Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Why programmers love their language, eine empirische Studie

Lukas Hoffmann
Matrikelnummer: 4896570
lhoff@zedat.fu-berlin.de

Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Ralf Romeike

Berlin, 26. Juni 2019

Zusammenfassung

Seit jeher ist die Frage, welche Programmiersprache wofür geeignet ist, großer Grund für Diskussionen. Persönliche Präferenzen spielen dabei vermutlich eine maßgebliche Rolle, wurden bisher aber kaum untersucht.
Forschungsfrage: Warum bevorzugen Softwareentwickler bestimmte Programmiersprachen? Welche Faktoren spielen eine Rolle in deren Wahrnehmung?
Methode: Befragung von 24 Softwareentwicklern unterschiedlichen Hintergrunds unter Verwendung von unterschiedlichen Interviewtypen. Qualitative Auswertung der Interviews mit Methoden aus der Grounded-Theorie-Methodik.
Ergebnis: Welche Faktoren für die Programmiersprachenpräferenz eine Rolle spielt, hängt maßgeblich davon ab, ob Entwickler die Domäne wechseln oder nicht. Außerdem ist die Erfahrung in einer Sprache der wichtigste Faktor für die Präferenz.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

26. Juni 2019

Lukas Hoffmann

Inhaltsverzeichnis

1 Einführung **7**
1.1 Forschungsfrage . . . 7
1.1.1 Ziel . . . 7
1.1.2 Nicht-Ziel . . . 7
1.2 Vergleichbare Arbeiten . . . 7
1.2.1 Popularität von Programmiersprachen . . . 8

2 Methodik **8**
2.1 Grounded Theory Methodik . . . 8
2.1.1 Offenes Kodieren . . . 9
2.1.2 Selektives Kodieren . . . 9
2.1.3 Die Methode des ständigen Vergleichens . . . 9
2.1.4 Theoretische Sensitivität . . . 10
2.1.5 Theoretisches Sampling . . . 10
2.1.6 Nicht verwendete Elemente . . . 10
2.2 Interviews . . . 10
2.2.1 Lange Interviews (Phase 1) . . . 11
2.2.2 Kurze Interviews (Phase 2) . . . 13
2.2.3 Lange Interviews (Phase 3) . . . 14
2.3 Transkription . . . 15
2.3.1 Kodieren . . . 15
2.4 Demografie . . . 15
2.5 Zusätzliche Datenquellen . . . 16
2.6 Verwendete Werkzeuge . . . 16
2.6.1 Interviews . . . 16
2.6.2 Transkription und Kodieren . . . 16

3 Erkenntnisse **17**
3.1 Kategorien . . . 17
3.1.1 Produktivität - "Was hilft mir" . . . 17
3.1.2 Befriedigung - "Was macht mir Spaß" . . . 20
3.2 Erkenntnis 1: Entwickler brauchen Herausforderungen, entweder durch wechselnde Domänen oder durch die Programmiersprache . . . 21
3.3 Erkenntnis II: Erfahrung ist der wichtigste Grund eine Programmiersprache zu mögen . . . 22
3.4 Erkenntnis III: Softwareentwickler reflektieren zu wenig über die verwendeten Programmiersprachen . . . 24
3.5 Vergleich mit bisheriger Arbeit . . . 24
3.6 Ausblick . . . 25

Literaturverzeichnis **26**

1 Einführung

Das wichtigste Werkzeug eines Programmierers/Softwareentwicklers [1] ist die Programmiersprache. Die Frage welche Programmiersprache sich für welchen Anwendungszweck gut eignet, ist dabei eine immer wiederkehrende Diskussion unter Softwareentwicklern. In einigen Studien wurden verschiedene Aspekte von Programmiersprachen und die Eignung für unterschiedliche Anwendungszwecke untersucht [11] [9] [10]. Dennoch ist die Fragestellung weiterhin nicht als geklärt anzusehen. Ein Artikel mit dem Titel: "Welche Programmiersprache soll ich lernen?" [2] der im September 2018 auf der Online-Newsseite Golem erschienen ist, hat zu regen Diskussionen mit über 400 Kommentaren geführt. Ein möglicher Grund dafür ist, dass auch persönliche Präferenzen eine nicht zu unterschätzende Rolle spielen. Softwareentwickler scheinen manche Sprachen mehr zu mögen als andere.

1.1 Forschungsfrage

Warum mögen Softwareentwickler manche Programmiersprachen lieber als andere? Gibt es Faktoren die wiederholt auftreten? Wie wichtig sind diese Faktoren?

1.1.1 Ziel

Ziel dieser Arbeit ist es durch Interviews mit Softwareentwicklern Konzepte zu entwickeln. Diese Konzepte und Erkenntnisse sollen nachvollziehbar aus den Aussagen bzw. Nicht-Aussagen der Interviewten abgeleitet werden.

1.1.2 Nicht-Ziel

Auch wenn mit einer theoriegenerierenden Methode gearbeitet wird, ist das Ziel nicht eine allgemeingültige Theorie zu erarbeiten. Die Gruppe der Befragten ist nicht repräsentativ und daher sind auch die Ergebnisse nicht allgemeingültig. Die Konzepte werden im Rahmen der Arbeit nicht durch eine quantitative Studie validiert.

1.2 Vergleichbare Arbeiten

Empirische Forschung zu der genauen Forschungsfrage gibt es bisher nicht. In einigen Studien wird eine verwandte Frage untersucht. Erwähnenswert sind hier vor allem Meyerovich und Rabkin [8], die in einer groß angelegten Studie mit verschiedenen Datenquellen Faktoren für die Adaption von Programmiersprachen untersucht haben und dabei auch ob intrinsischen Faktoren eine wichtige Rolle spielen. Unter

[1] Die Trennung der Begrifflichkeiten ist eher Unscharf. Auf Wikipedia steht "Umgangssprachlich wird zwischen den Begriffen "Programmierer" und "Softwareentwickler" oft nicht unterschieden. Man spricht allerdings selten von Programmierern, wenn etwa Softwarearchitekten oder Tester gemeint sind." [12]. Ein Blog Eintrag der Aussagen aus verschiedenen Quellen Vergleicht kommt zu folgendem Schluss: "Alles in allem bleibt der Eindruck des Vagen: Zu unterschiedlich sind die Auffassungen da draußen, als dass man strikt zwischen Entwickler und Programmierer unterscheiden könnte." [1]. Im Weiteren Verlauf dieser Arbeit sind die Begriffe als Synonym zu betrachten.

[2] https://www.golem.de/news/software-entwickler-welche-programmiersprache-soll-ich-lernen-1809-136638.html

anderem wurde auch die Frage "What programmers enjoy" beantwortet.
In einer empirischen Studie von Chen et al. [2] wurde über einen Zeitraum von zehn Jahren die Wahrnehmung von Programmiersprachen mit Hilfe einer Online-Umfrage erfasst und aus den Ergebnissen ein Vorhersagemodell für Programmiersprachen Trends entwickelt. Für die Umfrage wurde eine Liste von Faktoren (intrinsisch und extrinsisch) entwickelt und in der Umfrage die Relevanz des Faktors zusammen mit der primären Programmiersprache abgefragt.

1.2.1 Popularität von Programmiersprachen

Neben wissenschaftlichen Studien gibt es verschiedene Projekte, die versuchen die Popularität von Programmiersprachen zu messen. Ob eine Programmiersprache populär ist, spielt für meine Studie nur eine untergeordnete Rolle, da die Konzepte unabhängig von der Programmiersprache gültig sein sollten. Dennoch ist es sinnvoll sich der Popularität bewusst zu sein, um die Repräsentativität der Testgruppe beurteilen zu können.
Drei bekannte Indizies die zur Beurteilung herangezogen wurden sind folgende:

- **Tiobe Index** [3]
  Der Tiobe Index misst die Anzahl der Suchergebnisse in verschiedenen Suchmaschinen. Dabei wird der Name der Sprache mit dem Wort "programming" gesucht.

- **PYPL PopularitY of Programming Language** [4]
  Auf Basis der Google Suchtrends [5] wird die Häufigkeit der Suche nach dem Namen der Sprache und dem Wort "Tutorial" gemessen.

- **The RedMonk Programming Language Rankings** [6] In dem Redmonk Ranking werden die Rankings der Sprachen auf der Code-Hosting-Platform "Github"[7] und der Frage-und-Antwort-Plattform Stackoverflow [8] gegenübergestellt und ein gemeinsames Ranking gebildet.

2 Methodik

2.1 Grounded Theory Methodik

Zur Auswertung habe ich Elemente aus der Grounded Theorie Methodik (im weiteren GT) verwendet. Die Studie ist keine reine GT Studie, da sich einige der Konzepte nicht auf das untersuchte Feld anwenden lassen und andere in dem Zeitrahmen einer Bachelorarbeit nicht umsetzbar sind.
Die Grounded Theory Methodik ist ein Verfahren um ausgehend von einer Forschungsfrage, mittels empirischer Schritte eine Theorie zu entwickeln. Sie wurde

[3] https://www.tiobe.com/tiobe-index Abgerufen am 8.1.2019
[4] http://pypl.github.io/PYPL.html Abgerufen am 8.1.2019
[5] https://trends.google.de/
[6] https://redmonk.com/sogrady/2018/08/10/language-rankings-6-18/ Abgerufen am 8.1.2019
[7] https://github.com
[8] https://stackoverflow.com

in den 1960er Jahren durch die Sozialforscher Anselm Strauss und Barney Glaser entwickelt[5].
Die drei zentralen Schritte sind Datenerhebung, Datenauswertung und Konstruktion von Theorien und Teiltheorien. Diese drei Schritte werden dabei mehrfach durchlaufen. Jeder Durchlauf dient der Prüfung und zur Weiterentwicklung der bis dahin gefundenen Aussagen. Nach einer gewissen Anzahl an Durchläufen tritt eine Sättigung ein. Das heißt es kann davon ausgegangen werden, dass durch weitere Erhebungen keine neuen Erkenntnisse gewonnen werden.[7, S.162-168]
Die verwendeten Elemente sind im weiteren erläutert.

2.1.1 Offenes Kodieren

Der erste Schritt nach der Datenerhebung ist das offene Kodieren. In meinem Fall bedeutet das, das Transkript durchzulesen und alle Stellen die ich für relevant halte zu markieren und mit einem Code zu versehen. Relevant bezieht sich dabei zum einen auf die Forschungsfrage aber auch auf andere interessante Aussagen. Dieser Code oder auch Kategorie genannt, beschreibt die Aussage hinter den Aussagen. So kann die Aussage "Aber einfach das du so ganz schnell Produktiv sein kannst mit den Bibliotheken" [I3] mit der Kategorie "Schnelligkeit" kodiert werden. Zusätzlich könnte das Zitat auch noch die Kategorie "Produktivität" bekommen. Die Kategorien werden also nicht vorher festgelegt, sondern erst während der Auswertung. Deshalb sollte immer wieder einen Schritt zurückgegangen und die Kategorien miteinander verglichen werden. Oft beschreiben mehrere Kategorien denselben grundlegenden Mechanismus. Krotz [7, S.181-182] fasst das Zusammen als "Offenes Kodieren heißt also, dass man den Text des Protokolls Sinneinheit für Sinneinheit einordnet, abstrahiert, klassifiziert und in Bezug setzt".

2.1.2 Selektives Kodieren

Beim selektiven Kodieren geht es um die Findung der Kernkategorie. Die vorher gefundenen Kategorien werden auf einem abstrakten Level erneut klassifiziert. Im Zentrum steht dabei, dass die verschiedenen Kategorien in Bezug gesetzt werden, um Beziehungen zwischen diesen zu erkennen. Dabei entstehen neue Einsichten über den zugrundeliegenden Sachverhalt.

2.1.3 Die Methode des ständigen Vergleichens

Die Methode des ständigen Vergleichens beschreibt den zentralen Prozess der Grounded Theory Methodik. Die zentralen Phasen sind nach Glaser und Strauß [6, S. 111-119]:

1. Vorkommnisse für jede Kategorie vergleichen
2. Integration der Kategorien und ihrer Eigenschaften
3. Begrenzung der Theorie
4. Abfassen der Theorie

In meiner Arbeit habe ich mich auf die ersten beiden Schritte beschränkt.

2.1.4 Theoretische Sensitivität

Der Begriff theoretische Sensitivität meint, dass es für den Forscher wichtig ist die Daten mit offenen Augen auszuwerten, dadurch Einsichten zu gewinnen und neue Aspekte zu Erkennen. [4, 2.4]

2.1.5 Theoretisches Sampling

Theoretisches Sampling meint, dass die Auswahl der Daten, also in meinem Fall der Interviews, von den bisherigen Erkenntnissen abhängig zu machen. Glaser und Strauss formulieren folgende Frage: "Welchen Gruppen oder Untergruppen wendet man sich zwecks Datenerhebung als nächstes zu? Und mit welcher theoretischen Absicht?" [6, S. 55]. Ziel ist es den Punkt der Sättigung zu erreichen. Sättigung bedeutet, dass durch weitere Interviews keine neuen Erkenntnisse mehr gewonnen werden. Dazu werden Anfangs möglichst ähnliche Daten erhoben, um die Kategorien zu bilden und im weiteren Verlauf die Unterschiede maximiert um die Reichweite der Theorie erhöhen.

2.1.6 Nicht verwendete Elemente

Da es nicht Ziel der vorliegenden Arbeit war eine Theorie zu entwickeln, wurde der Schritt des axialen Kodierens übersprungen. Das Konzept des theoretischen Samplings wurde aus Zeitgründen nicht bis zu theoretischen Sättigung weitergeführt da dies den Zeitrahmen der Bachelorarbeit überschritten hätte.

2.2 Interviews

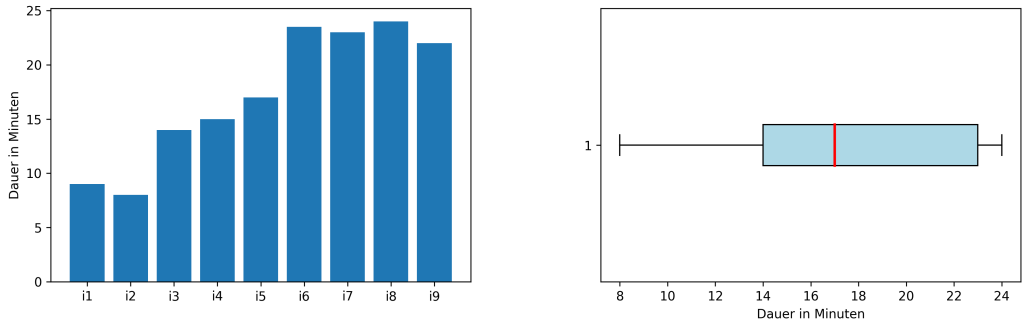
Die zentrale Datenquelle der Studie sind semistrukturierte Interviews. Davon gab es zwei verschiedene Typen. In der ersten Phase habe ich längere Interviews geführt in denen ich ein mit der Zeit leicht anwachsendes Set an Fragen gestellt habe, die dann jeweils als Einstieg in einen bestimmten Aspekt gedient haben. Außerdem ging es im größeren Teil des Interviews darum, zwei Programmiersprachen zu vergleichen in denen der Interviewte über einen längeren Zeitraum aktiv genutzt hat. Das Ziel der zweiten Phase mit deutlich kürzeren Interviews und nur einer Frage war, bisherige Erkenntnisse durch Antworten aus einem zweiten Blickwinkel weiterzuentwickeln. Als letzter Teil der Erhebung habe ich noch zwei längere Interviews geführt mit Entwicklern, die aus einer bisher eher unterrepräsentierten Gruppe kamen. Dabei habe ich einige Fragen aus den Interviews der Phase 1 wiederverwendet aber auch weitere hinzugenommen, die darauf abzielten die entstandenen Theorien zu falsifizieren.
Zitate aus den Interviews sind entsprechend gekennzeichnet und in der Aufnahmereihenfolge nummeriert. Die Interviews aus Phase 1 sind mit [i1] bis [i9] gekennzeichnet, die Kurzinterviews aus Phase 2 mit [k1] bis [k10] und die Interviews aus Phase 3 mit [i10] und [i11]. Aussagen aus den weiteren Quellen sind mit [y] für das Youtube Video [3] gekennzeichnet. In einigen wenigen Fällen sind die Zitate um Kontexthinweise ergänzt, um die Aussage außerhalb des kompletten Transkripts zu erhalten. Diese Hinweise sind (in Klammer und *kursiv*) gekennzeichnet.

2.2.1 Lange Interviews (Phase 1)

Teilnehmer Bis auf einen Teilnehmer habe ich alle Teilnehmer auf Meetups [9], die in Berlin im November und Dezember 2018 stattgefunden haben, kennengelernt. Insgesamt bin ich auf 9 Veranstaltungen gewesen, auf denen ich vermutet hatte Softwareentwickler anzutreffen.
Auf den Meetups, lassen sich die Interviews bedingt durch die Art der Veranstaltung, erst gegen Ende durchführen und da die Meetups in der Regel abends stattfinden wollen viele Teilnehmer sehr schnell nach der Veranstaltung nach Hause oder falls sie das Meetup als Teil einer Gruppe besucht haben, sich noch unterhalten. Die Geduld der Interviewten war dadurch bedingt eher gering. Deshalb habe ich nach den ersten beiden Interviews bei allen folgenden Veranstaltungen die Personen gebeten mir eine Kontaktinformation zu geben, sie nach der Veranstaltung kontaktiert und einen Termin für das Telefoninterview vereinbart. Leider haben mir dann nicht alle Personen geantwortet. Von 16 Personen habe ich von sechs keine Rückmeldung bekommen.
Vor dem Interview habe ich die Personen gefragt ob sie als Softwareentwickler arbeiten, schon mindestens drei Jahre Berufserfahrung gesammelt haben und schon mit mindestens zwei Sprachen produktiv gearbeitet haben. Dadurch ist die Chance höher, dass die Antworten auf eigenen Erfahrungen basieren und fundiert genug sind um eine Auswertung ermöglichen.
Falls die beiden Fragen bejaht wurden habe ich kurz das Thema der Studie so neutral wie möglich in ein bis zwei Sätzen erläutert, die ungefähre Dauer des Interviews erwähnt, betont, dass die Interviews und die darin gegeben Antworten anonym bleiben und dann gefragt ob sie bereit wären an einem Telefoninterview teilzunehmen.

Dauer Die Interviews dauerten zwischen 8 und 24 Minuten mit einer Durchschnittsdauer von 19,5 Minuten. Die ersten beiden Interviews waren die kürzesten mit acht und neun Minuten. Mit der Zeit wurden die Interviews immer länger da auch die Anzahl der Themenfelder, die ich durch die Fragen angesprochen habe größer wurde. Der Sprung zwischen dem zweiten und dritten Interview lässt sich auch noch durch den Wechsel von Vor-Ort zu Telefoninterviews erklären. Siehe auch Abbildung 1.

[9] Meetups sind Events die von Gruppen oder Firmen unabhänig von einander regelmäßige Treffen zu verschiedenen Themen veranstalten. Die Orte sind meist wechselnd. Bekanntgegeben werden die Veranstaltungen auf der namensgebenden Plattform "Meetup" (https://www.meetup.com/). Im Bereich Tech gibt es viele Veranstaltungen die sich mit einer bestimmten Programmiersprache, einem Framework oder eine Plattform beschäftigen. In der Regel gibt es dort ein bis zwei Vorträge die vorher bekanntgegeben werden und davor und danach die Möglichkeit sich in entspannter Atmosphäre auszutauschen.



(a) Dauer der Interviews

(b) Verteilung der Interviewdauer

Abbildung 1: Dauer und Verteilung der Interviews

Ablauf und Fragen Die Interviews liefen grob nach demselben Schema ab. Der Interviewleitfaden sollte aber nur einen ungefähren Rahmen an Themen liefern, die ich in den Interviews angesprochen haben wollte. Im Laufe der Interviews hat sich der Leitfaden durch die Erkenntnisse aus den vergangenen Interviews weiterentwickelt. Einige Punkte sind weggefallen und andere dazugekommen.
Zu Beginn habe ich bei jedem Interview folgende Punkte zur Einordnung und als Grundlage für das weitere Interview abgefragt:

- In welchen Programmiersprachen ist welches Erfahrungslevel vorhanden?
- In welcher Domäne arbeitet der Interviewte? (wenn schon aus dem Vorgespräch bekannt übersprungen)
- Welche Rolle nimmt er in der Organisation in der er arbeitet ein? (wenn schon aus dem Vorgespräch bekannt übersprungen)
- Festlegen von zwei Programmiersprachen die den höchsten und aktuellsten Erfahrungsgrad haben.
- Welche der beiden Sprachen ist die Favorisierte und warum?

Im zweiten Teil wurden dann die beiden Sprachen aus dem ersten Teil verglichen. Der Ausgangspunkt war dabei Antwort bei der letzten Frage aus dem ersten Teil. Der Teil lief weitgehend unstrukturiert ab und hing von den Antworten der Interviewten ab. Weitere Fragen vielen dabei im Wesentlichen in folgende Kategorien:

- Bitte um Spezifizierung eines Teils der Antwort
- Nachfrage zu der Antwort. Meist als neutrale Entweder-oder-Frage formuliert, um den Interviewten nicht zu beeinflussen.
- Fragen nach einem der folgenden Punkte, falls in den bisherigen Fragen noch nicht darauf eingegangen wurde: (Standardbibliothek, Syntax, Ökosystem, Features, Modularisierung, Kultur/Community)

Im Weiteren haben ich noch die Frage "Gab es schon mal ein Problem das du in der einen Sprache nicht lösen konntest und in der anderen gewusst hättest wie es

geht"gestellt. Ziel der Frage war, dass der Befragte mehr über seinen Alltag spricht. Diese Annahme hat sich auch weitestgehend bewahrheitet.
Zum Abschluss habe ich immer nach der Wichtigkeit der Programmiersprache bei der Arbeitgeberwahl gefragt. In dem ersten Interview hat der Befragte von sich aus dieses Thema angesprochen. Die Aussagen und Begründungen, die dort genannt wurden ließen Rückschlüsse auf den generellen Stellenwert der Programmiersprache zu, deshalb habe ich auch alle weiteren Teilnehmer dazu befragt. Auch hier wurde darauf geachtet, Suggestivfragen zu vermeiden.

2.2.2 Kurze Interviews (Phase 2)

Teilnehmer Die kurzen Interviews habe ich im Rahmen des "Chaos Communication Congress"[10] durchgeführt. Die Interviewten waren also alles Teilnehmer des Kongresses. Um einerseits bei den Befragten die Hürde zu antworten so gering wie möglich zu halten und andererseits die Spontanität zu fördern habe ich nicht nach Erfahrung oder Domäne gefragt sondern bin direkt mit der Interviewfrage eingestiegen.
Auf die Teilnehmer bin ich mit der Frage "Bist du ein Softwareentwickler" zugegangen. Nur wenn der Teilnehmer die Frage bejaht hat, habe ich das eigentliche Interview gestartet.

Dauer Die Interviewdauer variierte zwischen 30 Sekunden und 5:26 Minuten. Die Durchschnittslänge liegt bei 1:43 Minute. In Abbildung 2 lässt sich die Verteilung der Interviewdauer erkennen.
In Fällen in denen zwei Personen nebeneinander saßen und sich kannten habe ich beide Personen auf einmal Interviewt. In zwei Fällen ist die zweite Person auf die Aussage der ersten Person eingegangen.

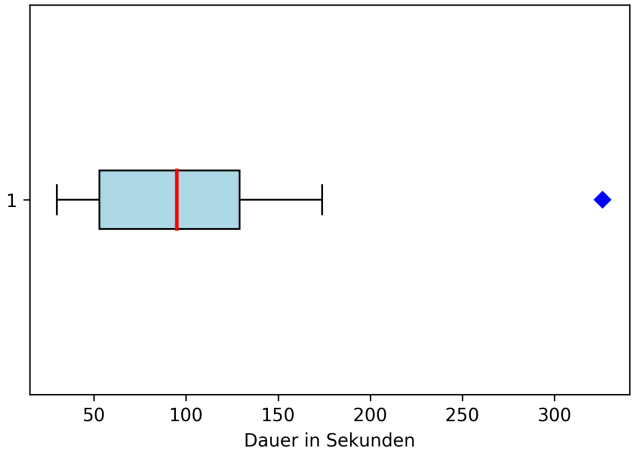


Abbildung 2: Verteilung der Interviewdauer

[10] https://events.ccc.de/congress/2018/wiki/index.php/Main_Page

Ablauf und Fragen Vor dem Start des Aufnahmegeräts habe ich die Befragten um Erlaubnis gebeten und darüber aufgeklärt, dass die Antworten nur anonym weiterverarbeitet werden.
Zu Beginn des Interviews habe ich dem Befragten eine Karte (Abbildung 3) in die Hand gegeben auf der die Fragestellung "Warum ist deine Lieblingsprogrammiersprache deine Lieblingsprogrammiersprache" stand und als Trigger dieselben Begriffe die ich in den langen Interviews abgefragt hatte in geringerer Schriftgröße unterhalb der Frage. Auf der Rückseite war derselbe Inhalt auf Englisch abgedruckt, um auch Personen zu befragen die kein Deutsch sprechen.
Falls als Antwort nur die Programmiersprache kam, habe ich nach den Gründen gefragt. Ansonsten habe ich auf Rückfragen verzichtet.

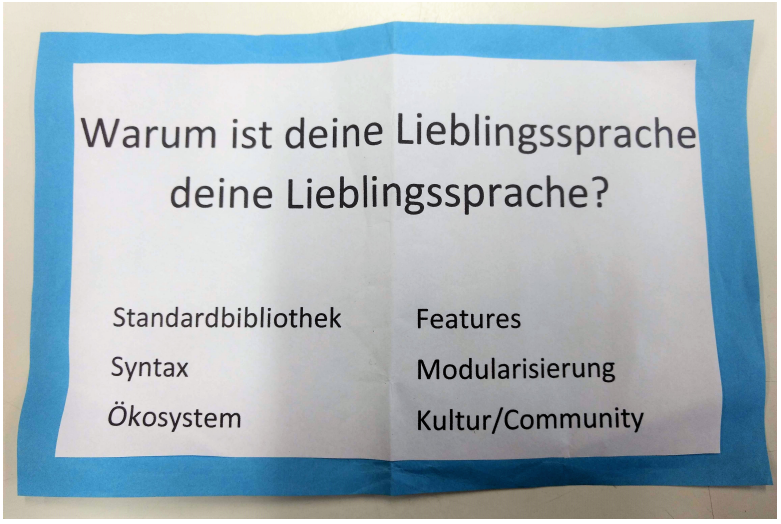


Abbildung 3: Karte mit Interviewfrage

2.2.3 Lange Interviews (Phase 3)

Teilnehmer Nach dem Prinzip des theoretischen Samplings habe ich zum Schluss noch zwei Softwareentwickler interviewt, die im Gegensatz zu den in Phase 1 Interviewten systemnahe Software entwickeln. Diese kamen nicht aus dem Umfeld der Meetups, da diese Themen dort stark unterrepräsentiert sind, sondern wurden über persönliche Kontakte angesprochen.

Dauer Die beiden Interviews lagen bei 15 Minuten und 25 Minuten und unterschieden sich in der Hinsicht nicht von den Interviews aus Phase 1

Ablauf und Fragen Der Grundablauf war im Vergleich zu den Interviews aus Phase 1 ähnlich. Statt jedoch im zweiten Teil durch Nachfragen alle sechs Punkte anzusprechen habe ich stattdessen direkte Fragen gestellt die auf die bisher ermittelten Konzepte abzielten um so zu validieren ob die Konzepte auch auf Entwickler aus einer anderen Gruppe zutreffen oder noch weiterentwickelt werden müssen.

2.3 Transkription

Nach den Interviews habe ich die Tonaufnahmen transkribiert, um sie in den folgenden Schritten besser weiterverarbeiten zu können. Sequenzen am Anfang und Ende die nicht Teil des eigentlichen Interviews waren wurden grundsätzlich nicht transkribiert. Die Interviews habe ich weitestgehend komplett transkribiert. In einigen Fällen wurden Sequenzen übersprungen, die keinen Bezug zu der Forschungsfrage hatten und auch sonst keine relevanten Details enthielten. Die Stellen sind in den Transkripten mit "(...)" gekennzeichnet.
Die Transkripte können auf Github `https://github.com/lhoff94/TranskripteBA` eingesehen werden (Commit-Hash: 7be5de4ffd83035439c09a720f2023e5e916decf).

2.3.1 Kodieren

Nach der Transkription wurden die Interviews nach dem Prinzip des offenen Kodierens ausgewertet. Dabei habe ich im ersten Durchlauf jeweils alle relevanten Stellen mit einem Code versehen, der sehr spezifisch die markierte Aussage zusammenfasst. Das Bilden von ersten Kategorien war ein Prozess der parallel dazu gestartet ist. Kam ein neuer Code dazu, der eine ähnliche Aussage beschreibt wie ein anderer zuvor erstellter Code, habe ich eine Kategorie hinzugefügt und die gefundenen Codes dort untergeordnet. MaxQDA bietet hier die Möglichkeit Codes hierarchisch zu ordnen. Textstellen, die mit einem der untergeordneten Codes versehen wurden, werden auch dem übergeordneten zugeordnet.
So entstanden mit jedem weiteren Interview größere und allgemeinere Kategorien, die jeweils mehrere Unterkategorien hatten. Nach dem letzten Interview der Phase 1 bin ich zum selektiven Kodieren übergegangen und habe die Kategorien hinterfragt und teilweise umsortiert. Dabei war vor allem das Vorkommen bzw. Nicht-Vorkommen von bestimmten Kategorien wichtig. Das heißt die Kategorien wurden nicht einzeln bewertet, sondern im Gesamtkontext gesehen. Die so entstandene Struktur der Codes hat der Validierung durch Phase 2 und 3 weitgehend standgehalten und wurde nur an wenigen Stellen noch verändert.
Die Auswertung der Interviews von Phase 2 lief dann dementsprechend so ab, dass ich versucht habe die Aussagen in das bisherige System einzufügen. Das ist in allen Fällen bis auf einen gelungen. In dem einen Fall musste das System ein noch nachjustiert werden.

2.4 Demografie

Alle Befragten in der Studie waren männlich. Generell ist der Frauenanteil auf Meetups ziemlich gering. Zwei Softwareentwicklerinnen haben auf der Veranstaltung zugesagt, dann aber leider nicht auf die Kontaktanfrage geantwortet. Um zu einer theoretischen Sättigung zu kommen, sollte versucht werden die Diversität der Befragten noch zu erhöhen.
Von den insgesamt 24 Befragten (alle Phasen) waren 21 aus Deutschland, eine Person aus Großbritannien, eine Person aus Polen und eine Person aus Spanien.

2.5 Zusätzliche Datenquellen

Zusätzlich zu den Interviews wurden noch weitere Datenquellen verwendet. In der Grounded Theory Methodik gilt der Grundsatz, dass beliebige Datenquellen verwendet werden können.
Folgende weitere Quellen wurden mit ausgewertet:

- Das Youtube Video "What's your Favourite Programming Language? (sound check Q)" des Kanals "Computerphile" [11]. Auf den Kanal werden üblicherweise Experten zu verschiedenen Themen aus dem Themenspektrum Informatik/-Computer befragt. Im vergangenen Jahr 2018 wurde zu Beginn des Interviews als ein Soundcheck die Frage "What is your favorite Programming Language and Why" gestellt und die Antworten am Ende des Jahres in dem Video veröffentlicht [3]. Von den insgesamt 27 befragten Personen haben zehn Personen einen Grund für ihre Präferenz genannt bzw. eine anderweitig interessante Antwort gegeben.

2.6 Verwendete Werkzeuge

Für die Erstellung der Bachelorarbeit habe ich eine Reihe von Werkzeugen verwendet. Im Folgenden sind diese und die Erfahrungen damit kurz erläutert.

2.6.1 Interviews

Die Interviews wurden teilweise in Person und teilweise mittels eines VoIP-Dienstes durchgeführt. Bei den Interviews, die in Person aufgenommen wurden, kam ein Diktiergerät vom Typ "Philips Voice Tracer LFH0612" zum Einsatz. Die Tonqualität war dabei ausreichend sofern die Sprecher nicht mehr als ein Meter von dem Gerät entfernt werden. Umgebungsgeräusche wurden dabei kaum aufgenommen, so dass auch die Interviews die auf dem Kongress in lauter Atmosphäre aufgezeichnet wurden gut zu verstehen waren.
Bei den telefonisch durchgeführten Interviews kam entweder Skype [12] oder Google Hangouts [13] zum Einsatz. Alle Befragten hatten bei einer der beiden Plattformen schon einen Account. Aufgenommen wurden die Interviews mit der Software "MP3 Skype Recorder"[14] (Skype) oder "OBS Studio" [15]. Die Tonqualität war sowohl bei dem Telefonat also auch in der Aufnahme bei Google Hangouts besser als bei Skype.

2.6.2 Transkription und Kodieren

Für die weitere Auswertung wurde die Software "MaxQDA Standard"[16] in der Version 2018.1 verwendet. Für die Transkription der Interviews stellt die Software eine entsprechende Funktionalität bereit, die das Setzen von Zeitmarken unterstützt und

[11] https://www.youtube.com/user/Computerphile
[12] https://www.skype.com/de/
[13] https://hangouts.google.com
[14] https://voipcallrecording.com/MP3_Skype_Recorder
[15] https://obsproject.com/de
[16] https://www.maxqda.de/produkte/maxqda-standard

durch automatisches Zurückspulen nach einer Abspielpause den Prozess deutlich erleichtert. Bei dem Kodieren der Interviews unterstützt die Software durch ein eingebautes Kodiersystem.

3 Erkenntnisse

In diesem Kapitel beschreibe ich zuerst das entstandene Code-System und im weiteren die Erkenntnisse die ich aus den Daten gewonnen haben. In der Beschreibung verwende ich dabei immer wieder Quantifizierungen. Diese sind unter Betrachtung der Stichprobengröße nicht auf andere Gruppen übertragbar.

3.1 Kategorien

Während dem Kodieren sind wie in 2.3.1 beschrieben viele Codes und Kategorien entstanden. Im weiteren Verlaufe haben sich daraus Hierarchien von Kategorien entwickelt. Durch ständiges umsortieren und erweitern ist ein System aus zusammenhängenden Kategorien entstanden. Das zentrale Element ist dabei die Forschungsfrage. Die beiden Schlüsselkategorien, in die sich fast alle anderen Kategorien einfügen, sind "Produktivität" und "Befriedigung". Im Weiteren werde ich auf die zentralen Kategorien eingehen, die Zusammenhänge darlegen und beispielhafte Zitate, die die Kategorie beschreiben, einfügen.

3.1.1 Produktivität - "Was hilft mir"

Viele Aussagen beschreiben Faktoren, die die Produktivität betreffen. Aussagen die direkt das Wort "Produktivität" beinhalten gibt es in allen Interviews nur vier. Dort wird das Wort im Zusammenhang mit einem speziellen Grund genannt. "Ich benutze da nur ganz normale Features, Aber einfach, dass du so ganz schnell produktiv sein kannst mit den Bibliotheken. Das gefällt mir da sehr gut." [i3]. Dennoch ist Produktivität, auch wenn nicht so benannt, ein zentrales Ziel eines professionellen Softwareentwicklers da sehr viele Aussagen zu anderen Kategorien auch eine Antwort auf die Frage "Was hilft mir" bzw. "was stört dich" sein könnten. In allen langen Interviews und mit einer Ausnahme auch in allen kurzen Interviews gibt es Aussagen, die in eine der untergeordneten Kategorien passen:
" I kind of feel that Kotlin is more expressiv. It allowes you to solve more complex problems in a simpler fashion. " [i6]
"Also, wenn ich das vergleiche, wie viel Code ich für manche Sachen schreiben musste die ich vor einem Jahr gemacht habe. In Kotlin sind es drei Zeilen, in Java sind es 30 oder 20." [i1]

Funktionalität Die größte Unterkategorie ist Funktionalität. In zehn von elf Interviews wurden funktionale Eigenschaften einer Programmiersprache als Argument genutzt. Die Aussagen beschreiben Funktionen oder Eigenschaften, die eine Programmiersprache hat:

"Java ist auch relativ gut beim Testen. Also man kann auch relativ gut Testcases schreiben" [i2]

"Was ganz gut ist, dass man eine Programmiersprache hat für die Darstellung im Frontend und auch eine Programmiersprache hat um im Backend bestimmte Parameter, bestimmte Logiken halt eben zu machen. Das ist sehr praktisch, wenn man auf beiden Seiten arbeitet." [i5]
Außerdem werden Anforderungen an eine Programmiersprache genannt:
"I need to write my logic and under the hood the things should be solved by the language, they should be solved by the actual creators of the language. They shouldn't offload the decision on to me." [i7]

Ökosystem Eine etwas speziellere funktionale Eigenschaft ist das Ökosystem. Für mehr als die Hälfte der befragten Entwickler ist das eine wichtige Eigenschaft. Funktionen, die von Bibliotheken bereitgestellt werden sind, genauso wie Frameworks, streng genommen nicht Teil der Sprache aber spielen in der Wahrnehmung der Befragten dennoch eine große Rolle:
"Die Standartbibliotheken, das Ökosystem allgemein, du bekommt im Open Source Bereich so gut wie alles was du brauchst" [i3]
"du kannst auf viele Module zurückgreifen und die einfach installieren. " [i5]
"Also bei mir ist es jetzt Python geworden, einfach weil es auch so viel gibt. Also man kann halt so viele Librarys die was bringen in meinem Bereich einbinden" [k4]
Sowohl in den langen als auch in den kurzen Interviews wurde die Standardbibliothek erwähnt:
"Bei Php würde ich sagen sind noch mehr Grundfunktionen in der Standartbibliothek dabei. " [i8]
"Also bei Java fand ich immer gut, dass schon sehr viele Klassen vorhanden waren, also für String, für Netzwerk und ähnliches." [i5] "(*Python*)Hat eine sehr umfangreiche Standartbibliothek" [k5]

Typisierung Obwohl auch in vier langen Interviews eine der zum Vergleich herangezogenen Sprachen eine dynamisch typisierte Sprache war, gab es ausschließlich Aussagen, die in statischer Typisierung einen Vorteil gesehen haben:
"Auch gerade, wenn die Programmiersprache halt statisch typisiert ist und dich dadurch halt zwingt wirklich mit anzugeben ob das jetzt Integer oder Float ist. Also meiner Meinung nach ist das ein Vorteil." [i10]
"Die starke Typisierung mag ich auch noch. Also nicht irgendwie, es ist was es gerade Lust hat zu sein. Es ist genau eine Sache." [k10]
"Also mit dem Typsystem, das gefällt mir ganz gut" [i3]
"its also a statically typed language which i like" [y]

Erfahrung Ein weitere Faktor, der die Produktivität beeinflusst ist die Erfahrung die ein Entwickler mitbringt. In fast der Hälfte aller Kurzinterviews wurde die Erfahrung in einer Sprache als Antwort auf die Forschungsfrage genannt. Einige sahen das als kritisch an:
"Also meine Lieblingsprogrammiersprache ist C weil ich da am besten mit zurecht

komme." [k3]
"Ja vielleicht. Weil ich die Sprache schon ganz gut kenne. Ist total erbärmlich." [k5]
Auch in den langen Interviews wurde über den Einfluss von Erfahrung gesprochen, dort aber weniger direkt als Grund für die favorisierte Programmiersprache. (Dazu mehr in 3.3):
"Natürlich ist ein Experte für eine Sprache unglaublich viel schneller als jemand der gerade erst anfängt" [i9]
"Wie gesagt, Startvorteil für eine Auswahl ist auf jedenfall da wo ich mich schon auskenne und wohlfühle und mich dementsprechend auch direkt einbringen kann." [i8]

Tooling Eine weitere Kategorie die für einige der Befragten eine Rolle spielt ist die Unterstützung durch Werkzeuge die den Entwickler unterstützen: "totale Toolunterstützung in dem Bereich und die ganze Umgebung die finde ich sehr gut." [i3]
"Was ich in beiden Sprachen mag, du hast eine sehr gute Paketorganisation, das heißt du kannst auf viele Module zurückgreifen und die einfach installieren. Das eine ist halt NPM und das andere ist dann Pip und das sind dann sehr sehr gute Tools." [i1] "Bei Javascript, zumindest das Frontend finde ich sehr angenehm, dass man halt in der Debugkonsole vom Browser ein komplettes Debugging hat. Da du da einen Breakpoint setzten kannst, dass du da direkt an der Stelle eigenen Code eingeben kannst und der wird evaluiert und halt so debuggen kannst und richtig gut durch den Code durchsteppen kannst." [i8]
"... auch die exzellente Unterstützung durch die Entwicklungsumgebung, die weiß halt relativ genau, was kann hier aufgerufen werden, was kommt hier zurück. " [k10]

Geschwindigkeit Aus den Aussagen einiger der Befragten hat sich die Kategorie Geschwindigkeit herausgebildet. Dabei geht es um die Geschwindigkeit mit der Fortschritte in zum Beispiel einem Projekt gemacht werden.
"The way the language is build, it makes it really easy to prototype something very quickly" [k7]
"Aber einfach, dass du so ganz schnell produktiv sein kannst mit den Bibliotheken" [i3]
"Aktuell finde ich so Sprachen wie Go und Rust persönlich interessanter und finde es auch schneller damit Sachen umzusetzen." [i4]

Lesbarkeit Die letzte Kategorie aus dem Bereich Produktivität ist die Lesbarkeit des Codes. Aussagen die in diese Kategorie fallen sind verstärkt in den langen Interviews gefallen (Sieben von Elf) und nur einmal in einem der kurzen Interviews. Das Verständnis von Schönheit ist natürlich höchst Subjektiv. Das zeigen auch schon die ersten beiden Zitate:
"Also ich halte C++ für eine sehr lesbare Sprache" [i10]
"C++ kann manchmal ziemlich kryptisch werden." [i2]
"Whereas with Kotlin a lot of the complexity is abstracted away so you are sort of working with a little bit more high level code and it's much more functional and it's much easier to read and write" [i6]

"Aber ich habe schon das Gefühl, dass auch viele von den Bibliotheken sehr sehr viele Sachen vor einem verstecken, so dass das dann sehr elegant aussieht aber man übersieht dann manchmal Sachen." [i1]

3.1.2 Befriedigung - "Was macht mir Spaß"

Die zweite zentrale Kategorie beschreibt sehr subjektive Faktoren. Auch hier habe ich als Oberbegriff ein Wort gewählt, das zwar nicht in den Interviews vorkommt aber die darunterliegenden Konzepte sehr gut beschreibt. Alle Kategorien beschreiben ein Bedürfnis, das von der Sprache befriedigt oder nicht befriedigt wird.
In alle langen Interviews wurden Konzepte erwähnt, die unter die Schlüsselkategorie fallen. Einigen Personen war das wichtiger (15 Aussagen[i6]) einigen eher weniger (1 Aussage[i11]).

Leichtigkeit Einige der Befragten argumentieren, dass ihnen die Programmiersprache nicht "nicht im Weg steht" [k5] bzw. sie nicht bei der Arbeit behindert und keine Dissonanz entsteht zwischen dem was der Entwickler erwartet und dem was die Software tut:
"Also vor allem weil gefühlt Kotlin einfach sehr sehr gut funktioniert." [i1]
"Wenn ich im Datenanalysebereich etwas machen muss will ich mich nicht um Typen, nicht um den Classpath kümmern muss, sondern es funktioniert einfach in der sehr gut abgestimmten Umgebung" [i3]
"Weil ich mich nicht mit zu vielen Kleinstdetails rumschlagen muss." [k10]

Spaß Spaß ist ein schwierig zu definierendes Konzept. Der Begriff wurde aber von der Hälfte der Befragten in den langen Interviews benannt. In den englischen Interviews ist öfter der Begriff "Enjoy" gefallen. In dem verwendeten Kontext wurden die Begriffe semantisch äquivalent verwendet. Deshalb habe ich die beiden Begriffe gleichgesetzt. Erwähnung fand der Begriff nur in den langen Interviews.
"Also manches davon, klar, ist persönliche Einschätzung und wie viel Spaß mir die Sache macht." [i4]
"And I think as well that I started really enjoy coding again when I was writing in Kotlin." [i6]
"Nichts desto trotz würde ich trotzdem noch versuchen das in Java noch alles nachzuprogrammieren aber das macht mehr Arbeit, als dass es Spaß macht." [i3]

Vorhersagbarkeit Eine weitere Kategorie, die in der Hälfte der Interviews zu Sprache kam ist die Vorhersagbarkeit. Einer der Befragten hat dabei den Begriff "berechenbar" verwendet. Um die Doppelbedeutung die er auslöst zu umgehen habe ich den Begriff "Vorhersagbarkeit" als Kategorie gewählt.
"Es ist sehr berechenbar was man tut, also nicht berechenbar von der Laufzeit, sondern es ist sehr klar, wenn man was schreibt, was los ist." [i3]
"Die sind irgendwie groß und nicht cool und funktionieren halt irgendwie." [i4]
"Yes that's true but in the relationship with Java and Kotlin with Kotlin for Android it still compiles to JVM so in essence it's an abstraction of Java code so coming from a Java background you can understand what Kotlin is doing." [i6]

Schönheit Ähnlich wie Lesbarkeit ist auch Schönheit sehr subjektiv. Einige Benutzen den Begriff in Bezug auf die Syntax, andere auf die Art und Weise wie etwas gelöst wird.
"Ich glaube was einen begeistert, ist wenn etwas elegant einfach und eingängig ist." [i5]

"Es geht schon darum, dass man eine möglichst elegante Lösung zu finden. Also meiner Meinung nach ist eine Lösung immer dann elegant, wenn du mit möglichst wenig Code möglichst viele Eingaben erschlägst." [i10]
"Stylistically Java is probably my favorite, I used that for many years." [y]

Innovation In der letzten Kategorie, geht es um die Weiterentwicklung der Sprache und die Adaption von neuen Konzepten. Das Abstraktionslevel spielt dabei auch eine Rolle. In dieser Kategorie sind die Aussagen am uneinheitlichsten. Die Unterschiede werde ich in 3.2 noch genauer betrachten.
"Also Java ist ja schon ein bisschen älter. Typescript ist moderner und man merkt auch, dass da einige Konzepte sind die ein bisschen mehr können." [i9]
"Was natürlich cool bei so sprachen wie Go und Rust ist, dass die sich wesentlich schneller weiterentwickeln. Bei Java ist häufig das Problem, dass eine neuen Java Version dann ein paar Jahre gedauert hat." [i4]
"For Dart for example I don't like it as much as Kotlin because it seems to be still quite an early language, really crude" [i7]

3.2 Erkenntnis 1: Entwickler brauchen Herausforderungen, entweder durch wechselnde Domänen oder durch die Programmiersprache

Einigen der Interviewten sind neue Features sehr wichtig und sie wollen neue Features und Konzepte, die ihnen erlauben Probleme auf eine neue, bessere Art lösen zu können (" I felt like i have to be on the forefront of technology. So I would be very uncomfortable to use a technology that I think is dated or deprecated so to me thats quite important" [i6]). Aus der Auswertung ergibt sich, dass es vier Advokaten dieser Haltung gibt. Alle vier eint, dass sie Software für die immer gleiche Domäne oder sogar dasselbe Projekt entwickeln (drei entwickeln Apps für Android [i1, i6, i7], einer arbeitet an einer Server-Applikation [i4]). Das Bedeutet, dass die Probleme die sie in ihrer täglichen Arbeit lösen müssen häufig ähnlicher Natur sind. Dazu kommt, dass diese vier Personen keine Aussagen getätigt haben, die unter das Konzept Geschwindigkeit fallen.

Hypothese: Wer lange in derselben Domäne die immer gleichen Probleme löst ist schnell. Da die Lösung der Domänenprobleme nicht mehr als Herausforderung im Vordergrund steht, suchen sich die Entwickler neue Herausforderungen. Eine der Herausforderungen scheint zu sein, mit neuen Sprachen elegantere Wege zu finden, um das bekannte Problem einfacher zu lösen.
"But in our community of Android developers in Berlin, most of them are young and they are still keen on learning new stuff, and that's amazing, and the problem with Android was that you really didn't have the opportunity that much to learn new stuff. The language didn't change for so long, so people wanted, including me, a breeze of

new fresh air." [i7]
"Java war halt immer so hinterher bei dem was gerade hip ist. Es fühlte sich halt immer sehr altbacken an." [i4]
"were as with something like Kotlin were it's more exciting and vibrant and you get a lot of Android developers and a lot of very , I don't want to say opinionated but i mean very passionate developers who are very keen to adopt the leading technologies, who want to try out new things, who then go to champion these features and become advocates for them in a community, so it seem that there is more involvement in the community." [i6]

Entwickler die in unterschiedlichen Domänen arbeiten bewerten Sprachen öfter im Bezug auf die Geschwindigkeit und das Ökosystem [i3,i8, i9, i10]. Die Programmiersprache ist dabei Mittel zum Zweck (vergleichbar mit 3.1.2[i1,i3,i9,i10]) da die Herausforderung in der Bewältigung des Problems liegt. Die Erwartungshaltung an die Programmiersprache ist dann, dass diese der Lösung des Problems nicht im Weg steht.
"Also neue Sachen probiert man vor allen Dingen dann aus wenn man die Zeit hat. Solange es keinen äußeren Zwang gibt kann man das gerne machen. Wenn das nicht so ist dann sagt man sich meistens "Ok ich nehme jetzt das von dem ich weiß, dass es funktioniert"" [i10]
"Mit C++ und Qt können wir relativ einfach Library die wir auf dem Desktopsystem oder auf Embedded entwickeln auf dem jeweiligen anderen System wiederverwenden." [i11]
"Also die Programmiersprache alleine ist nur die halbe Miete, sondern du brauchst halt auch noch gute Librarys." [i10]

Validität: Da in den Kurzinterviews, vermutlich durch die Natur der gestellten Frage, keinerlei Aussagen getroffen wurden, die den hier angesprochenen Sachverhalt betreffen, ist die Datenbasis noch geringer als im Rest der Studie. So ist gerade die erstgenannte Gruppe demografisch sehr homogen. Deshalb ist es möglich, dass die ähnlichen Positionen in der besprochenen Sache nicht auf andere Personen übertragbar ist.

3.3 Erkenntnis II: Erfahrung ist der wichtigste Grund eine Programmiersprache zu mögen

In den Kurzinterviews und den Aussagen aus dem Youtube-Video gibt es eine Häufung von Antworten, die die Präferenz auf der Erfahrung in der Sprache basieren. ("Das ist die Sprache die ich am besten kann eigentlich." [k5], "favorite? probably PHP because it's where I'm best at. It's messy" [y])
In den langen Interviews spielt das Thema Erfahrung eine untergeordnete Rolle. Viele der funktionalen Argumente sind allerdings nicht sehr spezifisch und treffen auch auf einige andere Sprachen aus derselben Klasse zu.
"Also die Programmiersprache alleine ist nur die halbe Miete, sondern du brauchst halt auch noch gute Librarys." [i10]
"Also von der Programmiersprache, klar die Paradigmen dahinter. Also Javascript ist deutlich mehr funktional." [i8]
Softwareentwickler, die öfter die Sprache gewechselt haben sehen keine großen Un-

terschiede.
"So it wasn't a problem to jump from Kotlin (*to Dart*) because it's the same class of languages. It's not like going to Clojure or Haskell, which provides lots of issues for somebody who never did it. If it's the same class of languages it's just a different toolkit, ... different Hammer." [i7]
"Ich denke ich könnte in allen Sprachen dasselbe machen. (...)Ich mag das eigentlich gerne mal die Sprache zu wechseln." [i4]

Auf die Frage welchen Einfluss die Programmiersprache bei der Arbeitgeberwahl hat, hat sich niemand darauf festgelegt nur in einer spezifischen Sprache programmieren zu wollen, aber wenn auf zukünftige Entscheidungen eingegangen wurde, wollte auch niemand zwingend eine andere Sprache.
"Wäre da aber schon offen auch für andere Sprachen, allerdings würde ich durchaus schon schauen welche Sprachen das sind. Also bei mir mit Haskell hätte ich jetzt weniger Lust drauf, oder Cobol. Sollten schon Sprachen sein, wo ich sage, ja, das liegt mir jetzt." [i8]
"Ich würde jetzt eher das bevorzugen was ich kenne, also C++, C und Java. Und ausschließen würde ich denke ich alles was mit Webprogrammierung zu tun hat, da ich da bisher noch fast gar nichts gemacht hab." [i11]
"Ich würde schon sagen, dass man sich wohler da fühlt was man schon kennt oder was man interessant findet." [i9]

Hypothese Funktionale Argumente sind oft vorgeschobene Argumente die das eigentliche Argument "Erfahrung" verdecken. Erfahrung und Befriedigung bei der Arbeit mit der Sprache bedingen und verstärken sich gegenseitig. Um in einer Sprache Erfahrung zu sammeln darf sie nicht Nicht-Gefallen. Wenn die Erfahrung da ist wird sie schnell zur favorisierten Sprache, da die Eingewöhnung gefühlt viel Energie kostet und mit dem Verlust an Leichtigkeit verbunden ist.
"Also, wenn man sehr lange sehr viel mit einer Sprache etwas gemacht hat, ist man da natürlich sehr vertraut damit und es fühlt sich gut an ein Spezialist zu sein und auch wirklich so die Details zu kennen. Das glaube ich ist so ein Grund warum jemand seine Programmiersprache liebt." [i5]
"Ich habe keine Lieblingsprogrammiersprache. Ich programmiere einfach in den Programmiersprachen die ich gelernt habe." [k2]
"Man kennt sich mit irgendwas aus. Man könnte in anderen Sprachen wahrscheinlich auch etwas ähnliches machen aber die Hürde da jetzt zu wechseln. Ist halt erst wieder ein bisschen Arbeit da reinzukommen. Ob das jetzt soviel mehr Vorteile bietet." [k5]

Validität In der Implikationskette ist besonders der Mechanismus der Gegenseitigen Beeinflussung von Erfahrung und Befriedigung nicht durch die Daten ausreichend belegbar. Dabei besonders die Grundlage auf der eine Entscheidung für eine Sprache gefällt wird. Einige Anhaltspunkte dafür gibt es in den Daten aber für eine definitive Aussage reicht dies nicht aus.

3.4 Erkenntnis III: Softwareentwickler reflektieren zu wenig über die verwendeten Programmiersprachen

Wie in 3.3 schon dargelegt sind viele der funktionalen Argumente eher oberflächlich. Vereinzelt werden auch Eigenschaften erwähnt, die sehr speziell eine Programmiersprache betreffen.
"Du kannst aber auch ohne Probleme Sachen außerhalb dieser Klasse definieren. Also du schreibst ne Klasse in die Datei rein, Also zum Beispiel ein "Apple" Objekt und irgendwelche static Variablen die eigentlich zu diesem Objekt gehören definierst du außerhalb." [i9]
Da bei Argumenten dieser Art fast immer von Erfahrungen berichtet wird, in denen etwas nicht funktioniert hat, liegt die Vermutung nahe, dass diese Beispiele Erfahrungen aus der näheren Vergangenheit widerspiegeln und keine Zeichen von Reflexion sind.
Die Entscheidung für eine Programmiersprache wird oft mit dem Anwendungszweck begründet.
"Also wie gesagt, bei meinem vorherigen Job musste ich ja Objektiv-C lernen obwohl ich das nicht so toll fand. Aber ich musste so LowLevel Sachen auf MacOS machen und das geht halt nicht anders." [i10]
" C++ Zwang kommt durch Embbeded da fällt Java eigentlich weg." [i11]
"Wenn ich irgendwas Hardware näheres machen will, würde ich halt C nehmen" [i8]
Den Einfluss der Stellenauswahl auf die Programmiersprache sind sich auch alle Befragten bewusst.
"Das war schon auch ein Grund sozusagen da hin zu gehen." [i1] "Es war die Sprache der Firma, es gab da auch keine Diskussion. Ich hatte auch nicht vor das zu diskutieren. Für mich war das auch kein Grund nicht dahin zu gehen." [i4] "But I also would be flexible if the company had said we are using this, this and this and would think ok" [i6]

Hypothese Softwareentwickler vergleichen wenig welche Faktoren ihre tägliche Arbeit mit einer Programmiersprache beeinflussen. Die Entscheidung der Programmiersprache wird in Verbindung mit dem Anwendungszweck getroffen und deshalb bei der Stellenauswahl getroffen.

Validität Vergleiche von Programmiersprachen die in derselben Domäne eingesetzt werden, waren unter den Befragten unterrepräsentiert. Nach dem Prinzip des theoretischen Samplings wäre hier eine entsprechende Auswahl der Befragten im weiteren Verlauf sinnvoll.

3.5 Vergleich mit bisheriger Arbeit

Viele der in 3.1 entdeckten Kategorien finden sich in den Faktoren von Chen et al. [2] wieder. Die laut ihrer Analyse wichtigsten Faktoren lassen sich bis auf einen auf die hier gefundenen Kategorien Abbilden "generality"(Leichtigkeit und Funktionalität), "reliability"(Vorgersagbarkeit), "Machine independance" (keine Äquivalente Kategorie) und "extensibility"(Funktionalität/Ökosystem).
Meyerovich und Rabkin untersuchen unter anderem Faktoren, die die Entscheidungsfindung hin zu einer neuen Programmiersprache beeinflussen [8, 4.]. Viele der hier

gefunden Faktoren werden von den Befragten aus dieser Studie ähnlich eingeschätzt. Auch wenn die Größe der Stichprobe hier zu klein ist, eine Übereinstimmung hier durchaus ein Zeichen, dass eine gewisse Aussagekraft in den Aussagen zu finden ist. Der wichtigste Faktor in der zitierten Studie ist die Verfügbarkeit von Open Source Bibliotheken. 65% der Befragten stimmen der Aussage zu. In 15 von 21 Interviews gab es in meiner Studie ebenfalls Aussagen, die die Wichtigkeit von Bibliotheken betrafen (ich habe keine Unterscheidung zwischen Open Source und kommerziellen Bibliotheken vollzogen).
In der Rangfolge der Faktoren liegt bei Meyerovich und Rabkin die "Personal familiarity" mit unter 60% hinter den Faktoren "Extending existing Code" und "Already used in group". Dies kann als Indiz für 3.4 gesehen werden.
Andere wichtige Faktoren wie "Development Speed"(45% zu 5 von 11) und "Tools"(40% zu 6 von 11) werden Ebenfalls von ähnlich vielen Personen erwähnt.
In Kapitel 6.2 der selben Veröffentlichung wurden aus einem anderen Datensatz die Zustimmung zur Aussage "I enjoy using the language" mit Zustimmung zu anderen Aussagen verglichen. Die Fragestellung bezeichnen die Autoren selbst als die den "closest proxy" zum Konzept von "liking a language"
Die stärkste Korrelation der Antwortverteilung gibt es zu den Aussagen "This language is expressiv" (Korrelation 0.76), "I find code written in this language very elegant" (Korrelation 0.73) und " I rarley have difficulty in abstracting patterns I find in my code" (Korrelation 0.66). Auch diese Aussagen lassen sich auf die von mir gefundenen Kategorien abbilden.

3.6 Ausblick

Um aus den Hypothesen Theorien zu entwickeln wären nach dem Konzept des theoretischen Samplings noch weitere Interviews nötig. Gruppen, die für weitere Interviews in Frage kommen würden, wären folgende:

- Sprachentwickler
- Entwickler, die sehr erfahren sind und mit vielen verschiedenen Sprachen gearbeitet haben
- Softwarearchitekten, die in Projekten die Entscheidung für die verwendeten Programmiersprachen treffen

Insbesondere wäre es auch notwendig Softwareentwickler aus anderen Ländern zu befragen und dabei auch Frauen mit in die Befragung einzubeziehen.

Im Weiteren wäre es sinnvoll, die aus den qualitativen Daten gewonnenen Einsichten durch quantitative Umfragen zu evaluieren. Hierfür würden sich primär Erkenntnis 1 (Siehe: 3.2) und 2 (Siehe: 3.3) eignen.

Literaturverzeichnis

[1] Selim Baykara. Sind Sie Programmierer oder Software-Entwickler? https://jaxenter.de/sind-sie-programmierer-oder-software-entwickler-1545, 2015. [Online; accessed 10-January-2019].

[2] Yaofei Chen, Rose Dios, Ali Mili, Lan Wu, and Kefei Wang. An empirical study of programming language trends. *IEEE software*, 22(3):72–79, 2005.

[3] Computerphile. What's your Favourite Programming Language? (sound check Q). https://www.youtube.com/watch?v=p8-rZOCn5rQ, 2018. [Online; accessed 05-January-2019].

[4] Vito Flaker. *Von der Idee zur Forschungsarbeit: Forschen in Sozialarbeit und Sozialwissenschaft*. Böhlau Verlag Wien, 2006.

[5] Barney G Glaser et al. *The discovery of grounded theory: Strategies for qualitative research*. Routledge, 1967.

[6] Barney G Glaser, Anselm L Strauss, and Axel T Paul. *Grounded Theory: Strategien qualitativer Forschung*. Huber, 1998.

[7] Friedrich Krotz. *Neue Theorien Entwickeln*. Herbert von Halem Verlag, 2005.

[8] Leo A Meyerovich and Ariel S Rabkin. Empirical analysis of programming language adoption. *ACM SIGPLAN Notices*, 48(10):1–18, 2013.

[9] Sebastian Nanz and Carlo A Furia. A comparative study of programming languages in rosetta code. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 778–788. IEEE, 2015.

[10] Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[11] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[12] Wikipedia. Softwareentwickler. https://de.wikipedia.org/wiki/Softwareentwickler, 2019. [Online; accessed 10-January-2019].