Anzahl gedanklicher Vergleiche, welche benötigt werden um den betrachteten Quellcode zu schreiben [Halstead, 1977][Al-Qutaish und Abran, 2005]. Definiert wird es wie folgt:

$$V = N \log_2 n$$

Die Autoren zeigen, dass für das Volumen, das Vokabular und die Länge des Programmes jeweils eine negative Korrelation zur Bewertung der Lesbarkeit besteht. Allerdings korrelieren die Eigenschaften auch stark untereinander.

Dies bedeutet, dass zwar sowohl Volumen, Vokabular, als auch die Länge eines Programmes eine Auswirkung auf die Lesbarkeit haben können, Aufgrund der Korrelation untereinander sollte jedoch die Berücksichtigung einer der Eigenschaften im Modell ausreichen. Die Autoren entscheiden sich für das Volumen.

Dieser Entscheidung liegt ihre Hypothese zugrunde, dass sich mit Halsteads Volumen, durch die Kombination des Vokabulars und der Länge des Programmes, der Informationsgehalt von Quellcode beschreiben lässt. Sie folgern daraus eine enge Beziehung zur Lesbarkeit.

Zur Stärkung ihrer Hypothese zeigen sie, dass ein einfaches Modell, bestehend aus Halstead Volumen und der Anzahl der Zeilen, bereits eine bessere Vorhersagekraft besitzt als das Model von Buse und Weimer.

Entropie Als weitere Eigenschaft, welche Auswirkungen auf die Lesbarkeit von Quellcode haben könnte, identifizieren die Autoren die Entropie des Quellcodes.

In der Informationstheorie wird Entropie mit dem Maß für den mittleren Informationsgehalt einer Nachricht beschrieben. Grundlage dieses Verständnisses von Entropie bildet die

Arbeit von Claude E. Shannon „A Mathematical Theory of Communication“ [Shannon, 1948]

Berechnet wird sie aus der Anzahl aller Ausdrücke (z.B. Zeichen oder Tokens) sowie der Anzahl unterschiedlicher Ausdrücke.

Sei X ein Dokument und x ein Ausdruck in X. Dann ist $count(x_i)$ die Anzahl der Vorkommen von x_i in X. Die Wahrscheinlichkeit p für das Auftreten von x_i wird beschrieben mit $p(x_i) = \frac{count(x_i)}{\sum_{j=1}^{n} count(x_j)}$. Die Entropie H(X) wird definiert als

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

[Shannon, 1948]

Die Autoren betrachten die Auswirkung von Token- und Zeichen-Entropie zunächst isoliert.

Die Token-Entropie wird dabei von den Autoren berücksichtigt, da sich auch das Halstead Volumen auf die Zählung von Tokens stützt. Eine Betrachtung der Entropie auf Token-Ebene erscheint ihnen daher sinnvoll. Sie zeigen, dass eine positive Korrelation zur der Bewertung der Lesbarkeit besteht.

Versuche, in welchen das Halstead Volumen im vereinfachten Modell durch Token-Entropie ersetzt wird zeigen jedoch, dass die Vorhersagekraft eines solchen Modells nicht ganz so hoch ist wie bei Berücksichtigung des Halstead Volumens. Zudem besteht eine negative Korrelation zwischen beiden Werten.

Die Zeichen-Entropie korreliert, anders als die Token-Entropie, leicht negativ mit der Bewertung der Lesbarkeit. Eine größere Anzahl an (verschiedenen) Zeichen, kann also die Lesbarkeit eines Code-Ausschnittes verringern. Die Hinzunahme der Zeichen-Entropie zum einfachen Halstead Modell, welches das Volumen sowie die Anzahl der Zeilen enthält, führt zu einer leicht besseren Performance des Modells.

Das entwickelte Modell, bestätigt die Hypothese, dass sich der Ansatz von Buse und Weimer deutlich vereinfachen lässt, ohne an Vorhersagekraft zu verlieren. Es beinhaltet lediglich drei Maße, welche sich leicht und automatisiert aus dem Quellcode extrahieren lassen. Die Anzahl der Zeilen ist dabei positiv, die Zeichen-Entropie und das Halstead Volumen negativ mit der Lesbarkeit assoziiert.

Die Autoren weisen darauf hin, dass das entwickelte Modell nicht den Anspruch erhebt, die Essenz von Lesbarkeit allgemein abzubilden. Ebenso lässt es sich nicht blind auf beliebigen Quellcode anwenden. Es ist beschränkt auf den Kontext des verfügbaren Datensatzes, für welchen es realistische und erklärbare Ergebnisse liefert. Dabei kommt es mit deutlich weniger Variablen als das Modell von Buse und Weimer aus und liefert dennoch bessere Ergebnisse.

Im Rahmen dieser Arbeit ist daher das Modell in Gänze weniger Relevant. Die Beobachtungen in der Entwicklung des Modells bieten dagegen wichtige Erkenntnisse, welche in Teilen aufgegriffen und überprüft werden sollen.

3.2.3 A general software readability model [Dorn, 2012]

Ein weiteres Modell zur Lesbarkeit von Quellcode, wurde 2012 von Jonathan Dorn, im Rahmen seiner Master Thesis veröffentlich. Betreut wurde er dabei von Westley Weimer, einem der Autoren der zuvor vorgestellten Arbeit "Learning a metric for code readability".

Dorn analysiert zunächst die Modelle von Buse und Weimer sowie Posnett et al. Er kritisiert, dass beide Modelle sich nur schlecht verallgemeinern lassen. Als Gründe nennt er die Beschränkung auf sehr kleine Code-Ausschnitte, die geringe Datengrundlage von lediglich 100 Ausschnitten und 120 Studienteilnehmern, die Beschränkung auf Java, als einzige Programmiersprache sowie die Nichtberücksichtigung linguistischer, visueller und struktureller Eigenschaften.

Auf dieser Grundlage entwickelt er ein eigenes Modell zur Lesbarkeit von Quellcode, welches die genannten Probleme adressiert.

Dabei berücksichtigt er, anders als zuvor betrachteten Arbeiten, auch strukturelle Eigenschaften (z.B. die Veränderung der Einrückungen von Zeile zu Zeile), visuelle Eigenschaften (z.B. die Auswirkung von Syntax-Highlighting) sowie Eigenschaften der natürlichen Sprache (z.B. der verbale Inhalt von Bezeichnern). Diese stellen einen großen Teil seiner Weiterentwicklungen und Verbesserungen der vorherigen Modelle dar[2].

[2]Da diese Aspekte im Kontext dieser Arbeit jedoch nur geringe Relevanz besitzen, werden sie im weiteren Verlauf nicht weiter erläutert. Der Grund ist, dass sie weitestgehend unabhängig von speziellen Programmiersprachen sind und somit voraussichtlich auch keine Rückschlüsse auf die Lesbarkeit einer bestimmten Sprache wie z.B. Kotlin oder Java ermöglichen.

Um eine Datenbasis für sein Modell zu erhalten, führt Dorn eine Studie durch, welche ähnlich der Studie von Buse und Weimer aufgebaut ist.

Die Studie

Aufgabe der Teilnehmer ist es Code-Ausschnitte auf einer Skala von eins bis fünf nach ihrer Lesbarkeit zu bewerten. Anders als Buse und Weimer, setzt Dorn jedoch auf deutlich mehr und differenziertere Code-Ausschnitte.

Er extrahiert jeweils 120 Code-Ausschnitte in drei verschiedenen Programmiersprachen aus Open Source Projekten. Zu den Sprachen zählen Python, Java und CUDA. Des Weiteren unterscheidet er zwischen drei Größenklassen: *short* (ca. 10 Zeilen), *medium* (ca. 30 Zeilen) und *long* (ca. 50 Zeilen). Pro Sprache verteilen sich die Ausschnitte in gleichen Teilen (je 40 Stück) auf diese Klassen.

Ingesamt basiert die Studie somit auf 360 Code-Ausschnitten, von welchen jedem Teilnehmer jedoch lediglich 20 zufällig gewählte zur Bewertung gezeigt werden.

Im Gegensatz zu den Vorgaben von Buse und Weimer, muss ein Ausschnitt keinen logisch geschlossenen Block darstellen. Es kann sowohl mitten in einem Block anfangen oder enden als auch mehrere, logisch nicht zusammenhängende Blöcke umschließen. Dorn verspricht sich davon, den Blick eines Entwicklers so möglichst realistisch zu simulieren.

Mit 5000 Rückläufen, erreicht die Studie deutlich mehr Teilnehmer als Buse und Weimer.

Um eine weitere Differenzierung der Ergebnisse zu ermöglichen, werden die Teilnehmer zudem nach ihrer Arbeitserfahrung gefragt. Diese berücksichtigt Dorn in der Auswertung der Studie und stellt fest, dass Teilnehmer mit mehr als fünf Jahren Berufserfahrung durchaus andere Faktoren als relevant für die Lesbarkeit bewerten.

Relevante Ergebnisse

Die Einbeziehung verschiedener Programmiersprachen, die Klassifizierung der Ausschnitte nach Größe sowie die Berücksichtigung der Erfahrung der Teilnehmer, ermöglichen Dorn eine differenziertere Betrachtung der Ergebnisse.

So erzeugt er, neben einem allgemeinen Modell, unter Berücksichtigung aller Code-Ausschnitte, weitere spezialisierte Modelle, in welchen jeweils nur ein Teil der Ausschnitte berücksichtigt wird. Das Ergebnis sind Modelle spezialisiert auf Java und Python, sowie auf die Erfahrung der Teilnehmer.

Für Jedes dieser Modelle werden die statistisch signifikantesten Eigenschaften identifiziert, welche in Assoziation zur Bewertung der Lesbarkeit stehen. Dazu zählen strukturelle, syntaktische, visuelle und linguistische Eigenschaften.

Tabelle 3.2: Signifikante Code-Eigenschaften nach Dorn [Dorn, 2012]

Modell	Eigenschaft	Korrelation
Allgemein	avg. Zeilenlänge	-
	lange Zeilen	-
Java	lange Zeilen	-
	avg. Zeilen zwischen Bezeichnern	-
	# Schlüsselwörter	-
Python	avg. verschiedener Bezeichner pro Zeile	-
+5 Jahre Arbeitserfahrung	lange Zeilen	-
	# Leerzeichen	-
Studierende	lange Zeilen	-
	# arithmetische Operatoren	-
	# numerischer Literale	+

Die Eigenschaften sind pro Modell, absteigend nach ihrer Vorhersagekraft sortiert.
Die Spalte zur Korrelation, zeigt die Richtung der Korrelation an.
negativ (-) bzw. positiv (+). # wird als “Anzahl” gelesen

In der Tabelle 3.2 werden die, für diese Arbeit relevanten, signifikanten Eigenschaften für die Modelle dargestellt. Verschiedene Modelle zu Java und Python werden dabei jeweils zu einem Modell zusammengefasst. Nichtberücksichtigt werden strukturelle, visuelle und linguistische Eigenschaften.

Die Tabelle zeigt, dass die Zeilenlänge, wie auch schon in der Arbeit von Buse und Weimer, die größte Auswirkung auf die Bewertung der Lesbarkeit allgemein hat. Es wird jedoch auch sichtbar, dass sich die signifikanten Eigenschaften, für die verschiedenen Modelle zum Teil deutlich unterscheiden. So spielt in Python z.B. die durchschnittliche Anzahl Bezeichner pro Zeile eine große Rolle, während die Zeilenlänge nicht weiter relevant ist.

Auf Grundlage dieser und weiterer Erkenntnisse, entwickelt Dorn ein Modell, welches

in der Bewertung der Lesbarkeit besser mit den Bewertungen der Studienteilnehmer übereinstimmt als die Modelle von Buse und Weimer sowie Posnett et al.

Er fokussiert sich dabei stark auf die strukturellen, visuellen und linguistischen Eigenschaften von Quellcode, welche in dieser zusammenfassenden Darstellung jedoch nicht weiter betrachtet werden. Somit hat diese auch nicht den Anspruch seinen Ansatz in Gänze abzubilden. Sie beschränkt sich auf die, für diese Arbeit, relevanten Aspekte.

3.2.4 A comprehensive model for code readability [Scalabrino u. a., 2018]

Ein weiteres Modell zur Lesbarkeit von Quellcode wurde 2017 von Simone Scalabrino, Mario Linares-Vásquez, Rocco Oliveto und Denys Poshyvanyk veröffentlich.

Das Modell baut unter anderem, auf allen zuvor vorgestellten Modelle auf und soll diese Ansätze fortführen und erweitern.

Den Fokus legen die Autoren dabei jedoch auf Eigenschaften der natürlichen Sprache, welche in den existierenden Modellen nicht, oder nur wenig berücksichtigt werden.

Das Vorgehen ist dabei ähnlich dem Vorgehen von Buse und Weimer oder Dorn. Eine Umfrage, in welcher die Teilnehmer die Lesbarkeit von Code-Ausschnitten bewerten sollen bildet die primäre Datengrundlage. Diese gestaltet sich, mit etwa 5000 Teilnehmern und 600 Coe Ausschnitten noch etwas Umfangreicher als in der Arbeit von Dorn.

Anschließend wird die Auswirkung textueller Eigenschaften auf die Bewertung der Lesbarkeit analysiert und ein Modell erzeugt, welches die Erkenntnisse der zuvor betrachteten Modelle ergänzt.

Die Autoren zeigen, dass ein solches, erweitertes Modell, die Übereinstimmung mit den Bewertungen der Studienteilnehmer noch einmal erhöht.

Da textuelle Eigenschaften nur begrenzt Rückschlüsse auf die Lesbarkeit einer bestimmten Programmiersprache zulassen, wird die Arbeit nicht weiter ausgeführt. Die kurze Vorstellung soll dem Leser jedoch verdeutlichen, dass es weitere Faktoren gibt, welche bei der Bewertung der Lesbarkeit von Quellcode eine wichtige Rolle spielen. Diese sollten in einer ganzheitlichen Betrachtung nicht vernachlässigt werden.

4 Bewertung der Lesbarkeit von Kotlin und Java

In den bisherigen Kapiteln wurden zunächst die grundlegenden Eigenschaften und Charakteristika von Kotlin (2) sowie Metriken als Mittel zur Bewertung von Software beschrieben (3). Anschließend wurde die Problematik bei der Messung der Lesbarkeit von Quellcode erläutert sowie einige Modelle und Metriken vorgestellt, welche eine solche Messung, trotz der schwierigen Vorraussetzungen in Ansätzen ermöglichen.

Im Folgenden wird auf Grundlage dieser Ansätze ein erster Entwurf eines Modell zur Messung von Lesbarkeit entwickelt, welches sich auf Kotlin und Java Quellcode anwenden lässt. Anders als die bisher betrachteten Modelle sollen Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache ermöglicht werden. Das Vorgehen orientiert sich dabei am Vorgehen der zugrunde liegenden Arbeiten.

Zunächst wird ein Konzept erstellt, in welchem die Anforderungen an ein solches, sprachspezifisches Modell erarbeitet werden. Anschließend wird analysiert, in welcher Form sich diese Anforderungen umsetzen lassen, welche Erkenntnisse aus den bestehenden Arbeiten auf ein solches Modell potentiell übertragbar sind und welche weiteren Faktoren für die Bewertung berücksichtigt werden können.

Durch eine eigene Studie zur Lesbarkeit von Kotlin und Java Quellcode wird zunächst überprüft ob Kotlin Quellcode tatsächlich als besser lesbar als Java Quellcode bewertet wird. Unabhängig vom Ergebnis dieser Überprüfung wird anschließend untersucht welche Auswirkungen die einzelnen Faktoren, welche für den initialen Entwurf berücksichtigt wurden, auf die Bewertung der Lesbarkeit haben.

Auf Grundlage dieser Erkenntnisse lässt sich der initiale Entwurf des Modells bewerten und kann weiter modifiziert und verbessert werden. Das Ergebnis ist ein Vorschlag für ein allgemeines, aussagekräftiges Modell zur Bewertung der Lesbarkeit einer Programmiersprache.

4.1 Ansatz für ein allgemeines Modell zur Lesbarkeit von Quellcode

Um ein vorläufiges Modell zur Lesbarkeit von Kotlin und Java zu entwickeln, werden zunächst die Anforderungen an ein solches Modell erläutert. Anschließend wird geprüft wie sich diese praktisch umsetzen lassen und welche Erkenntnisse der unter 3.2 vorgestellten Arbeiten übertragen und genutzt werden können.

4.1.1 Anforderungen an das Modell

Um die Lesbarkeit einer Programmiersprache beschreiben zu können, muss zunächst festgelegt werden, was genau unter Lesbarkeit verstanden wird. Die Übergänge zur Verständlichkeit oder zur Komplexität sind fließend und liegen auch im Auge des Betrachters.

Anschließend muss eine Möglichkeit gefunden werden, die Lesbarkeit von Quellcode messbar und vergleichbar zu gestalten. Als Richtlinie können dabei die unter 3.1.2 beschriebenen, grundlegenden Anforderungen an Metriken genutzt werden. Die weitgehende Erfüllung dieser Anforderungen ist ein wichtiger Indikator für die Qualität des entwickelten Modells.

Da die Bewertung der Lesbarkeit des Quellcodes, Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache ermöglichen soll, dürfen zudem ausschließlich Faktoren berücksichtigt werden, welche diese auch zulassen.

4.1.2 Umsetzung der Anforderungen

In dieser Arbeit wird Lesbarkeit nach dem Verständnis von Buse und Weimar definiert.

> „We define readability as a human judgment of how easy a text is to understand.“[Buse und Weimer, 2010, S.1]

Der Begriff der Lesbarkeit begründet sich also auf der subjektiven, menschlichen Bewertung, wie leicht ein vorgelegter Text zu Verstehen ist. Um eine solche Bewertung messbar und vergleichbar zu gestalten, wird diese zudem quantifiziert, also auf einen Zahlenwert abgebildet. Die Maßskala dieser Werte umfasst dabei Werte von eins bis fünf, wobei Fünf eine sehr gute- und Eins eine sehr schlechte Lesbarkeit bedeutet.

Da es sich um subjektive Bewertungen handelt wird jedoch bereits die Objektivität als grundlegendste Voraussetzung einer Metrik nicht erfüllt. Das Ziel muss es also sein, von diesen subjektiven Bewertungen auf andere Faktoren zu schließen, welche eine objektivere und allgemeinere Betrachtung ermöglichen.

Dabei stützt sich diese Arbeit zunächst auf die Annahme von Buse und Weimer, wonach die Lesbarkeit von Quellcode primär von lokalen, zeilenbasierten Eigenschaften, wie z.B. der Länge einer Zeile oder der Anzahl an Schlüsselworten abhängig ist.

Die Schwierigkeit liegt darin zu bestimmen, welche und wieviele dieser Faktoren für die Bewertung der Lesbarkeit einer Programmiersprache berücksichtigt werden sollen. Ziel ist es ein möglichst einfaches Modell zu entwickeln, welches eine hohe Vorhersagekraft besitzt, sich jedoch auf möglichst wenige Faktoren stützt. Die Einbeziehung zu vieler Faktoren führt schnell zu einer hohen Komplexität, welche die spätere Verwertbarkeit des Modells erschwert.

Die unter 3.2 beschriebenen Arbeiten bieten dazu verschiedene Ansätze, welche sich zum Teil überschneiden und ergänzen, jedoch jeweils einen anderen Schwerpunkt setzen. In dieser Arbeit werden einige relevante Faktoren aus diesen Arbeiten, aufgegriffen, kombiniert und um weitere Faktoren ergänzt.

So kann ein Modell entwickelt werden, welches auf objektiv messbaren Werten beruht. Dies ermöglicht die Erfüllung weiterer Anforderungen an eine Metrik. Da es sich um statische Werte handelt, welche sich leicht aus dem Quellcode extrahieren lassen, wird das Kriterium der Zuverlässigkeit erfüllt. Mehrfache Messungen desselben Quellcodes werden stets zum gleichen Ergebnis führen. Ebenso lassen sich verschiedene Messungen in Relation zueinander stellen, womit die Vergleichbarkeit der Ergebnisse gegeben ist.

Die Anforderungen der Verwertbarkeit und Ökonomie werden für dieses vorläufige Modell vorerst nicht in Gänze berücksichtigt.

4.1.3 Identifizierung relevanter Faktoren

Nachdem im vorherigen Abschnitt beschrieben wurde, wie die Anforderungen an das Modell erfüllt werden sollen, wird dies im Folgenden konkretisiert. Dafür wird zunächst geprüft, welche Ergebnisse sich aus den vorgestellten Arbeiten auf das Modell übertragen lassen. Anschließend werden weitere Faktoren betrachtet, welche das Modell potentiell, sinnvoll ergänzen können.

Zeilenbasierte Eigenschaften

Die Beschreibung der Lesbarkeit von Quellcode anhand zeilenbasierter lokaler Eigenschaften im Quellcode hat sich in den bisherigen Arbeiten als sinnvoller Ansatz bewiesen. Buse und Weimer haben dabei den Grundstein gelegt und eine Reihe solcher Eigenschaften auf ihre Relevanz für die Bewertung der Lesbarkeit geprüft. Gemessen wurden zumeist zwei Werte: die durchschnittliche und die maximale Ausprägung einer Eigenschaft pro Zeile.

Die Tabelle 3.1 zeigt die betrachteten Eigenschaften, sowie die gemessene positive bzw. negative Korrelation zur Bewertung der Lesbarkeit.

10 der 18 betrachteten Eigenschaften haben nach den Erkenntnissen von Buse und Weimer eine mittlere bis hohe Auswirkung auf die Bewertung der Lesbarkeit. Diese relevanten Eigenschaften werden für das Modell in Betracht gezogen. Es muss jedoch überprüft werden, ob sie Rückschlüsse auf die Bewertung der Lesbarkeit einer konkreten Programmiersprache zulassen oder eher unabhängig von der eingesetzten Sprache sind. Die Anzahl von Bezeichnern pro Zeile, deren durchschnittliche Länge oder das Vorkommen von leeren Zeilen lassen solche Rückschlüsse z.B. eher schlecht zu.

Für das Modell berücksichtigt wird daher nur ein Teil der Eigenschaften

- **Die durchschnittliche und maximale Länge der Zeilen:** Beide Werte, haben eine starke, negative Auswirkung auf die Bewertung der Lesbarkeit. Zu diesem Schluss kommen sowohl Buse und Weimer als auch Dorn in seiner Arbeit.
- **Die durchschnittliche Anzahl an Schlüsselwörtern:** Diese hat zwar nur eine mittelstarke negative Auswirkung auf die Bewertung der Lesbarkeit, sie steht jedoch klar in Beziehung zur eingesetzten Programmiersprache.
- **Die durchschnittliche und maximale Einrückung des Codes:** Beide Eigenschaften wirken sich mittelstark und negativ auf die Bewertung der Lesbarkeit aus. Sie lassen potentiell Rückschlüsse auf sprachspezifische Konstrukte zu.
- **Die durchschnittliche Anzahl an Punkten:** Das Vorkommen von Punkten hat eine starke, negative Auswirkung auf die Bewertung der Lesbarkeit
- **Die durchschnittliche Anzahl an Klammern:** Auch diese haben eine starke negative Auswirkung auf die Bewertung der Lesbarkeit

Die erwartete Auswirkung der ausgewählten Eigenschaften auf die Bewertung Lesbarkeit wird zunächst so übernommen. In der eigenen Studie wird geprüft werden, ob sich diese Bewertung bestätigt.

Größe und Informationsgehalt

Ein Faktor, welchen Buse und Weimer nicht berücksichtigen ist die Größe eines betrachteten Codeabschnittes. Diese lässt sich z.B. durch die Anzahl der Zeilen, Wörter oder einzelnen Zeichen beschreiben. Sie begründen die Nichtberücksichtigung damit, dass die Größe, nach ihrer Auffassung, eher Auswirkung auf die Komplexität des Quellcodes hat und weniger auf die Lesbarkeit. Die Abgrenzung zur Komplexität möchten sie erhalten.

Diese Auffassung wird von Posnett et al. in ihrer, unter 3.2.2 beschriebenen, Arbeit kritisiert. Sie entwickeln ein Modell, welches, unter anderem die Größe in Form der Anzahl an Zeilen berücksichtigt. Sie zeigen, dass sich durch ihr Modell die Lesbarkeit von Quellcode sogar besser beschreiben lässt, als durch das von Buse und Weimer.

In dieser Arbeit wird daher die Anzahl der Zeilen eines Codeabschnittes für das Modell mit berücksichtigt. So lässt sich auch überprüfen ob die Knappheit von Kotlin im Vergleich zu Java tatsächlich zur von JetBrains erhofften besseren Lesbarkeit der Sprache führt.

In diesem Zusammenhang ebenfalls interessant sind die weiteren Faktoren welche von Posnett et al. berücksichtigt werden. Zum einen das Volumen des Quellcodes nach Halstead, sowie die Entropie auf Zeichenebene. Mit beiden Maßen lässt sich der Informationsgehalt des Quellcodes beschreiben. Sie unterscheiden sich jedoch in der Berechnung. Die Entropie hängt von der relativen Verteilung der Token bzw. Zeichen im Quellcode ab. Eine gleichmäßige Verteilung führt dabei zu einer hohen Entropie, eine stark verzerrte Verteilung zu einer geringen. Das Halstead Volumen beschreibt dagegen die Anzahl Bits, die zur Darstellung der Implementierung nötig sind bzw. die Anzahl gedanklicher Vergleiche, die beim Schreiben des Programmes benötigt werden. Gemessen wird es anhand der Vorkommen von Operatoren und Operanden im Code.

Beide Eigenschaften weisen nach den Erkenntnissen von Posnett et al. eine negative Korrelation zur Bewertung der Lesbarkeit auf. Eine größere Anzahl (verschiedener) Zeichen führt also zu einer schlechteren Lesbarkeit. Auch hier wirkt sich die häufig knappere Schreibweise von Kotlin Quellcode also potentiell positiv aus.

In das vorläufige Modell werden daher beide Eigenschaften zunächst integriert.

Spezifische Eigenschaften für Kotlin

Neben den Eigenschaften, welche bereits in den vorgestellten Modellen berücksichtigt wurden, gibt es einige weitere, welche bei der Bewertung der Lesbarkeit von Kotlin durchaus relevant sein können. Sie lassen eventuell Rückschlüsse auf bestimmte grundlegende Sprachfeatures zu.

Da sich diese Arbeit mit der allgemeinen Bewertung der Lesbarkeit von Quellcode auch mit der konkreten Bewertung der Lesbarkeit von Kotlin beschäftigt, werden einige dieser Eigenschaften separat betrachtet. Für ein allgemeines Modell werden sie jedoch nicht berücksichtigt.

Folgende weitere Eigenschaften werden betrachtet:

- **Durchschnittliche Anzahl an Fragezeichen pro Zeile:** Ein Fragezeichen kennzeichnet in Kotlin einen nullfähigen Typ bzw. wird für die sichere Benutzung von Referenzen dieses Typs genutzt. Die Auswirkung des Vorkommens von Fragezeichen auf die Bewertung der Lesbarkeit, lässt daher potentiell Rückschlüsse auf die Lesbarkeit dieser grundlegenden Eigenschaft zu.
- **Durchschnittliche Anzahl an Typbezeichnungen pro Zeile:** Kotlin und Java sind jeweils statisch typisierte Sprachen. In Kotlin ist es jedoch möglich auf die Typbezeichnung im Quellcode zu verzichten, falls sich dieser aus dem Kontext des Codes zweifelsfrei bestimmen lässt. Durch die Betrachtung der Typbezeichnungen lässt sich potentiell feststellen ob diese Typinferenz eine Auswirkung auf die Bewertung der Lesbarkeit hat.

4.1.4 Entwurf eines vorläufigen Modells

Ein erster Entwurf eines Modells zur Bewertung der Lesbarkeit von Kotlin und Java basiert also zunächst auf einer größeren Anzahl von Faktoren, welche aus dem Code extrahiert werden müssen. Die Tabelle 4.1 zeigt die Eigenschaften, welche vorläufig berücksichtigt werden.

Extrahiert werden diese Eigenschaften aus 70 Code-Ausschnitten. Davon jeweils 35 in Java und 35 in Kotlin. Die Auswahl sowie der Aufbau dieser Ausschnitte wird unter 4.2.4 beschrieben.

Tabelle 4.1: Vorläufige berücksichtige Code-Eigenschaften

Eigenschaft	Erwartete Korrelation zur Lesbarkeit
max. Zeilenlänge	▽
avg. Zeilenlänge	▽
avg. Anzahl Schlüsselwörter pro Zeile	▽
max. Einrückung	▽
avg. Einrückung	▽
avg. Anzahl Punkte pro Zeile	▽
avg. Anzahl Klammern pro Zeile	▽
Zeilenanzahl	△
Anzahl aller Operatoren	-
Anzahl unterschiedlicher Operatoren	-
Anzahl aller Operanden	-
Anzahl unterschiedlicher Operanden	-
Halstead Volumen	▽
Anzahl Vorkommen aller Zeichen	▽
Zeichen-Entropie	▽

▽ zeiget eine negative, △ eine positive Korrelation zur Bewertung der Lesbarkeit an.
Die erwartete Stärke der Auswirkung wird farblich dargestellt:
hoch = ▽, mittel = ▽, nicht bekannt = ▽

Die Anzahl der Operatoren und Operanden wird für die Berechnung des Halstead Volumens benötigt[1], die Anzahl der Vorkommen aller Zeichen für die Berechnung der Entropie auf Zeichenebene.

Neben den voraussichtlich allgemein verwendbaren Eigenschaften, werden zur genaueren Bewertung der Lesbarkeit von Kotlin, zusätzlich die vorgestellten, spezifischen Eigenschaften aus dem Quellcode extrahiert. Eine Korrelation mit der Bewertung der Lesbarkeit ist hier nicht bekannt sondern wird lediglich vermutet.

[1] Da es, wie unter 3.2.2 beschrieben, keine eindeutige Definition zur Bestimmung von Operatoren und Operanden in konkreten Sprachen wie Java oder Kotlin gibt, orientiert sich diese Arbeit an einigen gängigen Ansätzen zu Zählstrategien [HalsteadCounter][VirtualMachinery][Abbas, 2010].

4.2 Vergleichende Studie zur Lesbarkeit von Kotlin und Java

Eine vergleichende Studie zur Lesbarkeit von Kotlin und Java Quellcode stellt einen wichtigen Teil dieser Arbeit dar. Durchgeführt wurde sie, in Form einer vierwöchigen Online-Umfrage, bei der CoreMedia AG sowie an der Hochschule für angewandte Wissenschaften Hamburg (HAW).

4.2.1 Ziel

Mit der Studie werden zwei Ziele verfolgt. Sie dient zum einen zur Untersuchung der konkreten Lesbarkeit von Kotlin im Vergleich zu Java, zum anderen zur Validierung des zuvor entwickelten, vorläufigen Modells zur Lesbarkeit von Quellcode.

Zunächst wird also allgemein betrachtet, wie die Teilnehmer die Lesbarkeit von Kotlin im Vergleich zu Java bewerten. Berücksichtigt werden dabei eventuelle Unterschiede in der Bewertung zwischen den studentischen Teilnehmern und den Mitarbeitern der CoreMedia AG. Auch eine eventuelle Auswirkung der Entwicklungserfahrung auf die Bewertung wird mit einbezogen.

Die Bewertung der Lesbarkeit ist natürlich nicht ausreichend um über den Einsatz von Kotlin bei CoreMedia zu entscheiden. Dennoch wird auf Grundlage dieser Bewertung eine eingeschränkte Empfehlung für bzw. gegen den Einsatz von Kotlin bei CoreMedia gegeben. Warum die Lesbarkeit ein wichtiges Kriterium bei der Einscheidung über den Einsatz einer Sprache ist, wurde bereits einleitend unter 1.1.2 erläutert.

Der Vergleich mit Java erscheint sinnvoll, da Kotlin sich in vielen Bereichen eng an Java orientiert. Zudem ist Java die derzeit vorwiegend eingesetzte Programmiersprache bei der CoreMedia AG. Ein Vergleich kann also sehr gut zeigen, welche Vor- und Nachteile die Nutzung von Kotlin bringen kann.

Nach der allgemeinen Betrachtung der Lesbarkeit, wird im Anschluss geprüft wie die Bewertungen der Studienteilnehmer zustande kommen, also welche Eigenschaften im Quellcode der Code-Ausschnitte dabei ein Rolle spielen. Betrachtet werden alle Eigenschaften, welche im zuvor entwickelten Modell zur Lesbarkeit berücksichtigt werden (siehe Tabelle 4.1). So lässt sich die Relevanz der Eigenschaften für ein solches Modell überprüfen und

die erwartete Auswirkung, welche in anderen Studien festgestellt wurde, kann im besten Fall bestätigt werden.

Abschließend wird das Modell auf Grundlage dieser Erkenntnisse angepasst und verbessert.

Separat betrachtet werden zudem die unter 4.1.3 beschriebenen Eigenschaften, welche potentiell Erkenntnisse über die Auswirkung einiger Kotlin Sprachfeatures auf die Lesbarkeit der Sprache ermöglichen.

4.2.2 Aufbau

Den Kern der Studie bilden 70 Code-Ausschnitte. Davon sind 35 in Java und 35 in Kotlin implementiert. Jeweils ein Java und ein Kotlin Ausschnitt beschreiben dabei dieselbe Funktionalität. Die Aufgabe der Teilnehmer ist es, diese Ausschnitte nach ihrem subjektiven Empfinden von Lesbarkeit auf einer Skala von 1 (sehr schlecht lesbar) bis 5 (sehr gut lesbar) zu bewerten. Dabei werden ihnen die funktionsgleichen Implementierungen in Java und Kotlin jeweils zeitgleich angezeigt (sieht Abb. 4.1).

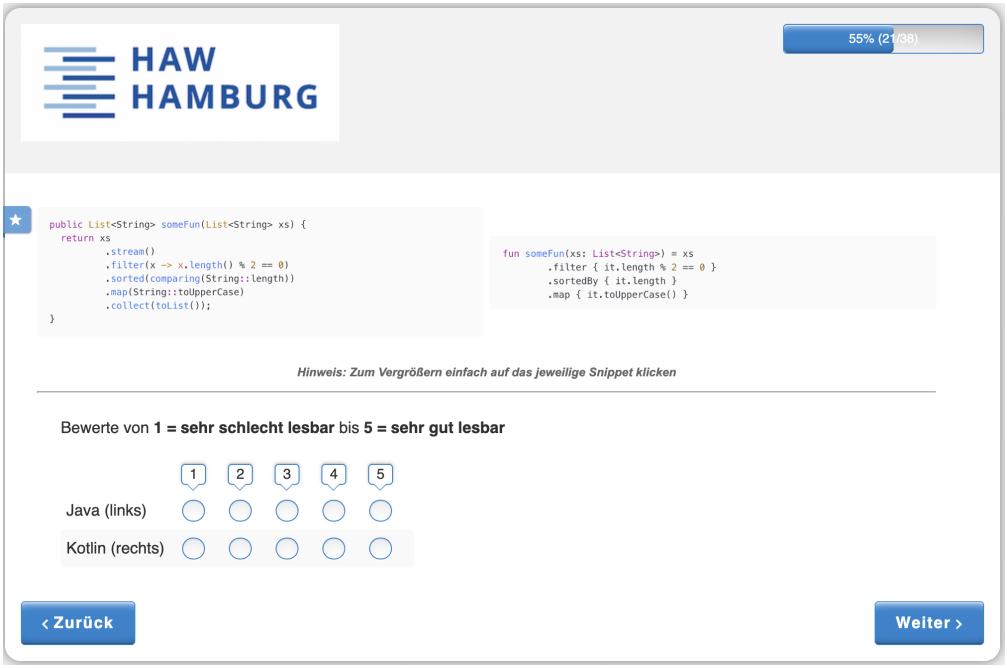

Abbildung 4.1: Beispiel der Bewertung zweier funktionsgleicher Code-Ausschnitte

Die zeitgleiche Anzeige soll dem Teilnehmer den Vergleich beider Lösungen erleichtern und so eine fundiertere Bewertung der Lesbarkeit ermöglichen. Würden zwei funktionsgleiche Ausschnitte nacheinander angezeigt werden, fehlt dem Teilnehmer gerade beim betrachten des ersten Ausschnitts die Relation.

Zusätzlich zu den Bewertungen der Code-Ausschnitte, werden die Teilnehmer zu ihren Erfahrungen in der Softwareentwicklung allgemein sowie ihren Erfahrungen mit Kotlin und Java befragt. Dafür gibt es zwei Gründe. Zum einen zeigt Dorn in seiner, unter 3.2.3 vorgestellten Arbeit, dass sich die Arbeitserfahrung der Studienteilnehmer auf die Bewertung der Lesbarkeit auswirkt, zum anderen wird angenommen, dass die Teilnehmer über deutlich bessere Kenntnisse in Java, als in Kotlin verfügen.

Bei der CoreMedia AG wird derzeit hauptsächlich in Java programmiert, an der HAW Hamburg durchlaufen alle Studierenden der Informatik in den ersten beiden Semestern mindestens eine Pflichtveranstaltung zu Java. Zudem ist Kotlin im Vergleich zu Java eine sehr junge Sprache. Die Wahrscheinlichkeit als Entwickler bereits Erfahrung mit Kotlin gesammelt zu haben, ist also auch abseits der HAW oder der CoreMedia AG deutlich geringer.

In der Folge wird angenommen, dass die Teilnehmer Java potentiell als besser lesbar bewerten, da sie mit der Syntax und den Konstrukten der Sprache vertrauter sind. Um diesen erwarteten Fehler in den Ergebnissen der Studie zumindest sichtbar zu machen, wird in der Auswertung die Erfahrung der Teilnehmer berücksichtigt. Zusätzlich wird den Teilnehmern in einer kurzen Ausführung der Umgang mit Nullreferenzen im Typsystem von Kotlin erläutert um die Hürde für das Verständnis einiger Code-Ausschnitte zu verringern. In dieser Arbeit werden dieser und weitere Aspekte in Kapitel 2 etwas ausführlicher erläutert.

4.2.3 Teilnehmer

Die Teilnehmer einer Studie sind ein kritischer Aspekt wenn es um die Einordnung der Ergebnisse geht. Dies gilt gerade dann, wenn die Ergebnisse der Studie auch auf den subjektiven Eindrücken der Teilnehmer beruhen. Um in der Auswertung der Ergebnisse von diesen subjektiven Eindrücken auf messbare, objektive Faktoren zu schließen, ist eine gewisse Anzahl an Teilnehmern erforderlich. Je höher die Anzahl, desto eher erfüllen die Ergebnisse das Kriterium der Objektivität und können als allgemeingültig bzw. repräsentativ betrachtet werden.

In dieser Studie stellt die Bewertung der Lesbarkeit der Code-Ausschnitte durch die Teilnehmer einen solchen subjektiven Aspekt dar.

Ein weiterer wichtiger Aspekt ist die Größe der Datengrundlage der Studie. Einen großen Teil dieser Datengrundlage bilden in dieser Studie die ausgewählten Code-Ausschnitte, welche durch die Teilnehmer bewertet werden. Auch hier gilt: eine höhere Anzahl zu bewertender Ausschnitte erhöht die Wahrscheinlichkeit der Allgemeingültigkeit der Ergebnisse.

Diese beiden Faktoren stehen sich in gewisser Weise gegenüber. Eine größere Anzahl an zu bewertenden Code-Ausschnitten führt dazu, dass sich der zeitliche Aufwand für die Teilnehmer der Studie erhöht. Da die Teilnahme freiwillig und nicht vergütet ist, erhöht sich durch einen höheren Zeitaufwand die Wahrscheinlichkeit, dass weniger Teilnehmer die Studie bis zum Ende durchführen bzw. überhaupt daran teilnehmen.

Die Anzahl der zu bewertenden Ausschnitte und damit die Länge der Studie ist also ein Kompromiss zwischen einer belastbaren Datengrundlage und einer ausreichenden Anzahl an Teilnehmern. Um einen weiteren Anreiz für die Teilnahme zu schaffen wurde den Teilnehmern zudem die Möglichkeit geboten, auf freiwilliger Basis an der Verlosung einiger Gutscheine im Wert von über 50 Euro teilzunehmen.

Die angestrebte Mindestanzahl von 50 Teilnehmern konnte so erreicht und sogar überschritten werden. Ingesamt nahmen 65 Personen an der Studie Teil, von welchen 45 diese auch vollständig durchführten. Die Abbruchquote liegt somit bei 31%. Die Teilnehmer setzen sich dabei aus 34 Beschäftigten der CoreMedia AG, 29 Studierende sowie 2 Professoren der HAW Hamburg zusammen.

Die Erfahrung der Teilnehmer in der Softwareentwicklung variiert stark, wie sich in Abbildung 4.2 ablesen lässt. Für die spätere Auswertung der Studie werden die dargestellten Gruppen auf drei Gruppen reduziert.

- Weniger als drei Jahre Erfahrung
- Drei bis zehn Jahre Erfahrung
- Mehr als zehn Jahre Erfahrung

Die Verteilung auf diese Gruppen unterscheidet sich bei separater Betrachtung der studentischen Teilnehmer sowie der Beschäftigen der CoreMedia AG, erwartungsgemäß relativ stark.

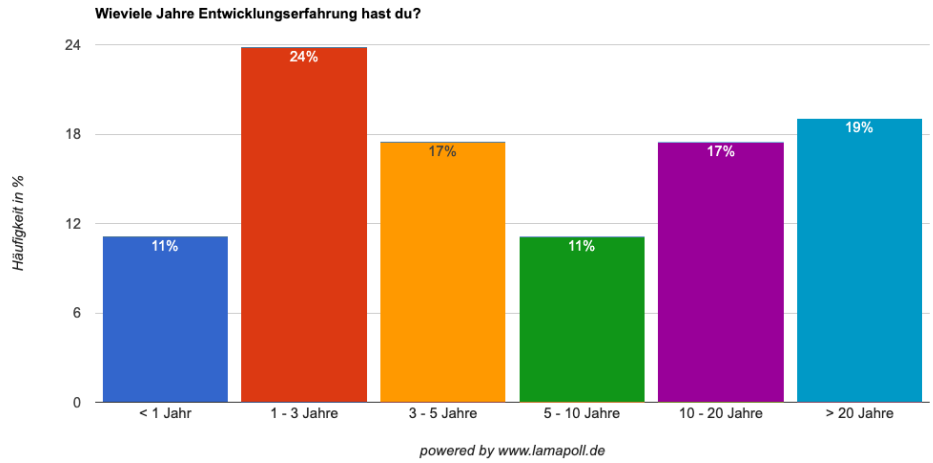

Abbildung 4.2: Erfahrung der Teilnehmer in der Softwareentwicklung

Die Entwicklungserfahrung bei den Studierenden liegt zu 67% bei bis zu drei, zu 29% bei bis zu zehn und lediglich zu 4% bei über zehn Jahren. Bei den Beschäftigten der CoreMedia AG zeichnet sich ein gegensätzliches Bild. Lediglich 6% verfügen über weniger als drei, 29% über drei bis zehn Jahre und 64% über mehr als zehn Jahre Erfahrung.

Ebenfalls große Unterschiede gibt es, wie erwartet, in der Erfahrung der Teilnehmer mit Kotlin und Java (siehe Abb. 4.3 und Abb. 4.4). So haben alle Teilnehmer Kenntnisse in Java, wobei 40% sogar über mindestens fünf Jahre Erfahrung verfügen. Immerhin 67% der Teilnehmer nutzen zudem regelmäßig Sprachfeatures, wie z.B. Lambdas oder Streams, welche mit Java 8 oder später eingeführt wurden. Diese sind in ihrer syntaktischen Struktur zum Teil sehr nah an Kotlin. Ein Verständnis dieser Eigenschaften kann also das Verständnis von Kotlin Quellcode erleichtern.

In Kotlin sind dagegen, wie bereits angenommen, beim Großteil der Teilnehmer keine Erfahrungen vorhanden. Lediglich 14% geben an, zumindest über geringe Erfahrungen in Kotlin zu verfügen. Inwieweit sich dies auf die Bewertung der Code-Ausschnitte auswirkt, wird in der Auswertung der Studie untersucht. Anders als erwartet zeigt sich hier auch kein nennenswerter Unterschied zwischen den studentischen Teilnehmern und den Mitarbeitern der CoreMedia AG.

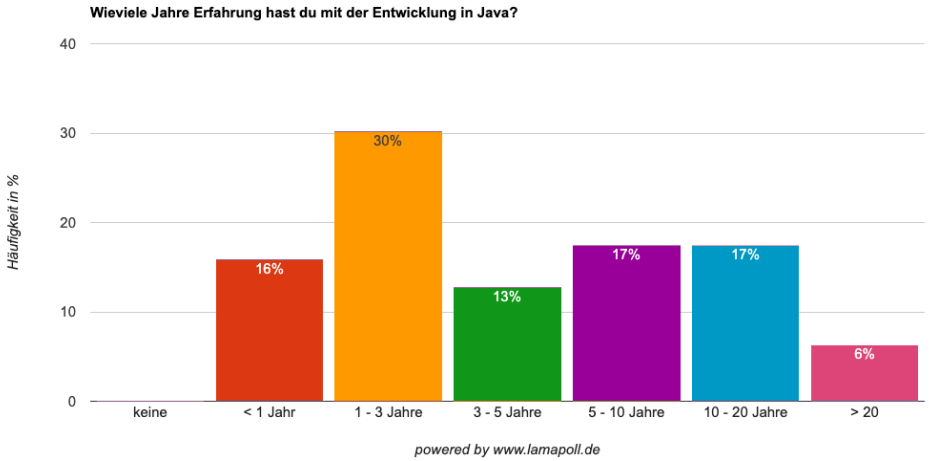

Abbildung 4.3: Erfahrung der Teilnehmer mit der Entwicklung in Java

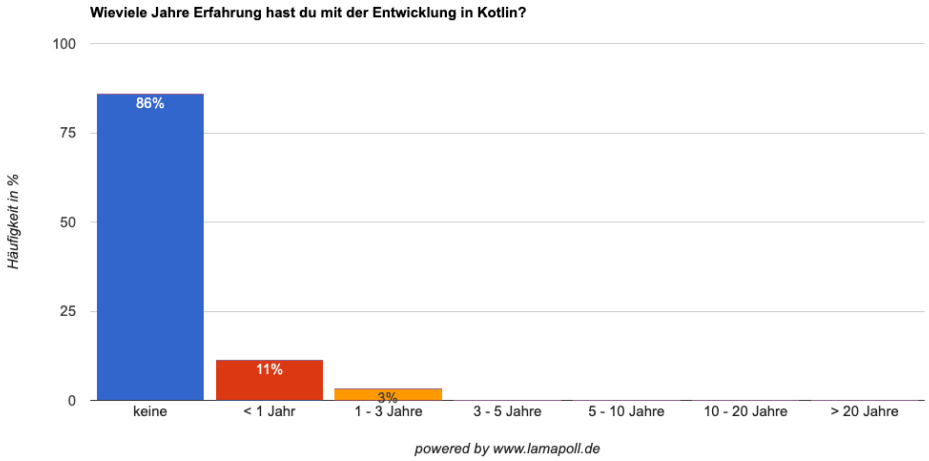

Abbildung 4.4: Erfahrung der Teilnehmer mit der Entwicklung in Kotlin

4.2.4 Auswahl der Code-Ausschnitte

Die 70 Code-Ausschnitte, welche durch die Teilnehmer nach ihrer Lesbarkeit bewertet werden, bilden den kritischen Teil der Datengrundlage für die Studie. Die Anzahl ist dabei, wie bereits unter 4.2.3 beschrieben, ein Kompromiss zwischen einer möglichst großen Datengrundlage und der Dauer, welche die Teilnehmer für die Studie benötigen.

Die Ausschnitte müssen den Teilnehmern dabei soweit wie möglich einen umfassenden Eindruck über die Lesbarkeit von Kotlin und Java vermitteln sowie einen fairen Vergleich der beiden Sprachen ermöglichen. Anders als in den vorgestellten Studien können sie also nicht einfach aus anderen Projekten extrahiert werden sondern müssen eigens für die Studie implementiert werden. Dabei wurde darauf geachtet, dass sie ohne das Wissen über den eventuellen Kontext einfach zu verstehen sind. Eine zu große Komplexität der Ausschnitte würde dazu führen, dass statt der Lesbarkeit, die Komplexität das ausschlaggebende Kriterium für die Bewertung ist.

Ein Code-Ausschnitt stellt immer einen logisch zusammenhängenden Block dar. Dies Kann z.B. eine Funktion, mehrere, inhaltlich zusammenhängende Funktionen oder eine Klasse sein. In einigen Ausschnitten ist zudem ein einfacher Aufruf von implementierten Funktion enthalten.

Da die Ausgangslage eine Evaluation darüber ist, ob Kotlin aufgrund seiner Lesbarkeit bei der CoreMedia AG eingesetzt werden soll, wird besonderer Wert darauf gelegt die, in Kapitel 2 vorgestellten, grundlegenden Eigenschaften und Funktionen der Sprache in den Ausschnitten zur Geltung kommen zu lassen. Dazu zählen z.B. der Umgang mit Nullreferenzen im Typsystem, die Herleitung von Typen aus dem umgebenden Kontext oder der Einsatz von String Templates. Ebenso berücksichtigt wurde zudem das Arbeiten mit Collections, da die Kotlin Standard Library hier deutlich mehr Funktionalität und Unterscheidung bietet als Java.

Die Einbeziehung dieser Eigenschaften und Funktionen lässt in der Auswertung der Ergebnisse eventuell Rückschlüsse darauf zu, wie sich einige Sprachfeatures auf die Lesbarkeit der Sprache auswirken.

Durch die Berücksichtigung vieler grundlegender Kotlin Features besteht jedoch die Gefahr, dass die ausgewählten Code-Ausschnitte keinen fairen Vergleich der Sprachen ermöglichen. Die Umsetzung vieler Features in Kotlin beruht auf den Erfahrungen, welche die Entwickler bei JetBrains mit der Entwicklung in Java gesammelt haben [Jemerov

und Isakova, 2017]. Somit kann angenommen werden, dass deren Umsetzung in Kotlin häufig eine Verbesserung zu Java darstellt.

Um dieser Gefahr entgegen zu wirken, sind in den Ausschnitten neben grundlegenden Idiomen beider Sprachen auch standard Such- und Sortieralgorithmen sowie verbreitete Entwurfsmuster umgesetzt.

Neben den grundlegenden Eigenschaften von Kotlin soll auch die Knappheit der Sprache für die Teilnehmer ersichtlich werden. Auch hier lässt sich in der Auswertung prüfen, ob diese sich tatsächlich wie von Jemerov und Isakova angenommen, positiv auf die Lesbarkeit auswirkt oder aber ob sich die Erkenntnisse von Posnett et al. bestätigen, wonach die Größe des betrachteten Ausschnitts eine positive Korrelation zur Bewertung der Lesbarkeit aufweist. Anders als in der Studie von Buse und Weimer variiert die Länge der Ausschnitte daher deutlich.

Die durchschnittliche Länge der Code-Ausschnitte liegt bei 13 Zeilen wobei der kürzeste Ausschnitt lediglich eine Länge von zwei Zeilen hat, während der längste Ausschnitt eine Länge von 46 Zeilen aufweist. Kotlin Ausschnitte haben eine durchschnittliche Länge von 10,1 Zeilen und sind somit im Schnitt 36,5% kürzer als Java Ausschnitte, welche im Schnitt 15,9 Zeilen lang sind.

Eine weitere Schwierigkeit stellt die Vergleichbarkeit der jeweils funktionsgleichen Code-Ausschnitte dar. Für beide Sprachen gibt es jeweils viele verschiedene Möglichkeiten die gewünschte Funktionalität zu implementieren. Eine möglichst äquivalente und faire Implementierung in beiden Sprachen kann daher nur nach besten Wissen und Gewissen erfolgen. Um sich hier nicht auf die Einschätzungen eines Einzelnen zu verlassen, wurden alle Ausschnitte mehrfach durch erfahrenen Entwickler überprüft und gegebenenfalls modifiziert.

4.3 Auswertung der Studie und Validierung des Modells

Im folgenden Abschnitt werden die Ergebnisse der Studie zur Lesbarkeit von Kotlin und Java ausgewertet und diskutiert. Aufgrund der eher geringen Teilnehmeranzahl werden dabei auch unvollständige Rückläufe mit einbezogen. Das bedeutet, die Datengrundlage nimmt für die im Fragebogen weiter hinten platzierten Code-Ausschnitte von 65 auf bis zu 45 Teilnehmer, also um bis zu 31% ab.

Zunächst werden die Bewertungen der Code-Ausschnitte allgemein betrachtet. Dabei werden die Ergebnisse der studentischen Teilnehmer und der Mitarbeiter der CoreMedia AG gegenübergestellt und verglichen. Ebenso wird untersucht ob und wenn ja, in welcher Form sich die Software-Entwicklungserfahrung der Teilnehmer auf die Bewertung auswirkt.

Anschließend werden diese Betrachtungen separat für Kotlin und Java Ausschnitte durchgeführt um einen Vergleich der Bewertung beider Sprachen zu ermöglichen. Ergänzend wird hier die Auswirkung der Erfahrung in der Entwicklung mit Java und Kotlin auf die Bewertung der Lesbarkeit untersucht.

Nach dieser allgemeineren Auswertung der Ergebnisse, wird anschließend das entwickelte Modell zur Lesbarkeit anhand der Ergebnisse validiert. Dafür wird die Relevanz aller berücksichtigten Eigenschaften im Einzelnen überprüft und das Modell dementsprechend überarbeitet. Neben den allgemeinen Faktoren werden auch weitere Kotlin spezifische Eigenschaften überprüft, welche ggf. Rückschlüsse auf die Lesbarkeit einzelner Kotlin Sprachfeatures ermöglichen.

4.3.1 Allgemeine Bewertung der Lesbarkeit

Bei der Betrachtung der Studienergebnisse wird deutlich, dass die ausgewählten Code-Ausschnitte als eher gut lesbar bewertet werden, wie in Abbildung 4.5 abzulesen ist. Die durchschnittliche Bewertung der Lesbarkeit liegt bei 3,79, wobei eine durchschnittliche Standardabweichung von 0,78 eine hohe Übereinstimmung der Bewertungen untereinander anzeigt. Die durchschnittlich niedrigste Bewertung liegt bei 2,72, die höchste bei 4,57.

Eine durchschnittlich hohe Übereinstimmung in den Bewertungen der Teilnehmer zeigt, dass diese ein ähnliches Verständnis von Lesbarkeit haben. Dies lässt vermuten, dass einige Faktoren im Code, welche die Bewertungen beeinflussen, durchaus eine gewisse Allgemeingültigkeit besitzen können. Es bestärkt zudem die Hypothese von Buse und Weimer, wonach Software Entwickler eine ähnliche, intuitive Vorstellung von Lesbarkeit besitzen [Buse und Weimer, 2010].

Ein Vergleich der beiden Teilnehmergruppen, den Studierenden der HAW Hamburg sowie den Mitarbeitern der CoreMedia AG zeigt, dass die Übereinstimmung der Bewertungen zwischen den beiden Gruppen sehr hoch ist.

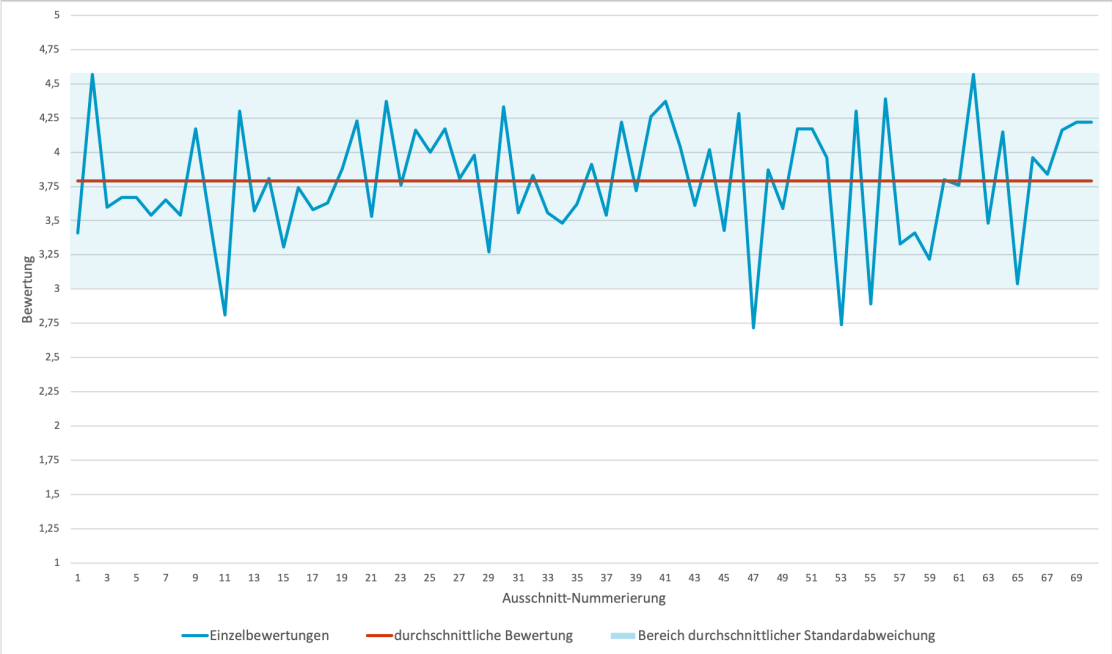

Abbildung 4.5: Allgemeine Bewertung der Code-Ausschnitte

Dies zeigt zum einen die durchschnittliche Bewertung, welche bei den Studierenden bei 3,84 und bei den Mitarbeitern der CoreMedia AG bei 3,72 liegt sowie der Verlauf der Kurven in Abbildung 4.6.

Die durchschnittliche Bewertung beider Gruppen unterscheidet sich also lediglich um 0,12, wobei die studentischen Teilnehmer in ihrer Bewertung leicht über dem allgemeinen Durchschnitt und die Mitarbeiter der CoreMedia AG leicht unter dem Durchschnitt liegen. Die Übereinstimmung der Bewertungen innerhalb der Gruppen ist ähnlich hoch wie bei der allgemeinen Betrachtung, mit einer durchschnittlichen Standardabweichung von 0,78 bei den Studierenden bzw. 0,74 bei den Mitarbeitern.

Diese hohe Übereinstimmung zwischen beiden Gruppen ist interessant, da sie zeigt, dass die studentischen Teilnehmer, welche ihre Erfahrungen zum großen Teil aus den entsprechenden Lehrveranstaltungen und Projekten erhalten, eine durchaus ähnliche Vorstellung von Lesbarkeit besitzen wie Teilnehmer, welche Programmierung als Werkzeug in der täglichen Arbeit nutzen und bei denen die entsprechende theoretische Ausbildung zum Teil schon weit zurückliegt.

Eine leichte Tendenz lässt sich bei der Gruppierung der Ergebnisse nach der Erfahrung der Teilnehmer erkennen. Mit zunehmender Erfahrung verschlechtert sich die Bewertung der

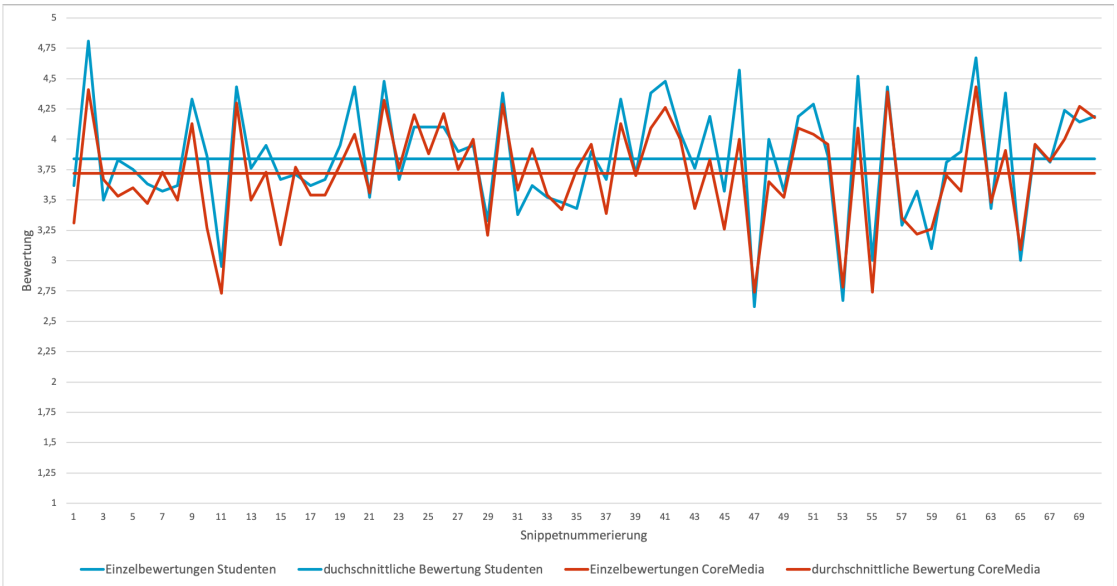

Abbildung 4.6: Bewertung der Code-Ausschnitte, gruppiert nach Studierenden und Mitarbeitern der CoreMedia AG

Code-Ausschnitte. Zwar herrscht weiterhin eine große Übereinstimmung in der Tendenz der Bewertung, Bei der Betrachtung von Abbildung 4.7 ist jedoch zu erkennen, dass gerade Teilnehmer mit mehr als zehn Jahren Erfahrung in der Softwareentwicklung, die Lesbarkeit der Ausschnitte schlechter bewerten als Teilnehmer mit weniger Erfahrung. Im Vergleich mit Teilnehmern, welche weniger als 3 Jahre Erfahrung vorweisen können, bewerten sie im Durchschnitt immerhin 3% schlechter.

Die Grafik zeigt, dass gerade erfahrene Teilnehmer die schlechter lesbaren Code-Ausschnitte deutlich negativer bewerten als die weniger erfahrenen. Andersherum werden besonders gut lesbare Ausschnitte von Teilnehmern mit wenig Erfahrung noch etwas positiver bewertet als durch erfahrene Teilnehmer. Daraus lässt sich schließen, dass bei erfahrenen Entwicklern die Toleranz gegenüber schlecht lesbarem Code zum Teil deutlich geringer ausfällt. Gleichzeitig sind sie von vermeintlich gut lesbaren Code Konstrukten weniger stark beeindruckt als die weniger erfahrenen Entwickler.

Die wichtigste Erkenntnis, welche sich aus der allgemeinen Betrachtung der Bewertungen ergibt ist jedoch, dass über die verschiedenen Teilnehmergruppen hinweg, insgesamt eine große Übereinstimmung im Verständnis von Lesbarkeit existiert.

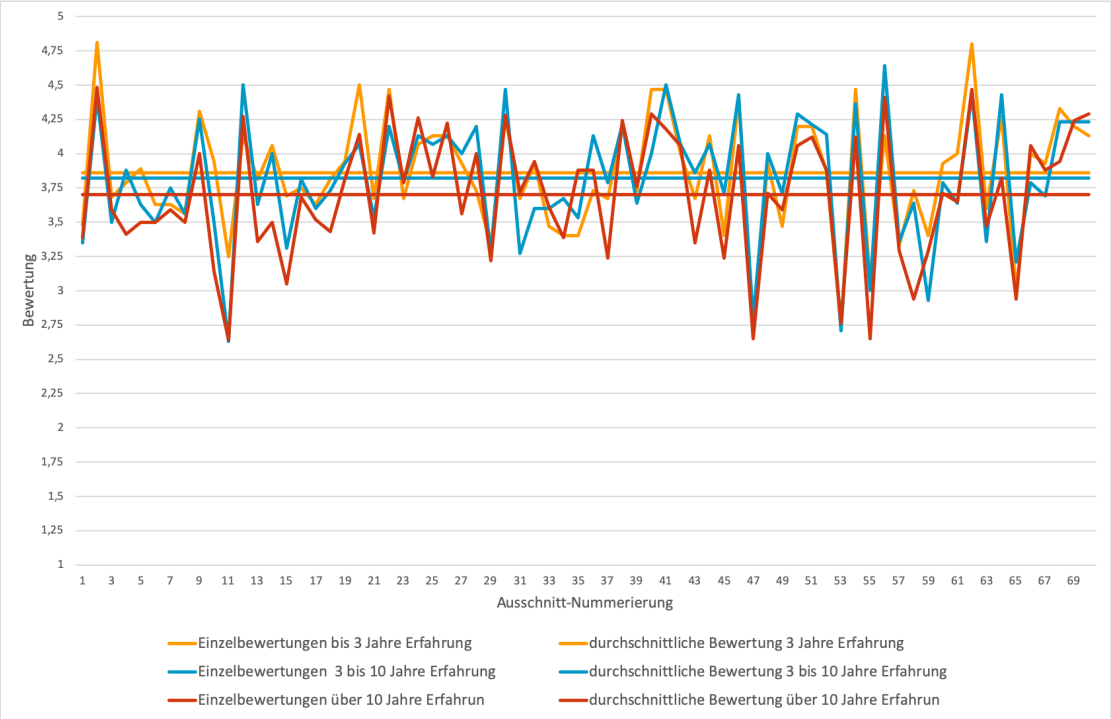

Abbildung 4.7: Bewertung der Code-Ausschnitte, gruppiert nach Erfahrung der Teilnehmer

4.3.2 Bewertung von Java und Kotlin

Bei der separaten Betrachtung der Bewertungen von Kotlin und Java Ausschnitten (siehe Abb. 4.8) zeigt sich, dass die Bewertung von Kotlin mit durchschnittlich 4,01 deutlich besser ausfällt als die Bewertung der Java Ausschnitte, welche im Schnitt mit 3,56 bewertet wurden. Die Lesbarkeit von Kotlin wurde also durchschnittlich 9% besser bewertet.

Die sehr hohe Bewertung der Kotlin Ausschnitte legt nahe, dass Kotlin Quellcode, wie von JetBrains erhofft und angenommen, über eine sehr gute Lesbarkeit verfügt, welche zudem deutlich besser ausfällt als die Lesbarkeit von Java Quellcode.

Zudem muss berücksichtigt werden, dass 86% der Studienteilnehmer über keinerlei Kenntnisse in Kotlin verfügen, während Erfahrungen in Java bei allen Teilnehmern vorhanden sind. Die deutlich bessere Bewertung von Kotlin, trotz der augenscheinlichen Vertrautheit der Teilnehmer mit Java, wertet diese Aussage nochmals deutlich auf.

Bei der separaten Betrachtung beider Teilnehmergruppen zeigt sich ein ähnliches Bild,

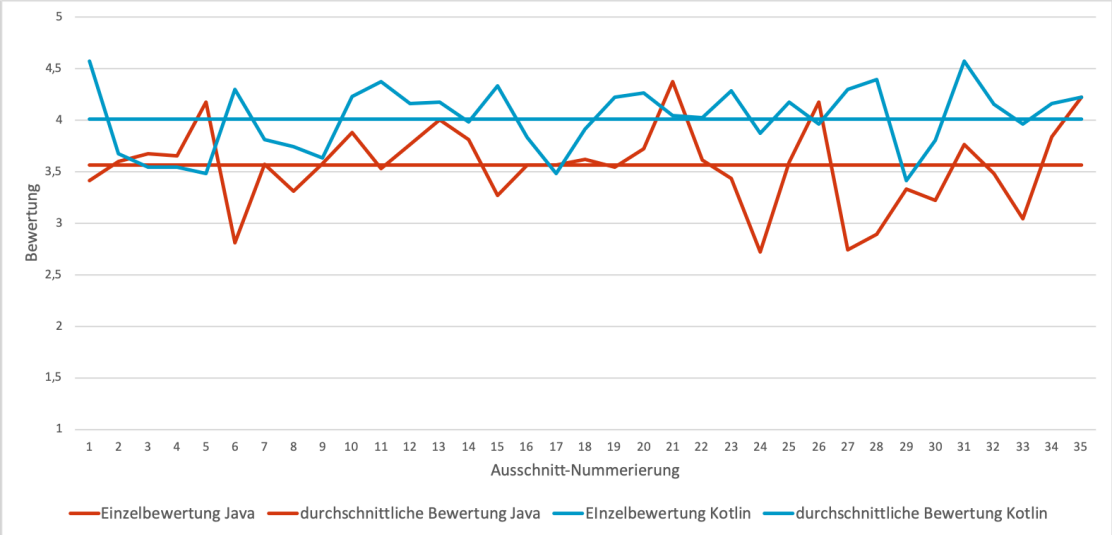

Abbildung 4.8: Bewertungen der Code-Ausschnitte, gruppiert nach Kotlin und Java

wie beim allgemeinen Vergleich dieser Gruppen. Die Studierenden bewerten beide Sprachen durchschnittlich noch einmal etwas besser als die Mitarbeiter der CoreMedia AG. In der Tendenz liegen beide Gruppen jedoch nah beieinander, wobei Kotlin von beiden Teilnehmergruppen als deutlich lesbarer eingestuft wird als Java (siehe Abb. 4.9)

Sehr interessant gestaltet sich die Betrachtung der Bewertungen bei der Gruppierung der Teilnehmer nach ihren Erfahrungen mit Kotlin bzw. Java.

Die Daten in Abbildung 4.10 bestätigen die Annahme, wonach Teilnehmer eine Sprache mit der sie vertraut sind besser bewerten als eine ihnen unbekannte Sprache. Kotlin wird zwar dennoch über alle Teilnehmergruppen hinweg als deutlich lesbarer eingestuft, die Bewertung verschlechtert sich jedoch mit zunehmender Erfahrung in Java. Die Bewertung der Java Ausschnitte steigt dagegen zunächst an und verschlechtert sich erst bei über zehnjähriger Erfahrung in Java, jedoch weniger stark als die von Kotlin.

Ein ähnliches Bild zeigt sich beim Vergleich der Bewertungen von Teilnehmern mit und ohne Kotlin Kenntnissen. Die Bewertung der Lesbarkeit von Kotlin fällt bei Teilnehmern mit Kotlin Kenntnissen um 6,8% besser aus als bei Teilnehmern welche über keine Kotlin Kenntnisse verfügen. Die Bewertung von Java unterscheidet sich dagegen bei beiden Gruppen um lediglich 2,6%.

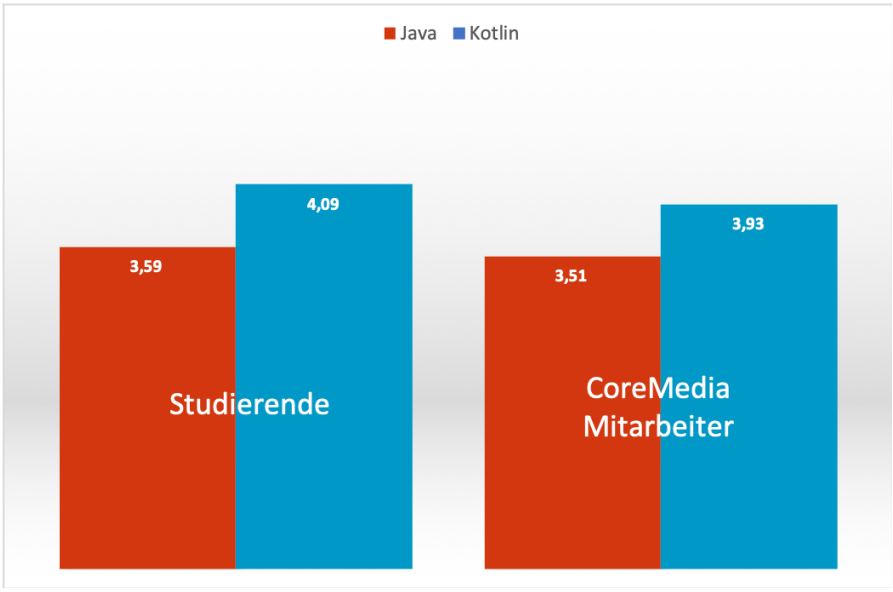

Abbildung 4.9: Durchschnittliche Bewertung von Kotlin und Java, gruppiert nach Studierenden und Mitarbeitern der CoreMedia AG

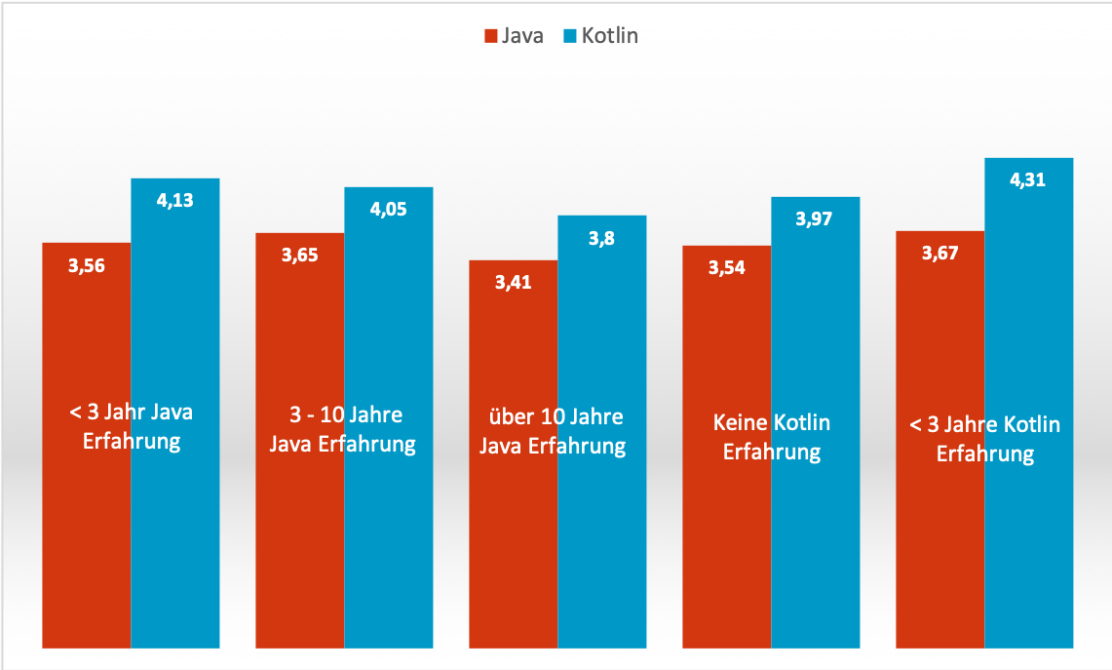

Abbildung 4.10: Durchschnittliche Bewertung von Kotlin und Java, gruppiert nach Erfahrung in der jeweiligen Sprache

4.3.3 Überprüfung des vorläufigen Modells zur Lesbarkeit

Nachdem die bisherige Auswertung sich auf die Bewertungen der Studienteilnehmer beschränkt, wird im Folgenden untersucht wie diese Bewertungen zustande kommen.

Das zuvor entwickelte Modell stellt eine Hypothese darüber dar, welche Eigenschaften im Quellcode einen Einfluss auf dessen Lesbarkeit haben. Durch die Überprüfung der für das Modell gewählten Faktoren, kann überprüft werden inwieweit es sich bei dem Modell um einen validen Ansatz zur Bewertung der Lesbarkeit von Quellcode handelt. Anhand der gewonnen Erkenntnisse kann das Modell entsprechend modifiziert und verbessert werden.

Um die Auswirkung der einzelnen Eigenschaften auf die Bewertung der Lesbarkeit zu beschreiben wird geprüft ob eine Korrelationen zwischen den Bewertungen der Code-Ausschnitte und der Ausprägung der betrachteten Eigenschaft vorhanden ist. Anhand der Stärke einer Korrelation lässt sich feststellen wie stark die Auswirkung der Eigenschaft auf die Bewertung der Lesbarkeit ausfällt.

Spearman Korrelation

Zur Bestimmung der Korrelationen wird Spearmans Rang Korrelationskoeffizient genutzt. Mit diesem lässt sich die monotone Beziehung zwischen zwei kontinuierlichen oder ordinalen Variablen beschreiben [Sprent, 2001].

Anders als die ebenfalls häufig genutzte Pearson Korrelation, mit welcher sich die lineare Beziehung zwischen kontinuierlichen Variablen beschreiben lässt, wird die Spearman Korrelation nicht direkt auf den Variablen angewendet. Es wird stattdessen eine Rangfolge für beide Variablen erzeugt, auf welchen die Berechnung der Korrelation stattfindet. Der Spearman Korrelationskoeffizient eignet sich daher gut zur Beschreibung von Beziehungen, wenn keine lineare Beziehung zwischen den Variablen angenommen wird.

Da die Beziehungen zwischen den Bewertungen der Code-Ausschnitte und der Ausprägung bestimmter Eigenschaften im Code voraussichtlich nicht linear sind, ist Spearmans Korrelationskoeffizient also sehr gut geeignet um diese zu beschreiben.

Die Werte des Korrelationskoeffizienten liegen dabei zwischen -1 und 1, wobei 1 eine perfekte positive und -1 eine perfekte negative Korrelation anzeigt. Nimmt er einen Wert nahe 0 an, ist keine Korrelation zwischen den Variablen gegeben.

Für die Interpretation der Werte zwischen 0 und 1 bzw. -1 gibt es keine einheitliche Vorgabe. Diese Arbeit orientiert sich an der Konvention von Jacob Cohen [Hemphill, 2003]

- 0.1 = geringe Korrelation
- 0.3 = mittlere Korrelation
- 0.5= hohe Korrelation

Der Spearman Korrelationskoeffizient wird in der Folge auch mit r_{Sp} angegeben.

Allgemeine Betrachtung der Korrelationen

Die Auswertung der Beziehungen zwischen den Bewertungen der Code-Ausschnitte und den gewählten Code-Eigenschaften zeigt, dass nur ein Teil dieser Eigenschaften eine Korrelation zur Bewertung der Lesbarkeit aufweist (siehe Abb. 4.11). Lediglich drei der zehn betrachteten Eigenschaften weisen eine niedrige bis mittlere Korrelation auf.

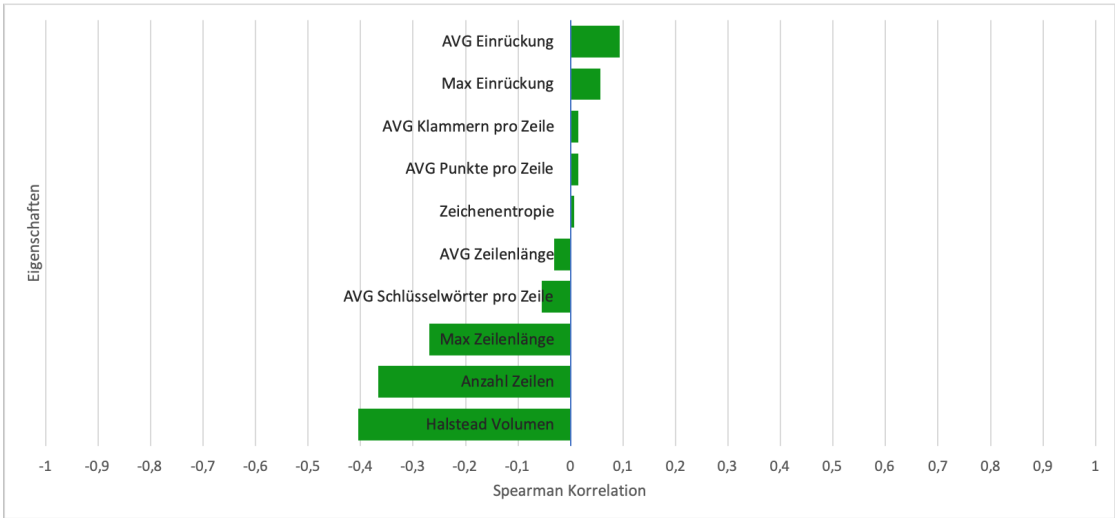

Abbildung 4.11: Korrelationen zwischen den Bewertungen der Teilnehmer und den extrahierten Eigenschaften

Halstead Volumen: Den höchsten Spearman Korrelationskoeffizienten besitzt die Beziehung der Lesbarkeit zu Halsteads Volumen. Mit einem Wert von -0.4 weist sie eine mittlere, negative Korrelation auf. Eine höhere Anzahl an (verschiedenen) Operatoren und Operanden führt also tendenziell zu einer Verschlechterung der Lesbarkeit des Quellcodes. Dies bestätigt die Erkenntnisse welche Posnett et al. aus ihrer Studie gewonnen haben (siehe 3.2.2).

Da das Halstead Volumen neben den verschiedenen Operatoren und Operanden auch deren Gesamtanzahl berücksichtigt, ist dieses Maß auch stark abhängig von der Größe des betrachteten Quellcodes. Dies zeigt auch ein Korrelationskoeffizient von 0.8, welcher eine hohe, positive Korrelation zur Anzahl der Codezeilen bedeutet .

Anzahl der Codezeilen: Neben dem Halstead Volumen weist die Anzahl der Zeilen eines betrachteten Ausschnitts mit r_{Sp} = -0.37 ebenfalls eine mittlere, negative Korrelation zur Lesbarkeit auf. Die Länge eines betrachteten Ausschnitts wirkt sich demnach ebenfalls negativ auf die Lesbarkeit aus. Dies widerspricht den Erkenntnissen von Posnett et al., die eine positive Auswirkung auf die Bewertung der Lesbarkeit festgestellt haben.

Die Tatsache, dass das Halstead Volumen und die Anzahl der Codezeilen die größte Auswirkung auf die Lesbarkeit unter den betrachteten Eigenschaften aufweisen, bestätigt dagegen weitere Erkenntnisse von Posnett et al. wonach die Größe des betrachteten Quellcodes bei der Bewertung der Lesbarkeit stets berücksichtigt werden muss. Eine Bewertung, welche ausschließlich auf größenunabhängigen, zeilenbasierten Eigenschaften beruht, wie etwa beim Modell von Buse und Weimer, ist demnach in seiner Beschreibung der Lesbarkeit begrenzt.

Maximale Zeilenlänge: Als einzige größenunabhängige Eigenschaft weist die maximale Zeilenlänge eine Korrelation zur Bewertung der Lesbarkeit auf. Mit r_{Sp} = -0.27 fällt diese gering und negativ aus. Lange Zeilen führen also zu einer schlechteren Lesbarkeit des Quellcodes.

Zur Erkenntnis, dass die Zeilenlänge zu den Eigenschaften mit der stärksten, negativen Auswirkungen auf die Lesbarkeit gehört, kommen alle unter 3.2 vorgestellten Arbeiten. Damit lässt sich dieser Eigenschaft eine gewisse Allgemeingültigkeit zusprechen.

Vergleich relevanter Eigenschaften für Studierende und Mitarbeiter der CoreMedia AG

Bei der separaten Betrachtung der Bewertungen der studentischen Teilnehmer (siehe Abb. 4.13) sowie den Bewertungen der Mitarbeiter der CoreMedia AG (siehe Abb. 4.12) wird deutlich, dass für beide Gruppen das Halstead Volumen, die Anzahl Zeilen sowie die maximale Zeilenlänge die größte Auswirkung auf die Bewertung der Lesbarkeit haben.

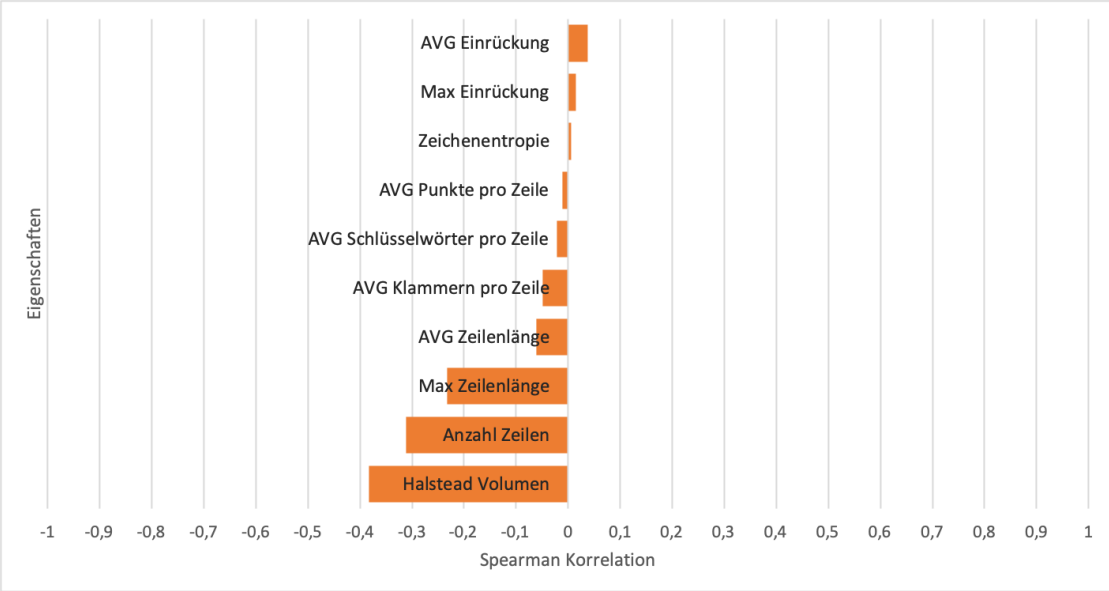

Abbildung 4.12: Korrelationen zwischen extrahierten Eigenschaften und Bewertungen der Mitarbeiter der CoreMedia AG

Bei den studentischen Teilnehmern fallen die Korrelationen jedoch zum Teil stärker aus. So weist die maximale Zeilenlänge in diesem Fall mit r_{Sp} = -0.34 schon eine mittlere, negative Korrelation zur Lesbarkeit auf. Zudem kommen mit der durchschnittlichen und maximalen Einrückung des Codes zwei Eigenschaften hinzu, für welche ebenfalls eine Korrelation erkennbar ist. Diese Eigenschaften korrelieren schwach, positiv mit der Lesbarkeit. Vermehrte Einrückungen erhöhen also potentiell die Lesbarkeit des Quellcodes. Dies widerspricht den Erkenntnissen von Buse und Weimer, wonach die Einrückung des Codes eine negative Auswirkung auf die Bewertung der Lesbarkeit hat

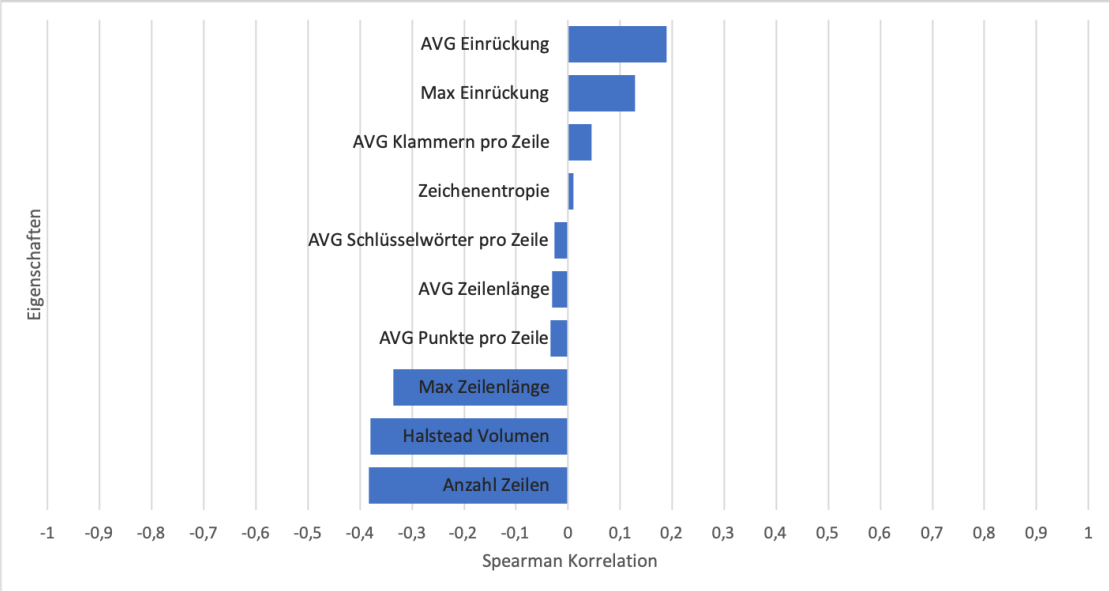

Abbildung 4.13: Korrelationen zwischen extrahierten Eigenschaften und Bewertungen der Studierenden

Betrachtung der Korrelationen für Java und Kotlin

Bei der separaten Betrachtung der Korrelationen zwischen Code-Eigenschaften und den Bewertungen der Lesbarkeit für Java und Kotlin wird deutlich, dass sich diese stark unterscheiden. Dies bestätigt die Erkenntnisse von Dorn. Er stellte in seiner unter 3.2.3 vorgestellten Arbeit bereits fest, dass die Code-Eigenschaften, welche Einfluss auf die Lesbarkeit des Quellcodes haben, je nach Sprache variieren.

Relevante Eigenschaften für Java (Abb. 4.14): Die Korrelationen in den Bewertungen der Java Code-Ausschnitte weisen eine gewisse Übereinstimmung mit den allgemein betrachteten Korrelationen auf. Die stärkste Auswirkung auf die Lesbarkeit haben demnach ebenfalls das Halstead Volumen und die Anzahl der Zeilen, mit jeweils r_{Sp} = -0.44. Auch die maximale Zeilenlänge weist hier eine schwache negative Korrelation zur Lesbarkeit auf.

Ebenfalls relevant sind zudem die maximale Einrückung sowie die durchschnittliche Anzahl an Klammern und Schlüsselwörtern pro Zeile. Die Korrelation zwischen der maximalen Einrückung ist mit r_{Sp} = -0.27, jedoch schwach negativ, während sie bei den

allgemeinen Bewertungen der studentischen Teilnehmer mit $r_{Sp} = 0.13$, schwach positiv ausfällt.

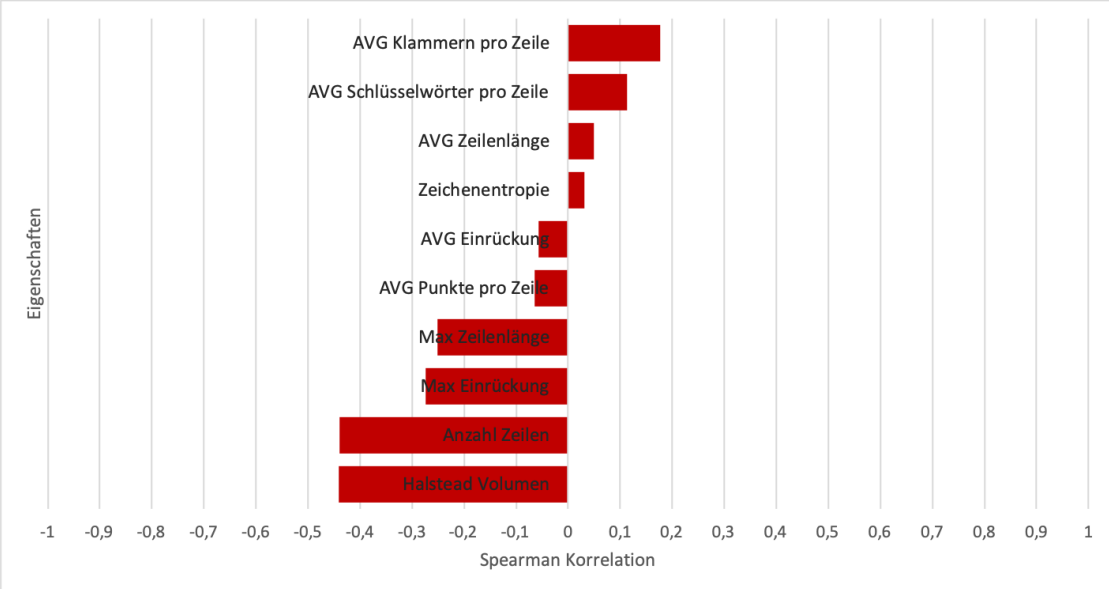

Abbildung 4.14: Korrelationen zwischen extrahierten Eigenschaften und Bewertungen der Java Code-Ausschnitte

Relevante Eigenschaften für Kotlin (Abb. 4.15): Im Gegensatz zu den Korrelationen der Java Code-Ausschnitte, ist die Stärke der Korrelationen der Kotlin Ausschnitte deutlich geringer. Zudem gibt es wenig Übereinstimmungen bei den relevanten Code-Eigenschaften. Die stärkste Korrelation zur Lesbarkeit weist die durchschnittlichen Zeilenlänge auf. Sie fällt mit r_{Sp} = -0.22 schwach negativ aus. Ebenfalls eine schwache negative Korrelation besteht zwischen der Lesbarkeit und der maximalen Zeilenlänge.

In Kotlin spielt also, anders als in Java, die Zeilenlänge die zentralste Rolle bei der Bewertung der Lesbarkeit. Das Halstead Volumen und die Anzahl der Zeilen besitzen bei der Bewertung der Lesbarkeit von Kotlin, mit $r_{Sp} = 0.05$ bzw. $r_{Sp} = 0.01$ keinerlei Relevanz. Die durchschnittliche Anzahl an Klammern und Schlüsselwörtern pro Zeile weisen im Gegensatz zu Java eine schwache negative Korrelation zur Bewertung der Lesbarkeit auf, was den Erkenntnissen von Buse und Weimer entspricht.

Diese große Diskrepanz zwischen den beobachten Korrelationen bei Kotlin und Java Quellcode lässt sich nur begrenzt erklären.

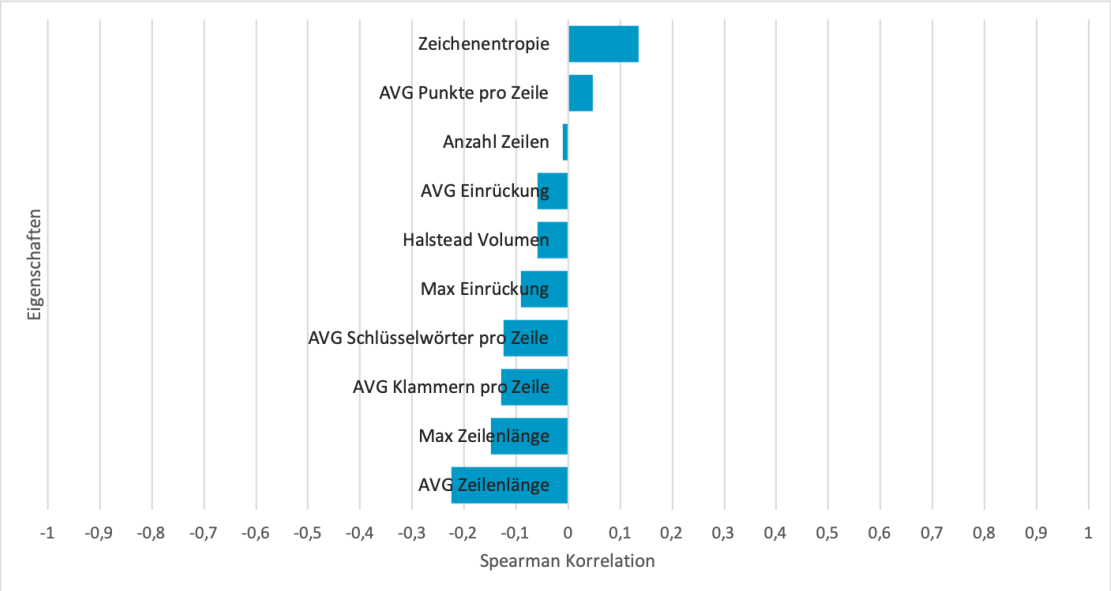

Abbildung 4.15: Korrelationen zwischen extrahierten Eigenschaften und Bewertungen der Kotlin Code-Ausschnitte

Ein Ansatz ist die geringe Datengrundlage, welche diesen Beobachtungen zugrunde liegt. Da jeweils 35 der 70 betrachteten Code-Ausschnitte in Java und 35 in Kotlin implementiert wurden, beruht die Bewertung lediglich auf diesen 35 Ausschnitten, was einer Halbierung der Datengrundlage im Vergleich zur allgemeinen Betrachtung entspricht.

Ungeachtet dessen und mit dem Wissen über ähnliche Erkenntnisse von Dorn, legen diese Beobachtungen nahe, dass für verschiedene Programmiersprachen jeweils andere Faktoren bei der Bewertung der Lesbarkeit beachtet werden müssen. Ein allgemeines, sprachunabhängiges Modell zur Lesbarkeit erscheint daher nur schwierig umsetzbar zu sein.

Kotlin spezifische Eigenschaften:

Neben den sprachunabhängigen Eigenschaften des Quellcodes, wurden zwei weitere, Kotlin spezifische Eigenschaften untersucht. Aus diesen Eigenschaften lässt sich potentiell die Lesbarkeit bestimmter Sprachfeatures von Kotlin ableiten. Auf Grund der geringen Menge an Kotlin Code-Ausschnitten beschränken sich die Beobachtungen auf zwei Eigenschaften und Sprachfeatures.

Anhand der durchschnittlichen Anzahl an Typbezeichnungen pro Zeile, wird die Auswirkung der Typinferenz, also die Herleitung von Typen aus dem umgebenden Code, auf die Lesbarkeit bewertet. Ein Korrelationskoeffizient von -0.27 zeigt eine geringe, negative Korrelation zur Lesbarkeit. Die Angabe von Typbezeichnungen wirkt sich demnach leicht negativ auf die Lesbarkeit von Kotlin aus. Im Umkehrschluss lässt sich ein leicht, positiver Effekt der Typinferenz auf die Lesbarkeit vermuten.

Als weiteres, wichtiges Sprachfeature von Kotlin wurde das Arbeiten mit nullfähigen Referenzen bezüglich seiner Auswirkung auf die Lesbarkeit der Sprache untersucht. Extrahiert wurde dabei die durchschnittliche Anzahl an Fragezeichen pro Zeile, welche den sicheren Aufruf solcher Referenzen kennzeichnen. Ein Korrelationskoeffizient von -0.15 zeigt auch hier eine geringe, negative Korrelation zur Lesbarkeit an. Eine negative Auswirkung solcher sicheren Aufrufe von nullfähigen Referenzen auf die Lesbarkeit der Sprache ist daher wahrscheinlich.

Relevanz der Eigenschaften im Zusammenhang zur Entwicklungserfahrung

Nachdem Dorn in seiner Arbeit ebenfalls Auswirkungen der Entwicklungserfahrung auf die Beziehungen zwischen den Bewertungen der Lesbarkeit und den Eigenschaft des Quellcodes beobachtet hat, wurde dieser Zusammenhang auch in dieser Arbeit untersucht.

Abbildung 4.16 zeigt die Korrelationen zwischen Code-Eigenschaften und Lesbarkeit, gruppiert nach der Erfahrung der Studienteilnehmer. Es ist deutlich abzulesen, dass es Unterschiede in der Ausprägung der Korrelationen gibt. Die Tendenz der relevantesten Korrelationen ist jedoch bei allen Teilnehmern, unabhängig von ihrer Erfahrung die gleiche. Demnach haben Halsteads Volumen, die Anzahl der Zeilen sowie die maximale Zeilenlänge die größte Auswirkung auf die Lesbarkeit. Bei Teilnehmern mit weniger als zehn Jahren Entwicklungserfahrung, spielen die Einrückung des Codes sowie die durchschnittliche Anzahl an Punkten pro Zeile ebenfalls eine Rolle. Diese Korrelationen sind jedoch deutlich schwächer ausgeprägt.

4.3.4 Gültigkeitsbeschränkung der Ergebnisse

Bei der Interpretation der Daten, welche auf Grundlage der Studie generiert wurden, müssen einige Faktoren beachtet werden, die deren Gültigkeit beschränken können.

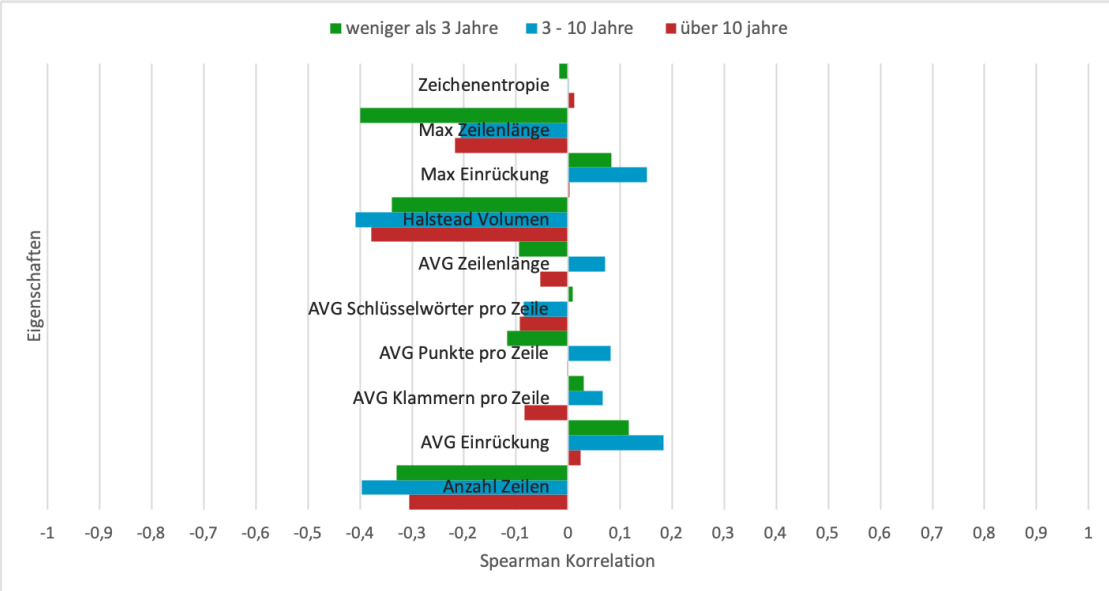

Abbildung 4.16: Korrelationen zwischen extrahierten Eigenschaften und Bewertungen der Teilnehmer, gruppiert nach Erfahrung

Die Studienteilnehmer: An der Studie haben ingesamt 65 Personen teilgenommen. 45 haben die Studie erfolgreich beendet, während 20 nur einen Teil der Fragen beantwortet haben.

Bei einer Anzahl von 70 Code-Ausschnitten beruhen die gewonnenen Erkenntnisse demnach auf etwa 3500 bis 4000 Bewertungen. Während diese Anzahl für die generelle Bewertung der Ergebnisse durchaus ausreichend ist, gestaltet sich die Betrachtung einzelner Teilnehmergruppen, etwa studentische Teilnehmer, Mitarbeiter der Coremedia AG oder Teilnehmer mit mehr als zehn Jahren Entwicklungserfahrung als schwierig. Die Menge der zugrundeliegenden Bewertungen fällt hier zum Teil deutlich geringer aus. Demnach sind Erkenntnisse, welche sich auf bestimmte Teilnehmergruppen beziehen mit Vorbehalt zu betrachten.

Um die Allgemeingültigkeit der Studienergebnisse auch für bestimmte Teilnehmergruppen zu erhöhen, müsste also eine höhere Teilnehmerzahl erreicht werden.

Ebenso kann nicht garantiert werden, dass die Bewertungen der Zielgruppen, also der Studierenden der HAW Hamburg bzw. der Mitarbeiter der CoreMedia AG die allgemeine Auffassung von Entwicklern zur Lesbarkeit von Quellcode widerspiegeln. Dieses Risiko lässt sich durch eine breitere Zielgruppe der Studie minimieren.

Die Auswahl der Code-Ausschnitte: Die Wahl der zu bewertenden Code-Ausschnitte ist ein kritischer Aspekt für die Ergebnisse der Studie.

Es muss beachtet werden, dass die Ausschnitte von einer einzelnen Person erstellt und lediglich von drei weiteren Personen überprüft wurden. Die Entscheidung darüber, welche Aspekte der betrachteten Sprachen berücksichtigt werden, ist demnach die Entscheidung weniger Personen und wurde nach bestem Wissen dieser Personen getroffen. Es handelt sich also um eine subjektive Auswahl, welche auch durchaus anders getroffen werden könnte.

Ebenso kritisch zu bewerten ist die Anzahl von 70 Code-Ausschnitten. Es handelt sich, wie unter 4.2.3 beschrieben, um einen Kompromiss. Gerade bei der separaten Bewertung von Kotlin und Java Ausschnitten ist die Bewertungsgrundlage mit 35 Ausschnitten eher gering. Eine höhere Anzahl an Ausschnitten würde dieses Problem minimieren, ließ sich im Rahmen dieser Studie jedoch nicht umsetzen, ohne die Teilnehmerzahl weiter zu verringern.

Der Vergleich von Kotlin und Java: In der Studie wurde die Lesbarkeit von Kotlin im Vergleich zu Java bewertet. Dieser Vergleich erfolgt aufgrund der Tatsache, dass die CoreMedia AG derzeit hauptsachlich auf Java bei der Entwicklung ihres Produktes setzt. Durch den Vergleich soll deutlich werden ob Kotlin, bezogen auf die Lesbarkeit des Quellcodes eine Verbesserung zu Java darstellt.

Unter Berücksichtigung des Prozesses für die Auswahl der Code-Ausschnitte lässt sich jedoch nicht mit Sicherheit feststellen ob sich ein solcher Vergleich unter diesen Voraussetzungen als fair gestaltet. Es besteht die Gefahr, dass persönliche Präferenzen Auswirkungen auf die Auswahl, der in den Code-Ausschnitten berücksichtigten Sprachfeatures haben.

Zudem gibt es häufig mehr als eine Möglichkeit die in einem Ausschnitt gezeigte Funktionalität zu implementieren. Die Entscheidungen darüber, welche Implementierungen beider Sprachen die inhaltlich höchste Äquivalenz aufweisen beruht auf den Einschätzungen weniger Personen.

4.4 Empfehlung für den Einsatz von Kotlin

Das primäre Ziel dieser Arbeit liegt darin eine objektiv begründete Empfehlung für den Einsatz von Kotlin bei der CoreMedia AG zu geben. Auf Grundlage der Ergebnisse der durchgeführten Studie und mit Hilfe des entwickelten Modells zur Lesbarkeit von Quellcode, wird eine solche Empfehlung ermöglicht.

Die Auswertung der Studienergebnisse zeigt, dass die Lesbarkeit von Kotlin Quellcode im Vergleich zu Java Quellcode deutlich besser bewertet wird. Auf einer Skala von Eins bis Fünf wurde die Lesbarkeit von Kotlin durchschnittlich mit 4,01 bewertet, was eine sehr gute Lesbarkeit der Sprache bedeutet. Im Vergleich zu Java fielen die Bewertungen in Schnitt 9% besser aus.

Wichtig ist dabei auch die Tatsache, dass sich in den Bewertungen beider Zielgruppen der Studie eine hohe Übereinstimmung zeigt. Demnach bewerten die Studierenden der HAW Hamburg und die Mitarbeiter der CoreMedia AG die Lesbarkeit von Kotlin ähnlich gut und jeweils besser als die Lesbarkeit von Java. Auch die erwartete Verschiebung in den Bewertungen aufgrund der größeren Vertrautheit der Teilnehmer mit Java, fällt weniger stark aus als angenommen. Unabhängig von der Entwicklungserfahrung in Java oder Kotlin, wurde Kotlin stets als besser lesbar im Vergleich zu Java bewertet. Es darf zudem davon ausgegangen werden, dass die Bewertungen noch deutlicher „pro Kotlin“ ausgefallen wären, wenn eine ähnlich hohe Vertrautheit zu Kotlin bei den Teilnehmern gegeben wäre.

Durch die Validierung des vorläufigen Modells zur Lesbarkeit von Quellcode kann die bessere Bewertung der Lesbarkeit von Kotlin zudem auf objektive Faktoren zurückgeführt werden. Die Auswertung der Korrelationen zwischen Eigenschaften im Code und der Bewertung der Lesbarkeit zeigt, dass die Lesbarkeit von Quellcode auch maßgeblich von seiner Größe abhängig ist. Demnach wirken sich eine größere Anzahl an Zeilen, eine größere Anzahl (verschiedener) Operatoren und Operanden sowie lange Codezeilen negativ auf die Lesbarkeit des Quellcodes aus. Es ist also schon allein aufgrund der Knappheit von Code wahrscheinlich, dass dieser im Vergleich zu längerem Code besser lesbar ist.

Eine Betrachtung der ausgewählten Code-Ausschnitte zeigt, dass in den Kotlin Ausschnitten durchschnittlich 36,5% weniger Codezeilen als in den Java Ausschnitten benötigt wurden, um die gleiche Funktionalität abzubilden. Dieser Wert liegt nahe an den von JetBrains geschätzten 40% [KotlinLangFaQ]. Die bessere Bewertung dieser Ausschnitte

geht demnach voraussichtlich auch stark auf ihre Knappheit im Vergleich zu den Java Ausschnitten zurück.

Da die Knappheit der Sprache eine zentrale Eigenschaft von Kotlin ist, ist es wahrscheinlich, dass ein solcher positiver Effekt auf die Lesbarkeit auch in einem größeren Projekt skaliert.

Aufgrund dieser Erkenntnisse ist Kotlin für den Einsatz bei der CoreMedia AG zu Empfehlen. Die Empfehlung stützt sich dabei allein auf die Lesbarkeit des Quellcodes, kann also nur einen Teil zur Entscheidungsbildung beisteuern. Eine vollständige Evaluation der Sprache ist weiterhin zu empfehlen.

4.5 Vorschlag eines allgemeinen Modells zur Lesbarkeit

Aus dem primären Ziel dieser Arbeit, der Bewertung der Lesbarkeit von Kotlin Quellcode, ergibt sich ein weiteres Ziel. Dies begründet sich auf dem Umstand, dass nach aktuellen Erkenntnissen keine belastbare Möglichkeit existiert die Lesbarkeit einer Programmiersprache objektiv zu bewerten. Zwar existieren einige Arbeiten, in welchen Modelle und Metriken zur Lesbarkeit von Quellcode entwickelt wurden. Es handelt sich jedoch lediglich um Ansätze, welche sich weder direkt auf beliebigen Quellcode anwenden lassen, noch eine ausreichende Allgemeingültigkeit besitzen. Zudem lassen diese Modelle und Metriken keine, oder nur begrenzte Rückschlüsse auf die Lesbarkeit einer bestimmten Programmiersprache zu.

Ziel dieser Arbeit ist es daher, einen Vorschlag für ein allgemeines Modell zur Lesbarkeit von Quellcode zu unterbreiten, welches Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache zulässt.

Auf Grundlage der bisherigen Erkenntnisse wurde daher zunächst ein vorläufiges Modell entwickelt (siehe 4.1), welches im Anschluss mit Hilfe einer Studie zur Lesbarkeit von Kotlin und Java validiert wurde (siehe 4.3.3).

Es umfasst eine Menge von Code-Eigenschaften, welche sich in den vorherigen Arbeiten als relevant für die Bewertung der Lesbarkeit erwiesen haben. Dabei werden größenunabhängige, zeilenbasierte Eigenschaften aus dem Modell von Buse und Weimer mit größenabhängigen Eigenschaften, sowie bekannten Größenmaßen aus dem Modell von Posnett et al. kombiniert und um weitere, sprachspezifische Eigenschaften ergänzt.

Die Überprüfung des vorläufigen Modells in der Auswertung der Studie zeigt, dass sich für drei der allgemeinen Eigenschaften eine Beziehung der Lesbarkeit bestätigt. Dazu zählt das Volumen nach Halstead, als größenabhängige Eigenschaft, die maximale Zeilenlänge, als größenunabhängige Eigenschaft, sowie die Anzahl der Zeilen als verbreitetes Größenmaß von Software. Eine Kombination dieser Eigenschaften kann demnach die Grundlage für ein allgemeines Modell zur Lesbarkeit von Quellcode bilden. Es bleibt jedoch zu überprüfen ob sich mit einem solchen Modell, welches einzig auf diesen drei Faktoren basiert, die Lesbarkeit von Quellcode ausreichend beschreiben lässt.

Die Bestätigung der Erkenntnisse von Dorn [Dorn, 2012], wonach sich die, für die Lesbarkeit relevanten Code-Eigenschaften je nach Programmiersprache deutlich unterscheiden, stellt zudem infrage, ob ein allgemeines Modell die Lesbarkeit von konkreten Programmiersprachen überhaupt hinreichend genau beschreiben kann.

Ein Ansatz ist es, das allgemeine Modell als Grundgerüst zu nutzen und je nach Sprache um weitere ggf. sprachspezifische Faktoren zu ergänzen.

Für Java legen die Ergebnisse der Studie eine Einbeziehung der maximalen Einrückung des Quellcodes sowie der durchschnittlichen Anzahl an Schlüsselwörtern und Klammern pro Zeile nahe.

In Kotlin erweisen sich ebenfalls die durchschnittliche Anzahl an Schlüsselwörtern und Klammern pro Zeile als relevante Eigenschaften. Hinzu kommt die zeichenbasierte Entropie sowie zwei Kotlin spezifische Eigenschaften. Dazu zählt das durchschnittliche Vorkommen von Fragezeichen sowie das durchschnittliche Vorkommen von Typbezeichnungen pro Zeile. Fragezeichen werden zur Kennzeichnung von sicheren Aufrufen nullfähiger Referenzen genutzt, die Anzahl an Typbezeichnungen steht im Zusammenhang mit der Möglichkeit der Typinferenz. Beide Eigenschaften haben, nach den gewonnen Erkenntnissen, eine negative Auswirkung auf die Lesbarkeit des Quellcodes (siehe 4.3.3).

Solche sprachspezifischen Erweiterungen des allgemeinen Modells versprechen eine bessere Abbildung davon, wie sich Lesbarkeit für die entsprechende Sprache definiert.

Es ist zu berücksichtigen, dass es sich lediglich um einen Vorschlag für ein Modell und nicht um eine anwendbare Metrik handelt. Durch die Studie zur Lesbarkeit von Kotlin und Java, konnte ein vorläufiger Entwurf des Modells validiert und ein begründeter Vorschlag für ein allgemeines Modell zur Lesbarkeit von Quellcode unterbreitet werden. Die Aussagekraft sowie die Allgemeingültigkeit dieses Modells ist noch zu überprüfen.

5 Fazit

Die CoreMedia AG steht vor der Entscheidung ob Kotlin neben Java in der Produktentwicklung genutzt werden soll. Um eine begründete Entscheidung darüber treffen zu können, sollte eine Evaluation der Sprache durchgeführt werden. Ein wichtiger Aspekt, welcher dabei überprüft werden sollte ist die Lesbarkeit der Sprache.

Das Ziel dieser Arbeit war es, eine solche Überprüfung möglichst objektiv durchzuführen und auf Grundlage dieser Überprüfung eine Empfehlung für bzw. gegen den Einsatz von Kotlin bei der CoreMedia AG auszusprechen.

Um eine Bewertung der Lesbarkeit von Kotlin zu ermöglichen, wurden zunächst bestehende Ansätze von Metriken und Modellen zur Lesbarkeit von Quellcode herangezogen (siehe 3.2). Dabei wurde deutlich, dass eine einfache Bewertung auf Grundlage dieser Modelle nicht durchgeführt werden kann. Zum einen da die Allgemeingültigkeit dieser Ansätze nur sehr begrenzt gegeben ist, zum anderen da sich diese Modelle auf Aspekte stützen, welche keine Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache zulassen.

Aufgrund dieser Erkenntnis wurde im Rahmen dieser Arbeit ein Vorschlag für ein allgemeines Modell zur Lesbarkeit von Quellcode erarbeitet, welches Rückschlüsse auf die Lesbarkeit einer konkreten Programmiersprache ermöglichen soll (siehe 4.5). Begleitend dazu wurde eine Studie zur Lesbarkeit von Kotlin und Java unter Mitarbeitern der CoreMedia AG und Studierenden der HAW Hamburg durchgeführt. Ziel der Studie war es, Erkenntnisse darüber zu sammeln, wie gut die Lesbarkeit von Kotlin und Java bewertet wird und welche Faktoren bei der Bewertung eine Rolle spielen.

Auf Grundlage der Studienergebnisse und des entwickelten Modells zur Lesbarkeit, ist Kotlin für den Einsatz bei der CoreMedia zu Empfehlen. Die Sprache wurde von Studienteilnehmern unabhängig von allen berücksichtigten Aspekten als deutlich besser lesbar im Vergleich zu Java bewertet (siehe 4.4). Durch das entwickelte Modell lassen sich diese Bewertungen zudem zumindest in Teilen objektiv begründen.

6 Ausblick

Ausblickend bleibt anzumerken, dass zwar auf eine umfassende Validierung des vorgeschlagenen Modells im Rahmen dieser Arbeit verzichtet wurde, weiterführend sollte eine solche ausführliche Überprüfung jedoch in Betracht gezogen werden und ist mit Sicherheit sinnvoll. Diese kann zum Beispiel nach Vorbild der unter 3.2 beschriebenen Arbeiten, mit Hilfe eines Klassifikationsalgorithmus erfolgen. Durch dieses Vorgehen ließe sich die Aussagekraft des Modells fundiert beschreiben und seine Allgemeingültigkeit überprüfen.

Diese Arbeit versteht sich als grundlegender Ansatz zur Beschreibung der Lesbarkeit von Quellcode. Sie überprüft die Erkenntnisse vorheriger Arbeiten auf einem eigenen Datensatz, bestätigt diese teilweise und folgert auf Grundlage der durchgeführten Studie weitere Schlüsse, welche zu einer besseren Beschreibung der Lesbarkeit von Quellcode beitragen.

Es bleibt wünschenswert, dass diese Erkenntnisse in weiteren Arbeiten aufgegriffen und ergänzt werden, um so Stück für Stück von einem Modell zu einer anwendbaren Metrik zur gelangen, mit welcher eine einfache, automatisierte Bewertung der Lesbarkeit von Programmiersprachen möglich ist.

Literaturverzeichnis

[KotlinBlog] *blog.jetbrains.com Kotlin Blog.* https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/. – Zugriff: 2019-04-25

[HalsteadCounter] *HalsteadCounter.* https://github.com/abramhindle/ngram-complete-dist/blob/master/antlr/HalsteadCounter.java. – Zugriff: 2019-04-09

[KotlinLangFaQ] *kotlinlang.org FAQ section.* https://kotlinlang.org/docs/reference/faq.html. – Zugriff: 2019-01-12

[KotlinLang] *kotlinlang.org Why Kotlin?* https://kotlinlang.org/. – Zugriff: 2019-01-12

[VirtualMachinery] *virtualmachinery.com The Halstead Metrics.* http://www.virtualmachinery.com/sidebar2.htm. – Zugriff: 2019-04-09

[Abbas 2010] Abbas, N.: Properties of "Good" Java Examples. (2010)

[Al-Qutaish und Abran 2005] Al-Qutaish, Rafa ; Abran, Alain: An Analysis of the Design and Definitions of Halstead's Metrics, 09 2005

[Boehm und Basili 2001] Boehm, B. ; Basili, V. R.: Software Defect Reduction Top 10 List. In: *Computer* 34 (2001), Januar, Nr. 1, S. 135–137. – URL http://dx.doi.org/10.1109/2.962984. – ISSN 0018-9162

[Buse und Weimer 2010] Buse, R. P. L. ; Weimer, W.: Learning a Metric for Code Readability. In: *IEEE Trans. Softw. Eng.* 36 (2010), jul, Nr. 4, S. 546–558

[Collar und Valerdi 2014] Collar, E. J. ; Valerdi, R.: *Role of Software Readability on Software Development Cost.* S. 1, ResearchGate, 2014

[Dorn 2012] Dorn, Jonathan: A General Software Readability Model, 2012

[Halstead 1977] Halstead, M. H.: *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA : Elsevier Science Inc., 1977. – ISBN 0444002057

[Hemphill 2003] Hemphill, James: Interpreting the Magnitude of Correlation Coefficients. In: *The American psychologist* 58 (2003), 02, S. 78–9

[Hoffmann 2008] Hoffmann, D. W.: *Software Qualität*. Springer-Verlag, 2008. – ISBN 978-3-540-76322-2

[Jemerov und Isakova 2017] Jemerov, D. ; Isakova, S.: *Kotlin in Action*. Manning Publications Co., 2017. – ISBN 9781617293290

[Liggesmeyer 2009] Liggesmeyer, P.: *Software Qualität*. Spektrum Akademischer Verlag Heidelberg, 2009. – ISBN 978-3-8274-2056-5

[Posnett u. a. 2011] Posnett, D. ; Hindle, A. ; Devanbu, P.: A Simpler Model of Software Readability. In: *Proceedings of the 8th Working Conference on Mining Software Repositories*. New York, NY, USA : ACM, 2011 (MSR '11), S. 73–82. – URL `http://doi.acm.org/10.1145/1985441.1985454`. – ISBN 978-1-4503-0574-7

[Radatz u. a. 1991] Radatz, Jane ; Katki, Freny ; McMonegal, Louise ; Meyer, Bennett ; Lane, John ; Wilson, Paul ; Geraci, Anne ; Yee, Mary ; Porteous, Hugh ; Springsteel, Fredrick: *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. Piscataway, NJ, USA : IEEE Press, 1991. – ISBN 1559370793

[Scalabrino u. a. 2018] Scalabrino, S. ; Linares-Vásquez, M. ; Oliveto, R. ; Poshyvanyk, D.: A comprehensive model for code readability. In: *Journal of Software: Evolution and Process* 30 (2018), jun, Nr. 6, S. 1–29

[Schneidewind 1997] Schneidewind, N.: IEEE Standard For A Software Quality Metrics Methodology Revision And Reaffirmation. In: *Proceedings of IEEE International Symposium on Software Engineering Standards*, June 1997, S. 278. – ISSN 1082-3670

[Shannon 1948] Shannon, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), Nr. 3, S. 379–423

[Shen u. a. 1983] Shen, V. Y. ; Conte, S. D. ; Dunsmore, H. E.: Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support. In: *IEEE Transactions on Software Engineering* SE-9 (1983), March, Nr. 2, S. 155–165. – ISSN 0098-5589

[Sneed u. a. 2010] Sneed, M. H. ; Seidl, R. ; Baumgartner, M.: *Software in Zahlen, Die Vermessung von Applikationen.* Carl Hanser Verlag München, 2010. – ISBN 978-3-446-42175-2

[Sprent 2001] Sprent, P.: *Applied Nonparametric Statistical Methods.* Chapman and Hall/CRC, 2001. – ISBN 1-58488-145-3

[Thomson 1889–94] Thomson, W. 1. Baron K.: *Popular Lectures and addresses Vol. 1.* S. 73, JLondon Macmillan, 1889-94

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original