universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Wer verbindet Technik mit den Nutzer/innen? –

Programmierer/innen und ihre Vermittlungsposition zwischen Technik und Gesellschaft“

verfasst von / submitted by

Johanna Goldbeck, BA

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Arts (MA)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 905
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Soziologie
Betreut von / Supervisor:	Assoz. Prof. Mag. Dr. Maximilian Fochler

Danksagung

An dieser Stelle bedanke ich mich ganz herzlich bei meinem Betreuer, Dr. Maximilian Fochler, der mir immer alle Fragen beantworten konnte und auf einfache, unkomplizierte Weise zur Seite stand, wenn es Unklarheiten oder Unsicherheiten gab. Dank ihm konnte ich, aus jedem Besprechungstermin in meinem Vorhaben bestärkt, die nächsten Schritte wagen.

Außerdem danke ich meiner Familie, besonders meinen Eltern Christine und Martin, die mir ermöglichten, ein Studium zu absolvieren und mich in allen meinen Entscheidungen unterstützten. Sie haben mich dazu angehalten, das Masterstudium bis zum Ende durchzuziehen und mir offene Arme und ermunternde Worte geliehen.

Abschließend bedanke ich mich ganz besonders bei meinem Mann, für die endlose Geduld, trotz der ständigen Verzögerungen, und die bestärkenden oder beruhigenden Worte, wenn die Motivation einen Tiefpunkt erreichte.

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Publikationen entnommen sind, sind als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt."

Wien, 20.09.2019

Inhaltsverzeichnis

1 Einleitung

Was hat sich nicht alles verändert in den vergangenen 25 Jahren? Wir können mittlerweile mit strom-betriebenen Autos fahren, die wir bald vielleicht nicht einmal mehr steuern müssten, weil sie der Kategorie „*self-driving*“ zugeordnet werden. Anstatt eines großen, schweren Kastens, kaufen wir jetzt einen großen, flachen Bildschirm als Fernsehgerät. Die junge Generation verwendet gar kein herkömmliches Fernsehprogramm mehr, sondern nutzt verschiedene Streaming-Plattformen um Lieblingssendungen dann zu sehen, wenn es ihr zeitlich passt. Die neuesten Schlagzeilen lesen wir auch nicht mehr in der Zeitung im großen Papierformat. Wir bekommen sie zum Beispiel auf unser Tablet geschickt. Wenn wir etwas wissen wollen, denken wir uns nicht, dass wir das später im Lexikon nachschlagen sollten. Nein, wir “*googeln*” es ganz einfach im selben Moment, in dem es uns einfällt. Wir geben einer Frauenstimme aus einer schwarzen Box namens Alexa Befehle und wir steuern unser Smart-Home mit unserem Smartphone, ganz gleich ob es die Heizung, die Jalousie, die Kaffeemaschine oder das Radio ist. Unsere Kommunikation läuft fast ausschließlich über ein kleines flaches Gerät auf dessen Bildschirm wir herumtippen und wischen. Selbst wenn wir am anderen Ende der Welt sitzen, können wir entspannt unseren Müttern durch das Smartphone zuwinken (sofern sie sich schon auf dieses Gerät eingelassen haben versteht sich).

Laut einem Bericht der Statista Webseite vom August 2019, nutzen rund 77 % der österreichischen Bevölkerung ein Smartphone fast jeden Tag. Bei den unter 30-jährigen sind es sogar 96 % (Schultz 2019, statista.com). Manche Menschen würden ohne dieses “Wunderwerkzeug” sogar in ein chaotisches Loch fallen, weil sie ihr gesamtes Leben damit organisieren und es wie ein zweites Hirn fungiert. Unsere Gesellschaft und die Art und Weise, wie wir miteinander interagieren hat sich also grundlegend verändert aufgrund des Smartphones und wie wir mit dem Gerät selbst umgehen.

Doch wie kommt es, dass wir so an dieses Gerät gebunden sind? Das Smartphone alleine ist ja noch nicht alles. Der wesentliche Bestandteil eines jeden Smartphones sind die Apps (Applikationen/Anwendungen)[1], die darauf installiert sind, ohne die ein Otto-

[1] App kommt vom englischen application software, was wörtlich übersetzt Anwendungsprogramm bedeutet. Eine App ist also ein Computerprogramm (meist auf einem Smartphone oder Tablet), das in einem bestimmten Lebensbereich oder zu einem bestimmten Thema spezifische Aufgaben erfüllt und dabei den Nutzer/innen hilft, Probleme zu lösen oder einfach Spaß bereiten soll (SEO-Analyse.com https://www.seo-analyse.com/seo-lexikon/a/app/; gruenderszene.de https://www.gruenderszene.de/lexikon/begriffe/app?interstitial).

Normalverbraucher/in nicht viel mit dem Smartphone machen könnte.
Was steckt also hinter diesen Apps und wer ist dafür verantwortlich?
Es sind App-Entwickler/innen, die die Apps für die beiden großen Betriebssysteme iOS (Apple) und Android (Google) konzipieren und programmieren. Mit dem Code den sie schreiben, sagen sie der Technologie, was sie tun soll um die User (Nutzer/Innen; AnwenderInnen) mit der App zu unterstützen. Dazu verwenden sie eigene Programmiersprachen, mit denen sie ihren Willen auf "Computerisch" übersetzen. Programmierer/innen können daher als Mittelspersonen gesehen werden, die App-Technologie zur Gesellschaft bringen und sie damit verbinden. Gleichzeitig entwickeln sie nicht nur irgendwelche Apps, wie weiter unten ersichtlich wird, sondern sie richten sich auch nach den Wünschen der Individuen in einer Gesellschaft.
So wie ein paar Zeilen weiter oben schon aufgelistet, zeigt auch eine höchst interessante anthropologische Studie über „*computer engineers*" von Downey (1998) wie stark unsere Gesellschaft von Technologie durchtränkt ist. Seiner Ansicht nach, sind Gesellschaften und Computer wechselseitig co-produziert, was bedeutet, dass sich ein Teil von uns in der Maschine befindet und ein Teil der Maschine befindet sich in uns. Das lässt sich heutzutage, 20 Jahre nach seiner Studie, kaum leugnen, wenn man bedenkt, wie wir unsere Computer und Smartphones personalisieren oder wie sehr diese wie ein zweites Hirn für uns geworden sind. Mensch und Maschine sind also deutlich voneinander abhängig.
Noch direkter sieht man diese Beziehung am Beruf Programmierer/innen, die der Maschine ihr „Leben einhauchen", ihr Funktionen und auch (Bildschirm-)Ästhetik verleihen und dabei einen Teil ihrer selbst und ihrer Wahrnehmung von der Gesellschaft einschreiben. Ich sehe Programmierer/innen daher als wesentliche „Gestalter/innen" der Zukunft unserer Gesellschaft, deren Zusammenleben mit der Technik sich in den kommenden Jahren noch weitaus intensivieren und ausbreiten wird. Das erinnert auf gewisse Weise an den SCOT Ansatz, bei dem es im Grunde darum geht, dass auch Technologie nicht einfach von der Natur gegeben ist, sondern in sozialen Prozessen konstruiert wird (Bijker 2009). Ich nehme also an, dass die gesellschaftliche Umgebung in der ein/e Programmierer/in sich befindet, sich im geschriebenen Code widerspiegelt, was durch die Technologie des Endprodukts sichtbar wird. Das Endprodukt wiederum beeinflusst die Teile der Gesellschaft, die dieses anwenden. Deshalb verstehe ich die programmierenden Personen als eine Art von Bindeglied oder Verbindungsstück zwischen Technologie und Gesellschaft.
Umso erstaunlicher ist es, dass es trotz der fortgeschrittenen technologischen Entwicklung und Durchtränktheit der Gesellschaft, noch relativ wenig soziologische Untersuchungen zu Programmierer/innen und App Technologie gibt. Es gibt zwar ein paar wenige

Forscher/innen, die diese Berufsgruppe in ihrer Arbeit ähnlich wie ein Verbindungsstück betrachten (Heim 1974; Kupka 1983; MacKenzie and Wajcman 1985; Flanagin et al. 2010; Petersen 2015), aber explizit als ein solches wurden die programmierenden Personen von kaum jemandem bezeichnet.

Die vorliegende Arbeit nimmt sich diese Forschungslücke zum Anlass, App-Entwickler/innen als Verbindungsstück zwischen Gesellschaft und Technologie genauer zu erforschen, indem qualitative Interviews themenanalytisch untersucht werden. Aus den resultierenden Themenkategorien wurde versucht Typen von Programmierer/innen zu entwickeln, die eine unterschiedliche Sichtweise auf die Gesellschaft, ihre Produkte und ihre Rolle haben.

Zu Beginn werden die Grundlagen und die Weiterentwicklung der Techniksoziologie, sowie einige verschiedene Forschungsbeispiele, die sich mit Programmierer/innen beschäftigen, behandelt. Daraufhin wird klargestellt, welches Ziel diese Arbeit verfolgt und warum sie für die Wissenschaft und die Gesellschaft relevant ist. Als nächstes folgt mit der Empirie der Kern der Arbeit, in der viele interessante Beispielzitate aus den Interviews angeführt werden. Zum Abschluss wird versucht die Ergebnisse mit der Literatur zu verknüpfen, gefolgt von einem Fazit und Ausblick.

Im nächsten Abschnitt folgt eine kurze Begriffsbestimmung, woraufhin der Forschungsstand zu diesem Thema beleuchtet wird.

2 Begriffsbestimmung

2.1 Programmierer/innen - Entwickler/innen

Theoretisch ist es möglich die Begriffe Programmierer/in und Entwickler/in zu trennen und es gibt auch Menschen, die sich die Mühe machen, diese Berufe differenziert mit ihren jeweiligen Besonderheiten zu betrachten (Weddehage, 2015 - entwickler.de; Kaplan, 2014 – sleepeaseysoftware.com; Skorkin 2010 – skorks.com).

So kann man Programmierer/innen als jene Personen verstehen, die sich stärker auf das Schreiben des Codes fokussieren und das meisterhaft beherrschen. Entwickler/innen hingegen, werden oft so dargestellt, dass sie sich außerdem noch viele Gedanken zu Spezifikationen und Design von technologischen Lösungen machen. Zudem weisen sie mehr kommunikative Fähigkeiten, sowie Problemlösungskompetenzen auf. Keiner der beiden Berufe sollte aber als besser oder schlechter als der andere verstanden werden. In der realen Arbeitswelt oder im Gespräch mit Nicht-Programmierer/innen tut diese Unterscheidung jedoch kaum zur Sache und die Bezeichnungen werden meist synonym verwendet. Hauptsächlich deshalb, weil es sich bei den beschriebenen Definitionen eher um Idealtypen handeln dürfte, welche in der Realität kaum in ihrer Reinform auftreten. Das heißt, die Grenzen zwischen den Tätigkeitsbereichen beider Berufe sind so verschwommen oder überlagernd, dass es für den täglichen Gebrauch nicht wirklich notwendig ist, diese genaue Trennung vorzunehmen (ebd.). Da es nicht das Ziel dieser Arbeit ist, die einzelnen Berufspezifikationen und deren Besonderheiten zu beschreiben, werden die beiden Begriffe auch in dieser Arbeit synonym verwendet.

2.2 Start-up

Als ein Start-up wird ein Unternehmen bezeichnet, das erst vor kurzem gegründet wurde, eine innovative Idee umsetzen will und häufig wenig finanzielle Mittel zur Verfügung hat, das aber hohes Wachstumspotenzial besitzt und bis zur Etablierung am Markt meist von externen Investitionen wie *Crowdfunding*[2], *Venture Capital*[3] oder *Business Angels*[4]

[2] Wörtlich übersetzt heißt Crowdfunding Gruppenfinanzierung, ein im Deutschen bekannterer Ausdruck ist Schwarmfinanzierung. Es geht dabei darum, Internetnutzer (eine große Menge davon), die sich für das eigene Projekt interessieren könnten, um eine Spende zu bitten. Es ist also eine Art, das eigene Projekt zu finanzieren, wenn man selbst nicht die Mittel hat. Der Aufruf zur Spende, also zum Crowdfunding findet meist über eine bestimmte Webseite statt und in vielen Fällen im Zusammenhang mit der Unternehmensgründung (Bendel, Gabler Wirtschaftslexikon; gründerszene.de).

[3] Venture Capital bedeutet zu Deutsch Risikokapital/Wagniskapital. Es handelt sich um Investment, das häufig an relativ junge Unternehmen vergeben wird als Unterstützung oder Entwicklungshilfe für die erste Zeit der Etablierung. Den Investor/innen ist das Risiko dabei bewusst, dass sie beim Scheitern der jungen Firma, ihr Kapital nicht wieder bekommen (gründerszene.de).

[4] Ein Business Angel ist eine Person, die ein junges Unternehmen mit Eigenkapital unterstützt, sowie mit Know-How aus

abhängig ist. Solche Unternehmen können grundsätzlich in den verschiedensten Branchen gegründet werden, solange die Geschäftsidee innovativ ist, also nicht nur in der Technologiebranche. Dennoch werden die meisten Start-ups in dieser Branche gegründet, da Technologieentwicklung und Innovation meist Hand in Hand einhergehen. Die häufigsten Start-ups kommen in den Bereichen *Electronic Business*, Kommunikationstechnologie oder *Life Sciences* vor (gruenderszene.de; Gabler Wirtschaftslexikon, 2017). Für diese Arbeit wurden Programmierer/innen von Start-ups oder Unternehmen, die laut einzelner Interviewpersonen bis vor kurzem noch Start-ups waren, für die Interviews ausgewählt.

2.3 Nicht-Techniker/innen

Da weiter unten in den Ergebnissen immer wieder “Nicht-Techniker/innen” erwähnt werden, soll hier kurz geklärt werden, was diese Arbeit darunter versteht. Als Nicht-Techniker/innen werden schlicht und einfach jene Personen gemeint, die keine professionelle computer-technische Ausbildung absolviert haben. Das heißt auch andere technische Berufe, die nichts mit Computertechnik zu tun haben, werden in dem Fall als Nicht-Techniker/innen bezeichnet.

2.4 Usability

Der Begriff Usability kann mit Gebrauchstauglichkeit (im Fall von Produkten) oder Benutzerfreundlichkeit (für Anwendungen) übersetzt werden. Usability wird erreicht, wenn die Handhabung des Produkts oder die Navigation durch eine App einfach und intuitiv gestaltet ist, sowie auf die Bedürfnisse der Benutzer/innen zugeschnitten ist. Je einfacher die Ziele der BenutzerInnen ohne Probleme erreicht werden könne, desto höher ist die Usability, was in weiterer Folge auch zu höherer Zufriedenheit führt. Das heißt die Funktionen des Geräts oder der Software sind leicht zu bedienen und das Design ist ansprechend. Es handelt sich im Wesentlichen also um die Nutzungsqualität bei der Interaktion mit technischen Systemen (Hübsch 2014, https://entwickler.de/online/web/ui-und-ux-design-kenne-den-unterschied-143641.html). Dabei spielen auch *User Interface* und *User Experience (UI/UX)* eine wichtige Rolle. Als User Interface versteht man die Oberfläche oder Plattform, auf der die Interaktion der Nutzer mit dem Gerät von statten geht. Im Fall einer App ist das also der Bildschirm, bzw. die Steuerungselemente, die darauf angezeigt werden. Das User Interface lässt sich wiederum in verschiedenen Varianten unterteilen, die beiden wichtigsten davon sind das *Graphical User Interface (GUI)* und das

jahrelanger Erfahrung und Geschäftskontakten als Berater/in zur Seite steht (WKO Gründerservice 2019).

Web User Interface (WUI). Ersteres ist wie der Name schon sagt, die graphische Aufbereitung der Benutzeroberfläche, und letzteres ist dafür zuständig, dass die Inhalte einer Webseite im Browser richtig dargestellt werden.

Bei der *User Experience* geht es mehr um den gesamten Auftritt des Produkts. Das heißt es geht um den Gesamteindruck und die individuellen Erfahrungen, wie Emotionen oder Ansichten, die eine App oder eine Webseite hinterlassen. Dazu gehören zum einen natürlich die optische/ästhetische Aufmachung und zum anderen die Funktionalität und der reibungslose Ablauf. Um eine gute *UX* zu schaffen, ist es also von Vorteil die Personen in der Zielgruppe zu kennen um das Design und die Funktionalität entsprechend zu designen. Es ist nicht einfach *UI* und *UX* zu unterscheiden, da es Überlappungen gibt. Zum Beispiel ist es möglich eine passable *UX* zu schaffen ohne gute *UI*, da wahrscheinlich nicht alle Menschen so viel Wert auf die Optik legen. Umgekehrt wird es schon viel schwieriger, da ein Produkt einfach nicht zu gebrauchen ist, wenn es die Funktionsanforderungen nicht erfüllt, da kann es noch so schön designet sein (ebd.).

3 Problemstellung und Forschungsstand

Da diese Begriffe nun geklärt sind, soll an dieser Stelle gezeigt werden, wie Entwickler/innen in der techniksoziologischen Literatur, bzw. den *Science and Technology Studies* beleuchtet werden.

Obwohl Techniksoziologie als eigene etablierte soziologische Sparte noch recht jung ist, haben sich eigentlich schon seit den Anfängen der Soziologie bekannte und berühmte Soziolog/innen immer wieder mit Technik, ihrer Entwicklung und Verwendung in der Gesellschaft befasst. Von Marx, Weber und Schumpeter, über Ogburn, Beck, Horkheimer und Adorno, sowie Luhmann bis hin zu Hughes und Latour um nur ein paar wenige zu nennen, haben sich alle unter anderem mit dem Verhältnis von Technik und Gesellschaft beschäftigt, manche mehr, manche weniger (Häußling, 2019).

Auf internationaler Ebene ist Techniksoziologie viel stärker an die Wissenschaftsforschung gekoppelt, weshalb man meist von *Science and Technology Studies* spricht (ebd.) und obwohl die Wissenschaften und die akademische Wissensproduktion bereits ausgiebig erforscht wurden, fällt die Arbeit in der Technikforschung vergleichsweise gering aus und hielt sich eher im Hintergrund (Downey & Lucena 2004). Allein beim Betrachten des Inhaltsverzeichnisses der aktuellen Ausgabe des “Handbook of Science and Technology Studies” (Felt et al., 2017), fällt auf, dass inmitten der großartigen Bandbreite an Themen, Technologie im weiteren Sinn immer wieder am Rande dabei ist und in ein paar Kapiteln, der Fokus darauf gelegt wird, aber Computertechnologie im engeren Sinn und Programmierer/innen verschwindend gering bzw. gar nicht thematisiert werden. Natürlich ist es schwierig aus der Fülle von Forschungsbeiträgen und Artikeln, die im Moment in der wissenschaftlichen Welt kursieren, eine passende aber limitierte Auswahl für ein Handbuch zu wählen, dennoch zeigt es, worauf der Fokus in diesem Feld liegt. Das ist lediglich eine Feststellung und keine Wertung, aber es macht ersichtlich wie wenig dieses Thema in der sozialwissenschaftlichen Forschung angekommen ist, obwohl in der Einleitung richtigerweise geschrieben steht, dass “all knowledge is local and reflects the specific historical moment, cultural context as well as the networks within which it is made.” (ebd., 1). Dementsprechend ist es merkwürdig, dass die Entwicklungen in Computertechnologien und ihr Einfluss auf die Gesellschaft noch eher spärlich in den großen wissenschaftlichen Büchern vertreten sind. Umso verwunderlicher ist das in weiterer Folge, da sich Wissenschaft und Technologie eigentlich gegenseitig bedingen, insbesondere seit der Industrialisierung sind sie miteinander verkettet und “...whether it [Technology] develops and grows depends on the institutional framework as well as the cultural and intellectual milieu.”

(Das & Kolack, 1990). Das heißt ohne Wissenschaft gäbe es keine neuen Technologien und nur mit Hilfe verbesserter und innovativer Technologien, lassen sich auch in der Wissenschaft neue Erkenntnisse erreichen (Sarewitz 2016). Außerdem müsste es doch bei dem Ausmaß an Durchtränktheit der Gesellschaft mit Technologie, wie weiter oben schon erwähnt, intensive, soziologisch forscherische Auseinandersetzung mit dieser Thematik geben.

Ungeachtet der Dringlichkeit, mit der mehr soziologische Untersuchungen angebracht wären, haben sich in den Jahren ihrer Existenz in der Techniksoziologie unterschiedliche Herangehensweisen und Betrachtungsperspektiven geformt. Eine Einteilung von Häußling (2010) sieht so aus, dass man Technik von einer Makro-, von einer Meso- und von einer Mikroperspektive betrachten kann. Hinzu kommt, dass sich manche Herangehensweise auf die Herstellung von Technik fokussieren und andere auf die Verwendung von Technik. Auf der Seite der Herstellung befindet sich der sozialkonstruktivistische Ansatz der „Laboratory Studies“, zu Deutsch auch Laborkonstruktivismus auf der Mikroperspektive. Dieser wurde im Wesentlichen von Latour und Woolgar (1979) aber auch von Knorr-Cetinas Arbeit (1984) entwickelt und setzt sich mit sozialer Konstruktion von Technik und Wissenschaften durch Aushandlungsprozesse auf der individuellen Ebene in wissenschaftlichen Laboren auseinander.

Eine Stufe höher, in der Mesoebene, wird zum Beispiel der SCOT-Ansatz *(Social Construction of Technology)* verortet. Hier wird besonders das “kontroverse Zusammenspiel relevanter sozialer Gruppen” (Häußling 2010, S.635; Bijker et al. 2012) untersucht und auf der Makroebene finden sich die Pfadkonzepte wieder. Das bedeutet, dass Technik und wissenschaftliches Wissen auf verschiedenen Ebenen durch soziale Phänomene und gesellschaftliche Strukturen geprägt und geformt werden.

Betrachtet man die Seite der Verwendung von Technik, findet sich zum Beispiel die Systemtheorie auf der Makroebene, Studien zu virtuellen Gemeinschaften werden auf der Mesoebene verortet und die kulturalistische Techniksoziologie befindet sich auf der Mikroebene (Häußling 2010).

Diese Seite der Forschung wird jedoch hier kaum weiter beschrieben, da in dieser Arbeit mit Programmierer/innen ein Fokus auf die Herstellung von Technik gelegt wird.

Wie wird also Technik produziert?

Es ist ein interessantes Phänomen, dass diverse Produktionsprozesse; ob in Firmen abgeschlossen von der Öffentlichkeit oder mit potentiellen Nutzer/innen, die miteinbezogen werden; von einem geradlinigen Ablauf weit entfernt sind. Stattdessen werden die Akteure/Akteurinnen in diesen Prozessen ständig mit Herausforderungen, langen

Aushandlungsdiskussionen sowie „Trial & Error“ konfrontiert und müssen dementsprechend viel Zeit und Geduld für ihre Arbeit aufbringen (Vinck 2003; Hyysalo, Jensen, Oudshoorn 2016). Ein wesentlicher Grund für diese oftmals langwierigen Hin-und-Her-Prozesse ist laut Vinck (2003) in vielen Fällen, dass an der Entwicklung eines technischen Produkts nicht nur die Techniker/innen arbeiten, sondern auch Personen aus anderen Abteilungen, wie zum Beispiel Sales, Projektmanagement oder Design, welche möglicherweise die Technik nicht verstehen, oder als nicht so wichtig ansehen. Und dann stellt sich die Frage, ob auch die zukünftigen Nutzer/innen involviert werden. Das Wissen der einzelnen Projektmitarbeiter/innen aus diversen Fachbereichen, das für die Entwicklung eines bestimmten Produkts aufeinandertrifft unterscheidet sich dementsprechend maßgeblich voneinander. Manche haben mehr Ahnung von der Technik, andere von den Anforderungen für die Nutzer/innen und wieder andere werden auf Einschränkungen hinsichtlich Zeit oder Kosten des Projekts hinweisen. Des Weiteren wird Jede/r versuchen, die eigenen Interessen im Entwicklungsprojekt möglichst gut durchzusetzen. Der Erfolg bei der Entwicklung eines technischen Produkts hängt also stark von der Bereitschaft und Fähigkeit zur Kooperation der involvierten Personen ab. In weiterer Folge gestaltet sich der Entwicklungsprozess als kontinuierlicher Aushandlungsprozess zwischen den involvierten Parteien (ebd.). Das wird auch später in den Ergebnissen noch sichtbar, dass die Entwicklungsprozesse eher als sich wiederholende Schleifen gesehen werden können.
Programmierer/innen als Berufsgruppe und Technik produzierender Teil der Gesellschaft kommen in der soziologischen Literatur aber noch relativ selten vor, weshalb sich dieser Abschnitt eher als Mosaik vorhandener Literatur gestalten wird, mit einem späteren Fokus auf Usability.

3.1 Soziologie des Computers

Gestartet wird nun aber mit dem Arbeitsmittel von Programmierer/innen, dem Computer in der Soziologie. Dem Computer wurde unter anderem von Norbert Wiener (1961) Beachtung in seiner Arbeit geschenkt, wo er sich damit auseinandersetzt, welche menschlichen Prozesse diese Maschine einmal übernehmen kann oder wird. Für ihn sind Computer mechano-elektrische Systeme mit Sensoren und Rechenleistung, die eine bestimmte Ausgangssituation mittels eines festgelegten Schemas verarbeiten und auch kommunizieren können. Nachrichten können nur Systeme verarbeiten, die auch in der Lage sind, durch einen Speicher Informationen aus der Vergangenheit so zu modifizieren und für die gegenwärtige Situation anzupassen, damit das System in der Gegenwart agieren kann. “Hier handelt es sich beispielsweise um das systeminterne Instruieren, eine konkrete

Situation als alltägliches Begrüßungsritual zu identifizieren, indem sie mit bereits erlebten Situationen dieses Typs in Bezug gebracht wird." (Wiener 1961 zit. nach Häußling 2019, S.129). Systeme sind bei Wiener außerdem Träger von entweder Materie, Energie oder Information und jene Systeme mit viel Information und wenig Energie oder Materie seien in der Lage die informations-armen Systeme mit viel Energie und Materie zu steuern. Dazu werden die internen Informationen zu *"messages"* gebündelt, mit denen anderen Systeme kontrolliert werden können. Regelung findet also durch Kommunikation statt und letztere sieht Wiener auch zwischen Menschen und Maschinen oder nur zwischen Maschinen möglich. Damit die Kommunikation funktioniert, wird zwischen den kommunizierenden Systemen ein "Soll-Ist-Abgleich" durchgeführt und über sogenannte "Melder" (Sinnesorgane bzw. Sensoren) werden Rückmeldungen ausgetauscht. Nicht nur sieht Wiener somit die Kommunikation zwischen Mensch und Computer als natürlich an, er ist auch damals schon der Meinung, "dass Maschinen lernen können und damit in eine menschliche Domäne vorstoßen, die vormals als nicht technisierbar galt: die Welt der Kognitionen. Diese Prognose Wieners – lange vor der Künstlichen Intelligenz-Debatte – hat sich in den letzten Jahrzehnten mit sogenannten selbstlernenden Systemen, deren Softwarearchitektur oder, mit Wiener gesprochen, deren Regelungsprozesse den Prinzipien neuronaler Netze folgen, bewahrheitet." (Wiener 1961 zit. nach Häußling 2019, S.131). Seine Ansprüche an lernfähige Systeme sehen so aus, dass sie "weitreichendste Anpassungsfähigkeit, beste Fehlertoleranz und höchste systemische Dynamik" durch "flexible, wechselseitige Verschaltungen" erreichen sollen. Außerdem sind Kommunikation und Kontrolle maßgeblich für das 20. Jahrhundert, da aufgrund von "Sensorik, informationellen Input- und Output-Ströme sowie operativen Interventionen ein Wechselbezug mit dem Umfeld" hergestellt wird (ebd.). Computer sind nach Wiener also dazu in der Lage, ihre gespeicherten Informationen so zu kombinieren, dass sie damit in einen wechselseitigen Austausch mit ihrem Umfeld treten und sinnvoll kommunizieren können, sowie in weiterer Folge auch lernen können.

Bei Herbert Simon (1994) entwickelt sich die Ansicht, dass im "Computerzeitalter" Komplexität rapide zunimmt, da das Ausmaß an Informationen explosionsartig ansteigt, während die verfügbare Aufmerksamkeit gleich limitiert bleibt, also die Lösungsmöglichkeiten der Komplexität weniger werden und eine "Armut an Aufmerksamkeit" entsteht, wie Simon es nennt. In diesem Fall sind Softwareprogramme dazu entwickelt worden, um als eine Art Sieb die essentiellen Informationen für eine bestimmte Aufgabe herauszulösen und vorzubereiten. Des Weiteren sieht er nun Informatiker/innen damit beauftragt "ein geeignetes Kommunikationsdesign zu finden, das

Nutzer in die Lage versetzt, sich trotz Unkenntnis souverän im Möglichkeitsraum der digitalen Welt zu bewegen.” (ebd. 1994, zit. nach Häußling 2019, S. 134). Winograd und Flores (1989) haben zu Simons Überlegungen kritisch angemerkt, dass er von der Existenz konkreter Probleme und der Bekanntheit der dabei enthaltenen Variablen ausgeht, für die auch die Lösung gleich zur Hand ist. Außerdem sehen sie, den Kontext der jeweiligen Situation und den Hintergrundbezug von Akteuren außer Acht gelassen (Winograd & Flores 1989, zit. nach Häußling 2019).

Sie sehen den Design- und Entwicklungsprozess von technologischen Systemen als “zyklisch”, da nach ersten Entwürfen und Entwicklungen und Prototypen usw. die gesammelten Erfahrungen immer wieder in weiteren Entwürfen eingearbeitet werden (ebd.). Das zyklische an Entwicklungsprozessen wird auch später in den Ergebnissen deutlich.

Ihre Sichtweise, dass die Anwendung eines technologischen Werkzeugs, die Gewohnheiten der Nutzer/innen verändert (ebd.), lässt sich auch auf mobile Apps anwenden. Je nachdem wie die App oder die Features in ihr gestaltet wurden, wird es die Nutzungsweise unterschiedlich beeinflussen.

> “Ein Programmierer erzeugt demzufolge einen Handlungsrahmen, innerhalb dessen der Benutzer agiert – so Winograd und Flores. Dabei bilde das eigentliche Nadelöhr der Systemkonstruktion, den ‘Verstehensbereich der Benutzer’ (ebd.: 270) zu berücksichtigen. Denn nur wenn die betreffende Software an die Bedürfnisse, Erfahrungen und Kenntnisse der Nutzer bzw. betroffenen Berufsgruppen ankoppelbar ist, könne sie auf Akzeptanz und damit auf Benutzung stoßen. In Anlehnung an Simon sprechen auch Winograd und Flores (1989: 269) von Schnittstellen. Dabei sei die Schnittstelle so zu gestalten, dass sie die Transparenz erhöht. [sic]” (Häußling 2019, S.137).

Das macht die Notwendigkeit von Usability-Tests mit potentiellen Nutzer/innenn verständlich, durch die ausgelotet wird, was der “Verstehensbereich” beinhaltet und wie der Handlungsrahmen aussieht, den ein Feature bereitstellt. Unternehmen würden sich selbst nichts Gutes tun, wenn sie auf das Feedback von Nutzern verzichteten.

In diesem Zusammenhang des Computers als Schnittstelle kann er zudem als "Hilfsmittel von Sprache" verstanden werden und Software als "Gestaltungsangebot", das dem Nutzer/innen ermöglicht, die einzelnen Features einer Software in verschiedenen Konstellationen zu kombinieren, um zu einer hilfreichen Lösung zu gelangen oder ein ansprechendes Design zu entwickeln (Winograd & Flores, 1989). Wenn auch Computer bei Winograd und Flores noch nicht sinnverstehend kommunizieren können, so befinden sie sich trotzdem auf der Ebene der Sprache, wenn wir mit ihnen interagieren. Das ist deshalb

möglich, weil Programmierer/innen zuvor bei der Entwicklung des Computers oder der Software einen Handlungsbereich definiert haben, in dem Nutzer/innen sich bewegen können, während der Kommunikation mit der Maschine. Da während des Entwicklungsprozesses und der Definition des möglichen Handlungsbereichs Sinn eingeflossen ist, der die Bedürfnisse und Verstehensbereiche der Nutzer/innen vorhersehen soll, muss dieser Prozess kritisch und reflektiv analysiert werden (ebd.). Zu der Zeit, als Winograd und Flores diese Überlegungen anstellten, gab es in der Entwicklung von Computertechnologie noch kaum Ambitionen, die Meinung von potentiellen Nutzer/innen einzuholen und die Gestaltung der Software an die Bedürfnisse und Alltagspraxen dieser anzupassen. Umso angebrachter war es, dass sie genau diese Herangehensweise empfahlen.

3.2 Design in der Computertechnologie

Nach diesem Blick in die Geschichte der Soziologie des Computers, beschäftigt sich der nächste Abschnitt mit einem aktuelleren Thema dieser Soziologie-Sparte: die Digitalisierung. Einen wichtigen Anteil der Digitalisierung übernimmt das Design, da für viele innovative, technische Lösungen ein ansprechendes *Interface* und einfache, verständliche Handhabung wesentlich zum Erfolg beitragen und Programmierer/innen meist eng mit der Designabteilung eines Unternehmens zusammenarbeiten oder das Design selbst übernehmen.

> “Nutzer sind also in der Regel nicht mit der Technik selbst, sondern mit einem auf individuelle Bedürfnisse angepassten Design konfrontiert. Die Technik verschwindet regelrecht hinter dem Design - das nennt man dann verheißungsvoll smart. Das Design wird auch immer bedeutsamer, je mehr digitale Technologie Einzug erhält. [...] Gerade auch an der Geschichte des Smartphones sieht man, dass der Designprozess unabschließbar geworden ist - und viel stärker die Interaktion mit den Nutzern einbezieht.[sic!]” (Häußling 2019, S.296).

Was Winograd und Flores vor 30 Jahren empfahlen, hat sich also tatsächlich durchgesetzt, was später auch in den Ergebnissen sichtbar wird.

Designer haben laut Schön (1983) die besondere Fähigkeit, während der Ausübung ihrer Designaufgaben darüber zu reflektieren, was er “*reflecting in action*” nennt. Um die diversen und komplexen Anforderungen (einfache Bedienung, optisch ansprechend, technische Funktionen, ökonomische Faktoren, etc.) des Designprozesses in den Griff zu bekommen, ist es notwendig, andauernd zwischen der gestaltenden Arbeit und der Bewertung dieser,

wechseln zu können, was längerfristig einem oder mehreren kleinen Experimenten gleich kommt.

Umgelegt auf den Programmierprozess, könnte das so verstanden werden, dass die Entwickler/innen als Experiment ihren Code schreiben und zur Reflexion und Bewertung der Ergebnisse daraus, die Nutzer/innen für Tests und Feedback herangezogen werden. Dadurch zeigt sich dann in weiterer Folge, wie zufrieden die Designer mit ihren Versuchen sein können.

Während dieses Prozesses kommt es immer wieder zu ungewollten Änderungen, auf die die Designer/innen reagieren müssen, das nennt Schön "*the situation talks back*" (Schön 1983, zit. nach Häußling 2019).

Ganz ähnlich ist das auch beim Programmierprozess der Fall, wenn zum Beispiel durch eine Korrektur oder eine Änderung des Codes an der einen Stelle, sich gänzlich unabsichtlich an einer völlig anderen Stelle etwas ändert. Das könnte als "*the code talks back*" bezeichnet werden.

3.3 Digitalisierung und Soziologie

Vielerorts hört man immer wieder, dass wir uns im Zeitalter der Digitalisierung befinden, doch wie genau sind wir dort angelangt? Die Entwicklungsgeschichte der Computernutzung kann laut Häußling (2019) in fünf Phasen eingeteilt werden: Computerisierung, PC-isierung, Internetisierung, Mobilmachung und Ubiquisierung von Mikroprozessoren, Digitalisierung. In der ersten Phase wurden vermehrt Großrechner in Wissenschaft, Militär und Wirtschaft eingesetzt. In der zweiten Phase wurden mit der Individualisierung PCs (*Private Computer*) populär, die sich immer mehr Privathaushalte leisten konnten, dabei sind die Handlungsmöglichkeiten noch sehr auf das eigene Gerät beschränkt. Bei der Internetisierung als der dritten Phase, war wieder eine Entwicklung entgegen der Individualisierung der vorherigen Phase bemerkbar. Durch die Vernetzung dank des Internets entwickelte sich ein reger kommunikativer Austausch und die sogenannten Internet-*Communities* entstanden, in denen sich einzelne Personen zu Gruppen (*Communities*) mit diversen Zwecken zusammenfinden und gemeinsam interagieren. In der vierten Phase konnten mit Mikroprozessoren die herkömmlichen technischen Geräte um ein Vielfaches verkleinert werden, bei ähnlicher Leistungsstärke, was zum Erfolg der *smart devices*[5] und in weiterer Folge zum Internet der Dinge[6] führte. In der aktuellen Phase, der

[5] Elektronische Geräte, die mit dem Internet verbunden sind und dadurch über das Smartphone oder Tablet gesteuert werden können, wie die Kaffeemaschine, der Fernseher.

[6] Internet, das mit oben genannten elektronischen Geräten verbunden ist, welche darüber von der Ferne gesteuert werden können.

Digitalisierung, “werden die Potentiale einer Vernetzung unterschiedlichster Daten ausgelotet. Nicht nur werden „*in situ*“- und „*in silicio*“-Daten miteinander kombiniert, sondern das „Internet der Dinge“ greift in unsere Handlungsvollzüge ein, und die dort anzutreffenden Daten vernetzen sich mit der Offline-Wirklichkeit.” (Häußling 2019, S. 327). Daten aus der Wirklichkeit, sowie aus Simulationen werden also nun automatisch gesammelt, ausgewertet und für die weitere Vorgehensweise, z.B. weitere Berechnungen durch Simulationen etc., herangezogen. So können Handlungs-, Produktions- oder Entwicklungsschritte fast ohne menschliche Akteure angestoßen werden.

3.4 Daten, Algorithmen, Protokolle

Hinter diesen verschiedenen Prozessen mit Daten stehen Codes, Algorithmen und Protokolle, die unter anderem festlegen, was mit diesen Daten passiert. In den *Software Studies, Critical Code Studies* und *Critical Algorithm Studies* werden diese Strukturen näher untersucht, häufig in Kombination mit Themen wie *Nudging*[7], Macht, Performanz, Agency, etc. Diese genannten Ansätze sehen die Entstehung von Protokollen, Codes und Algorithmen nicht von der Gesellschaft getrennt, sondern als Ergebnis sozialer Kräfte, da Ideen, Werte, Visionen, Bedeutungen, Normen etc. in sie eingeschrieben sind (auch im wörtlichen Sinne) (ebd.).

Diese Ansicht teilt auch Mackenzie (2006), wenn er davon spricht, sich in den Computer hineinzuversetzen und dessen Inneres, also den Code verständlich zu machen, welcher sich in unterschiedliche Abstraktionsgrade aufteilt. Zum Beispiel ist die menschliche Sprache noch in den Programmiersprachen erkennbar, während Quellcode oder Objektcode schon abstrakter sind. Hinter den abstrakten Codes, egal ob Quellcode oder Objektcode, steckt aber immer eine konkrete Idee und ein konkretes Ziel. Der Code kann also als inhaltlicher Text aber gleichzeitig als direkt ausführbarer Befehl verstanden werden. Für Mackenzie und Vurdubakis sind Codes eine Art Hilfsmittel mit den diversen Arten von Krisen, die die Gesellschaft heutzutage plagen (Klimakrise, finanzielle, politische, kulturelle Krisen etc.), fertig zu werden, indem zum Beispiel durch Simulationen verschiedene Lösungswege aufgezeigt werden. Jedoch sollte man sich gleichzeitig bewusst sein, dass ebendiese Codes zur Finanzkriese beigetragen haben, da sie im Bereich des *high frequency tradings* zum Einsatz kamen (Mackenzie & Vurdubakis 2011). Da sich mit dem Einsatz von Codes eine enorme Vielfalt an Möglichkeiten auftut, die ab einem bestimmten Zeitpunkt

[7] Nudging bedeutet übersetzt so viel wie lenken, anregen oder in eine Richtung steuern. Es wird häufig als neuere Marketingstrategie für Produkte oder Ansichten eingesetzt, dabei werden den jeweiligen Personen Informationen zu ihren Auswahl- oder Handlungsmöglichkeiten dargelegt, welche sie unterbewusst zu einer bestimmten Entscheidung führen.

kaum oder nicht mehr überschaubar ist und die Welt mit Ungewissheit instabiler macht; also Problemlösungsstrategien, die sich auf Codes verlassen, immer mit einem gewissen Risiko verbunden sind, braucht es einen kritischen, reflektierten Umgang damit, bei dem immer auch die alternativen Codes als Lösungsmöglichkeiten transparent dargestellt werden müssen (ebd.).

Auch Algorithmen müssen laut Beer (2017) ähnlich kritisch betrachtet werden, weil sie einen starken, gestalterischen aber auch limitierenden Einfluss auf soziale und gesellschaftliche Geschehnisse haben. Die besten Beispiele sind die Ampelschaltung, *Filter Bubbles* oder sogenannte Echokammern in sozialen Medien, aber auch beim *Online-Dating*, in der Arbeitsvermittlung (AMS mit zwei Pilotprojekten), personalisierte Werbung, zur Bestimmung der Kreditwürdigkeit, zur Entscheidungsfindung ob jemand Sozialhilfe bekommen soll uvm., um nur ein paar zu nennen.

Die Funktionsweise von Algorithmen und wie sie lernen, kann ganz schnell problematisch und diskriminierend werden. Wenn zum Beispiel bei der Bewertung von Arbeitsmarktchancen oder von Bonität einer Person, die Daten der Herkunft, des Geschlechts, des Bildungsgrades, Daten vergangenen Verhaltens etc. herangezogen werden und schlechtere Chancen am Arbeitsmarkt oder auf einen Kredit als Ergebnis in die Bewertung einfließen. Das kann breiter gesehen dazu führen, dass die Ungleichheitsverhältnisse in Gesellschaften stagnieren oder noch ungleicher werden (Berger & Schöggl 2019).

Dieses paternalistische Eingreifen und Vorwegnehmen von Entscheidungen, das auch keine menschlichen Interventionen mehr zulässt und wie eben erwähnt auch zu Diskriminierungen führen kann, nennt Beer "*Hyper-Nudging*[8]" (ebd. 2017, S.5). Trotzdem sieht er Algorithmen als soziale Prozesse, die in Form von Codes zum Ausdruck kommen, da sich die Werte der Personen die den Code und die Algorithmen geschaffen oder in Auftrag gegeben haben, im Code und der daraus resultierenden Prozesse widerspiegeln.

3.5 Schnittstellen

Das führt wieder zurück zu Häußlings Sichtweise, dass Codes oder Daten als Schnittstellen zwischen algorithmischen und sozialen Prozessen fungieren.

> "Als Schnittstelle schieben sich Daten zwischen soziale und technische Prozesse

[8] Nudging bedeutet übersetzt so viel wie lenken, anregen oder in eine Richtung steuern. Es wird häufig als neuere Marketingstrategie für Produkte oder Ansichten eingesetzt, dabei werden den jeweiligen Personen Informationen zu ihren Auswahl- oder Handlungsmöglichkeiten dargelegt, welche sie unterbewusst zu einer bestimmten Entscheidung führen. Bei Hyper-Nudging wird den Personen eigentlich keine andere Wahl mehr gelassen, als sich für die durch Nudging „empfohlene" Möglichkeit zu entscheiden. Entscheidungen werden also vorweggenommen und wird nur ein Anschein von Entscheidungsmöglichkeiten erhalten.

> und nehmen dort eine mittlere – auch im Sinne von transferierende – Stellung ein. Einerseits werden Phänomene der Offline-Wirklichkeit in binäre Daten transferiert, fast alle Lebensbereiche erfahren auf diese Weise eine Datafizierung. Andererseits wird mit Daten binär gerechnet bzw. gearbeitet, um Ergebnisse zu produzieren, die wiederum an die sozialen Prozesse angekoppelt werden" (ebd. 2019, S. 341).

Die Verkoppelungsprozesse von Daten mit sozialen Tatsachen teilt er in fünf nicht chronologische Formen: "Die (1) Produktion von Daten, ihre (2) Strukturierung, die (3) Distribution von Daten, ihre (4) Visualisierung sowie die (5) datenbasierte oder sogar – induzierte Steuerung sozialer Prozesse." (ebd. S. 342). Mit diesem Verkopplungsmodell sieht Häußling die Möglichkeit, eine angemessene techniksoziologische Analyse der vielfältigen, digitalen Welt durchzuführen.

3.6 Technologisierung der sozialwissenschaftlichen Methoden

Die Soziologie ist aber nicht nur gefragt, wenn es um die Erforschung der Zusammenhänge und wechselseitigen Einflüsse zwischen Gesellschaft und Technologie geht. Es wäre auch an der Zeit, dass sich die Soziologie die Technik zu Nutze macht, um den neuen Herausforderungen, die die Digitalisierung mit sich bringt auch auf analytischer Ebene gewachsen zu sein. In der *Digital Sociology* und der *Computational Social Science (CSS)* wird debattiert, welche Methoden für welche Problem- und Fragestellungen als nützliches Werkzeug dienen. Um die Datenflut der heutigen Zeit mit Big-Data-Analysen sinnvoll zu durchforsten und relevante aber vor allem aussagekräftige Ergebnisse daraus zu generieren braucht es Kompetenzen in der Programmierung, im Datenbankmanagement und komplexitätstheoretische Auswertungsverfahren, sowie Kenntnisse in *machine learning, text mining* und formale Netzwerkanalyse. Damit näherte man sich einer Digital Sociology an, wobei aber noch nicht definiert ist, was Digital Sociology genau bedeuten soll (Häußling 2019). Daniel McFarland, Kevin Lewis und Amir Goldberg (2016) haben zunächst drei verschiedene Möglichkeiten entwickelt, wie *Digital Sociology* aussehen könnte. Der erste Weg, der eingeschlagen werden kann ist "*Interdisciplinary Joining*", bei dem die Aufgaben zwischen Informatiker/innen und Soziolog/innen deutlich getrennt bleiben. Informatiker/innen sind für die Analyse der Daten zuständig und Soziolog/innen sollen die Theorien daraus entwickeln. Im zweiten Weg "*Interdisciplinary Embedding*" nutzen Informatiker/innen soziologische Theorien um damit die Ereignisse aus ihren Analysen zu erklären, während Soziolog/innen zum Testen ihrer Theorien Big-Data-Analysen heranziehen. Beim "*Full mixing*" des dritten Weges stellen sich die Autoren eine enge

Kooperation der beiden Disziplinen vor, was auf lange Sicht so weit führen wird, dass sich ohnehin eine komplett neue Disziplin entwickeln würde, welche sie “*forensic social science*” nennen (ebd.).

Nach diesen allgemeinen Ausführungen über Techniksoziologie und die Soziologie des Computers bis hin zur *Digital Sociology*, sollen nun spezifischere Untersuchungen vorgestellt werden, die zeigen inwiefern Programmierer/innen in der soziologischen Forschungswelt vorkommen.

3.7 Mensch-Maschine-Vermittlung durch Programmierer/innen

Ein größerer Themenpunkt bei dem Programmierer/innen in der sozialwissenschaftlichen Forschungswelt aufscheinen, ist die Vermittlung oder Kommunikation zwischen Mensch und Maschine. Ein Beispiel, dass Wissenschaftler/innen vor 20 bis 30 Jahren schon ähnlich dachten, ist Heim (1974), der einen Austausch zwischen automatisierten Datenverarbeitungssystemen (ADV-Systemen) und den Programmierer/innen bzw. Systemanalytiker/innen erkennt. In seiner Inauguraldissertation sieht er sich aus einer systemtheoretischen Perspektive den Beruf der Programmierer/innen und Systemanalytiker/innen an und sieht dabei diesen Beruf als "Filter" zwischen den ADV-Systemen und seinen Umsystemen. Anders ausgedrückt, diese Berufsgruppe vermittelt zwischen Gesellschaft und Technik, da sie sowohl mit der Technik der ADV-Systeme, als auch mit ihrer sozialen Umgebung in der Gesellschaft in einer wechselseitig abhängigen Beziehung stehen laut Heim (1974).

Nicht nur als Filter, sondern als direkt mit dem Computer kommunizierendes Individuum bezeichnet Kupka (1983) die Programmiererin/den Programmierer:

> "Der Programmierer übergibt nicht nur Programme und übernimmt Ergebnisse, sondern er verständigt sich mit dem Computer über die Arbeit und via Computer auch mit anderen. Mit Recht können wir daher sagen, daß der Umgang des Menschen mit dem Computer ein *Kommunikationsphänomen* ist. Die Entwicklung komfortabler Dialogsysteme und dem Menschen stark angepaßter Verständigungsmechanismen wie quasi-natürlichsprachliche Dialoge oder audiovisuelle Ein-/Ausgabetechnik stützen diese Auffassung. [sic]" (ebd. 1983, S.12f).

Auch Nievergelt und Ventura (1983) sehen das ganz ähnlich, wenn sie vom Computer als “Zweiwegkommunikationsmittel” sprechen. Hier steht zwar eher die Maschine im Mittelpunkt, die als Dialogpartner der Programmiererin/des Programmierers fungiert, dennoch sind es letztere, die mittels Computer als Technologie Problemlösungen für die

Gesellschaft erzeugen und somit mit beiden Seiten kommunizieren müssen. Des Weiteren müssen Programmierer/innen dem Computer "beibringen" auch mit der Benutzerin/dem Benutzer verständlich zu kommunizieren (z.B. Fehlermeldungen mit Hinweisen darauf, warum ein Befehl nicht ausgeführt werden kann).

3.8 Arbeitsstile prägen den Habitus und die Programmierer/innen-Identität

Ein Aspekt, der auch ganz interessant ist und der mit der Identität der Programmiererin/des Programmierers in Verbindung steht, ist der Arbeitsstil. Dzida, Spittel und Sylla (1983) unterscheiden grob zwischen zwei gegensätzlichen Typen: zum einen "der Hacker [sic]" und zum anderen "der kooperative Programmierer [sic]" (S.141).

Hacker/innen werden demnach folgendermaßen charakterisiert: "er spezifiziert nur ungenau seine Arbeits-Ziele und arbeitet gern 'drauf los'; während der Programmerstellung produziert er kaum Dokumente, die über das Programm und dessen Entwicklungsgeschichte Auskunft geben; er ist Einzelkämpfer; er hat grenzenloses Vertrauen in die eigene 'Cleverness' [sic]" (Dzida, Spittel, Sylla 1983; S.141).

Kooperative Programmierer/innen hingegen weisen folgende Eigenschaften auf: "er diskutiert mit anderen über seine Arbeit; er macht ausgedehnte Vorarbeiten, bevor er mit dem Programmieren im engeren Sinn anfängt; er antizipiert Folgearbeiten; er arbeitet umsichtig, d.h. mit Rücksicht auf die Produkte seiner Kollegen; er erledigt nebenher administrative Arbeiten [sic]" (ebd.; S.142). Der Begriff *Hacker* wird heute mittlerweile auch für Personen verwendet, die es schaffen, Sicherheitseinstellungen auf verschiedenen Ebenen zu umgehen und mit den Daten oder Informationen, die sie dahinter finden, zum Beispiel deren Besitzer erpressen, was als „*Black Hat Hacker*“ bezeichnet wird. Andererseits gibt es auch „gut gesinnte“ *Hacker*, die ihre Fähigkeiten dazu nutzen, Firmen auf ihre Sicherheitslücken aufmerksam zu machen und somit Sicherheitstechnologie verbessern, sogenannte „*White Hat Hacker*“. Diese Arbeit, wird immer mehr zu einem Beruf, bei dem die *White Hat Hacker* für eben diesen Zweck angestellt werden und aufgrund der hohen Nachfrage ein sehr gutes Gehalt verdienen (norton.com; karriere.de)[9].

Auf ähnliche Art und Weise wie Dzida, Spittel, Sylla hat Strübing (1992) die Berufsgruppe der Programmierer/innen untersucht und sich dabei im speziellen eben den Arbeitsstil bzw. die Arbeitskultur und den Habitus[10] der Programmierarbeit angesehen. Wie manch andere

[9] Wenn in der vorliegenden Arbeit der Begriff Hacker vorkommt, so ist der Black Hat Hacker gemeint.

[10] Der Habitus nach Bourdieu ist grob erklärt die Gesamtheit aus Erscheinung und Auftreten einer Person, dazu können unter anderem Lebensstil, Kleidung, Sprache und Geschmack gezählt werden. So kann, laut Bourdieu, der Habitus die Zugehörigkeit zu einer gesellschaftlichen Gruppe oder Schicht wiederspiegeln, in diesem Fall zur Gruppe der Programmierer/innen (Rehbein 2006).

Berufsgruppen, so wird auch den Programmierer/innen eine gewisse Sonderbarkeit zugesprochen mit einer Reihe spezifischer Stereotypisierungen. Strübing (ebd.) geht in seinem Werkstattbericht der Frage nach, ob an diesen Stereotypisierungen etwas dran ist und welches Selbstbild bezogen auf die Arbeitskultur die Mitglieder der Softwarebranche entwickeln.

Es wurde in dieser Untersuchung angenommen, dass Programmierer/innen eine ganz eigene Arbeitsweise aufweisen und sich von anderen Abteilungen der Firma, in der sie arbeiten, abgrenzen, also einen eigenen Habitus entwickeln. Die Ergebnisse zeigen dann auch, dass Programmierer/innen tatsächlich unterschiedliche Arbeitsstile entwickeln, abhängig von ihrer eigenen Persönlichkeit und von der auszuführenden Tätigkeit. Manche arbeiten sehr klar strukturiert, einen Schritt nach dem anderen, sie erstellen genaue Pläne des Projektvorhabens und schreiben während der Arbeit und am Ende des Projekts detaillierte Berichte. Auch das Testen der Software und das Finden der Fehler im Code, wenn etwas nicht so läuft wie es soll, wird in klar strukturierten Abläufen durchgeführt. Andere arbeiten eher drauf los und sind dabei etwas unordentlich, chaotisch und springen von einem Teil des Codes zum anderen. Sie schreiben nur ungern genaue Pläne und Berichte und verzetteln sich gerne in bestimmten Funktionen der Software (Strübing 1992). Erstere können mit dem oben erwähnten kooperativen Typ in Verbindungen gebracht werden und Letztere wiederum mit dem Hacker-Typ. In seiner Untersuchung bezeichnen sich die Befragten sogar selbst teilweise als “‘Hacker bzw. als Tüftler, die gerne’ durch ‘rumprobieren’ zu einer Lösung gelangen." (Strübing 1992, S.135).

Dabei entwickelt sich auch ein eigener Habitus im Vergleich mit anderen Berufsgruppen. Es findet eine Schließung nach außen statt, indem sie sich von anderen Berufen abgrenzen. Gleichzeitig gibt es eine Differenzierung nach innen, da sie sich nach Spezialisierung, auch (nach Geschlecht), nach Arbeitsstil etc. unterscheiden. Zum Beispiel wird gegenüber Vertriebs- und Marketingmitarbeiterinnen/Marketingmitarbeitern desselben Betriebs eine sehr starke Abgrenzung vorgenommen, die als "'zu glatt', zu unehrlich, zu förmlich, zu sehr auf Äußeres orientiert und technisch zu wenig fachkompetent." (Strübing 1992, S.105) gelten.

Ein weiterer wichtiger Aspekt bezüglich der Abgrenzung zu Anderen aber auch bezüglich der Identität als Programmierer/in, ist das äußerliche Auftreten. Es wird erwähnt, dass sie im Gegensatz zu Mitarbeiter/innen aus anderen Firmenabteilungen (Verkauf/Vertrieb) eher locker-lässig und gemütlich gekleidet sind, was auch mit dem "Verhaltensstil korrespondiert"

(Strübing 1992, S.119).

3.9 Verschmelzung von Programmierer/in und Computer

Eine andere Studie, die spannende Erkenntnisse liefert ist "Geistmaschine. Faszination und Provokation am Computer" von Schachtner (1993). Für diese Untersuchung wurden leitfadengestützte Interviews durchgeführt und zusätzlich dazu hat Schachtner die Kandidat/innen gebeten ein "Körperbild" von sich selbst bei der Arbeit zu zeichnen.

Die Fragestellung ist recht allgemein gehalten zur Arbeits- und Erlebnissituation von Programmierer/innen und der Beziehung zwischen ihnen und der Technologie (Computer + Systeme) mit der sie arbeiten. Es werden Themen angesprochen, wie "Vermenschlichung" des Computers indem mit ihm kommuniziert wird, Identität und Persönlichkeit der Programmierer/innen, (nicht vorhandene) Körperlichkeit der Programmierarbeit (aus den Körperbildern), Denk- und Wahrnehmungsweisen der Softwareentwicklerinnen und -entwickler, ihre Tätigkeitsmotive, ihre Selbstverwirklichungswünsche, ihre Lebensgeschichten aber auch die Ansprüche z.B. was die Technologie können muss, wie programmierte Ergebnisse auszusehen haben, die Beziehung und Kommunikation mit den Kund/innen. (Schachtner 1993).

Auch hier sind die Kommunikation mit dem Computer bzw. durch den Computer, die Arbeitsweise und die Identität (als Architekt/innen einer schönen Programmstruktur zum Beispiel, oder als Kunstschaffende eines ästhetischen Programmdesigns), ein wesentlicher Bestandteil der durchgeführten Interviews (ebd.).

Eine Studie, die sich ebenso mit der Verschmelzung oder wechselseitiger Verinnerlichung beschäftigt, ist anthropologischer Herkunft. Bei dieser Studie hat Downey (1998) unter anderem mit einer Reihe von Student/innen aus der technischen Informatik (*computer engineering*) Interviews geführt und mit ihnen über das Zeichenprogramm CAD/CAM (*computer-aided-design, computer-aided-modeling*) gesprochen, welches in einem Kurs unterrichtet wurde an dem er selbst teilnahm für seine Beobachtungen. Aus den Interviews geht im Grunde hervor, dass der Computer alleine nicht alles machen kann, sondern dass es immer noch den Menschen braucht um dem Computer oder dem Programm die richtigen Anweisungen zu geben. Gleichzeitig muss sich der Mensch an den Computer anpassen und mit dessen "Eigenheiten" fertig werden oder Geduld haben. So sagt eine seiner interviewten Personen:

"You are also a slave to the computer. You can only work as fast as the computer can work. You have to tell the computer exactly what you want done. If you have a conceptual outline of what you want done, you have to translate that into the format of the commands that the

CAD/CAM system recognizes. If the system is not user-friendly, then you can run into difficulties." (Downey 1998, S.137). Man ist also einerseits in den Funktionsmechanismen des Programms "gefangen" und andererseits muss man ihm zeigen "wo es lang geht".
"Rather the humans had to give themselves over to the machine and accept the task of becoming part of a technological system." (ebd. 145). Es gab in den Interviews immer wieder Hinweise auf intime Verbindungen zum Computer und einer seiner Interviewkandidaten brachte ein schönes bildliches Beispiel, er stellte sich oft vor, er sei ein Orchesterdirigent im inneren des Computers, der die einzelnen Prozesse wie verschiedene Instrumentengruppen dirigierte. Andererseits wird sehr häufig erwähnt, dass das Programm oder der Computer nicht das macht, was man will. Stattdessen macht er nichts oder etwas anderes. Manchmal gibt es Voreinstellungen im Computer, so wie er programmiert wurde, die an manchen Stellen auftauchen und dazu führen, dass etwas anderes passiert als man sich vorstellt. In diesen Fällen muss man lernen, damit umzugehen. Einer der Studenten hat dies geschafft und er bezeichnet sich selbst als "Computer-Person", er ist eins mit dem Computer und er würde es nicht schaffen, ohne ihn zu leben (ebd.).
Durch das Einloggen in das Programm, so der Autor, wurde ein ganz besondere Verbindung zwischen Mensch und Maschine hergestellt aber mehr als das, wurden dadurch die Handlungsmächte (*agency*) beider verflochten, indem die Handlungsmacht des Users in die Handlungsmacht der Software übertragen wurde. Durch die Macht und Kontrolle, die man dadurch erlangte, entwickelte sich aber auch eine Art Verwundbarkeit. Dadurch man sich an die Handlungsmacht der Software gewöhnt hat und gelernt hat mit den Wirkungsweisen, des Computers umzugehen, fühlte man laut Downey das Eindringen der Maschine in eine/n selbst (ebd.). Ein Beispiel dafür, wie ein ehemaliger Student in einem Programm weiterhin in der Firma/an diesem Institut anwesend blieb, obwohl er schon lange einen anderen Job hatte, ist dass er den Code für die Software geschrieben hat, dieser Code so einen wesentlichen Anteil am Funktionieren der Software hatte und er damit so "verschmolzen" ist, dass die Leute nach wie vor mit seinem "Geist" zu tun haben. "Expertise [wie im Fall dieses ehemaligen Studenten] was only one dimension of selfhood alongside others. In becoming CAD persons, students and faculty were building the agencies of CAD/CAM technologies into their bodies, justifying the sense that they were becoming experts." (ebd., S. 232). In mancher Hinsicht können Expert/innen in dem Gebiet auch als Handwerksmeister gesehen werden, wie es einer von Downeys Interviewpartner sah: "There's no reason why the designer can't take the role that the craftsman had at one time. I want to integrate the concept of doing good design together with having the best tools available to do it and make them learn the process rather than learn about the tools." (ebd.,

S.236). Beim Handwerk wie in der Software Entwicklung werden gewissen Tätigkeiten so internalisiert und in Körper und Geist eingewachsen, sodass diese Tätigkeiten ganz natürlich und automatisch ausgeführt werden können, als hätte man das schon immer so gemacht. Man hat also zu diesem Zeitpunkt die Technologie schon so verinnerlicht, dass sie Teil des Selbst geworden ist.

So wie in der Studie über die CAD/CAM Software das Programm immer wieder einmal etwas Unerwartetes macht, mit dem sich Entwickler/innen dann abkämpfen müssen, um es zu korrigieren; so stolpert auch Whitson (2018) in ihrer Studie zu Kooperation und Konflikt in der Spiele Entwicklung, immer wieder über den Ausdruck, dass die Software magisch sei. Das ist dann der Fall, wenn etwas nicht funktioniert oder das Programm ein anderes Ergebnis produziert, als man geplant hat. Das Ziel der Studie war, zu untersuchen wie die "*game engine*" Unity 3D und die Designsoftware 3D studio MAX als "*boundary-object*" eine Gruppe junger Menschen aus unterschiedlichen Disziplinen (Kunst, Design, Programmierung und Projektmanagement) zusammenhielt und sie Konflikte überwinden half in ihrer Arbeit an einem vorgegebenen Spiel, das sie entwickeln sollten. Im Rahmen dieses Versuchs kam es auch immer wieder zu Problemen mit der Software, bei denen niemand recht wusste oder herausfinden konnte, woran das Problem lag, dass also wie erwähnt, die Software etwas Unintendiertes macht.

> "On site, again and again, Unity 3D, 3D Studio Max and other tools such as Photoshop are anthropomorphized by the team, especially when things were not going according to the user's plans. For example, Jessie speaks quite vehemently about Photoshop and Unity both being 'retarded' when they are forced to work together. Every day, software is depicted as recalcitrant, wilful and needlessly abstruse. However, these negative characteristics are balanced by framing the software as something magical in the sense that it is so complex as to be fully unknowable, definitely not something simply taught or learned quickly, but something arcane that one must tread lightly around." (Whitson 2018, S.2322).

In den Fällen, in denen man sich die Probleme nicht erklären konnte und offensichtlich niemand aus dem Team Schuld daran hatte, wurde der Software Handlungsmacht übertragen und ihr die Schuld zugewiesen. Die Software sei widerspenstig und magisch, weil etwas Unerklärliches, fast wie in Zeiten des Mittelalters, als etwas Übernatürliches gesehen wurde. "In this case, Alex initially assumed that he made a mistake or that the software had a bug. But as more experts, both programmers and artists, are enrolled to unsuccessfully discover the problem, the interaction becomes imbued with 'magic'." (ebd.,

S.2323). In anderen Studien, wie diese von Schachtner weiter oben (oder Fogg 2003; Turkle 2005) über die Mensch-Maschine-Beziehung wurde bereits gezeigt, dass Nutzer/innen von Technologie, diese während ihres Gebrauchs häufig personifizieren, wie es auch in Whitsons Untersuchung der Fall war. So war es den untersuchten Personen möglich, für ein unerklärliches Problem einen Sündenbock zu kreieren und diesen mit übernatürlichen Kräften auszustatten.

"And rather than operating according to rational dictates, when ‘misbehaving’ it was seen as acting in ways that were ‘magical’, mysterious and indecipherable – a ghost in the machine. This was the role of ‘voodoo software’, a term arising from this fieldwork which resurfaced again and again when developer’s input into the software resulted in inexplicable forms of output." (Whitson 2018, S.2324). Das Erzeugen einer Person in der Technologie, also die Kreation des Sündenbocks half jedoch dem Team auch dabei, das Problem als solches anzunehmen und einen alternativen Weg, um das Problem herum, zu finden. Es war also hilfreich, indem sie sich damit abfinden konnten, das Problem sowieso nicht lösen zu können und da niemand aus dem Team Schuld daran hatte, konnte man gemeinsam daran arbeiten auf andere Art und Weise voran zu kommen. Das Problem von Voodoo-Software, die sich verselbstständigt, tritt außerdem nicht nur bei jungen Techniker/innen und Designer/innen auf, auch erfahrene Expert/innen kämpfen immer wieder mit "Software-Eskapaden", die sie sich nicht erklären können (Whitson 2018). Auch in den Ergebnissen wird das Adjektiv "magisch" verwendet, wenn auch häufiger aus der Sicht von anderen Menschen aus der Gesellschaft, also Nicht-Entwicklern.

3.10 (soziale ?) Programmierer/innen bei der Arbeit

Wie in dem Whitsons Artikel über Voodoo-Software schon erwähnt wurde, müssen für ein Entwicklungsprojekt Personen aus diversen Fachbereichen zusammenarbeiten. Das ist auch bei der Agile-Methode der Fall, die in diesem Absatz beschrieben wird und eine gängige Arbeitsweise in der Software Entwicklung ist. Bjorn, Soderberg und Krishna (2019) schreiben in ihrem Artikel, wie Agile eigentlich von allen Seiten als praktisches Projektmanagementkonzept gepriesen wird und sie überrascht waren, dass die Entwickler/innen in den untersuchten Softwarefirmen die sogenannte Wasserfall-Methode bevorzugen. Bei der Agile-Methode werden Aufgaben in kleinere Aufgaben, sogenannte “*Tickets*”, zerlegt, die in einem bestimmten Zeitraum erledigt werden müssen und am Ende jeden Tages, werden Status-updates über erreichte Ziele eingefordert. Sind Ziele nicht erreicht worden, gibt es natürlich Verzögerungen, was am Ende eines „*Sprints“* (eine kurze ca. zwei wöchige Entwicklungsphase, in der ein Teilziel eines Projektes erreicht werden soll)

wahrscheinlich zu Überstunden führen wird. Innerhalb eines Sprints müssen alle Aufgaben erledigt werden. Wenn, wie in den untersuchten Fällen, die Firmen unter Konkurrenzdruck stehen, müssen die Deadlines der Sprints unbedingt eingehalten werden, da ansonsten eine konkurrierende Firma den nächsten Auftrag bekommen könnte. Die Software Entwickler/innen in diesen Firmen stehen also unter konstantem Druck und Stress, jeden Tag ihre Aufgaben zu erfüllen, die keinen Raum lassen für private Notfälle oder dergleichen. Ursprünglich sollte diese Methode den Programmierer/innen Freiraum und Ermächtigung bieten, ihre Aufgaben für jeden Tag bzw. für die Sprint-Wochen selbst zu planen und somit die Arbeitsmotivation zu steigern. Das Problem bei den untersuchten Software Firmen ist allerdings, dass der Wunsch nach der Agile-Methode nicht von den Firmen selbst kommt, sondern von ihrem Auftraggeber/innen, ein großer, internationaler Konzern mit Sitzen in den USA und Europa. Diese Tatsache und "the global agile implementation in combination with the multiple vendor strategy created a most stressful environment for the developers and testers located "offshore," due to short intervals, tough deadlines and lack of flexibility in planning the work." (Bjorn et al. 2019, S. 186). "*Multiple Vendor Strategy*" meint in diesem Fall, mehrere Software Firmen, die um die Aufträge des Konzerns in Konkurrenz stehen und unter denen sich der Konzern das "passende" Angebot aussucht. Ein weiterer Stressfaktor für die Programmierer/innen in Pakistan und Indien, ist die Zeitverschiebung an die sie sich anzupassen haben. Da der Konzern das *daily stand-up meeting* am Beginn der Arbeitszeit in den USA durchführen möchte, sind die Entwickler/innen in Asien dazu gezwungen, ihren Arbeitstag um diese Meetings herum zu gestalten, das für sie spät am Abend um 22:00 oder 23:00 Uhr stattfindet, also am Ende des Arbeitstages. Das macht es praktisch unmöglich, sinnvoll Zeit mit der Familie zu verbringen oder sich um Beziehungen zu kümmern. Der Konzern fordert zudem, dass die gestellten Aufgaben in immer kürzeren Sprints durchgeführt werden.

"Project periods with the US Bank are extended from 4–5 months to 1 year or maybe even into projects of a duration of several years. With the highly structured day, 8 hours a day including daily stand-up meetings every evening at 10:30 pm., the global agile methodology tips the work/life balance toward still more work—especially since taking vacation is no longer an option." (ebd., S.187). Dementsprechend hatten die Angestellten in vielen Jahren keinen längeren durchgehenden Urlaub mehr. Ein Beispiel ist ein *Head of Operations*, der in neun Jahren nur drei Tage hintereinander als "längeren" Urlaub nehmen konnte. Anstatt einen Freiraum für Selbstorganisation zu schaffen, nutzt der Konzern die Agile-Methode um die Software Firmen zu kontrollieren und gegeneinander auszuspielen "we see that agile mechanisms can also be used by US-based clients to monitor and micromanage the work

conditions of IT developers working out of India." (ebd., S. 189). Es hat sich laut den Autoren also ein unausgeglichenes Machtverhältnis entwickelt, das an Post-koloniale Strategien erinnert. Bjorn, Soderberg und Krishna stellen noch eine Reihe weiterer interessanter Überlegungen an, die hier leider nicht mehr angeführt werden können. Es empfiehlt sich aber, diese erkenntnisreiche Studie genauer zu lesen.

Für die Zusammenarbeit an Software Projekten braucht es auch „*Soft-Skills*", welche, wie manche Menschen behaupten würden, den Programmierer/innen fehlen, weil sie Programmierer/innen mit den Begriffen "*Nerd*" oder "*Geek*" verbinden. Der folgende Artikel von Zoric & Stojanov (2018) hat ein zum Teil zu tun, dass Entwickler/innen häufig als weniger sozial kompetent gesehen werden. Zoric und Stojanov gingen der Frage nach, wie Programmierer/innen "*Soft-Skills*" sehen, dabei haben sie vier Problempunkte definiert, die die Bestimmung von Anforderungen erschweren, welche wären "(1) poor communication, (2) resistance as a physical and emotional expression, (3) articulation of problems, and (4) different perspectives on problem to be solved." (ebd. 2018, S. 56). Diese Problempunkte treten am ehesten auf bzw. wirken sich am schwerwiegensten in der Kommunikation mit Kunden aus, weshalb der Einsatz von speziellen Trainings für die Angestellten angedacht werden könnte. Dazu führten die Autor/innen Interviews mit Entwickler/innen, in denen sie Fragen zur Wahrnehmung von *Soft-Skills* stellten. Die Ergebnisse daraus sind, dass ihnen die Existenz von Soft Skills nicht immer bewusst ist, die Vernachlässigung dieser jedoch sich schlecht auf den Entwicklungsprozess auswirkt. Die Mehrheit der Befragten Personen versteht den Begriff, manche kennen sogar ähnliche Bezeichnungen und obwohl unbewusst, machen sie in der Praxis von *Soft-Skill*s gebrauch. Die Entwickler/innen gaben auch an, dass diese Kompetenzen in speziellen Kursen und Workshops verbessert werden könnten und damit sie langfristig in Übung bleiben, müssen im Laufe der Karriere immer wieder solche Kurse belegt werden. Das Bewusstsein über soziale, kommunikative Kompetenzen ist dabei Voraussetzung, dass der Begriff verstanden wird und der Bedarf für Verbesserung besteht. "It can be concluded that there is a certain awareness of the existence of soft skills, but also that these skills are often understood as skills that are not related to technical aspects of software engineering. It is clear that majority of software engineers do not think about these skills, but use them unconsciously in practice." (Zoric & Stojanov 2018, S. 62). Wahrscheinlich wird es davon abhängen in welcher Sparte und in welcher Position die Entwickler/innen arbeiten, inwieweit spezielle *Soft-Skills* Kurse notwendig und hilfreich sein können.

Wie soeben klar wurde, sind soziale Kompetenzen auch für Entwickler/innen ein wichtiger Faktor für die Arbeitswelt, was sich auch in der Studie von Iden und Bygstad (2018) zeigt,

die zwei Gruppen von Angestellten untersuchten, welche für die Entwicklung bzw. den Betrieb von Software zuständig sind und immer wieder miteinander kommunizieren müssen. Die beiden im Prozess involvierten Gruppen sind zum einen Software Entwickler/innen, die nach dem Launch noch einige Zeit zur Verfügung stehen, sollten Probleme auftreten. Zum anderen sind das die Personen, "*IT operations staff*", die später das Programm warten und am laufen halten, sie sollten aber auch während der Entwicklungsphase schon mitarbeiten können um bestimmte „nicht-funktionelle" Anforderungen zu identifizieren und vorzubereiten (ebd., S. 490). Die Kooperation ("*partnership and share knowledge*") und Koordination ("*communication and integrated planning*") in diesen Zeitspannen, in denen sie ihre gegenseitige Expertise benötigen ist ein ganz zentraler Faktor für das spätere Funktionieren der Software, darüber sind sich auch alle Personen aus den beiden untersuchten Gruppen einig. Im Fall von fehlender oder nicht ausreichender Kommunikation und Zusammenarbeit sind viele Fehler in der Software die Konsequenz, mit denen man sich im Nachhinein ärgern muss. "It is clear from our analysis that one consequence of many of the problems encountered in the social interaction is that new software contains more errors and is less stable in operation than necessary." (Iden & Bygstad 2018, S. 495). Als weiteres Ergebnis führen die Autoren an, dass Schulungen und verbesserte Dokumentation, sowie Wissenstransfer, die Service und Support Mitarbeiter bzw. das Problemmanagement bei auftretenden Störungen oder Nutzeranfragen unterstützen würden. Iden und Bygstad empfehlen den Projektmanagern, die Interaktionen vermehrt einzuleiten und auf längere Sicht zu fördern und zu verstärken, sowie die zuständigen Personen für den Betrieb der Software schon von Beginn an in den Entwicklungsprozess zu inkludieren (ebd. 2018).

Klar wurde durch die letzten beiden Artikel, dass Kommunikation unter Programmierer/innen sehr wichtig ist, damit das Produkt später auch so funktioniert wie es soll, ansonsten wird es für alle im Entwicklungsprozess beteiligen anstrengend. Was passiert, wenn Programmierer/innen (un)glücklich sind, wird im nächsten Absatz dargestellt.

Graziotin et al. hatten zum Ziel herauszufinden wie glücklich die Personen als Programmierer/innen sind und wodurch sie bei ihrer Arbeit glücklich oder unglücklich werden.

> "We found that the distribution of (un)happiness among software developers, in terms of a quantitative, well established instrument for assessing happiness, pointed towards developers being a slightly happy population, with happiness scores higher than those reported for many other parts of the general human population." (Graziotin et al. 2018, S. 3).

Ein paar ausgewählte, der unglücklich machenden Faktoren, sind zum Beispiel Zeitdruck,

bei einem Problem nicht weiter zu kommen, schlechte Qualität des Codes, mangelhafte Arbeit von Kolleg/innen oder für sich für eine Aufgabe ungeeignet zu fühlen. Ein ganz wichtiger Faktor für eine glückliche Stimmung der Entwickler/innen ist der Zustand oder das Gefühl sich im "*Flow*" zu befinden. Wenn sie in diesen Zustand gelangen, fühlen sie sich voller Energie und sie könnten diese Tätigkeit den ganzen Tag lang ausführen und eine Zeile Code nach der anderen nur so hinauszuschleudern. Dabei fällt ihnen auch weniger auf, wie die Zeit vergeht, was auch in den Ergebnissen einmal angemerkt wird. Gleichzeitig sind Faktoren, die den Flow unterbrechen besonders unangenehm und führen vermehrt zu einem unglücklichen Gefühl, weil es dann häufig schwierig ist, wieder in den Arbeitsfluss hineinzufinden, das lange Verzögerungen herbeiführt, längere Pausen gemacht werden und ein Gefühl des Feststeckens entsteht.

Die Autoren weisen darauf hin, dass manche der befragten Personen mehr auf negative und andere mehr auf positive Faktoren eingingen, und sie vermuten, dass dies wahrscheinlich mit der Persönlichkeit der/des jeweilige/n Entwicklers/Entwicklerin zusammenhängt. Als Schlussfolgerung ihrer Untersuchung sind sie der Meinung, dass ihre Ergebnisse über die verschiedenen Faktoren besonders für Projektmanager/innen, Teamleiter/innen aber auch Teammitglieder relevant sind, um mit diesen Erkenntnissen ein Umfeld zu schaffen, in dem sich Programmierer/innen möglichst wohl fühlen und somit guten Code schreiben und gute Produkte entwickeln. Dass glückliche Entwickler/innen bessere Leistung und höhere Produktivität in ihrer Arbeit erbringen, wurde laut Graziotin et al. schon in früheren Studien belegt (2018). Ein Faktor, der viele Entwickler/innen auch glücklicher macht, ist wie aus den Ergebnissen weiter unten hervorgeht, ein gewisser sozialer Aspekt oder eine gewisse Sinnhaftigkeit des Software Projekts. Der nächste Abschnitt geht darauf ein, wie Werte und Normen sich im Code widerspiegeln können.

3.11 Gesellschaftliche Werte in der Programmierarbeit

Ein Artikel, der auch ganz hilfreich ist die Fragestellung der Arbeit zu entwickeln, beschäftigt sich damit, dass im Code die Werte und Normen aus der Umgebung der Programmierer/in eingebaut bzw. eingeschrieben sind. Flanagin et al. schreiben:

> "Social constructivist perspectives, for example, argue that technological design is a function of interconnected social, cultural, technical, and economic factors. Technological artifacts thus result from a complex interaction between technical capabilities and the interests and values of many individuals, groups, and organizations. In this manner, technologies reflect felt social needs and current technical capabilities and are both the result of and impetus for social behaviors."

(Bijker et al., 1987; Bijker and Law, 1992; MacKenzie and Wajcman, 1985 zit. nach Flanagin et al. 2010, S.180).

Weiters sehen sie die "*technical code perspective*" als starkes Analyse Tool

> "because it exposes this relation [between technical properties, values, and social outcomes] by correlating technical properties with the social values leading to them or facilitated by them. In this manner it extends social constructivism by exploring the assumptions that are designed into technologies as a way to illuminate the values and choices that become manifest in them." (Flanagin et al. 2010, S.180).

Dieses Verständnis von Code unterstreicht die Ansicht über Programmierer/innen als Verbindungsstück zwischen Technologie und Gesellschaft, weil man annehmen kann, dass die sozialen Gegebenheiten und Umgangsformen etc. sich im Code eines technischen Artefakts wiederspiegeln. Ein Beispiel dafür wäre eine Gesichtserkennungssoftware, die von einem jungen "weißen" Mann programmiert wurde, welcher kaum mit Menschen anderer Hautfarbe zu tun hat und das auch in seiner Software nicht berücksichtigt. Dementsprechend wird die Software Schwierigkeiten haben, die Gesichter von Menschen mit anderer Hautfarbe zu erkennen.

Ein weiteres Beispiel wäre das Design einer Website, worauf auch Flanagin et al. (2010) hinweisen: "Technological design is an important indicator of technical code since design features directly reflect the underlying values and assumptions manifest in a technology." (S.182). Ein weiteres Beispiel aus eigener Erfahrung, wie sehr das Design auf die Gesellschaft Einfluss nehmen kann, ist der Vergleich von verschiedenen Web Tools, um "*best practice*" Beispiele zu finden und Ideen zu sammeln, wie die österreichische Version dieses Tools erneuert werden soll. Ein besonders wesentlicher Bestandteil des Designs kann sein, ob eine Webseite barrierefrei zugänglich ist. Das zeigt nämlich ganz genau, ob sich die Programmierer/innen (oder auch Auftraggeber/innen des Tools/der Website) Gedanken darüber gemacht haben, wie Menschen mit Sehbeeinträchtigungen oder anderen Behinderungen auf diese Webseite zugreifen können oder nicht. Bei Online Recherchen wurden sehr wenige Webseiten gefunden, die sich in dieser Hinsicht wirklich bemüht haben. Das zeigt, dass sich die Verbindungsstücke (Programmierer/innen) kaum bewusst sind über diese Mitglieder der Gesellschaft. Und nicht nur diese indirekten Auswirkungen spielen eine wichtige Rolle.

Ein Weg wie es möglich ist, Programmierer/innen gezielt für die Bedürfnisse von beeinträchtigten Personen zu sensibilisieren, ist die Verwendung von Simulationen, das die Programmierer/innen in die Situation versetzen kann, als wären sie selbst zum Beispiel von

unterschiedlichen Sehbehinderungen betroffen, wie die Studie über das Simulationsprogramm DIAS (Disability Impairment Approximation Simulator) zeigt.
Dieses soll es Interface Entwicklern ermöglichen, ihre Software oder App, mit den Sinnen einer beeinträchtigten Person wahr zu nehmen, um so erfahrbar zu machen, wie das Interface der Anwendung gestaltet werden muss, um es auch für diese Gruppe der Gesellschaft benutzbar zu machen. In ihrer Evaluation des Simulators weisen Giakoumis et al. (2013) darauf hin, dass Programmierer/innen grundsätzlich meist "Selbstbeobachtungsmethoden" zur Bewertung ihres eigenen Produkts anwenden.
"Thus, they typically predict user—product interaction, primarily based on their experience as potential users. After all, it is usually difficult for designers and developers to understand the problems users with disabilities could face during interaction with their software implementations. This can easily lead to products that are eventually inaccessible to, e.g., growing older adult population and people with different types of impairments." (ebd., S.227).
Um ihre Produkte nicht erst nach dem Launch nachjustieren zu müssen und um dem Problem direkt am Ursprung zu begegnen, empfiehlt es sich, Programmierer/innen selbst und vor allem während des Entwicklungsprozesses erfahren zu lassen, wie es sich für diese Personen anfühlen würde, ihr Produkt mitsamt der Zugangsbarrieren zu nutzen. Das barrierefreie Ergebnis würde die Qualität der "*Human Computer Interaction*" (HCI) um ein Vielfaches erhöhen. Giakoumis et al. (2013) kamen zu dem Ergebnis, dass das Simulationstool ausreichend gut funktioniert, mit den visuellen, akustischen, physischen und kognitiven Einschränkungen, die angezeigt werden. Getestet haben sie das, indem sie Anwendungen der Entwickler/innen auch von tatsächlich beeinträchtigten Personen bewerten ließen und ihr Feedback mit jenem der Simulation verglichen. Da die beiden Feedback Varianten größtenteils übereinstimmten, wurde klar, dass die Simulation sehr gut darstellen konnte, wie die betroffenen Personen das Interface wahrnehmen und in weiterer Folge die Programmierer/innen entsprechend darauf eingehen konnten. Die Autoren sind zwar unsicher, ob in Zukunft mehr Entwickler/innen das Tool öfter anwenden um die unterschiedlichen Behinderungen durchzutesten, doch sie heben zwei wichtige Punkte des Simulators hervor:

> "In particular, the DIAS tool: (1) can be used so as for developers to verify the barriers experienced by specific users when the design is devoted to the latter, whereas (2) it can also contribute in teaching designers the real impact of accessibility barriers for each specific affectation. By enabling UI developers to understand accessibility limitations through visual, hearing, physical and cognitive impairment simulation, the developed tool is expected to significantly

enhance the future development of applications that are accessible to people with special needs.” (Giakoumis et al. 2013, S.246).

Wenn auch nicht jede Software oder App damit getestet wird, so soll DIAS zumindest dann immer zum Einsatz kommen, wenn es wichtig ist, dass Technik barrierefrei zugänglich ist und insbesondere dann, wenn Technik darauf abzielt, die beeinträchtigten Personen in ihrem Leben zu unterstützen (ebd.).

Neben Giakoumis et al. sehen auch Flanagin et al. (2010), dass viele Programmierer/innen oder Techniker/innen mit ihren Erzeugnissen ganz bewusst Einfluss auf die Gesellschaft nehmen wollen. Wenn es zum Beispiel darum geht Software oder Apps zu entwickeln, die die für eine möglichst breite Verteilung von Nutzer/innen in der Gesellschaft frei und kostenlos zugänglich ist, wie Open Access Programme.

Ein anderer Aspekt, den sie noch ansprechen ist, wenn bestimmte Technologie oder aber Mensch-Technologie-Interaktionen direkt aber auch indirekt nicht zustande kommen kann (oder soll). Sei es aus politischen, sicherheitstechnischen oder anderen Gründen. Kinder sind zum Beispiel so ein Thema, sie müssen häufig bei ihren Mensch-Technologie-Interaktionen vor den diversen Gefahren im Netz bewahrt werden, wozu es auch Gesetze gibt. "*Intellectual property rights*" sind ein anderes Beispiel, wo Gesetze die Technologie direkt beeinflussen um Raubkopien oder Kinderpornographie zu verhindern, andererseits beeinflussen die Gesetze die Technologie indirekt, wenn Hacker versuchen, die gesetzlichen und technologischen Schutzmechanismen zu umgehen.

Dieser Fall wurde in einem Artikel behandelt, in dem die Frage gestellt wurde, ob Code Sprache [*speech* im Original] ist. Drei Personen haben programmierten Code genutzt um ein System zu umgehen, das verhindern soll, die Filmdateien von DVDs zu "kopieren" und wurden daher von acht Filmstudios verklagt. Im Text geht es um die Diskussion während der Gerichtsverhandlungen in denen am Ende beschlossen wurde, dass Computer-Code zwar eine Art von Ausdrucksform ist, aber in diesem Fall nur für Programmierer/innen verständlich und nicht allgemein für Nicht-Techniker lesbar. Das heißt der/die Otto-Normalverbraucher/in würde nicht verstehen, dass es sich bei diesem Code um den Ausdruck einer politischen Sichtweise der Programmierer/innen handelt. Es wurde in den Verhandlungen nur auf die funktionale Form des Codes fokussiert, zu dem nun eine einfache Regelung zur Restriktion solcher Tätigkeiten beschlossen wurde. Was an diesem Artikel besonders faszinierend scheint, ist die Tatsache, dass die Angeklagten ihren Code als eine Form von Protest sehen und sich damit gegen das Verbot, die DVDs zu kopieren, auflehnen wollten (Petersen 2015). So haben sie ihrem Gefühl von Ungerechtigkeit, in Form

von ausführendem Code, Ausdruck verliehen.

Eine weitere Möglichkeit, als Programmierer/in den eigenen Ansichten Ausdruck zu verleihen, ist es die gesetzlichen Regelungen zu umgehen und sich auf eigene Faust als *Do-It-Yourself (DIY)* Entwickler/innen zu versuchen. Kaziunas et al. (2018) untersuchten eine Gruppe von Personen, die an Diabetes Typ 1 erkrankt waren und gemeinsam mit Programmierer/innen ein DIY System entwickelte, das basierend auf Messdaten über den Blutzuckerspiegel einer Person und mittels Algorithmen berechnete, wie viel Insulin diese Person spritzen muss, bzw. diese Menge automatisch gespritzt wurde. Da diese Gruppe die herkömmlichen Methoden und Systeme unzureichend empfand, entschlossen sie sich, ihre eigenen Messdaten zu sammeln, zu teilen und mit Hilfe von Programmierer/innen so aufzubereiten, dass sie für Behandlungsentscheidungen herangezogen werden konnten. Diese Entscheidungen wurden dann auch ohne Einbezug der Ärzte getroffen und wenn die Ärzte darüber informiert wurden, zeigten sie kaum Interesse an erfolgreicher Selbstmedikation oder waren der Meinung, dass sie trotzdem Recht behielten. “In the creation of algorithms that dispensed insulin and experiments with bodies, coding itself became the expertise of care work. And yet, developers, in becoming experts in DIY health, also had to increasingly shoulder the burden of risk and responsibility.” (Kaziunas et al. 2018, S. 74). Ziel dieser Experimente am eigenen Körper war es also, mit den eigenen Messdaten, ein System, bzw. Algorithmen zu entwickeln, die die benötigte Menge an Insulin empfahlen und spritzten. Dabei begaben sich die Programmierer/innen (einige davon waren selbst Diabetiker) in einen risikoreichen Graubereich, da die Algorithmen vielleicht nicht immer funktionierten oder die einzelnen Personen unterschiedlich auf eine Menge an Insulin reagieren, was im schlimmsten Fall tödlich ausgehen kann. Dennoch waren die Programmierer/innen von der Wichtigkeit des Teilens der Daten überzeugt, ganz im Sinne von *Open Data,* weil sie an die Verbesserungen im Alltag dachten, die ihre Arbeit haben kann. Diejenigen in der Gruppe, die selbst nicht programmieren konnten, stellten ihre Daten meist als Testsubjekt gerne zur Verfügung, weil sie mit einem/einer der Programmierer/innen bekannt waren und die Sache unterstützen wollten (ebd.). Programmierer/innen können also auch den Gesundheitsbereich maßgeblich und auf eigene Faust mitgestalten, was durch den Trend der Selbstvermessung mit Diversen *Smart- und Tracking Devices* kein Problem mehr darstellt. Wenn es aber um ernste Krankheiten geht, muss auch die Datensicherheit ernst genommen werden, man weiß nie, wer die Daten in die Hände bekommen könnte und leider finden sich auch immer Menschen, die anderen

Schaden zufügen wollen.

3.12 Das Smartphone als täglicher Begleiter und risikoreiche Ablenkung

Der nächste Artikel ist quasi ein Beleg dafür, dass Smartphones vom überwiegenden Teil der Gesellschaft jeden Tag benutzt werden, wie in der Einleitung bereits erwähnt wurde. Carolus et al. (2019) haben die Beziehung zwischen Nutzer/innen und ihren Smartphone untersucht und sehen diese trotz der täglichen, über mehrere Stunden dauernde Benutzung, nicht als Sucht induzierende Abhängigkeitsbeziehung, sondern als Kameradschaft bei der das Smartphone als "treuer Begleiter" eine Unterstützung für alltägliche Tätigkeiten gesehen werden kann. Bei ihrer Analyse von Smartphone Nutzer/innen fragen sie sich also, ob dieses Verhältnis ein "*digital companionship*" darstellt und ob grundlegende Charakteristiken einer menschlichen Beziehung auf die Beziehung mit dem Smartphone angewendet werden können. Die Studien auf Basis derer sie ihre Überlegungen anstellten fanden vor 20 Jahren mit Stand-PCs statt aber trotz der Größe und Immobilität der Geräte, senden diese soziale Signale auf die die Nutzer/innen automatisch und unbewusst mit sozialem Verhalten reagierten (Lee et al. 2000; Nass et al. 1994). Für ihre Untersuchung haben Carolus et al. die drei Eigenschaften Nähe, Vertrauen und Sorge mit Fragebögen zu mehreren dahinter liegenden Dimensionen getestet. Weiters haben sie die befragten Personen eine Art Familienaufstellung durchführen lassen, nur dass sie abgesehen von Familienmitgliedern und Freunden auch elektronische Geräte, wie Smartphone und Co. auf einem "Brett" platzieren durften. So konnten sie auch auf diese Weise Schlussfolgerungen über die emotionale Nähe oder Distanz des Smartphones ziehen. Entgegen der Annahmen der Forscher, wurde die Ansicht über diese Nahbeziehung von Nutzer/innen zu ihrem Smartphone auch von den befragten Personen als richtig angenommen und bestätigt, was sich auch in den statistischen Ergebnissen zeigte. Nämlich, dass das Smartphone unter anderen Geräten, das wichtigste ist und bezüglich Beziehungen zwischen engen, wertvollen Beziehungen und eher entfernten sozialen Kontakten einzuordnen ist. Weitere Ergebnisse ihrer Studie sind zum Beispiel, dass der Stress weniger durch das Smartphone ausgelöst wird. Stattdessen wirkt es eher unterstützend bei der Bewältigung von Stress. Nähe zum Smartphone kann als Grundlage für die Entwicklung von Vertrauen und Sorge gesehen werden, was in Mensch zu Mensch Beziehungen ähnlich funktioniert. "Older users reported lower levels of trust and preoccupation; more frequent users reported higher levels of closeness, trust, and preoccupation. For gender, however, effects were less straightforward: Females reported lower levels of trust, but higher levels of preoccupation compared with males." (ebd. 2019, S. 934). Neben der Erfüllung der herkömmlichen, täglichen Aufgaben,

wie Information, Organisation, Navigation oder Kommunikation, übernimmt das Smartphone auch das Stillen, gewisser menschlicher Basisbedürfnisse und erweitert so die Perspektive auf die Smartphone Technik. Dieser digitale Kamerad kann als Erweiterung und natürlich Unterstützung von Kommunikationskompetenzen gesehen werden, da er Zeit und Distanz überwindet und zumindest vorübergehend emotionale Nähe erzeugt. “Consequently, our phone is not only a technological device but also a psychologically relevant entity.” (Carolus et al. 2019, S.934).

Wie gefährlich jedoch diese ständige Verwendung des Smartphones in manchen Situationen werden kann, zeigt die Studie von Licoppe und Figeac (2018), in der sie sich damit beschäftigen, an welchen Punkten in der App Nutzung Blickwechsel weg vom Smartphone stattfindet.

Genauer gesagt haben sie das mobile App Nutzungsverhalten von Personen beobachtet, während diese am Weg zwischen Wohnort und Arbeitsplatz waren, dabei kamen sowohl Autofahrer/innen als auch Personen, die die öffentlichen Verkehrsmittel nutzten zum Zug. Durchgeführt wurde die Untersuchung mittels einer Kamerabrille, welche die Blickrichtung zeigte und einem *Smartphone-Tracking-System*, das genau aufzeichnet, was die Personen während der Nutzung ihres Mobiltelefons am Bildschirm sehen. In ihrem Artikel erklären sie ihre zentralen Ergebnisse anhand des Beispiels einer autofahrenden Frau, die währenddessen die Facebook App nutzt und sich somit in einer Situation von “*multiactivity”* befindet, wie die Autoren es benennen. Diese Situation gestaltet sich so, dass die Frau beim Halten an einer roten Ampel, ihre Aufmerksamkeit auf die App richtet und die Wartezeit an der Ampel damit verbringt mehrere Posts und die Kommentare darunter zu lesen. Wenn die Protagonistin auf eine andere Seite der App gelangen will, muss die Seite geladen werden, was einen Moment dauern kann. Das Warten während der Ladezeit gibt der Frau die Möglichkeit, einen Blick auf die Ampel bzw. den Verkehr zu werfen. Diese Pausen, beim Autofahren durch die Ampel, bei der App durch das Laden, führen zur Fortsetzung der jeweils anderen Tätigkeit, was Licoppe und Figeac als “*transition points*” bezeichnen. Durch diese Erkenntnis sind sie auf die Idee gekommen, dass man sich diese “*ruggedness*” (Holprigkeit) zu Nutze machen könnte, indem man mobile Apps absichtlich “holprig” macht um so, wie im erwähnten Beispiel mehr *transition points* zurück zur Ampel zu ermöglichen. Auf der anderen Seite würde diese Eigenschaft aber mit einem sehr unangenehmen, unpraktischen Interface außerhalb solcher Situationen einhergehen. Für Nutzer/innen, die ihr Smartphone nicht während dem Autofahren benutzen wäre das ein großer Nachteil, deshalb könnte man die Holprigkeit auch als eigene Funktion der App einbauen, damit sie nicht von Grund auf unangenehm zu bedienen ist, sondern nur in den bestimmten

multiactivity Situationen, in denen ein erhöhtes Risiko besteht.

> “Designing specifically for multiactivity environments where safety concerns are critical might therefore involve increasing the ruggedness of the mobile interfaces to augment the frequency of possible transition points and minimize temporal mismatches in the demands of the varied activities in which the user is engaged, even if such a design rationale might run against more conventional design strategies centered on user friendliness.” (ebd. 2018, S. 331).

Zugunsten der Sicherheit wären die Autoren also bereit, die Nutzerfreundlichkeit gegen ein Design einzutauschen, das die Aufmerksamkeit wieder vom Smartphone weg lenkt. Inwieweit diese Idee umsetzbar ist, ist eine andere Frage.

Es wurde nun gezeigt, in welchen Bereichen und auf welche Art und Weise Studien über und mit Programmierer/innen durchgeführt wurden. Von der Persönlichkeit, über die Arbeitsweise oder die Arbeitsbedingungen, sowie sozialem Engagement im medizinischen Bereich oder politischen Statements, bis hin zu Nutzungsverhalten von Usern. Ein Forschungsbereich, der ganz wesentlich mit dem Nutzungsverhalten zusammenhängt, ist die Usability Forschung, bei der es darum geht herauszufinden, wie angenehm und ansprechend eine Software oder eine App zu verwenden ist.
Das Thema Usability ist für die vorliegende Arbeit deshalb von besonderer Wichtigkeit, weil es sehr eng mit den Aushandlungsprozessen zwischen den einzelnen Akteuren (Vinck 2003) im Software- oder App-Projekt verbunden ist. Um eine gute Usability für ein Produkt zu gewährleisten, ist der Austausch mit der Zielgruppe von entscheidender Wichtigkeit, wie auch in den Ergebnissen weiter unten klar wird.
Im folgenden Abschnitt wird auf die Diskussion über Usability und deren Funktion als Konzept oder Konstrukt in der Forschung eingegangen.

3.13 Usability - Konstrukt oder nicht?

Das Thema Usability gewinnt immer mehr an Bedeutung, seit Technikunternehmen bemerkt haben, dass es einen großen Unterschied in den Verkaufszahlen macht, ob einem Produkt eine gute Usability zugesprochen wird oder nicht. Natürlich hat sich auch die Wissenschaft eingehender damit beschäftigt und man ist sich allgemein einig, dass der Begriff sehr breit ist und sehr viel Raum für Interpretation zulässt. Tractinsky (2018) zum Beispiel ist der Meinung, dass man dieses Problem des “*umbrella constructs*” beheben soll. Er unterscheidet dabei Konstrukte und Konzepte. Während letztere lediglich ein Phänomen aus der realen Welt als ein Abbild darstellen, so sollten erstere, dieses Phänomen auch

messbar machen. Ein Konstrukt macht ein Konzept messbar, wenn man so will. Dazu müssen bestimmte Parameter festgelegt werden anhand derer das Konzept messbar gemacht wird. “Obviously, how a construct is characterized is essential for how it may be measured. We cannot measure a construct accurately and reliably unless we know what our measures represent and what they are supposed to measure.” (Tractinsky 2018, S. 139f). Da ein *umbrella construct* jedoch so breit gefasst werden kann, wird es in Untersuchungen immer wieder mit verschieden definierten Parametern gemessen, was Tractinsky in seinem Artikel als großes Problem sieht, da es die Vergleichbarkeit, Reproduzierbarkeit und Validierung von Studienergebnissen deutlich schmälert oder gar unmöglich macht. Wenn keine Klarheit über ein Usability Konstrukt gegeben ist, erschwert das die Kommunikation in der Wissenschaft und die Wissensgenerierung darüber. Der Autor ist der Ansicht, dass die Verwendung dieses Konstrukts in eine Sackgasse führen wird, sofern nicht bald Klarheit und Einigkeit über die Definition der Charakteristika und der messbaren Dimensionen herrscht (ebd.).

> “[...] research on the usability construct has reached a dead end and cannot be relied upon to contribute to the scientific advancement of the field of HCI. I propose instead a different research trajectory that rests on collapsing the umbrella construct into its constituent subdimensions (e.g., efficiency, ease of use, satisfaction, enjoyment, etc.). This will liberate researchers to propose and empirically evaluate in new nomological nets the relations of each subdimension with various potential antecedents (e.g., design features), consequences (e.g., user attitudes or behavior), under varying contexts (e.g., task, user and culture characteristics).” (ebd., S.169).

Als Lösung schlägt Tractinsky vor, bei Untersuchungen anstatt des gesamten Konstruktes, nur seine Subdimensionen zu verwenden um das zu messen, was Usability ausdrücken soll. Kurz zusammengefasst wünscht sich der Autor dieses Vorgehen, um das dahinterliegende Konzept besser quantifizierbar zu machen. Offensichtlich argumentiert er aus einer eher quantitativen Forschungsperspektive.

Als Reaktion auf diesen Artikel gab es eine ganze Reihe an Kommentaren, die eine unterschiedliche Meinung vertreten. Zum einen antwortet Hertzum (2018) aus der Sicht von Herbert Blumer (1954), dass Konzepte relativ gering spezifizierte Abstraktionen eines empirischen Phänomens sind, das sie abbilden sollen. So sind sie für viele verschiedene reale Phänomene anpassbar und wären sie genau definiert, könnten sie nur für ein einziges Phänomen angewendet werden. Im Gegensatz zu diesem definitiven Konzept, schlägt Blumer ein “*sensitizing concept*” vor, das als allgemeines Referenzkonzept helfen soll, sich

in eine Richtung zu orientieren, wie an empirische Phänomene herangetreten werden soll (ebd.). Mit diesem Vorschlag stimmt auch Hertzum (2018) überein, der weiter ausführt, dass Konzepte, wie auch Usability, demnach nicht nach dem "*one fits all*" Prinzip zu behandeln sind, sondern sie müssen entsprechend der untersuchten Situation angepasst werden.

> "A definitive concept also entails a risk that usability research becomes a formal quest for conceptual precision at the expense of practical relevance, that characteristics consequential to the usability of empirical instances are rendered invisible because they are not among the common features that define the usability concept, or more generally that the a priori specification of the concept obscures the very thing we seek to understand. In addition, the usability concept will remain ambiguous in the sense that there will still be a need for interpretation and common sense in applying it to empirical instances." (Hertzum 2018, S. 179).

Das heißt, würde Usability als Konstrukt auf bestimmte zu messende Dimensionen reduziert, wären wichtige Charakteristika der Usability in dieser spezifischen Situation nicht mehr sichtbar. Nur wenn bekannt ist, wer die Nutzer und ihre Ziele sind, sowie der Kontext des Gebrauchs der Technik, ist es möglich Usability in diesem Zusammenhang zu untersuchen, da Nutzerzufriedenheit immer von unterschiedlichen Faktoren abhängig ist (ebd.).

Bertelsen (2018) weist in seinem Kommentar darauf hin, dass Usability nicht wie in Naturwissenschaften untersucht werden muss (oder kann) und dass das akademische Konzept von Usability nicht gleich dem Konzept in der praktischen Realität sein muss. "In contrast to Tractinsky's implicit assumptions, I believe that studies of practical usability work is a necessary foundation for an academic concept of usability." (ebd., S. 182). Wie wird Usability also praktisch erforscht? Ein wichtiges Beispiel wie diese Forschung aussehen kann, ist die "*think-aloud*" Methode (Nørgaard & Hornbæk 2006; Boren & Ramey 2000), bei der Nutzer ihre Gedanken zur Bedienung eines technischen Artefakts vor sich her sprechen, während sie es testen. Scheinbar gibt es aber Unterschiede zwischen akademischen Standards und der realen Anwendung (ebd.).

Bertelsen ist der Meinung es wäre wichtiger ein Forschungsfeld zu etablieren, das die Praxis ernst nimmt und für diese relevante Theorien und Methoden bietet. "Thus, I propose another way forward, starting with the acknowledgment of the primacy of practice in HCI. Users' practice when interacting with computers, as well as developers and designers' practice when building new systems." (Bertelsen 2018, S. 183). Der Autor des Kommentars ist der Meinung, dass HCI erstens untersuchen soll, was Usability für die Menschen bedeutet, die damit arbeiten und wie es sie beeinflusst. Zweitens sollte Designpraxis und

Entwicklungspraxis untersucht werden für ein besseres Verständnis des Desingablaufs und der Verbesserung von Designsystemen. Drittens sollten Methoden auf Basis des zweiten Punktes entwickelt werden um Usabilitytests weiterentwickeln zu können. Viertens sollten zum Abschluss Untersuchungen auf Basis kontrollierter Experimente durchgeführt werden, ähnlich wie Tractinsky es sich vorstellt. Zum Beispiel mit der Frage, wie gut Nutzer/innen bestimmte Interface-Elemente verstehen (Bertelsen 2018). Für ihn ist es also wichtig ganz nahe an der praktischen Realität der Usability Forschung zu bleiben, um darauf aufbauend Theorien und Methoden passend zur Vielfalt der Techniknutzung zu entwickeln.

Reeves (2018) schlüpft in seinem Kommentar sozusagen in die Perspektive von Usability-Forscher/innen, die die Tests mit den Nutzern durchführen, auf deren Basis die Technik verbessert werden soll. Dabei kann er auf eigene Erfahrungen durch die Beobachtung dieser Forscher/innen zurückgreifen und stellt fest, dass es in der Praxis kein Problem gibt mit einem Usability Konstrukt, wie Tractinsky es sieht. Der Autor merkt an, dass es natürlich Diskussionen im Zusammenhang mit Usability gibt, zum Beispiel bezüglich der Anzahl der untersuchten Nutzer/innen, wie viel *think-aloud* angewendet werden soll oder über die Unterschiede von lokalem Testen und Testen aus der Ferne (remote). Reeves ist also der Meinung, dass man Usability getrennt aus zwei Perspektiven sehen kann. Zum einen aus der Sicht der praktischen User-Test-Forschung, wo es darum geht technische Produkte zu verbessern und zum anderen aus der akademischen Sicht, bei der Forschungsberichte, Beiträge zum akademischen Wissen und die Weiterführung von Forschungsförderungen etc. das Ziel sind (ebd.). Er kommt zu dem Schluss, dass Usability in der Praxis nicht einfach a priori vordefiniert werden kann. “Usability-in-practice cannot necessarily be readily, formally specified beforehand. This is perhaps one of the key reasons for actually even doing usability testing, where troubles-in-use unfold as identifiable socially shared objects, “filled in” by the interactional work of moderators and test participants.” (ebd. 2018, S. 193). Auch er weist darauf hin, dass Usability Testing immer von mehreren Faktoren abhängt.

Die Kritik von Stage (2018) fokussiert sich auf die Einseitigkeit von Tractinskys Artikel: “Tractinsky focuses solely on theory with no concern for the practice that this theory is about. In an area like human–computer interaction (HCI), where practice is of major importance, that is a very radical and discussable position.” (ebd., S. 196). Des Weiteren ist es seiner Ansicht nach nicht klug, Theorie und Praxis strikt zu trennen, außerdem wird es nicht möglich sein, Usability komplett objektiv zu fassen, da es immer unterschiedliche Interpretationen geben wird. Das zweite Problem das Stage an Tractinskys Artikel sieht, ist sein einseitiger Fokus auf die Elemente der Definition von Usability und dass nur durch diese Elemente das Konstrukt gemessen und erklärt werden kann. Der Autor meint, es geht auch

anders: “[...] to see the definitions as ways of explaining what usability is, while the actual techniques for measuring it may be different.” (ebd., S. 196).

Zusammengefasst sind die zentralen Kritikpunkte:

- starres Anhalten an definitivem Konzept, das nur auf ein reales Phänomen anwendbar ist
- die a priori Definition des Konzepts
- Einseitigkeit und Trennung zwischen Theorie und Praxis
- Vernachlässigung der Praxis (Usability Testing)
- Vernachlässigung der Kontextgebundenheit von Usability (immer beeinflusst durch unterschiedliche Faktoren)

Aus den Reaktionen war nun zu lesen, dass deren Autoren Usability eigentlich nicht als Konstrukt bezeichnen würden.

Diese unspezifische Breite des Konzepts Usability, welche Traktinsky als problematisch wahrnimmt, sehen Andere als Vorzug davon. So zum Beispiel Morten Hertzum (2010), der sich mit der Vielfalt von Anwendungsmöglichkeiten von Usability eingehend beschäftigt. Er schreibt davon, dass dieser Begriff in sechs grobe “*images*” unterteilt werden kann, da es keine einfache, einheitliche Definition gibt, sondern immer in unterschiedlicher Weise aber entsprechend der betreffenden Situation verstanden wird. Hertzum unterteilt Usability in die Perspektiven: “*universal, situational, perceived, hedonic, organizational und cultural*”. Die Grenzen zwischen diesen Varianten von Usability sind zwar verschwommen aber greifen ineinander und der Autor plädiert bei der Anwendung herauszufinden, welche davon am ehesten zur Situation passt und diese als dominante Variante zu erklären. Für die Anwendung empfiehlt er in drei Stufen vorzugehen: 1. in “*discovery stage*” werden die unterschiedlichen Verständnisse von Usability eines Systems herausgefiltert. 2. die “*integration stage*” ist dazu da, eines der Usability Verständnisse als dominantes zu erklären und vielleicht ein oder zwei weitere als ergänzende Perspektiven hinzuzunehmen. 3. in der “*challenging stage*” soll die Stellung des dominanten Bildes immer regelmäßig hinterfragt werden, indem ein anderes Bild als dominant erklärt wird um so eventuell auf blinde Flecken der ursprünglich dominanten Variante zu stoßen (Hertzum 2010).

- *Universal Usability* soll für möglichst viele Nutzer anwendbar sein, jedoch ohne dem *one-size-fits-all* Prinzip, das nur auf den kleinsten gemeinsamen Nenner abzielen würde und somit eigentlich einen Großteil der Nutzer/innen ausschließen würde.
- *Situational Usability* fokussiert sich auf die bestimmten Personen, Aufgaben, Werkzeuge und andere kontextbezogener Faktoren, die eine Nutzungssituation einzigartig machen. Laut Hertzum kann hier die ISO-9241 Definition von Usability

angewendet werden: “Usability: Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. Effectiveness: Accuracy and completeness with which users achieve specified goals. Efficiency: Resources expended in relation to the accuracy and completeness with which users achieve goals. Satisfaction: Freedom from discomfort, and positive attitudes towards the use of the product.” (ebd. 2010, S.572). Daraus wird klar, dass sich die Definition immer auf eine bestimmte Nutzungssituation bezieht.

- für *Perceived Usability* ist die subjektive Nutzungserfahrung einer Person zentral
- *Hedonic Usability* fokussiert sich ähnlich wie *Perceived Usability* auf einzelne Individuen, hier geht es jedoch mehr um das Vergnügen und die Emotionen bei der Nutzung. Es dreht sich weniger um die Erfüllung von Aufgaben, als um die Ästhetik und Schönheit des Designs.
- *Organizational Usability* ist von Bedeutung für organisationale Gruppen die ein System im Rahmen ihrer kollaborativen Arbeit nutzen.
- *Cultural Usability* berücksichtigt die kulturellen Hintergründer der Nutzer. Zum Beispiel ist die *think-aloud* Methode mit amerikanischen oder generell westlich geprägten Nutzern recht einfach und normal durchzuführen, während Ost-Asiatische Nutzer es unangenehm empfinden würden, ihre persönlichen Gedanken, vielleicht auch Kritikpunkte vor einer fremden Person zu offenbaren.

“The six images of usability have distinct foci but are at the same time interwoven. The interactions between the images comprise their shared essence, which centers on the fitness for use, ease, and intuitiveness of systems, and a more extensive blending of borders between particular pairs of image.” (Hertzum 2010, S. 588f). Die Definition von Usability setzt sich also immer aus einer Reihe verschiedener Faktoren zusammen, die für bestimmte Situationen charakteristisch sind.

Mit dieser Auswahl an Artikeln über die unterschiedlichen Bereiche der Software Entwicklung und der Programmierer/innen dahinter wurde zumindest ein kleiner Einblick geboten in dieses ebenso relativ kleine Forschungsfeld.

4 Zielsetzung dieser Arbeit

Ganz konkret war das Ziel herauszufinden, ob, wie und in welchem Ausmaß Programmierer/innen sich als Bindeglied zwischen Technik und Gesellschaft wahrnehmen und wenn ja, ob sie dieses Bewusstsein auch aktiv bei ihrer Arbeit einsetzen. Es stellte sich auch die Frage, wie sie ihr Verhältnis zu Technologie einerseits und zur Gesellschaft andererseits einschätzen würden und wie sie zu dieser Perspektive gekommen sind. In weiterer Folge dann auch, wie sie ihre Rolle als Verbindungsstück in der Zukunft sehen.
Um das Thema etwas zu fokussieren stelle ich die Frage, wie gesellschaftliche Werte ihren Weg in ein Programm/eine Anwendung finden. Gehen die Entwickler/innen rein nach ihren Vorstellungen und ihrem Wissen vor? Wird im Team ausgehandelt, was die App können und wie sie aussehen muss? Oder werden sogar potentielle Nutzer/innen und Interessent/innen befragt, welche Funktionen sie für wichtig halten? Dabei beschränkte sich die Arbeit auf Programmierer/innen, die vorrangig im Rahmen eines Start-ups an einer Mobile App arbeiten und dafür den Code schreiben. Die Wahl fiel auf Mobile App, weil diese “kleinen, einfachen” Anwendungen auf Smartphones und Tablets mittlerweile einen zentralen Stellenwert in unserem Leben ausmachen. Ganz egal ob es Messenger-Apps, Mobile-Banking-Apps aus der *Fintech Branche* (*financial technology*), Sprach-Lern-Apps, Photobearbeitungsapps, Karten-Apps zur Orientierung oder Gaming-Apps sind, wir sind umgeben von diesen Anwendungen, die unser gesellschaftliches Verhalten maßgeblich mitgestalten.
Insbesondere war der Plan 4 - 5 Start-ups auszuwählen, die möglichst für unterschiedliche Bereiche in der Gesellschaft Mobile Apps entwickeln, um eventuell Unterschiede in deren Herangehensweise herauszufiltern. Je nach der Größe des Start-ups, sollten alle oder ein Teil des Entwicklerteams interviewt werden, so dass sich jedenfalls ca. 12 - 15 Interviews ergeben.
Grundsätzlich arbeiten ja auch an einer Mobile App Programmierer mit unterschiedlichem Fokus. Eine/r kümmert sich um die zugrundeliegende Struktur, ein/e Andere/r arbeitet am Front-End (der optischen Aufmachung) der App, usw. Eine weitere Differenzierung dahingehend wurde bei den Interviewpersonen jedoch nicht vorgenommen, da es sonst zu schwierig geworden wäre in der geplanten Zeit genügend Personen zu finden. Es wurden also ganz allgemein Mobile App Programmierer interviewt, unabhängig davon, ob sie das *Back-end* oder das *Front-end* der App entwickeln.
Es soll auch noch angemerkt werden, dass es nicht das Ziel dieser Arbeit ist, repräsentative Ergebnisse zu liefern. Diese Arbeit ist rein qualitativ durchgeführt worden und hegt somit

keinen Anspruch auf universale Gültigkeit.

5 Relevanz

Obwohl die Fragestellung eher breit und etwas unspezifisch klingen mag, was daran liegt, dass es dazu noch relativ wenige Quellen, so gibt es umso mehr den Grund, diese Lücke zu füllen und mit dieser Arbeit quasi einen explorativen Beitrag zur wissenschaftlichen Forschungswelt in diesem Bereich zu leisten. Darauf aufbauend besteht die Hoffnung, dass sich weitere Anknüpfungspunkte für Forschende in dieser Sparte, aber natürlich auch für alle Interessierten an diesem Kreuzungspunkt, seien sie aus Wirtschaft, Politik, Industrie, der "*Developer-Szene*", etc. ergeben.
Mit der Forschung soll vordergründig die Sichtweise der Programmierer/innen auf ihre eigene Rolle beleuchtet werden. Nach der Aufarbeitung des Forschungsstandes, wurde klar, dass der Technik -Aspekt in der Wissenschafts- und Technikforschung noch Aufholbedarf hat, und insbesondere diese Personengruppe in der Software-, App- und Webentwicklung noch eingehender untersucht werden sollten.

5.1 Forschungsfrage

Aus der Aufarbeitung des Forschungsstandes und der Zielsetzung ergibt sich diese vorläufige Forschungsfrage:
„Wie nehmen Programmierer/innen ihre Rolle als Verbindungsstück zwischen Technik und Gesellschaft wahr?"
Dazu stellt sich noch die Frage, ob sie diese Rolle überhaupt wahrnehmen und ob sie sie von sich aus ansprechen, oder würden sie diese Frage eher zurückweisen und diese Rolle als nicht-existent erklären.

5.2 Unterfragen:

Wie sieht die Verbindung zur Gesellschaft ihrer Meinung nach aus?
Die Frage lässt sich so erläutern: Mit welchen Akteuren aus der Gesellschaft stehen die Programmierer/innen in Kontakt, wie häufig findet dieser Kontakt statt, welche Konflikte treten dabei auf, was ist ihnen selbst dabei wichtig, wie formell oder informell ist dieser Kontakt.

Wie sieht die Verbindung zur Technologie ihrer Meinung nach aus?
Folgende Dinge könnten bei dieser Frage angesprochen werden: Wie wird über Technik gesprochen, was ist wichtig bei der Arbeit mit Code und Programmiersprachen, welchen

Bezug haben sie zur Technik, wann und wie haben sie Programmieren gelernt, gibt es bestimmte Regelungen oder Vorgehensweisen bei der Arbeit.

Wie sieht für sie die Zukunft in dieser Rolle aus?
Die Smartphone Nutzung befindet sich nach wie vor im Anstieg, und der vielbeklagte Fachkräftemangel reißt nicht ab, welche Zukunftsvisionen haben mobile App Programmierer/innen für den eigenen Beruf und die Beziehung sowohl zu Technik als auch zu Gesellschaft.

Wie hoch schätzen sie den Einfluss der Gesellschaft auf Ihr Tun, ihre Produkte ein?
Welche Einflussfaktoren werden angesprochen, welche davon sind die wichtigsten, wie nehmen Entwickler/innen diese Einflussfaktoren wahr und wie gehen sie damit um.

Wie hoch schätzen sie den Einfluss ihres Tuns, ihrer Produkte auf die Gesellschaft ein?
Wollen Programmierer/innen bewusst Einfluss auf die Gesellschaft nehmen und wenn ja, welche Ziele verfolgen sie dabei und wie setzen sie das im Code um; gibt es spezielle Zielgruppen, die sie mit ihrem Einfluss erreichen wollen.

Wie gelangen gesellschaftliche Werte durch die Programmierarbeit in die Technologie?
Wie läuft der Entwicklungsprozess einer mobile App ab, und an welchen Stellen in diesem Verlauf, treten Aushandlungsprozesse mit anderen Akteuren aus der Gesellschaft auf.

6 Methodenauswahl

6.1 Interview

Zur Frage, welche Methoden angewendet werden sollen, stellte sich ganz klar das Problemzentrierte Interview nach Witzel (2000) als angemessen heraus, da es als teilstrukturiertes Leitfadeninterview einerseits ausreichend Offenheit bietet, um Erzählungen zu generieren und andererseits sich an den von der Interviewerin vorgegebenen Themen entlang entspinnt (Atteslander 2010).

Kombinierte Interviews werden mittlerweile vermehrt angewendet, weil sie die Vorteile von unterschiedlichen Interviewmethoden in sich vereinen und so und die jeweilige Untersuchung entsprechend angepasst werden können. Einerseits ist das Interview durch den Fokus auf das Thema und den Leitfaden strukturiert und andererseits weist es die erzählgenerierende Qualität eines narrativen Interviews auf, indem jede Leitfadenfrage offen gestellt wird und versucht wird eine Erzählung anzuregen (Helfferich 2011, S.42).

Wenn in der Forschung, „einerseits subjektive Theorien und Formen des Alltagswissens zu rekonstruieren sind und so maximale Offenheit gewährleistet sein soll, und wenn andererseits von den Interviewenden Themen eingeführt werden sollen und so in den offenen Erzählraum strukturierend eingegriffen werden soll“ (ebd. 2011, S.179), dann eignet sich das problemzentrierte Interview besonders gut. In dieser Forschung ist das der Fall, da in erster Linie die subjektive Sichtweise der Programmierer/innen herausgefiltert werden soll, insbesondere ihre Sichtweise über ihren eigenen Berufsstand und dessen Verknüpfungspotenzial zwischen Technologie und Gesellschaft. Der Leitfaden wurde so gestaltet, dass zu Beginn eine sehr offene, eher narrative Einstiegsfrage stand, bei der ihr bisheriger Werdegang als Programmierer/in, vom ersten Kontakt mit Computertechnik bis zum aktuellen Tag, wiedergegeben wurde. Darauf folgten einige kürzere spezifischere Fragen bezüglich ihrer Arbeit und diverser Kontaktpunkte zur Gesellschaft. Eine Frage richtete sich an die Begeisterung an Technologie bzw. dem Programmieren. Die zweite Frage zielte auf ihre Einschätzung des Stellenwerts der Programmiertätigkeit in der Gesellschaft ab und die dritte Frage widmete sich den Zielen beim Programmieren hinsichtlich Technik einerseits und Gesellschaft andererseits. Die Nachfragen vier bis neun beschäftigen sich mit der Ideenfindung und dem Entwicklungsprozess einer App von der Idee bis zum „Abschluss“, wobei auch Teamarbeit, Testphasen und äußere Einflüsse, wie zeitlicher-/finanzieller Rahmen oder Kund/innen- und Nutzer/innenwünsche, eine Rolle spielen. Die Fragen zehn bis zwölf fragten nach dem eigenen Code und den allgemeinen

Anforderungen für guten Code. Am Ende des Interviews wurden noch Alter, Herkunft, Programmiersprache und wie lange schon programmiert wird abgefragt.
Zu guter Letzt gab es bei jedem Interview den Versuch, am Ende die Forschungsfrage zu stellen, welcher im Grunde die Vermutungen bestätigte, dass die Programmierer/innen sich zwar nicht direkt bewusst als Verbindungsstück wahrnehmen, sonst hätten sie es in ihren Antworten vielleicht direkter erwähnt. Unterbewusst fühlen sie sich aber als ein solches und wenn sie direkt darauf angesprochen werden durch meine Forschungsfrage, sind sie mit dieser Sichtweise einverstanden und bestätigen sie. Dazu mehr in den Ergebnissen.

6.2 Sampling

Da es zu diesem Thema noch wenige und vor allem wenig rezente Studien gibt, an denen man sich orientieren könnte, wurden die Auswahlkriterien für Interviewpersonen nach der Erkenntnis gewählt, dass mobile Apps mittlerweile vom Großteil der Gesellschaft verwendet werden. Wie in der Einführung schon erwähnt, sind rund 77 % der österreichischen Bevölkerung Smartphone Nutzer (Schultz 2019, statista.com), weshalb mobile App-Entwickler aus Wiener Start-ups für diese Untersuchung ausgewählt wurden. Die Begrenzung auf den Raum Wien ergibt sich aus dem einfachen Grund, dass es hier viele Start-ups gibt und nicht extra ein weiterer Weg zu fahren ist.
Für das Sampling wäre vorgesehen gewesen, aus vier bis sechs Start-ups zwei oder mehr mobile App Entwickler/innen zu interviewen und so eventuell Fallstudien auf Basis der Start-ups zu generieren. Da sich die Kontaktaufnahme jedoch schwieriger als erhofft erwies, wurden mit allen Entwickler/innen, die sich bereit erklärten, Interviews geführt. Das heißt aus manchen Start-ups, die kontaktiert wurden, war nur eine Person bereit, aus einem Start-up waren zwei Personen damit einverstanden und aus einem Start-up waren sogar sechs Personen bereit, ein Interview zu führen. Insgesamt war geplant zwölf bis fünfzehn Interviews zu führen. Da sich nach dem zwölften Interview keine besonders neuen Informationen mehr ergaben, wurde das Sampling entsprechend der theoretischen Sättigung bei zwölf Interviews belassen.
Abgesehen von den Kategorien mobile App Entwickler/in und Wiener Start-up gab es keine Einschränkungen. Es soll hier jedoch darauf hingewiesen werden, dass bei einer unterschiedlichen Auswahl (Web-Entwickler/in, Software-Entwickler/in, Freelancer/in, große Software Firma, etc.), die Ergebnisse eventuell andere wären. Wie oben bereits erwähnt, kam es zu der Entscheidung mobile App Entwickler zu interviewen, durch die Überlegung, dass Smartphone oder Tablet Apps unser Leben schon so maßgeblich und auf viele verschiedene Weisen beeinflussen, dass der Einfluss von gesellschaftlichen Werten hier

wohl noch am leichtesten auszumachen sei. Rückblickend auf die Gespräche in den Interviews scheint es so, als ob diese Gruppe von Programmierer/innen ein ganz eigener Typ sein könnte. Einerseits war in den Interviews immer wieder von anderen Programmierer/innen die Rede, welche nur stur ihre Aufgaben erledigen und eine Tätigkeit nach der Anderen abarbeiten, die ansonsten aber wenig mit der Produktentwicklung zu tun haben wollen und auch mit dem Rest des Teams wenig interagieren. Andererseits könnte allein schon das Ziel von mobile Apps, als tägliche Begleiter eines Menschen zu fungieren und die daraus resultierende engere Zusammenarbeit mit Nutzer/innen in den Feedbackschleifen, implizieren, dass mobile App Programmierer/innen tendenziell eher Charakterzüge aufweisen, die als offen, umsichtig und kontaktbereit bezeichnet werden können. Im Gegensatz dazu könnte vermutet werden, dass Programmierer/innen, die in einer großen Softwarefirma arbeiten diese Charakterzüge weniger häufig aufweisen als ihre App Entwicklungskolleg/innen.

Es wurden also Interviews mit zwölf Programmierer/innen aus sechs Wiener Start-ups geführt. Wie erwähnt stammt die Hälfte der Interviewpersonen aus einer Firma, zwei weitere Interviewpersonen stammen auch aus einer Firma und alle weiteren Interviewkandidat/innen kamen aus unterschiedlichen Start-ups. Zwei der interviewten Personen waren Frauen, der Rest Männer. Das sind ca. 16 % Frauenanteil, was sich zufälligerweise sogar dem Prozentsatz für ganz Österreich annähert. Laut einem Artikel der Zeit Österreich vom Frühjahr 2019, sind nur 15,6 % der IT-Fachkräfte in Österreich weiblich (Innerhofer 2019). Ein Mann war mit 59 Jahren ein etwas älteres Semester, der sogar noch mit Fortran und den dazugehörigen Lochkarten programmieren lernte. Alle weiteren Interviewpersonen können zur Alterskohorte mitte zwanzig (24 Jahre) bis ca. mitte dreißig (37 Jahre) gezählt werden.

Bis auf drei Personen, wurden alle in Österreich geboren, wobei die Interviewpartner aus der Slowakei und aus Serbien schon seit Kindestagen in Österreich bzw. Deutschland lebten. Die dritte Person stammt aus Rumänien und ist für ihren derzeitigen Job als Entwicklerin nach Wien gezogen.

Acht der zwölf Personen programmieren für Android Apps, drei für iOS Apps und ein Interviewkandidat programmiert mit Xamarin, das zur Cross-Plattform-App Entwicklung genutzt wird. Das heißt, damit können sowohl Android als auch iOS Apps entwickelt werden. Jener Interviewkandidat hat am frühesten, nämlich ab acht Jahren, zu programmieren begonnen. Alle weiteren lernten im Alter zwischen elf und zwanzig Jahren zu programmieren.

Die Firmen, in denen die Programmierer/innen arbeiten haben folgende Themenbereiche:

Jobvermittlung, Bitcoinwährung, Lern-App, Computervision (digitales Auslesen von Bildern, Fotos, PDFs usw.), App-Entwicklungs-Agentur, Gesundheits-App für Typ 1 Diabetiker/innen. Zu genau sollen die Firmen auch nicht beschrieben werde, da die Pseudonymisierung der Interviewpersonen nicht gefährdet werden soll.

6.3 Feldzugang

Der Feldzugang gestaltete sich etwas schwieriger als erwartet. Ursprünglich war geplant eine Anfrage in einer Facebook-Gruppe zu stellen und mit dem Schneeballverfahren weitere Interviewpartner/innen zu finden. Bei dieser Gruppe geht es darum, sich über das Lernen von Programmiersprachen auszutauschen. Ich habe vor einiger Zeit ein Online-Lerntool zum Lernen von Programmiersprachen ausprobiert und mit der Registrierung für das Tool geht die Aufnahme in die Facebook-Gruppe einher. Weil das Tool nicht nur Anfänger/innen nutzen, sondern auch erfahrenere Programmierer/innen sich damit weiterbilden können, erschien der Einstieg über diese Gruppe dennoch als passend. Nachdem sich auf eine öffentliche Anfrage niemand meldete, wurde einem der Leiter der Gruppe die Lage geschildert, mit der Frage, ob er jemand kenne, die/der sich bereit erklären würde. Nachdem der Kontakt, der von dem Leiter vorgeschlagen wurde, auch nicht antwortete, wurde der Plan geändert.

Bei der Recherche nach Start-ups in Wien wurden dann aus der Liste dieser Webseite: https://angel.co/vienna Start-ups ausgewählt, die eine App entwickeln, welche ich per E-Mail kontaktierte. Von wenigen kam sofort eine Antwort, dass sie sich gerne die Zeit nehmen, die meisten antworteten jedoch mit einer Absage, da sie keine Zeit oder keine Ressourcen hatten (obwohl einige Interesse zeigten).

Eine Interviewkandidatin wurde über einen Arbeitskollegen gefragt und fünf der interviewten Personen konnten direkt in einem Büro gefragt werden, in dem gerade ein Interview geführt wurde. Abgesehen von zwei Interviews (eines in meinem Wohnzimmer und eines in einem Café) fanden alle in den Büroräumlichkeiten der Start-ups statt. Eine genaue Auflistung der einzelnen Interviews befindet sich im Anhang.

Die Interviewpersonen wurden alle darauf hingewiesen, dass das Interview transkribiert wird und dass die Daten zur Person und zur Firma so gut es geht pseudonymisiert werden. Jeder Interviewpartnerin/jedem Interviewpartner wurden neue Namen gegeben nach Buchstaben aus dem griechischen Alphabet. Dahinter steckt keine Bedeutung und auch die Reihenfolge ist nicht korrekt. Die Buchstaben wurden random ausgewählt. Nach den Interviews wurden

Protokolle geschrieben zur Reflexion über das jeweilige Interview.

6.4 Auswertung

Für die Auswertung wurde die Themenanalyse von Froschauer und Lueger eingesetzt, deren Ziel es ist, den Text zu reduzieren und einen Überblick über die einzelnen erwähnten Themen zu schaffen (2003, 2010).

Weil das problemzentrierte Interview in dieser Studie nicht tief in die Deutungs- und Sinnstruktur der Befragten eindringt und somit in der subjektiven Ebene der Interpretation zu verorten ist (ebd.), eignet sich die Themenanalyse sehr gut zur Kombination mit dieser Interviewart.

Die Analyseschritte des angewandten Textreduktionsverfahrens gestalten sich recht einfach, indem als erstes ein Thema als wichtig erkannt und einer Kategorie zugeordnet wird, danach die Charakteristika dieses Themas erfasst werden. Im nächsten Schritt liegt der Fokus auf der Abfolge der Themen und auf Unterschieden innerhalb eines Themas aber zwischen mehreren Interviews. Im letzten Schritt werden die herausgearbeiteten Themen wieder auf die Forschungsfrage rückbezogen (vgl. Froschauer, Lueger 2003).

6.5 Methodenreflexion

Das Führen der Interviews hat im Grunde sehr gut funktioniert, nur würde ich mir für weitere Interviews genau überlegen, ob ich wirklich 2-3 Interviews gleich hintereinander führen möchte, da die Konzentration doch darunter leidet und die Gedanken möglicherweise noch beim vorigen Interview hängen. Andererseits ist es natürlich viel praktischer, mehrere Interviews an einem Nachmittag abhaken zu können und die Mitarbeiter der Firma weniger häufig mit einem Besuch zu stören. In diesem Fall hat es ganz gut funktioniert, aber es ist ein Faktor, dessen man sich bewusst sein sollte.

Jeder Interviewperson wurde vor dem Interview versichert, dass die Informationen vertraulich behandelt und anonymisiert werden. Zudem wurde jedem/jeder erklärt, wie das Interview ca. abläuft und warum das Interview aufgenommen werden soll, damit es zu keinen Missverständnissen kommt. Viele Menschen verstehen unter einem Interview eher eine Situation wie mit einem Reporter für ein kurzes Fernsehinterview, was hier eben nicht der Fall war.

Der Wunsch kleine Fallstudien zu machen und aus jeder Firma mehrere Personen zu interviewen, war leider nicht erfolgreich, da sich nicht immer alle Kolleg/innen bereit erklärten oder Zeit hatten. Da aber die Interviews auch innerhalb einer Firma unterschiedlich ausfielen und eher jedes Interview für sich zu betrachten ist, stellt das hier kein Problem dar.

Vielleicht wäre es anders gelaufen, wenn pro Firma ein Gruppeninterview gemacht worden wäre. Das wäre jedenfalls in zukünftigen Studien eine Überlegung wert.
Im Großen und Ganzen entstand der Eindruck, dass trotz einzelner feiner Unterschiede die Interviews relativ ähnliche Ergebnisse produzierten, weshalb entsprechend der theoretischen Sättigung 12 Interviews ausreichend waren.

7 Ergebnisse

Hier werden nun die Ergebnisse aus den Interviews und der Auswertung dargestellt.
Aus den Transkripten der 12 Interviews wurden im ersten Analyseschritt 20 Kategorien erstellt, von denen einige mehr den Bezug zur Gesellschaft behandeln und andere eher die Programmierarbeit. Natürlich sind alle Aussagen in diesen Kategorien interessant. Für die Beantwortung der Forschungsfragen sind jedoch nicht alle im selben Ausmaß relevant.
Da viele Aspekte aus den Gesprächen miteinander zusammenhängen, können sie in diesem Abschnitt mehrmals erwähnt werden.
Die übergeordnete Forschungsfrage wird zuletzt behandelt, da diese auch in den Interviews gestellt wurde und somit die Interviews am Ende abrundete. Den Anfang machen einige Punkte aus den Interviews, die die bisherige Laufbahn der ProgrammiererInnen beschreiben.

7.1 Vom ersten Kontakt mit Computertechnik bis zum heutigen Tag

Alle interviewten Personen sind im Kindesalter, spätestens aber im jugendlichen Alter das erste Mal mit Computertechnologie in Berührung gekommen. Hauptsächlich weil sich Väter damit beschäftigen und einen ersten Familiencomputer anschafften, oder weil Brüder und Nachbarsjungen schon Zugang zu einem Computer hatten. *“Also angefangen hab ich mit Websites programmieren, wie man es damals nannte, weil ich das bei meinem Vater mitbekommen hab. Der hat für einen Verlag gearbeitet und nebenbei eben auch Websites gemacht und ich fand das sehr faszinierend, und hab zunächst mal mit HTML und CSS und JavaScript angefangen und dann mit Visual Basic mit 15, na 13-14 wahrscheinlich.” (Delta).*
Bei fast allen waren Computerspiele der Einstieg, der das Interesse weckte, abgesehen von Psilon, der in der HTL im Testprogramm während dem Mathematik Unterricht mit Fortran zu programmieren begann, bei dem noch Lochkarten gestanzt wurden. *“Ah angefangen mit Programmieren hats eigentlich schon mit 8, 9 Jahren. Da hab ich in Quick-Basic so kleine Pixel-Spiele programmiert damals in Dos. Ahm sowas wie Pingpong und Space Invaders und ein ASCRPG wo du mit einem Buchstaben herumgehen kannst und Sachen aufsammeln kannst.” (Sigma).*
Der nächste Schritt war es, früher oder später in der Schule einen Schwerpunkt in Richtung Technik zu wählen. Bei manchen war das schon bei der Schulwahl am Wechsel in die Sekundarstufe II der Fall, wenn sie sich für eine HTL entschieden. Andere wählten im Gymnasium Informatik. Der Großteil der Befragten gab an eine HTL besucht zu haben (hauptsächlich Elektrotechnik oder Netzwerktechnik). Vier waren am Gymnasium, eine in einer HBLA und einer in der HAK. Und jene, die im Gymnasium waren, haben sich auch alle

für das Wahlfach Informatik entschieden, in dem ein Teil auch maturierte.
Was meiner Meinung nach auffällig ist, ist die Tatsache, dass alle eine tertiäre Ausbildung gemacht haben, oder zumindest begonnen haben und vorwiegend an Technischen Universitäten, nur vier waren auf der FH und es gab niemand die/der kompletter Queer Einsteiger/in ist und das komplette Wissen über Computertechnologie und Programmieren im Selbststudium erlernt hat. Es haben zwar alle vor dem Studium programmieren gelernt oder begonnen programmieren zu lernen aber dennoch schien es für alle selbstverständlich, sich ihrem Beruf auf der akademischen Schiene zu nähern.
Wie der Weg an der Uni verlaufen ist, ist jedoch sehr unterschiedlich. Beta hat nur begonnen aber nicht abgeschlossen, bei manchen hat es etwas länger gedauert mit Umwegen aber in den meisten Fällen wurden Bachelor und Master gleich hintereinander erledigt.
Ein für alle charakteristischer Aspekt in der formalen aber besonders in der informellen Ausbildung ist das „*learning-by-doing*“. Nicht nur in Kursen in Schule oder Universität lernten die Interviewkandidat/innen programmieren, sondern vor allem beim „Herumtüfteln“ und Ausprobieren. Obwohl alle ein Studium absolvierten oder zumindest begannen, ist *learning-by-doing* für sie deshalb wichtig, weil jede der interviewten Personen, diese Art des Lernens anwendet in unterschiedlichen Situationen aber besonders, weil man sich in diesem “schnelllebigen Business” (Zeta) ständig weiterbilden muss, um *up-to-date* zu bleiben.
Grundsätzlich haben alle Programmieren gelernt indem sie ständig Sachen ausprobiert haben und viele kleine Projekte aus eigenem Interesse und für den eigenen Gebrauch umgesetzt haben. Alpha wollte als neugieriger Jugendlicher einfach wissen, wie Computertechnologie funktioniert und musste sich das eben selbst beibringen, da der Informatik Unterricht im Gymnasium ja nicht vergleichbar sei, mit einer HTL.
Besonders Beta hat davon gesprochen, dass er diese oder jene App oder Webseite programmiert hat, weil an etwas interessiert war und selbst eine Lösung zu etwas finden wollte. *“und daun hob i eigentlich meine gaunzn Projekte, de wos i gmocht hob, hob i imma aus Eigeninteresse gmocht. Des haßt ahm, waun i irgendwie wos benötigt hob, daun hob i gschaut, dass i ma des irgendwie söba programmier” (Beta).*
Delta und Epsilon erwähnten sogar, dass man an der TU am besten schon programmieren kann, wenn man anfängt. Man lernt zwar dort auch Programmieren aber laut Epsilon ist es eher ein Mittel zum Zweck. In vielen Lehrveranstaltungen gehe es mehr um die Konzepte dahinter, wie Effizienz, Funktionalität, Objektorientiertheit und Logik zum Beispiel.
Programmierer/innen sehen sich in der Hinsicht sehr pragmatisch und nutzen die Vielfalt der Programmiersprachen in der Tat als Mittel zum Zweck. Sie wollen ein Ziel erreichen und lernen “*on-the-go*” wie sie dahin kommen, indem sie recherchieren und ausprobieren (früher

Bücher, heute "*googlen",* und meist auch Communities auf *GitHub, Stackoverflow*, etc.). Je nach Ziel, das man erreichen möchte, bieten sich andere Programmiersprachen an, dabei kann es schon sein, dass man immer wieder einmal eine ganz neue Programmiersprache lernt. *"Jeder kann das selber lernen, genauso wie jeder selber malen lernen kann oder Radfahren, was auch immer. Also wir sind nichts spezielles, wir sind ein Beruf, der halt, wenn er auch einem noch Spaß macht, der perfekt ist." (Epsilon).*

Weiterbildung funktioniert also einerseits in der Freizeit, wenn man für sich selbst etwas ausprobieren will, oder wie Lambda erwähnt hat, er wissen will, warum etwas (z.B. eine Sprache) im Moment im Trend ist und gehypt wird und ob das in der Firma auch nützlich sein könnte. *"Hob do eigentlich auf einem letzten Projekt, wo ich noch bei der Firma woa, dos woa in Budapest, i bin do immer mit dem Zug hingefahren und hob einfoch die Zeit im Zug ausgenützt, dass i mir selbst do diese Android Programmierung beigebrocht hob und hob don wie i von der Firma weg bin gsogt, so und des is eigentlich jetzt a guater Zeitpunkt, dos ich sowos, i hob schon die ersten boa Schritte gmocht und hob don einfoch weiter gmocht, mit dieser Android Programmierung" (Psilon)*. Andererseits funktioniert Weiterbildung auch im Studium und in der Arbeit. In der Ausbildungslaufbahn (Schule, Uni), wenn man für Seminarprojekte oder das Bachelor-/Masterprojekt (Delta, Epsilon, Zeta, Lambda, Sigma) eine neue Sprache lernen muss und in der Arbeit, wenn in der Firma nicht ausreichend Personal zur Verfügung steht und man sich um ein neues Projekt kümmern muss. Omega zum Beispiel wurde als Webentwickler eingestellt und es wurde ihm aber bald erlaubt oder aufgetragen die iOS Programmiersprache zu lernen, und mit der App Entwicklung zu beginnen. Er hatte zu dem Zeitpunkt zwar ein bisschen Vorwissen aber nur in sehr geringem Ausmaß, dass er gut ein Jahr lang eine sehr steile Lernkurve machte. *"Jo, es woa scho a hoate Lernkurve. I man waun ma so kloane Kurse mocht mit Hilfe vo Youtube oda sowos, des is jo imma gaunz überschauboa, des is so a kloans Projekt, ma faungt frisch aun. Des besteht aus zwoa, drei Screens vielleicht, ma lernt hoid so a boa Basics oba waun ma daun in de Codebase von uns einigworfn wird, daun is des gaunz, und do glaubt ma daun, ma kaun eh nix, weils afoch a riesen großes gwochsenes Projekt is. " (Omega).*

Der Arbeitseinstieg während der Schule oder nach der universitären Ausbildung ist prinzipiell bei jeder Person unterschiedlich aber es finden sich einige Tendenzen. Alpha gelang er durch ein paar Freelance Jobs in der Web-Entwicklung, danach arbeitete er immer wieder für einige Monate oder wenige Jahre bei unterschiedlichen Firmen bis er den jetzigen Arbeitgeber fand.

Beta hat während der Schule schon kleinere Aufträge in der Web-Entwicklung über Freunde und Bekannte bekommen. Später auch schon für kleinere Unternehmen die Webseiten erstellt und er hat auch bald seine eigenen ersten App Projekte in den App-Store gestellt.

Frau Gamma ist über ein Praktikum nach Wien gekommen und hat nach Abschluss des Studiums dann hier zu arbeiten begonnen, wo sie gleich ihr ganzes erlerntes Wissen aus

dem Studium anwenden konnte. Über Bekanntschaften aus der vorigen Firma ist sie dann bei ihrem jetzigen Arbeitgeber gelandet, wo sie seit ca. eineinhalb Jahren arbeitet. *"Jetzt bin i bei XYZ und bin durch eigentlich zwa Bekanntschaften, die dort gearbeitet haum und a jetzt nu arbeiten, zu eana kuma. Jo, oiso des woa wei irgendwie, ma kennt si hoid eben und soboidst, weil i hob vor da Selbstständigkeit bei ZYX goabeit und is hoid a a große Firma und do kennt ma hoid vü Leid und jo, waunst do moi irgendwie so de Leid kennst, daun jo, vernetzt ma si hoid recht guad scho." (Gamma).*

Delta hatte in der Schule schon Praktika (Webentwicklung) machen müssen und hat auch während dem Studium bei unterschiedlichen Firmen nebenbei gearbeitet.

Epsilon war während dem Studium an der TU beschäftigt als Tutor und am Institut und seine jetzige Firma ist sein erster richtiger Arbeitgeber nach dem Studium. Hier ist er nun seit dreieinhalb Jahren tätig.

Zeta hat, währenddessen er die Masterarbeit geschrieben hat, in Wien begonnen bei einem Start-up zu arbeiten. Nach zwei Jahren bei diesem Start-up hat er einige Zeit als Freelancer gearbeitet und weil ihm die klare Trennung zwischen Arbeit und Freizeit fehlte, gründete er dann mit ehemaligen Kollegen die jetzige Firma. Mittlerweile ist er dort nicht mehr nur für die App-Entwicklung zuständig, sondern vermehrt auch im Projektmanagement tätig, was ihm Spaß macht. *"Des hob i ned so laung gmocht kobt, oiso des woa a halbes Jahr. Woa a interessant, jo, oba im Endeffekt, is mir daun eigentlich liaba gwesn, in an Team zu sein und deswegen hod ma daun eigentlich a de Idee recht gfoin, dass ma gemeinsam a Firma gründen. Oiso prinzipiell woa scho immer da Plan, dass i irgendwos auf eigene Sache machen wollte. Und ahm des woa domois daun eigentlich a so a Verkettung von einigen Umständen, de do zusammengekommen sind und des hod daun afoch guad bast. Jo, genau. Oba es Freelancen an und für sich oiso is a ned schlecht." (Zeta).*

Auch Iota hat nach der Universität als Freelancer gearbeitet, aber weil es ihm zu stressig war, sich ständig nach dem nächsten Job umzusehen bewarb er sich bei einigen Firmen und ist so bei der aktuellen Firma gelandet.

Frau Kappa hat in Rumänien schon bei zwei Firmen gearbeitet neben dem Studium, weil es ihrer Meinung nach, im Curriculum zu wenig Möglichkeiten gab, das gelernte anzuwenden und zu üben. Es war zwar am Anfang schwierig, da sie quasi ins kalte Wasser gestoßen wurde aber sie lernte schnell. In diesen beiden Firmen hatte sie also sehr viele *learning-by-doing* Erfahrungen und musste nichts mehr für die Prüfungen lernen. Durch ihren Wunsch ins Ausland zu gehen, ist sie bei der aktuellen Firma in Wien gelandet.

Auch Lambda machte erste Arbeitserfahrungen während dem Studium in einem Nebenjob bei IBM. Durch das Pflichtpraktikum im Master, das er mit der Masterarbeit verknüpfen konnte, fand er dann seinen aktuellen Arbeitsplatz, wo er schon seit einigen Jahren arbeitet. *"Hab dann aber im Master, für die Masterarbeit wieder ein Praktikum braucht sozusagen, wo man a halbes Jahr oder ein Semester, oder ich weiß nicht mehr, ein paar Monate bei einer Firma is, dort ein Praxisprojekt macht. Was man aber auch, wenn mas gut macht oder wenns gut genug is, oder komplex genug is auch für*

die Masterarbeit verwenden kann als Praxisthema. Das heißt ich wollte das verbinden. Bin nachher zufälligerweise mit einem Freund auf ein Computervision Meetup gangen, das heißt da geht's um ah . Bildverarbeitung und Machine Learning und so Sachen. Und da bin ich halt mitgangen, weil er was in die Richtung in der Masterarbeit schreiben wollt, bei seiner Firma. Und ich hab gsagt, ja wurscht komm i halt mit und da hab ich en Daniel, unsern CTO kennengelernt und der hat ma irgendwie woll sympathisch gwirkt und hat halt auch a coole Präsentation ghabt, da bin ich halt zu ihm gangen, so wie schauts aus bei eich, ich würd gerne Masterarbeit schreiben. Er hat gsagt schreib a mal und so zwei Wochen später war ich dann bei der Firma oder ein Monat." (Lambda).

Sigma hat sich ebenso das Studium durch Arbeit finanziert, indem er Spiele programmiert hat.

Psilon in seinem Nebenjob während dem Studium bei einer Firma Software für das Hotellerie Gewerbe entwickelt, eine Art *Customer-Relationship-Management*.

Omega hat durch einen Sommerjob als Web-Entwickler eine fixe Anstellung gefunden, durch die sein Masterabschluss auf Eis gelegt wurde, weil er nach 6 Jahren einfach zu weit weg von der Materie war. Nach diesem Job hat er für einige Zeit als Freelancer gearbeitet und wurde auf die Stelle in der jetzigen Firma aufmerksam, die er schließlich auch bekam. Sein erstes großes Projekt war sehr anstrengend und frustrierend, also wurde ihm angeboten, er darf mit mobile App Programmierung beginnen und muss dazu eben die Sprache noch genauer lernen.

Natürlich ist jeder Weg in die Arbeitswelt individuell, dennoch zeichnen sich gewisse Gemeinsamkeiten ab, wie das Arbeiten in Nebenjobs oder Praktika wodurch auch einige ihre jetzige Position gefunden haben, oder dass fast alle einmal als Freelancer gearbeitet haben aber nicht besonders lange. Letzteres lässt darauf schließen, dass alle gern in Teams und geregelten Arbeitsverhältnissen arbeiten.

Nachdem hier ein kleines Portrait der Interviewpersonen gezeichnet wurde, sollen nun die Forschungsfragen detailliert beantwortet werden.

7.2 Verbindung zur Gesellschaft aus Sicht der Programmierer/innen

Wenn in den Interviews über die Gesellschaft gesprochen wurde, so wurde diese häufig als hauptsächlich unwissend dargestellt, und daher hilfsbedürftig und von Programmierer/innen abhängig. In manchen Fällen wurde sie auch als uninteressiert, ignorant oder gar abneigend gegenüber Technologie oder den Entwickler/innen dargestellt, meist in Verbindung mit den Geschichten über das Klischeebild (siehe weiter unten) von dieser Berufsgruppe. Da die Gesellschaft zum einen keine Ahnung von Technologie hat, wird sie auch als Akteur gesehen, den man über Technologie unterrichten muss. Zum anderen haben noch nicht viele Leute verstanden, wie wichtig die Userexperience ist, weshalb auch über die strenge

Arbeitsaufteilung geklagt wird in einem Interview. Gesellschaft äußert ihre Wünsche, was sie in der Technologie haben will und die Programmierer/innen sollen umsetzen. Gleichzeitig wird über die Gesellschaft auch so gesprochen, dass sie schön langsam endlich sie zur Einsicht kommt und mehr Anerkennung und Verständnis zeigt. In weiterer Folge wird die Gesellschaft meist mit Nicht-Techniker/innen bzw. Nicht-Programmierer/innen gleichgesetzt. Einzig junge Menschen werden sehr positiv gesehen, da sie mit der aktuellen Technologie wesentlich vertrauter umgehen. Sie werden ja nicht umsonst als "*digital natives*" bezeichnet.

Die Verbindung zur Gesellschaft kann sehr unterschiedlich sein. Zum einen geht aus den Interviews hervor, dass sie in der Einflussnahme der Programmierer/innen auf die Gesellschaft durch die Gestaltung der Technologie, die die Menschen tagtäglich nutzen, besteht. Zum anderen zeigt sie sich in der Arbeit mit Usern und Kunden, was hauptsächlich beim Testen und bei den Feedback Loops mit Usern der Fall ist, aber auch in der Zusammenarbeit mit Teamkolleg/innen oder anderen Firmen und Privatpersonen. Dazu mehr in den nächsten beiden Fragen.

Zu Beginn sollen hier Sichtweisen auf den Beruf Programmierer/in dargestellt werden, die das grundlegende Verhältnis zwischen Gesellschaft und Entwickler/innen aufzeigen. Die interviewten Personen nahmen bei diesem Thema auch immer wieder die Sichtweise der Gesellschaft auf ihren Beruf ein um diese Verbindung zu erklären. Ein wichtiger Aspekt dabei ist das Klischeebild über Entwickler/innen. Es ist allgemein bekannt, dass Programmierer/innen häufig als "*Nerds*", manchmal auch als "Streber" oder "*Geeks*" bezeichnet werden. Die Interviews machten deutlich, dass dieses Klischeebild nach wie vor verbreitet ist, wenn es sich auch in den vergangenen zehn bis fünfzehn Jahren abgeschwächt hat und das Berufsbild Programmierer/in immer mehr Beachtung findet.

Wenn die Interviewpersonen Klischees ansprechen, meinen sie meistens einen Programmierer (betont männlich), der introvertiert ist, fehlende soziale Kompetenzen aufweist und als "*Nerd*" bezeichnet wird. *"A Programmierer, des wird afoch imma in an Topf gworfn. Dass wir so der etwas stärkere Herr mit fettigen Haaren san und mit der Brille do sitzn und dass wir gewisse Freaks san, wos aunfaungs durchaus so woa." (Beta).*

Besonders Delta geht weiter auf dieses Bild ein. Er ist der Meinung, dass der typische "Klischeeprogrammierer" diesen Job oft wegen dem Geld macht und nicht so sehr an den sinnvollen Beitrag für die Gesellschaft denkt, es ihm also egal ist, in welcher Firma er seinen Code schreibt. Er weist auch darauf hin, dass in Österreich dieses Bild seiner Meinung nach noch sehr weit verbreitet ist und deshalb auch die Arbeit häufig getrennt wird in Arbeit mit Technologie, wofür Programmierer/innen zuständig sind, und Arbeit mit Menschen, worum

sich andere zu kümmern haben. Wie in der Literatur angemerkt hängt der Erfolg eines Entwicklungsprozesses häufig von den langanhaltenden Aushandlungsprozessen und der Bereitschaft zur Kooperation zwischen Mitgliedern unterschiedlicher Abteilungen und den Nutzer/innen ab (Vinck 2003). Das scheint im Fall dieses Interviews nicht der Fall zu sein. Der wichtige Aspekt der Usability ist seiner Ansicht nach in Österreich noch nicht so angekommen, dass man verstehen würde, dass Programmierer die Arbeit mit Menschen und die Arbeit mit Technologie verbinden können. *"Ich kenn auch einige Entwickler, die sind menschlich meistens für mich recht unspannend, denens völlig wurscht is, ob sie bei diesem großen Glückspielkonzern arbeiten oder bei einer Telekom Firma oder bei einem Start-up. Das ist denen wurscht, hauptsache das Geld stimmt. Sind halt sehr langweilige Menschen meistens" (Delta).*

Auch Iota schließt sich dieser Meinung an. *"Ich meine, wenn man sich berühmte Serien oder Filme anschaut, dann sind meistens die Hacker oder die Programmierer so ein bisschen Nerdy, das ist glaub ich allgemein in der Gesellschaft so ein bisschen verschrien, dass man automatisch so ein bisschen, irgendwie ein bisschen streberhaft ist vielleicht auch, ja. Und ich denke, dass es hier halt, also das Bild davon eher geprägt ist, vielleicht so durch die Medien und halt Filme, Serien und wo man das halt sieht." (Iota).*

Epsilon, Kappa und Omega weisen darauf hin, dass sie sich nicht so sehen und generell Programmierer/innen als ganz normale, coole, nette Menschen betrachten, die einen normalen Job, wie jeden anderen ausführen. *"Großteil der Studenten ahm halt Männer sind, ahm oftmals nicht ganz so sozial aktiv. Ahm ja, so halt, eher in die Richtung, dass das, es kann auch sein, dass wir vielleicht als ahm wie gesagt, weniger sozial aktiv betrachtet werden oder vielleicht ahm introvertiert vielleicht manche, vielleicht in die Richtung aber ich empfinde mich selber nicht so, wirklich." (Epsilon).*

"Des san glaub i so eher die Klischees, waun ma so von außen an Programmierer siagt, daun denkt ma si, jo genau so schaun de aus, de san oba a olles liabe Leid, oiso des sogt jetzt goa nix aus." (Omega).

Aber die befragten Personen, sehen dass dieses Bild bereits im Umbruch ist und sich zum Besseren wendet. Die meisten sind der Meinung, dass sich der Stellenwert ihres Berufes wesentlich gebessert hat. Die Wahrnehmung darüber hängt jedoch sehr stark von Freundes- und Bekanntenkreisen, dem Land oder auch von Erfahrungen in bisherigen Arbeitsverhältnissen ab. Die meisten der Interviewpersonen sind sich einig, dass das Bild der Programmierer/innen das die Gesellschaft mittlerweile hat, sich ca. im letzten Jahrzehnt zum Positiven gewendet hat, da mit Digitalisierung und dem Smartphone-Boom immer mehr Menschen das Ausmaß an Wichtigkeit von Computertechnologie erkennen, besonders auch im Hinblick auf die Zukunft. Sie meinen, dass vor einigen Jahren das Klischeebild (eher ungepflegt, und geringe soziale Kompetenzen und Kontakte) eines Programmierers (einer Programmiererin) noch mehr auf Wahrheit basierte, heute wird ein/e Programmierer/in eher wie jeder andere Beruf gesehen der von einer bestimmten Menschengruppe (junge, meist männliche, Technik affinen Personen) ausgeführt wird.

Delta, Iota und Lambda sind sich einig, dass dieser Beruf in anderen Ländern schon weit

mehr anerkannt ist (in USA oder Deutschland aber auch in Serbien, wo sie fast auf einer Ebene mit Ärzten stehen). Kappa und Omega meinen, dass es auch auf den Teil der Gesellschaft darauf ankommt, dass jüngere Menschen diesem Beruf einen höheren Stellenwert zuweisen als ältere Personen.

Gamma hat erkannt, dass nach wie vor viele Menschen keine Vorstellung davon haben, was Programmierer/innen machen und welche oder wie viel Arbeit hinter einer App steckt. *"Programmierer/innen machen etwas Lustiges, und dann funktioniert es einfach" (Gamma)*, die Tätigkeit oder die resultierende Technologie wird als magisch empfunden, was in weiterer Folge zu Skepsis, Angst und Missverständnissen über den Arbeitsaufwand führt. Deshalb ist es ihr ein Anliegen, dass Programmieren auch als Schulfach etabliert wird.

Psilon hat die Trendwende am Smartphone-Boom erkannt, denn vor den Smartphones wurden Programmierer/innen eher abschätzig oder gar abwertend betrachtet. Jetzt nachdem fast jede/r ein Smartphone benutzt und somit auch die Apps ist das Ansehen gestiegen, sowie die Wertschätzung, der Respekt und sogar die Bewunderung, obwohl es laut Psilon keinen großen Unterschied gibt zwischen mobile App Programmierer/innen und Software-Programmierer/innen. *"Wenn ma vorher gsogt hätt, sog ma vor dem Aufkommen des Booms von den Smartphones, wos du oabeitest, und daun hätt ich gsogt a i bin a Programmierer. Häd der gsogt aha Programmierer jo ok. Abschätzig vielleicht ein bisschen, abwertend jo. Unterschwellig abwertend oder so, Techi, Informatiker, keine Ahnung, jo. Wenn ma nach dem, also seit dem Handy Boom jetzt, jemandem sogt, wos ma mocht, so: i bin a App Entwickler. Sogn sie OH, KLASS, TOLL, COOL, jo usw. Des hoaßt es hot do schon a bissl einen Shift gegeben jetzt in der Reputation, die man hat jetzt als Programmierer. Obwohl i ois App-Entwickler genauso ein Programmierer von Apps bin, wie ich vorher halt Programmierer von irgendwos anderem woa und jo. Deswegen denk ich, dass es do sicha einen Shift gegeben hat, zum positiven jetzt" (Psilon).*

Durch die ständige Ausbreitung von Computertechnologie, wie den Smartphones, die heute überall zum Einsatz kommt, sieht Zeta auch den Einfluss von Programmierer/innen stark angestiegen. Durch einflussreiche Erfolge wie Facebook etc. wird Software (Computertechnologie im Allgemeinen) immer wichtiger. Heute werden wir laut Zeta schon so stark von Algorithmen beeinflusst oder kontrolliert, was auch die Abhängigkeit immer weiter steigen lässt. Mit großem Einfluss und vor allem starker Nachfrage (überall werden Programmierer/innen gesucht) steigt auch das Gehalt und Ansehen, aber Zeta weist darauf hin, dass er diese Ansicht möglicherweise seiner Entwickler-Bubble zu verdanken hat, in der er sich bewegt. Außerhalb dieser Bubble würde er vielleicht mehr wie Beta oder Delta sprechen, die noch eine etwas negativere Sichtweise haben, wahrscheinlich aufgrund von bestimmten Begegnungen und Erfahrungen, die sie gemacht haben. Sie finden, dass Programmierer/innen nach wie vor eher in eine negative Schublade gesteckt werden und

dass das Interesse und Verständnis noch sehr gering sind. Deshalb ist Delta auch der Meinung, dass Programmierer/innen relativ wenig Mitspracherecht haben, wenn es um Entscheidungen über die App geht, obwohl sie gewisse Dinge viel besser einschätzen könnten und appelliert an seine Kolleg/innen dieses Knowhow besser zu kommunizieren.

"ahm ich denke, dass man als Entwickler auf jeden Fall auch eine Mitverantwortung hat für das, was man tut, das betrifft einerseits Dinge wie Datenschutz und die Sicherheit aber auch ahm irgendwo auch eine Form der Ethik. Also . ahm an der Schnittstelle sonst zur Gesellschaft, wirds derzeit grad wieder besser, weil sich jetzt auch die Allgemeinbevölkerung Gedanken macht über Datenschutz, über Apps, über Dinge wie Facebook, wo Daten im Hintergrund sind und sich interessieren dafür, was passiert mit Social Media, mit Big Data, mit Daten, mit Finanztransaktionen, wo fließt das alles hin. Und als Entwickler, Informatiker, Programmierer, whatever hast du ein Gefühl dafür und weißt um was es geht und kannst den Leuten helfen Dinge einzuschätzen ahm und bist jetzt weniger der eigenbrötlerische Ingenieur so wie vor 30 Jahren." (Delta).

Sigma auf der anderen Seite empfindet den Stellenwert schon ziemlich hoch, da er mehrere Freunde hat, die sich jetzt auch in Richtung dieses Berufes bewegen und diese "Handwerkskunst" erlernen wollen. Eine ähnliche Erfahrung hat Lambda, der in der Unterhaltung mit einem Uber-Fahrer gehört hat, dass Programmierer/in eigentlich DER Beruf ist, den man machen sollte, weil man gutes Gehalt bekommt, gute Arbeitsverhältnisse vorherrschen usw., wenngleich ihm selbst das nicht so aufgefallen ist (wahrscheinlich auch wegen der Entwickler-Bubble).

Zusammengefasst, sehen sie, dass sich der Stellenwert im Großen und Ganzen stark gebessert hat und auch in Zukunft noch weiterhin zunehmen wird. In der Vergangenheit war das Klischeebild noch weiterverbreitet, von den Befragten sieht sich aber niemand so. Sie würden sich eher als normalen Menschen wie jede/r andere bezeichnen. Ihre Ansichten dazu dürften stark mit gemachten Begegnungen und Erfahrungen zusammenhängen.

Aus den Aussagen über den Stellenwert geht also unter anderem hervor, dass der Einfluss auf die Gesellschaft, dessen sich die Programmierer/innen selbst bewusst sind, auch beim Rest der Gesellschaft immer deutlicher wird und der Respekt und die Wertschätzung gegenüber dieser Berufsgruppe seit einiger Zeit zu nimmt.

Das Bewusstsein über ihre eigene Einflussmacht, wird auch dadurch deutlich, wie über andere gesellschaftliche Akteure gesprochen wird. Zum Beispiel wurden in einem Interview die Gründer von Runtastic erwähnt, die zu Frau Gammas Studienzeit einmal Tutoren waren. In einem anderen Interview spricht Zeta davon wie viel Einfluss man als Programmierer/in haben kann und verweist dabei auf Mark Zuckerberg als Beispiel. Die Art und Weise, wie von diesen Personen gesprochen wird, lässt darauf schließen, dass sie als Berühmtheiten dargestellt werden, als Helden vielleicht sogar und da die Interviewkandidatinnen und -kandidaten zur selben Berufsgruppe gehören wird damit unterstrichen, dass sie ebenso die

Menschheit um sich herum beeinflussen können, wenn auch noch nicht im selben Ausmaß wie Runtastic oder Facebook.
Ein weiteres Beispiel zur Verbindung mit der Gesellschaft ist, wie über Familie, Freunde, Nutzer oder die Gesellschaft im Allgemeinen gesprochen wird und wie sie ihren eigenen Berufsstand sehen. Zum einen werden diese Personen natürlich für Tests oder Feedback herangezogen oder sie werden als Ideenquelle betrachtet, wenn es einen Prozess gibt, den die Entwickler/innen vereinfachen wollen. Zum anderen kann man die Aussagen in den Interviews auch so verstehen, dass die Gesellschaft und die Individuen in ihr, als hilfsbedürftige Menschen angesehen werden, denen man mit ihren Problemen helfen muss. Entweder indem ein tatsächliches Problem gelöst, ein Prozess vereinfacht wird oder indem den Personen durch Entertainment Spaß bereitet wird (im Falle von Spielen). Kunden wiederum (sofern sie nicht selbst aus der Branche kommen) muss man als Programmierer/in nicht nur deren Wünsche von den Lippen ablesen, sondern auch ihnen erklären, warum etwas nicht so gemacht werden kann, wie sie sich das vorstellen und wie man es besser machen könnte. Sie als Programmierer/innen könnte man in weiterer Folge als “Zauberer” oder “Heilsbringer/innen” verstehen, da nur sie in der Lage sind, diese Probleme zu lösen und den Menschen ein bequemes Leben zu bescheren.
Ein weiter Grund Entwickler/innen als “Zauberer” oder“ Heilsbringer/innen” zu sehen, wird sichtbar, wenn sie darüber sprechen, dass aus ihrer Sicht Technologie für Nicht-Techniker etwas Magisches sein muss.
Diese Ansicht wurde ein paar Mal erwähnt aber nicht in jedem Interview direkt angesprochen. Vorgekommen ist sie in den Interviews von Alpha, Delta, Zeta und Lambda und wurde immer im Zusammenhang mit der vorhergehenden Kategorie Stellenwert angesprochen. Sie waren alle vier der Meinung, dass sich ein großer Teil der Gesellschaft bisher nicht viel vorstellen konnte oder heute noch nicht kann, unter dem Beruf Programmierer/Entwickler und dass für diese Leute Computertechnologie etwas “Magisches” ist, weil sie nur sehen, was es macht, aber die Funktionsweise und die Arbeit dahinter nicht verstehen. Sie sehen also Computer als eine Black Box, die Input bekommt und Output erzeugt, aber ein Großteil der Gesellschaft weiß nicht, was drinnen vor sich geht. Dieses Unverständnis und Unwissen führt auch immer wieder dazu, dass die Interviewpersonen gebeten werden einen Drucker zu reparieren oder in Microsoft Word etwas zu erklären usw.
Für Alpha selbst war der Computer zu Beginn ein “Zauberkistl”: *“ja aber irgendwie wollt ich halt dann auch wissen, was hinter dem ganzen ist und das Zauberkistl verstehen, weil im Prinzip war das eigentlich für mich eine riesen Blackbox. Ich hab zwar gwusst, wie man damit umgehn kann aber ich wusste jetzt nicht*

wie das Ding funktioniert." (Alpha).

Auch für Delta war es ein Grund für Faszination: *"Also diese Faszination, dass man etwas automatisieren kann, etwas programmieren kann und dann löst sich etwas fast schon magisch [...] Aber natürlich, das ist ein Spezialgebiet, das für alle anderen schwarze Magie ist." (Delta).*

Zeta hat sich in die Situation von Nicht-Programmierer/innen versetzt und sieht das sehr analytisch: *"Oiso i glaub, wenn mas jetzt von sich selber aus betrachtet, daun hod ma hoid in gewisse Sochn an Einblick, de für an aundan oder an Nicht-Entwickler so noch Black Magic ausschauen, oiso keine Ahnung. Dass a App des mocht, wos sie mocht is hoid für oa nicht verständlich waun ma si ned damit beschäftigt und ned woaß wie do die Obläufe sind." (Zeta).*

Die Teamarbeit in der Firma kann hier auch noch angeführt werden, da sie zeigt, wie ein Start Up als gesellschaftlicher Akteur seine Mitarbeiter/innen in das Team integriert und an die Firma bindet. Ein Aspekt, der auch recht deutlich wird in dieser Kategorie ist die gemeinschaftliche Unternehmenskultur bei der sehr stark auf flache Hierarchien, Wohlbefinden am Arbeitsplatz, Transparenz und gute Kommunikation wert gelegt wird.

Bezüglich der Teamarbeit sind fast alle sehr zufrieden und schreiben das der offenen und transparenten Kommunikation in den Teams zu, die sie deshalb auch als essentiell für gute Zusammenarbeit ansehen. In einigen Fällen (Computervision Firma, Medizinproduktfirma) werden sogar täglich kurze Meetings innerhalb des Entwicklungsteams gehalten, in den meisten Fällen aber zumindest ein wöchentliches Meeting, dann aber auch mit anderen Teams. *"Also Kommunikation ist das wichtigste würd ich sagen, wenn viele Leute daran arbeiten. Es gibt mittlerweile sehr ausgefeilte Tools und Techniken, wie man wirklich organisiert an so ein Projekt rangeht. Wir haben zum Beispiel täglich zehn, fünfzehn minütige Treffen, wo alle Developer, die vom Mobile Team sind, zusammen kommen und jeder erzählt was er gemacht hat, was er vor hat zu tun, ob er irgendwelche Probleme hatte und dann tauscht man sich halt aus, was vielleicht die beste Art ist, irgendwas zu lösen oder einfach auch dass andere Leute wissen, falls sie irgendwie von dir abhängen, ob du grade dabei bist, das zu lösen usw. und woran man arbeitet und ich würd sagen, Kommunikation ist extrem wichtig" (Iota).*

Gerade bei diesen Firmen ist auch in den Interviews klar hervorgegangen, wie wichtig die Atmosphäre rund um den Arbeitsplatz und die Arbeitszeit ist. In der Computervision Firma (Epsilon, Iota, Kappa, Lambda, Sigma, Psilon) werden wöchentlich unterschiedliche Aktivitäten durchgeführt, wie ein gemeinsames Training an Dienstag Abenden, ein nepalesisches Mittagessen von einer Köchin, die ins Büro kommt; ein Hugging Day, gemeinsames Frühstücken mit selbstmitgebrachten Lebensmitteln oder ein Kennenlern-Kaffee mit Mitgliedern aus anderen Abteilungen. Bei der Medizinproduktfirma gibt es einen Tischtennistisch, eine Sport-Ecke, eine Kinderbetreuungsecke, eine Dachterrasse, eine große Küche, aber auch kleine Rückzugskabinen, wenn man sich besser konzentrieren will. Bei dieser Firma (Omega) ist auch der Einstellungsprozess stark auf die Persönlichkeit einer Bewerberin/eines Bewerbers fokussiert (neben den IT-Kompetenzen). Dort ist es sehr

wichtig, dass die Person in die Firma passt und das wird in mehreren Bewerbungsgesprächen mit unterschiedlichen Leuten ausgelotet. *“Do gibts daun a Interviews die zielen nur drauf ob, quasi den Cultural fit von ana Person zu eruieren und wie guad de Leid zu uns passen würden.” (Omega).*

Was die Entscheidungsfindung betrifft, wird in den meisten Firmen auf flache Hierarchie und Transparenz wert gelegt. Das heißt, jeder kann seine Meinung offen äußern, ohne gewertet zu werden. Die Beiträge werden gleichwertig behandelt, man diskutiert darüber und niemand ist höhergestellt als der/die andere. Gleichzeitig gibt es trotzdem in den meisten Fällen gewisse Entscheidungsstufen, die sich nach der Position und der Erfahrung richten, wie ich annehme. Wenn es kleinere Entscheidungen sind, was Bugs oder kleinere Änderungen am Code betrifft kann das jeder selber entscheiden. Eine Stufe höher steht der/die Teamleiter/in und darüber der/die CTO oder Geschäftsführer/in. Die Beiträge werden zwar offen, gemeinsam besprochen, aber je nachdem wie groß die Entscheidung ist, wird sie dann von den Programmierer/innen, der Teamleitung oder von CTO/Geschäftsführung getroffen. *“Okay da gibts mehrere (unverständlich) sogn ma moi so, sehr kleine Entscheidungen, wie sollte das auf Android ausschauen, werd ich manchmal involviert und sie fragen okay wir ham jetzt die und die Optionen, was würdest du sagen, dann redn ma in a kleinen Gruppe drüber und entscheid mas. Oder wenns so ist, dass die Leute sagen, das ist so, dann machen sie es meistens auch einfach. Jeder hat über seine Plattform, also jeder hat über seine Plattform quasi die Verantwortung. Also a Plattform is jetzt iOS, Android, Windows, was auch immer, ich sags nur gern dazu. Wenns dann die Sache is, soll mas so oder so machen, damits auf allen Plattformen gleich ist, kommts zu mir und wenns dann wirklich was sehr Allgemeines ist, was vielleicht auch das R&D Team betrifft oder das Tooling Team betrifft oder den Release betrifft, wo da Dani noch immer da Chef von dem Ganzen ist” (Lambda).*

Alpha sieht sich eher in einer ausführenden Position, der nicht viel mitentscheidet aber er schien auch nicht so, als wolle er viel mitentscheiden. Im Gegenzug dazu Delta, der sich wünschen würde, mehr in die Entscheidungsprozesse eingebunden zu werden. Er ist der Meinung, dass der Geschäftsführer der Lern-App Firma zu viel alleine entscheidet, wenn er eigentlich nicht viel Ahnung von User Experience usw. hat.

Lambda weist außerdem darauf hin, dass er es für besser und angenehmer hält in kleineren Teams zu arbeiten. In der Computervision Firma waren sie zuerst ein großes Development Team und nun haben sie sich aufgeteilt in mehrere, kleinere Teams, die sich auf bestimmte Dinge spezialisieren.

Firmenkolleg/innen werden meist als Partner/innen gesehen, für die ein gutes “Miteinander-Verhältnis”, ein guter Zusammenhalt notwendig ist, mit denen man auch Spaß haben und einfach Plaudern kann. Gleichzeitig können sie auch als Tutoren oder “Vitamin B” fungieren. In allen Fällen schien es in den Interviews so, als würde man sich gerne gegenseitig

weiterhelfen.

Über Firmen als gesellschaftliche Akteure allgemein wurde recht unterschiedlich gesprochen. Zum einen wurden Firmen angeprangert, sich zu wenig Gedanken über *corporate social responsibility* zu machen, “Schubladendenken” anzuwenden, und somit auch Aufgabenbereiche zu streng abzutrennen (Programmierer/innen sollen nur programmieren und sich nicht über die Nutzung und Rezeption einer App Gedanken machen) was häufig größere Firmen betrifft. Auf der anderen Seite wurden Firmen so dargestellt, dass sie ein Ort der Chancen und Möglichkeiten sind, wo man gemeinsam daran arbeitet, die richtige Job-Position für ein Teammitglied zu finden. Aufgrund des Fachkräftemangels, der in manchen Fällen auch einfach ein Mangel an kompetenten Fachkräften ist, sind manche Firmen dazu gezwungen, sich ihre Fachkräfte selbst großzuziehen, wie es in ein paar der Interviews der Fall war. Diesen Firmen dürfte auch das Wohlbefinden seiner Mitarbeiter/innen ein großes Anliegen sein, wenn sie regelmäßige Teamevents anbieten, wie gemeinsames Essen oder Trainieren etc. Hier könnte man die Firmen bzw. Arbeitgeber sogar als Familie/Freunde bzw. als Entertainer oder sogar Lifestyle Coach verstehen.

Was die direkte wechselseitige Verbindung zur Gesellschaft betrifft, nämlich die Arbeit mit Kund/innen und Nutzer/innen oder der Einfluss der Programmierer/innen auf die Individuen, wird das in den nächsten Fragen eingehender behandelt.

7.3 Der Einfluss der Programmierarbeit auf die Gesellschaft

Aus den Interviews geht hervor, dass die Programmierer/innen sich ihrer gesellschaftsgestalterischen Macht sehr wohl bewusst sind. Am besten zeigt sich wenn sie darüber reden, was sie am Programmieren so begeistert.

Abgesehen von technischen Herausforderungen, vom “Probleme lösen” und dem Vergnügen daran, etwas Schönes, Funktionales, Greifbares zu gestalten, sowie der Zusammenarbeit im Team, ist es für die interviewten Personen von großer Bedeutung, wenn ihr Produkt bei den Nutzer/innen gut ankommt, diese zufrieden sind, es gerne verwenden und gutes Feedback im App-Store oder Google Playstore hinterlassen:

Für Frau Gamma ist es wichtig, dass das Produkt, das sie schafft eine große Reichweite hat. Denn wenn viele Menschen es benutzen, dann hat es ihrer Meinung nach auch Sinn. In ihrer Freelancer-Zeit hat sie eher an kleineren Projekten gearbeitet hat, die nicht so starke Verbreitung fanden und sie deshalb nun großen Wert darauflegt, dass möglichst viele Personen von ihrer Technik profitieren. Eine funktionierende App und zufriedene Nutzer/innen sind schön und gut, aber wenn es sehr viele zufriedene Nutzer/innen gibt, sieht

sie das natürlich schon als eine besondere Belohnung für die eigene Arbeit. Auch die Arbeit in einem größeren Team, wo sie nicht alleine entwickelt ist ein wichtiger Faktor für sie. Sie möchte jedenfalls nicht für die Schublade entwickeln. *“Okay de Leid finden des cool und do gibts vü Menschen des verwenden und i entwickels ned nur für die Schublade, sozusogn.” (Gamma).*

Delta wollte ursprünglich Erfinder werden, daher ist es auch naheliegend, dass er großes Interesse an Nutzerforschung, Usability, *User Experience* und *Userfeedback* hat. Leider fehlt ihm momentan der ideelle Aspekt an der Arbeit. Er sieht einen Unterschied in sinnvollem Programmieren und sinnlosem Programmieren in dem letzteres keinen (sozialen) Nutzen hat für die Gesellschaft.

“...ich hab mir gedacht, man kann Dinge erschaffen und Leute freuen sich dran, oder man kann Dinge erschaffen und die können was bewirken, die können irgendwie was sinnvolles machen, um die Welt zu einem besseren Ort zu machen. [...] weil ich gmerkt hab, du kannst ein Mechaniker sein und das Mechanikerwesen macht dir Spaß, aber es macht an Unterschied, ob du für Steyr Waffen herstellst oder ob du Spielzeug für Ärzte ohne Grenzen herstellst. Die Tätigkeit kann dir Spaß machen aber der Ideelle Zweck ist ein ganz ein anderer.” (Delta)

Was Delta auch Freude macht, ist wenn er das Entwickeln mit einem anderen Interesse kombinieren kann. Zum Beispiel hat er eine App zum Klarinette lernen entwickelt. Dieser Nutzen für die User/die Gesellschaft steht für ihn eigentlich an oberster Stelle, weshalb er sich auch in Richtung Userexperience etc. spezialisieren will.

Iota sieht die vielfältigen Möglichkeiten in der Computertechnologie, durch die man sich das Leben erleichtern und vereinfachen kann. Dass man relativ schnell ein fertiges Produkt vorzeigen kann, von dem auch noch Leute profitieren können, das begeistert Iota. Ebenso die Tatsache, dass man als Programmierer/in prägenden Einfluss auf die Zukunft haben kann. *“Ich finde Technologie wird halt die Zukunft stark prägen und ich wäre irgendwie auch dabei, dass ich wenigstens irgendwas dazu beitrage oder irgendwie da mithelfe, das Leben von irgendjemand zu erleichtern durch mein Programmieren.” (Iota)*. Er meint, dass man gewisse Apps auch als tägliche Begleiter sehen kann und ein besonderes Verhältnis damit entwickelt. Er selbst möchte auch so etwas schaffen und da ist es natürlich etwas Besonderes für ihn, wenn die eigene App viele Downloads hat.

Lambda möchte beim Voranbringen der Technologie und der Industrie dabei sein, bzw. macht es ihn stolz und glücklich dabei sein zu können, denn als *Teamlead* haben seine Entscheidungen bereits Einfluss auf das Produkt. Er ist aber auch der Meinung, dass ihm nicht nur das Programmieren an sich Spaß machen muss, sondern das ganze “Paket” muss passen. Was er macht, soll auch verwendet werden, also einen Nutzen für Leute haben.

Sigma möchte auch, dass die Gesellschaft etwas von seiner Arbeit mitbekommt. Er meint sie muss nicht wissen, was dahintersteckt aber die Menschen sollen sich an seinem Produkt

erfreuen. Natürlich ist auch für ihn das Probleme lösen, die Prozessoptimierung, Arbeitsschritte für andere Firmen oder Individuen besser, lustiger und einfacher zu machen ein wichtiger Faktor seiner Arbeit. Da ist er dann auch stolz darauf, wenn die erbrachten Leistungen, durch viele Nutzerzahlen gewürdigt werden.

Für Omega ist die interaktive Komponente wichtig, denn eine Webseite oder eine App kann von der ganzen Welt angesehen und benützt werden. Das ist etwas Besonderes für Omega. Bei dieser Kategorie erwähnen alle ein paar Faktoren, die sie gemeinsam haben und zwar hauptsächlich das Probleme lösen und Dinge vereinfachen; den sozialen/gesellschaftlichen Nutzen die eine App ihrer Meinung nach haben soll, welcher auch durch eine große Reichweite und gutes Feedback oder hohe Downloadzahlen ausgedrückt wird; dass ein Ergebnis/Erfolg rasch sichtbar ist; schöne Code Architektur, sowie vielfältige Möglichkeiten und die Herausforderung ein gut funktionierendes, ansprechendes, schönes Produkt zu erstellen, das eine gute User Experience bietet.

Wie es bei den Erzählungen über Begeisterung schon angeklungen ist, ist es jeder/jedem meiner Befragten ein Anliegen, die Gesellschaft oder Teile der Gesellschaft in irgendeiner Weise zu unterstützen, indem sie mit der App, die sie entwickeln, ein Problem lösen oder eine Situation verbessern, vereinfachen. In den meisten Fällen der Interviews wurde dieser Wunsch ausgedrückt nachdem nach einem Nutzen oder Sinn für die Gesellschaft gefragt wurde.

Beta, Omega und Psilon sehen direkt den Nutzen ihrer App, da Jobvermittlung, eine Medizin-App und eine Sprachlern-App direkt die Menschen ansprechen. Besonders Delta wünscht sich das auch, dass er mit seiner Arbeit einen gemeinnützigen und sozial sinnvollen Beitrag leisten kann. Von seiner derzeitigen Stelle ist er in dieser Hinsicht leider enttäuscht, da wie erwähnt weniger auf User Experience eingegangen wird. *“Ahm ich hab mir die Firma ausgesucht, weil ich dachte es erfüllt mich mehr, weils um einen Bereich geht, der mich interessiert, ums Lernen und Lernplattformen. Leider sind wir aber auch sehr privatwirtschaftlich orientiert und haben auch einen sehr großen Fokus auf Marketing. Und dadurch kommt man gar nicht so sehr dazu, entweder akademisch, wie lernt der Mensch, wie kann ich das erleichtern, zu forschen. Oder andererseits, ich will, dass Leute besser lernen, ich schenke der Welt jetzt was, wir haben was gemeinnütziges, wir machen tolle Lernplattformen, damit Leute leichter lernen, sondern es hat oft so einen privatwirtschaftlichen Aspekt.” (Delta).*

Bei der Firma die das *Software Development Kit (SDK[11])* zum Bilder Scannen und auslesen entwickelt, sehen die Interviewpersonen das ebenso. Auch sie wollen, dass ihr Produkt den Menschen einen Nutzen bringt, aber wie Kappa richtig beobachtet hat, ist der Nutzen im

[11] Ein Software Development Kit ist im Grunde ein vollständig funktionsfähiges Feature für ein Programm oder eine App, das relativ einfach in die eigene App integriert werden kann. Im Fall der oben genannten Firma, wird an die Programmierer/innen anderer Firmen, die das SDK integrieren sollen, auch eine Anleitung mitgeliefert, wie das gemacht werden kann.

Fall des SDKs eher nur auf einen Teil der Gesellschaft beschränkt, zumindest im Moment noch. Je mehr sich diese Technologie entwickelt, desto weiter wird es in Zukunft wahrscheinlich verbreiten. Da sie bei dieser Firma angestellt ist, ist nun für sie die Möglichkeit großen Einfluss auf die Gesellschaft zu nehmen auf einen Teil dieser beschränkt, aber sie ist der Meinung, wenn man privat eine App entwickeln will, steht einem die Welt offen.

Auch Epsilon merkt an, dass seine Arbeit jetzt nicht direkt Einfluss nimmt auf die Gesellschaft, aber dass es das Ziel der Firma ist, einen Prozess zu vereinfachen und dass er sich damit identifiziert. *"Ich identifiziere mich mit dem, was die Firma macht und die Firma versucht halt eher positiven Einfluss auf die Gesellschaft zu haben durch Optimierung. Und das find ich gut." (Epsilon).*

Gamma und Zeta sind der Meinung, dass der Sinn oder der Nutzen einer App nicht immer direkt sichtbar ist, aber man wird merken, dass sie für andere einen Sinn und Nutzen hat, wenn sie von vielen Menschen benutzt wird und diese zufrieden damit sind. Das trifft zum Beispiel auf Spiele zu. Ein Spiel wird jetzt nicht ein Problem in der Gesellschaft lösen, sondern es hat den Zweck Spaß zu bereiten und wenn es von vielen Menschen gespielt wird, kann man annehmen, dass es einen Sinn hat[12].

Für Iota ist es quasi ein Traum an der Technologie der Zukunft mitarbeiten zu können, was ihm bei der Firma auch zu einem gewissen Grad erfüllt wird. Er will Technologie voranbringen und so das Leben der Menschheit positiv beeinflussen, indem die App hilft Zeit zu sparen und die Lebensqualität zu verbessern. *"Dass ich irgendwie ein Produkt liefere, dass Leuten hilft zum Beispiel Zeit zu sparen, ihre Lebensqualität zu verbessern. Zum Beispiel das was wir bei [Firma xy] machen finde ich auch deswegen sehr spannend, weil ich denke, das ist Technologie der Zukunft und es macht Spaß wirklich hier zu arbeiten und äh irgendwie." (Iota).*

Delta und Omega erwähnen auch, dass es für ein erfüllendes Erfolgserlebnis ist, wenn sie gutes *Userfeedback* bekommen und merken, dass die Leute glücklich sind mit der App, sie also etwas nützliches, hilfreiches geschaffen haben.

In diesen Aussagen sieht man ganz deutlich, dass die Programmierer/innen ihr Umfeld mitgestalten wollen mit ihren Produkten. Sie wollen also gezielt Einfluss nehmen auf die Gesellschaft.

Die Ziele, die sich jeder für seine Arbeit setzt, können meist in Verbindung mit den Kategorien Arbeitsweise und Code auftreten, weil ein großes Ziel ist, die Aufgaben nach bestem Wissen und Gewissen gut und schnell zu erledigen und am Ende ein gutes, zufriedenstellendes Produkt zu liefern. Mit einem guten Produkt ist gemeint, ein schöner, cleaner Code, also gute Codequalität, Erweiterbarkeit, Lesbarkeit, schönes Design,

[12] Anmerkung der Autorin: Natürlich gibt es viele unterschiedliche Arten von Spielen, die man differenziert betrachten soll.

eleganter, effizienter, optimierter Code, der außerdem skalierbar und integrierbar ist. Modular, testbar und wartbar sollte er auch sein. Wenn der Code all diese Dinge erfüllt, dann müsste er laut meinen Interviewpersonen auch die Kund/innen zufriedenstellen und viele Nutzer/innen anziehen. Das ist ein wichtiger Punkt um viele Nutzer zu finden, ein anderer ist es, wie Zeta anmerkt, ob die Aufgabenstellung bzw. die App überhaupt Sinn macht also ein Problem löst oder einen Prozess vereinfacht, optimiert etc. Man muss sich also überlegen, ob es die Arbeit wirklich wert ist, für einen Kunden eine App zu entwickeln, wenn die Idee dahinter nicht viel Sinn macht, denn dann wird sie auch am Markt nicht gut ankommen.

In weiterer Folge ist es auch ihr Ziel durch diese App einen Betrag für die Gesellschaft zu leisten, wie Beta, Zeta, Iota, Lambda, Psilon und Omega erwähnen.

Dazu gehört auch, dass Zeta manche potentiellen Aufträge ablehnt, weil er der Meinung ist, dass es keinen Sinn macht diese oder jene Idee umzusetzen, weil sie keinen besonderen Nutzen hat oder nicht ausreichend User anziehen wird. *"do muaß ma si daun auf jedn Foi mehr Fragen stellen, wos ma jetzt daun im Endeffekt damit erreichn will. Ahm wozu ma jetzt wos mocht, wobei des a a so is, dass wir des daun, oda diese Frage eigentlich a aun unsere Kunden oft stellen, weil sehr oft glaub i afoch Produkte entwickeltn werden, die ned imma an Sinn mochn" (Zeta).*

Oder, dass Omega überhaupt verweigern würde in gewissen Industriezweigen zu arbeiten, wie Glückspiele oder im Bankensektor etc., das Medizinprodukt, das er in der Firma entwickelt entspricht genau seinen langfristigen Zielen, Menschen mit ihrer Gesundheit zu helfen. *"Ahm des woa wie i domois mitn Freelancen quasi aufhean woit, hob i recherchiert, wos Jobs gibt und do woa für mi gaunz klor, dass es quasi gewisse Wirtschaftszweige gibt, wo i auf koan Foi oabeiten mecht. Glücksspiel, eher a ned bei ana Bank oder bei ana Verischerung oda sowos." (Omega).*

Genauso wie Psilon, der mit seiner privat entwickelten Lern-app englisch-, türkisch-, oder arabisch sprachigen Migranten helfen will, Deutsch zu lernen und somit sich besser zu integrieren.

Auch Iota hat privat eine App entwickelt, bei der sich Leute auf einer Weltkarte ansehen können, wo sie schon waren und sich mit anderen Reiseenthusiasten austauschen können. Er wollte eine App erschaffen, zu der die User eine gewisse Beziehung aufbauen, so wie er es bei manchen seiner Apps hat. Seine Reise-App soll Leute weiterbringen, sie glücklich machen oder sie einfach interessieren. Weitere Ziele sind natürlich Spaß an der Arbeit zu haben, etwas Innovatives zu entwickeln und auch etwas Neues zu lernen, also Weiterbildung und sich selbst verbessern, wie Delta, Epsilon, Lambda und Sigma anmerken.

Um die Unterfrage, wie hoch Programmierer/innen ihren Einfluss auf die Gesellschaft einschätzen, zu beantworten: Die Interview Kandidaten und Kandidatinnen schätzen ihren

Einfluss als hoch ein, da sie sogar das Ziel verfolgen die Gesellschaft mit ihren Produkten positiv zu beeinflussen und einen sinnvollen Beitrag zu leisten. Sei es auch nur indem sie die Nutzer durch ein gut funktionierendes, optisch ansprechendes Produkt zufriedenstellen oder sogar dabei helfen eine chronische Krankheit in den Griff zu bekommen, eine neue Sprache zu lernen, sich mit anderen reiselustigen Mensch zu vernetzen oder einen neuen Job zu finden.

7.4 Einfluss der Gesellschaft auf Programmierarbeit

Die Interviewpersonen würden den Einfluss der Gesellschaft auf ihre Arbeit als hoch einstufen, da jede der Firmen aus denen die Interviewkandidat/innen stammen, eine oder mehrere unterschiedliche Methoden anwendet um zu erfahren, was “Andere”, also nicht Programmierer/innen oder zumindest Leute außerhalb der Firma über das Produkt oder gar nur eine Idee für ein Produkt denken. Es sind diverse unterschiedliche Aspekte, die den Einfluss ausmachen.

Aus den Interviews ist hervorgegangen, dass die zwei größten Einflussfaktoren auf den Entwicklungsprozess Zeit und User-/Kundenwünsche sind. Beta erklärt das damit, dass Programmierer/innen häufig schlecht darin sind, Zeit realistisch einschätzen. Es gibt einen Zeitplan, der eingehalten werden soll, aber oft ergeben sich Probleme, die behoben werden müssen und sehr zeitraubend sind. *“Zeit is a gaunz a große Gschicht, ah meiner Meinung noch Entwickler san allgemein sehr schlecht im Schätzen” (Beta).*

Eine weitere Schwierigkeit ergibt sich, wenn Vorgesetzte oder Auftraggeber/innen ganz außergewöhnliche Sonderwünsche haben und noch dazu vom Arbeitsaufwand nicht viel wissen und davon ausgehen, dass gewisse Aufgaben gleich erledigt werden können, obwohl dem nicht so ist. Das ist eines der häufigsten Probleme, wo es dann wichtig ist, dass jemand den Kund/innen verständlich erklären kann, warum etwas nicht möglich ist und weiter verhandeln kann.

Beta hat auch erfahren, wie es ist, wenn man so viel arbeitet, dass man kaum noch Freizeit hat und die Familie, die Freundin das “ausbaden” müssen, was sich umgekehrt auch auf die Qualität der eigenen Arbeit auswirken kann. Ein weiterer Punkt, den er anspricht, ist die fortschreitende Entwicklung der Technologie, mit der man arbeitet, also die Hardware, Software, Programmiersprachen etc. die einem zur Verfügung stehen.

Zeta erwähnt auch Einschränkungen bei den Ressourcen, weil die meisten Firmen eben unterbesetzt sind und deshalb Aufträge nicht so schnell ausgeführt werden können, wie sich die Kund/innen das wünschen würden. Des Weiteren können auch technische Vorgaben eine Einschränkung darstellen, wenn man zum Beispiel nur gewisse *tools, frameworks* und

libraries verwenden kann oder rechtliche Einschränkungen was zum Beispiel Datenschutz betrifft. Das ist meist dann der Fall, wenn ein Projekt, das seine Agentur begonnen hat, dann an die Entwickler/innen des Kunden weitergegeben wird und die nur bestimmte Programmiersprachen verwenden. Er sieht seine Agentur dann auch häufig in einer Beraterfunktion, weil sie immer wieder den Kunden Empfehlungen abgeben, was Sinn macht und was nicht. *"Daun kauns sein, dass technologische Einschränkungen gibt. Ahm dass ma sogt, okay es deafn nur gewisse Tools, Frameworks, Bibliotheken verwendet werden. Des wär eine Möglichkeit. Eine andere Einschränkung könnte sein, rechtlicher Natur. Zum Beispü oa Punkt is Datenschutzbestimmungen, daun a dass zwoa vielleicht der Kunde gern irgendwos mochn würde, ma ihn oba darauf hinweisen muass, dass des aus datenschutzrechtlichen Gründen ned möglich is. Genau. Jo und natürlich daun auch Einschränkungen von den Plattformbetreibern, jetzt wenn mas speziell für die App Entwicklung sieht, ahm do mochn afoch die Plattformbetreiber gewisse Vorgaben, die Apps erfüllen müssen, die im Store vertrieben werden." (Zeta).*

Auch Designrichtlinien der Plattformen (Google, Apple) müssen berücksichtig werden, wie Psilon anmerkt.

Für Iota ist ganz klar der Austausch mit anderen, erfahreneren Entwickler/innen ein wichtiger Einflussfaktor, der einem weiterhilft. Den Geldfaktor sieht er eher dann problematisch, wenn man, so wie er, privat eine App entwickelt. Bei Psilon war das möglich, weil er länger von seinem Ersparten leben konnte. Für Iota ist es eine Überlegung wert, ob man sich eine/n Designer/in oder eine/n Tester/in leisten kann und anheuern will, oder ob man das alles selber auf die Reihe bringt. *"Vielleicht auch ob man Geld investieren möchte in die App. Weil wenn ja, dann hast du äußere Faktoren, wie einen UI Designer, dann hat man ein professionelles Logo, für das man bezahlt hat. Man kann vielleicht sogar Tester anheuern oder sowas. Es gibt schon vieles, das dann das Endprodukt beeinflusst und ja." (Iota).*

Für Lambda und Sigma ist es eine Frage des Ausbalancierens von User- und Kundenwünschen und Zielen, die sich die Firma selbst gesteckt hat. Nicht alle Wünsche sind realisierbar und sinnvoll. Da sie aber wie ein Start Up noch recht abhängig sind, von zahlenden Kunden, müssen sie immer wieder auch Aufträge annehmen, die gutes Geld einbringen, auch wenn diese Aufträge nicht ganz oben auf der Prioritätenliste stehen. Man muss in diesen Fällen den Nutzen eines potenziellen Features mit dem damit verbundenen Aufwand und dem Ertrag abwägen. Man wird sich mit den eigenen Prioritäten auch an der Marktsituation orientieren. Was ist am Markt stark nachgefragt? *"Also ich glaub, dass Kunden einen sehr großen Einfluss haben, vor allem bei Start ups usw. Wir sind jetzt zwar kein Start-up mehr aber wir sind doch noch sehr abhängig von Kunden die zahlen und ähm ich glaub, dass da ein großer Kunde, der sehr viel zahlt großen Einfluss auf uns hat und welche die hobbymäßig mit uns entwickeln eher weniger außer es ist guter Input. Also wenn sie sich das wünschen und sagen, wir hätten das urgern, das ist aber für uns keine Ahnung ein Monat Aufwand übertrieben gsagt, dann werdn mas wahrscheinlich nicht machen. Aber das Feedback fließt von den meisten immer bei uns ein und es macht auch Sinn, weil wenn die Leute dies*

verwenden sagen, das funktioniert so nicht und wir hättens gern so und es macht Sinn für uns und is auch nicht viel Aufwand oder macht einfach Sinn, weils genug Leute wollen, dann ändern ma das gerne, weils halt unser Produkt besser macht." (Lambda).

Je nach Bereich für den eine App entwickelt wird sind andere Prioritäten zu setzen. Für die Bitcoin-App und die Medizin-App ist Sicherheit eine der obersten Prioritäten. Bei der Arbeit mit Geld oder Gesundheit ist oberste Vorsicht geboten und man muss sich an regulatorische Vorgaben halten.

Omega erwähnt auch, dass die regelmäßigen Meetings viel Zeit wegnehmen, aber er versteht auch, die Notwendigkeit dahinter. Bei seiner Firma kommt hinzu, dass sie vor ein paar Jahren von einem großen Konzern aufgekauft worden ist, der auch immer wieder gewisse Wünsche äußert, aber die lenken die Firma einfach in eine bestimmte Richtung und sind laut Omega nicht so einschränkend oder störend. Generell funktioniert die Arbeit mit dem Konzern nicht so schlecht.

Ein großer Einflusspunkt ist natürlich das Userfeedback, das weiter unten noch einmal aufgegriffen wird. Hier ein erster Einblick in das Thema, welches essentiell ist für die Entwicklung und Wartung einer App, da es ja eines der wichtigsten Ziele ist, die Nutzer zufriedenzustellen und ihnen ein Tool in die Hand zu geben, das ihnen das Leben erleichtert. Dementsprechend haben User eine ganz wichtige, und mehr oder weniger einflussreiche Position im Entwicklungsprozess und es wäre laut Delta unklug die Nutzer nicht miteinzubeziehen. Je nach App und deren Zweck aber auch Zielgruppe, wird das Userfeedback unterschiedlich aufgenommen und eingearbeitet oder gar zurückgewiesen. *"Do kaun er scho uns scho beeinflussen. Es kuman a immer wieder gaunz guade Ideen von Feedbacks aussa, de wos wir daun, waun wir des für guad empfinden a mit einbauen" (Beta).*

Manche der Firmen hatten sogar verschiedene Gruppen von Nutzer/innen. Bei der Jobvermittlungs-App (Alpha, Beta) gibt es einerseits die Firmen, die neue Mitarbeiter/innen suchen und andererseits die Personen, die einen neuen Job suchen. Dementsprechend sind die Wünsche dieser beiden Nutzergruppen unterschiedlich. *"Oiso de zwoa Seiten san extem verschieden. Oiso de User san vom Niveau her, oiso do muast wirklich schaun, ois beschreiben, ois, oiso de i wü jetzt ned sogn, se san dumm oba se san ned schlau, sogn ma so. Und do kummt eher mehr des Feedback, se verstehen wos ned, des geht ned. Und auf da Recruiterseite, auf da Firmenseite is hoid mehr, kummt mehr des Feedback he des wär nu cool, oda he des und des und des und des, oiso des Feedback is scho extrem unterschiedlich von de zwoa Seiten. Oba es is hoid a a aundare Zielgruppe, sog i moi." (Beta).*

Die Lern-App (Delta) richtet sich auch vorrangig an Firmen, die Onlinekurse anbieten und das Lernen in diesen Kursen auch per Smartphone ermöglichen wollen. Sie können also ihre Kursinhalte einfach in die App integrieren und ihren Kursteilnehmer/innen zur Verfügung stellen. Also gibt es auch hier zwei Arten von Nutzer/innen (unterrichtende Institutionen und deren Kursteilnehmer/innen). Das Problem das Delta in seiner Firma sieht ist, dass die

Wünsche der Institutionen zu stark befolgt werden und der Aspekt der User Experience, von dem die Kursteilnehmer/innen profitieren würden, bleibt eher unbeachtet.
Bei der Agentur von Zeta ist das alles etwas anders. Da seine Firma die Apps für andere Firmen entwickelt (welche die App Entwicklung im Grunde einfach auslagern), ist hier der Kontakt mit den Endnutzer/innen der App eher gering. Zeta ist sich dennoch bewusst, dass die *User Experience* ein ganz wesentlicher Bestandteil einer funktionierenden App ist und es ist eines seiner Ziele die Funktionen mit dem Design so zu verbinden, dass eine gute *User Experience* das Ergebnis ist.
Bei der Bildverarbeitungs-App (Computervision Firma - Epsilon, Iota, Kappa, Lambda, Sigma, Psilon) sind die Kunden ebenfalls andere Firmen, die das SDK in ihre eigenen Apps einbauen wollen, welche wiederum von Endnutzern verwendet werden. Ein weiterer Punkt ist hier, die Programmierer/innen der anderen Firmen, die das SDK in deren App integrieren müssen. Auch sie müssen berücksichtigt werden, bei der Entwicklung des SDK um den Integrationsprozess für alle möglichst einfach zu halten.
Lambda erwähnt auch das Feedback, das im App-Store und im Play-Store in den Bewertungen steht, welches im Großen und Ganzen auch immer recht konstruktiv ist. *"Ich glaub, dass da zum Beispiel die Bewertungen von Apps usw. nicht so schlecht sind. Teilweise sind sie einfach nur sehr unnütz, teilweise sind Leute die echt grad an schlechten Tag haben und irgendwas schreiben oder so. Ma muss da a bissl ausbalancieren, was wirklich sinnvolle Information ist und teilweise is einfach nur irgendwer, der sich aufregt über irgendwas oder dich beleidigt oder so. Aber ich glaub, dass quasi im Durchschnitt da teilweise sehr gutes Feedback kommt. Weil Leute sagen okay, richtig coole App aber ich hab das und das nicht gefunden, wo ist das, oder meiner Meinung nach sollt das so funtkionieren, und dann kannst du drüber nachdenken" (Lamda).*
Bei der Medizin-App (Omega) ist es etwas einfacher. Hier müssen sich die Entwickler/innen hauptsächlich Gedanken um die Nutzer/innen machen. Diese sind aber Leute aus allen möglichen Schichten der Gesellschaft, also muss auch die *User Experience* der App so einfach, selbsterklärend und unterstützend gestaltet sein.
Userfeedback ist also immer erwünscht und wird auch in den meisten Fällen eingearbeitet, aber auch bei den Firmen, die Wert darauflegen, *Userfeedback* einzuarbeiten, kommt es manchmal zu speziellen Sonderwünschen der User, die nicht immer sinnvoll sind und die abgewiesen werden müssen. Beziehungsweise wäre der Wunsch nicht so dumm, nur der Aufwand, diesen speziellen Wunsch für wenige Personen umzusetzen rentiert sich nicht und kann daher nicht berücksichtigt werden, wie Beta und Lambda erklären.
Bei Omegas Firma, kommt es auch immer wieder vor, dass Nutzer eingeladen werden für Forschungseinheiten, bei denen getestet wird, wie sie mit der Handhabung der App, mit der Navigation und einzelnen Features umgehen. Daraufhin überlegt man sich ob etwas

geändert werden muss. *“oiso wir haum während ma entwickeln hauma a research phase, wo ma User einladen für Usability tests, wos mit an Prototypen zum Beispü spün kinnan oda für die Navigation haum ma uns a, wie soi i sogn, unser Informationsarchitektur hauma ändern miassn, wir haum davor a Seitenmenü ghobt, wo olle Punkte drin woan. Und daun hauma uns entscheidn miassn, wos san de vier unten und wos is daun in an seperaten Screen. Und do hauma mit User Interviews gmocht um herauszufinden wos san de wichtigsten Sochn für se.” (Omega).*

Nun ist klar geworden, dass die Programmierer/innen den Einfluss einzelner Akteure oder der Gesellschaft sehr ernst nehmen. Ansonsten würde die Meinung von Familie, Freunde, Zielgruppe, Nutzer oder Testpersonen nicht so stark in den Entwicklungsprozess einbezogen. Es ist aber auch zu bedenken, dass die Programmierer/innen in den meisten Fällen (außer sie programmieren für sich selbst) dazu gezwungen sind Feedback einzuholen, da ihre App ansonsten kaum Erfolge einfahren wird und die Entwickler/innen eines ihrer wichtigsten Anliegen verfehlen würden, nämlich den Mitmenschen ein gutes, schönes, funktionierendes Produkt zu bieten.

7.5 Die Zukunft der Verbindungsrolle

Diese Frage ist nicht ganz so einfach direkt zu beantworten. Aber aus den Interviews kann geschlossen werden, dass Entwickler/innen an sich auch in Zukunft weiterhin diese wichtige Position als Verbindungsstück beibehalten werden (für einzelne Individuen, wie meine Interviewpartner/innen kommt es natürlich auf den persönlichen Lebensweg an).

Wie weiter oben schon beschrieben wurde, hat sich das Bild von Entwickler/innen in der Gesellschaft wesentlich gebessert und wird vermutlich auch in Zukunft noch weiter an Anerkennung zulegen, da App-Technologie unser Leben noch weiter durchtränken wird. Mit der weiterhin zunehmenden Digitalisierung und dem bisher anhaltenden Mangel an qualifizierten Fachkräften wird sich dieser Trend fortsetzen. Was durch die Entwicklungen in der Digitalisierung auch zunehmen wird, ist das Ausmaß an Verantwortung, das Programmierer/innen übernehmen (werden) müssen. *“ahm ich denke, dass man als Entwickler auf jeden Fall auch eine Mitverantwortung hat für das, was man tut, das betrifft einerseits Dinge wie Datenschutz und die Sicherheit aber auch ahm irgendwo auch eine Form der Ethik. Also. ahm an der Schnittstelle sonst zur Gesellschaft, wirds derzeit grad wieder besser, weil sich jetzt auch die Allgemeinbevölkerung Gedanken macht über Datenschutz, über Apps, über Dinge wie Facebook, wo Daten im Hintergrund sind und sich interessieren dafür, was passiert mit Social Media, mit Big Data, mit Daten, mit Finanztransaktionen, wo fließt das alles hin. Und als Entwickler, Informatiker, Programmierer, whatever hast du ein Gefühl dafür und weißt um was es geht und kannst den Leuten helfen Dinge einzuschätzen ahm und bist jetzt weniger der eigenbrötlerische Ingenieur so wie vor 30 Jahren.” (Delta).*

Auch bei Zeta klingt die höhere Verantwortung in Verbindung mit dem enormen Einfluss an, den Entwickler/innen mittlerweile besitzen. *„Ahm nochdem oba mittlerweile de Technik so*

allgegenwärtig is und eigentlich bei fost ollen Sochn irgendwie Software Entwicklung zum Einsatz kummt und ma damit eigentlich recht coole Sochn mochn kaun und einflussreiche Sochn mochn kaun, is do a der Stellenwert von an Software-Entwickler gesamt gesehen a a bisl angehoben worden würd i moi sogn. […] wenn ma sogt ma schaut si so success-stories au, wia a Mark Zuckerberg mocht afoch Facebook und is daun ana der reichsten Menschen der Wöd, weil er hoid des System entwickelt hod, oiso. Und prinzipiell is afoch a so, dass Software an immer wichtigeren Teil der Gesellschaft einnehmen wird, weil afoch Algorithmen Sochn kontrollieren und beeinflussen und i glaub de Obhängigkeit do afoch a immer größer wird."(Zeta).

Ähnlich wie mit dem Bewusstsein über ihre Verbindungs-Rolle, wird auch die Verantwortung kaum oder nicht direkt angesprochen, dennoch weisen diverse Aussagen darauf hin, dass sie sich der Verantwortung bewusst sind.

7.6 Verbindung zur Technologie aus der Sicht der Programmierer/innen

Das Verhältnis zur Technologie, in diesem Fall mobile Apps, kann einerseits als professionell und andererseits als spielerisch bezeichnet werden. Letzteres ist auch ein wichtiger Aspekt der Begeisterung am Programmieren. Die Erzählungen zu Begeisterung und Faszination an der Programmiertätigkeit gab häufig Einblicke in die Denk- und Handlungsstrukturen der Entwickler/innen.

Für Alpha ist es ganz klar das Lösen von Problemen und das Interesse an der Funktionsweise der Technologie, sowie der Lerneffekt, der sich aus beidem ergibt. *„Das coole ist halt und es is halt ja, . teilweise wie in Spielen, du kriegst halt Aufgaben gstellt, eher kleine Aufgaben und die gilt es halt zu lösen und jemand der gern sein Hirn anstengt, der mag das eben und das Tolle is auch, wenn du dann so ein Problem gelöst hast, dann ist das quasi so ein Aha-Erlebnis, also man bekommt irgendwas zurück, das is das selbe, wie wenn ma jetzt a mathematische Aufgabe löst, freut ma sich ja auch, juhu ich habs gschafft." (Alpha).*

Auch bei Beta steht das Probleme lösen im Vordergrund, dass er mit seiner Lösung einen Prozess oder Irgendetwas vereinfachen kann. Zum Probleme lösen gehört natürlich auch das tüfteln und herumprobieren, bei dem er in der Lage ist unbegrenzte Möglichkeiten zu entdecken und anzuwenden, um zu einer Lösung zu gelangen und eine schöne Web-Architektur zu schaffen, also schöne Technologie nachzubauen. Die visuelle Komponente ist eine Leidenschaft von Beta. In weiterer Folge ist es auch die Genugtuung, dass andere Leute begeistert sind von seinem Produkt und es als schön und cool bezeichnen. Des Weiteren hat es für ihn auch den Reiz, am Ende seiner Arbeit ein sichtbares, "greifbares" Ergebnis vorweisen zu können und mit seinen Apps Geld zu verdienen. Als einziger sprach er auch Interesse am Hacken aus, um bestimmte Barrieren zu umgehen. *„Wos mi a sehr fasziniert hod ah, dieses Hacken zum Beispü wos hoid imma des große, die bösen, die bösen Männer und*

des hod mi a sehr motiviert i sog amoi so. Do einfoch irgendwie wos zu versuchen, zu umgehen und soiche Sochn oiso des hod mi durchaus a in die Richtung getrieben." (Beta).

Für Epsilon ist es wieder die Vereinfachung von Alltäglichem durch Programme und Algorithmen, das ihn so fasziniert. Ein Beispiel wäre ein automatischer Feuchtigkeitsmesser und Bewässerung für Pflanzen. Also Automatisierung, Optimierung, Probleme lösen. Das in Kombination mit seinem Interesse an Computervision ist ideal für ihn. *„Also im Prinzip, jetzt mit ein klein wenig Lernen, könnte ich mir einen Mikrochip einfach besorgen, was ein bisschen anders vom Computer ist, viel viel simpler, aber ich könnte mir einen Mikrochip hernehmen, irgendwas draufprogrammieren und dann hätte ich ein automatisches System, das in einer Pflanze, die Feuchtigkeit von der Erde misst und dann automatisch das bewässert oder sowas und solche Tasks." (Epsilon).*

Bei Zeta ist es einerseits die technische Herausforderung eines Problems und andererseits die vielseitigen Möglichkeiten Dinge zu verbinden. Für ihn sind es mehrere Faktoren, die ein gutes Produkt ausmachen. Zum Beispiel kann man sich die Fragen stellen, ob man a) ein Problem löst und ob man b) gutes Design/gute *User Experience* entwickelt? In diesem Fall sind es Funktionalität und ein schönes User-Interface die ein gutes Produkt ausmachen und diese eigentlich konträren Dinge zu verbinden fasziniert Zeta. *„Oba a so a bisl an ästhetischen Aspekt, oiso afoch, dass ma sogt es schaut afoch cool aus, oda is sche. Genau und des san teilweise gaunz konträre Sochn a, oiso des Funktionelle. Es kaun sei, dass jemand voi guad is in dem Umsetzen vom Funktionellen und afoch fürs User-Interface koa Auge hod und des afoch irgendwie mochn würd und daun würd oba so a Produkt, a ned wirklich erfolgreich sein. Weil nur weils guad funktioniert hoaßt des nu ned, dasd Leid daun a verwenden, sondern es muas a oafoch zu bedienen sei und sche sei." (Zeta).*

Für Frau Kappa bringt Programmieren eine Möglichkeit sich die Zeit mit etwas interessantem zu vertreiben. Sie liebt die technischen Herausforderungen, die ihr das Leben kurzweilig machen. Da sie häufig selbst im *learning-by-doing* Modus herausfinden musste, wie sie Probleme lösen kann, machen ihr solche Herausforderungen besonders Spaß und wenn sie dann eine funktionierende Lösung gefunden hat, ist das ein großes Erfolgsgefühl. Sigma fasziniert eine schöne Code Architektur, Konzepte umsetzen und herauszufinden wie Ideen und Anforderungen so umgesetzt werden können, dass alles funktioniert. Er liebt es, wenn etwas gut läuft und davor die Planung, das Ausprobieren, die Umsetzung. *„Ich bin sehr interessiert am ganzen architektonischen. Egal oba das jetzt Softwarelösungen oder das was wir machen das SDK, is einfach Konzepte umzusetzen oder also auch bei Spielen, einfach zu überlegen wie kann ich etwas von Gedanken oder von der Anforderung so umsetzen, dass es gut funktioniert, dass es in allen Fällen funktioniert, dass es erweiterbar ist und trotzdem schnell läuft und ja ich mag irrsinnig gern einfach das Planen und dann umsetzen und dann seh ich halt wirklich ah cool das funktioniert oder ah das funktioniert nicht, das muss man dann so und so machen." (Sigma)*. Des Weiteren hat er als Xamarin Programmierer den

Vorteil sowohl mit Android ProgrammiererInnen als auch mit iOS ProgrammiererInnen zusammenzuarbeiten und erhält so Einblick in die Eigenheiten beider Plattformen.
Auch Psilon genießt es, dass Erfolg recht schnell sichtbar ist im Vergleich zu seiner vorigen Tätigkeit als Consultant. Ein Prototyp ist schnell gebaut und man erhält auch schnell Feedback. Da Programmieren strukturiertes Arbeiten verlangt, passt das für ihn als strukturierten, ordentlichen Menschen sehr gut. Er vergleicht Programmieren mit dem Aufschreiben eines Kochrezeptes und meint strukturiertes Arbeiten ist im Grunde das Wesen eines Programmierers/einer Programmiererin. *„Programmieren selbst ist eine Tätigkeit, die vergleich i einfoch mit, zum Beispiel mit Kochen auch. Auch beim Kochen hat man ein Rezept, ma hod a bestimmte Abfolge von Schritten, die man einhalten muas, jo. Ahm natürlich is des Programmieren don so, dass man sehr sehr viel Zeit jetzt zum Programmieren aufwendet, dass man Befehle einbaut und bestimmte exceptions oder bestimmte ahm Konditionen, die jetzt nicht im normalen Ablauf drinnen sind, dass man das einbaut. Umgelegt auf das Kochen, wie so a Rezept is, nimm eine Zwiebel, schneid diese Zwiebel auf, gib Öl in die Pfanne, gib die Zwiebel dazu, okay vorher noch, scholt den Herd ein. Sonst nutzts nix. (lacht). Und beim Programmieren is es so, dass man jetzt halt viele Konditionen noch obfrogn muas. Was ist wenn ich keine Zwiebel hob? Oiso des heißt, der Schritt wäre don, schau ob Zwiebeln im Kühlschrank sind, wenn nid, laf obi zum Billa, kaf da ne Zwiebel jo. Und hoid im Grund kummt ma aufs Gleiche auch mitn Öl usw. jo. Ahm dos heißt des Programmieren ist einfach eine Tätigkeit, die auch sehr viel struktur verlangt.“ (Psilon)*. Weiters macht es ihm Spaß, Ideen umzusetzen und dabei viele Komponenten und Anforderungen zu berücksichtigen. Die Herausforderungen, die sich dabei auf vielen Ebenen ergeben sind wichtiger Bestandteil dieses Spaßes. Bei seinem privaten App Projekt versucht er Lernen so herunterzubrechen, dass es für andere zum Erlebnis wird, dabei ist die richtige Mischung an Content Erstellung, Funktionalität und Userexperience sehr wichtig. Die Programmiertätigkeit selber hat für ihn sogar etwas von einem meditativen Charakter.
Für Omega, der als Kind gerne Lego gespielt hat, ist es die Faszination, dass beim Programmieren die Möglichkeiten fast unbegrenzt sind und es keine Einschränkungen wie eine begrenzte Zahl von Bausteinen gibt. Es ist fast alles umsetzbar, wenn man genügend Zeit hat. Er meint, dass das Klischee, Programmierer wollen alles vereinfachen und automatisieren (und deshalb faul sind), das stimmt schon.
Bei dieser Kategorie erwähnen alle ein paar Faktoren, die sie gemeinsam haben und zwar hauptsächlich das Probleme lösen und Dinge vereinfachen; den sozialen/gesellschaftlichen Nutzen die eine App ihrer Meinung nach haben soll, welcher auch durch eine große Reichweite und gutes Feedback oder hohe Downloadzahlen ausgedrückt wird; dass ein Ergebnis/Erfolg rasch sichtbar ist; schöne Code Architektur, sowie vielfältige Möglichkeiten

und die Herausforderung ein gut funktionierendes, ansprechendes, schönes Produkt zu erstellen, das eine gute *User Experience* bietet.

Die professionelle Seite des Verhältnisses zur Technologie wird in Beschreibungen ihrer Arbeitsweise und ihres Codes (weiter unten) erkenntlich.
Beim Thema Arbeitsweise finden sich wieder sehr viele Ähnlichkeiten zwischen den Interviews. Alle wollen ihre Arbeit gut erledigen, arbeiten dazu clean, strukturiert, wiederverwendbar, lesbar, verständlich und wollen eine sauber gecodete, funktionierende Lösung schaffen. Was das Kommentieren[13] betrifft, sind sie sich jedoch uneinig. Zeta ist der Meinung, dass aussagekräftige Bezeichnungen für Klassen etc. im Code häufig ausreichen sollten und Kommentare eher in besonderen Fällen anzuwenden sind. So sieht das auch Sigma, der sagt, Code sollte grundsätzlich auch ohne Kommentare lesbar sein, aber er meint es braucht Kommentare um zu zeigen, wenn und was in einem noch nicht fertig ist und welche To-Dos es für bestimmte Codeabschnitte gibt.
Delta kommentiert erst im Nachhinein, Frau Kappa meint, Kommentare sind sehr wichtig und Omega sagt auch der Code sollte dort und da Kommentare aufweisen, die erklären, was in diesem Codeabschnitt passiert.
Für Epsilon wurde nach seiner Masterarbeit bald klar, dass er nicht noch den PhD machen und in der Wissenschaft bleiben wollte, da ihm dort die Arbeitsweise nicht so sehr gefällt. Er meint in der Wissenschaft wird beim Programmieren nur geprototyped was unsauber und unschön ist (unter anderem weil manche PhD Studierende nicht so gut programmieren können, weil sie aus einer anderen Studienrichtung kommen). Es geht nämlich nur darum, möglichst schnell und einfach die richtigen Ergebnisse zu produzieren, danach wird der Code nicht mehr benötigt. *„Ich war mir nicht sicher ob ich jetzt zum Beispiel ein PhD Studium machen will oder direkt arbeiten aber hab mich eher gegen das PhD Studium entschieden, weil ich eigentlich das Programmieren an sich selber sehr mag, Man während des ganzen Research aber eher Prototyped und nicht schön und clean programmiert, einfach nur um seine Idee und das ganze durchzusetzen und halt zu sagen, ja es ist so, wie ich denk oder nicht und den Code verwirft man dann, das ist dann eher zu unclean.“ (Epsilon).*
Laut Epsilon gibt es aber Bemühungen, den Code für die wissenschaftliche Gemeinschaft bereitzustellen und somit Reproduzierbarkeit zu ermöglichen. In seinem Research &

[13] Ein Kommentar im Code schreiben, das vom Programm nicht ausgelesen wird und daher für Nutzer nicht sichtbar ist. Es wird im Code angezeigt, bleibt aber ohne Funktionen und wird häufig angewendet als Notiz an sich selbst oder an Kollegen, wenn ein Feature noch nicht fertig ist zum Beispiel.

Development Team wird zwar auch geprototyped um für die App etwas Passendes in High Level Code herauszufinden. Danach wird es in Low Level Code in der App umgesetzt.
Zeta weist noch darauf hin, dass es auch nicht immer von Vorteil ist, wenn sehr viele Programmierer an einem Projekt arbeiten, sonst steigt der Anteil an Overhead. Außerdem sollte man auch nicht zu perfektionistisch sein, da man sich sonst in Kleinigkeiten verliert, was in so einem schnelllebigen Business eher unpraktisch ist. *„des hoaßt i bin jetzt koana, der unbedingt super schnell zu ana Lösung kommen will, sondern is wü daun a Lösung hobn, die afoch guad funktioniert und die auch sauber durchgecodet ist. Genau, ahm wobei, oiso früher woa i nu vü mehr so perfektionistisch veranlagt und des is daun eigentlich so a) im Laufe vom Studium und b) in der professionellen Laufbahn a bisl zurückgegangen, weil afoch ned imma in dem Ausmaß notwendig is. Oiso ma, es is a a sehr schnelllebiges Business und oft entwickeln si Lösungen sehr schnell weiter.“ (Zeta).*
Lambda und Omega weisen auf Code reviews und automatische Tests und Reports hin, letztere sollen es für die Zukunft schwieriger machen unabsichtlich Bugs zu schreiben.
Iota denkt auch an den Zeitdruck, der seine Arbeitsweise und die Codequalität beeinflusst.
In Verbindung mit der Arbeitsweise, wird in den Interviews sehr häufig über den eigenen Code gesprochen, aus dem sich eine ganze Liste an Eigenschaften bzw. Anforderungen ergibt. Die wichtigsten Punkte aus dieser Liste sind wiederverwendbar, wartbar, testbar, dokumentiert, kommentiert, schön, strukturiert, performant (= für effizientes und ressourcenschonendes Funktionieren der App), leserlich, sauber, ordentlich, aktuellen Standards entsprechend, erweiterbar, flexibel, modern, defensiv, einfach Bugs zu fixen, low-level Code, sprechende Funktionsnamen (= selbsterklärender Code), fehlerfrei, stabil, um nur einige zu nennen.
Was das kommentieren des Codes betrifft, sind sie sich nicht einige, wie schon bei der Kategorie Arbeitsweise erwähnt. Die einen halten es für sehr wichtig (Alpha, Kappa), die anderen haben dazu schlicht oft keine Zeit (Beta, Iota, Sigma), wieder andere sind der Meinung, dass guter Code auch ohne Kommentare auskommen kann (Zeta, Psilon). Langfristig würde es aber Zeit sparen und jedenfalls nicht schaden sind sie sich einig.
Beta und Gamma erwähnen auch, dass es nicht DEN perfekten Code gibt, sondern, dass es viele Wege zu gutem Code gibt. *„Er [der Code] soi die Idee oder die Funktion fehlerfrei ausführen. Des soi der Code erfüllen. Er soi des, wos si wir dabei denken umsetzten oder ausführen. […] najo eigentlich perfekter Code, an perfekten Code gibts ned.“ (Beta). „Najo, beim Programmieren, gibts jo ned foisch und richtig, du kaunstas auf viele verschiedene Wege mochn und wie strukturierst du dein Code, wie benennst du Variablen und lauter soiche Sochn hoid, jo.“ (Gamma).*
Kappa und Omega sind der Meinung, dass man guten Code ungefähr so flüssig lesen können müsste, wie ein Buch. *„Oba im Grund versuach ma, dass ma unseren Code, oiso des is oana von meine Aunsprüche is quasi, dass er a so gschriebn is, dass man quasi laut lesn kaun und so verstehen.*

Oiso jetzt ned direkt wie an englischen Sotz oba hoid noh draun, oiso dass ma quasi des so lesn kaun, wie okay jetzt passiert des, jetzt passiert des, jetzt passiert des.“ (Omega).

Guter Code sollte auf jeden Fall die Anforderungen erfüllen und es erleichtern Bugs zu finden.

Iota und Sigma fügen noch hinzu, dass es manchmal recht stressig ist und sie dann im Nachhinein den Code nachbessern müssen (*to refactor*). Außerdem sollten laut Sigma nicht Änderungen an einer Stelle Probleme an einer anderen Stelle erzeugen. *„Der Code muss auch so geschrieben sein, dass wenn ma jetzt was ändert, dass nicht was anderes kaputt geht.“ (Sigma).*

Es zeigt sich also, dass die Arbeit an Apps zum einen sehr konkrete Arbeitsweisen verlangt, um der Technologie klar zu machen was sie wann, wo und wie machen soll und aber auch den Kolleg/innen verständlich zu machen, was die Technologie an dieser Stelle macht. Durch den Code wird also nicht nur mit der Technologie selbst kommuniziert, sondern auch mit anderen Programmierer/innen.

Die Idee, den Code wie ein Buch zu lesen versuchen bzw. wie ein Kochrezept, macht durchaus Sinn, wenn man bedenkt, dass Code im Grunde nichts anderes ist als eine Sprache. Damit die App am Ende etwas ausführt, wie man es sich vorstellt, muss man das sehr genau kommunizieren. Wenn ich jemanden um einen Gefallen bitte, muss dies auch klar und deutlich ausgedrückt werden und nicht genuschelt oder zu leise etc.

Über die Verbindung zur Technik kann noch angeführt werden, dass über die Technologiebranche wurde in verschiedenen Tönen gesprochen. Einerseits wurde die *Gamedevelopment* Branche in Österreich als zu unterentwickelt bezeichnet, weshalb es eher schwierig ist, einen guten Job zu finden. Die *Softwaredevelopment* Branche (bzw. die Privatwirtschaft an sich) ist für andere zu wenig ideell und man hat das Gefühl damit in der Gesellschaft nichts Gutes bewegen zu können. Ein weiterer negativer Punkt ist, dass der Markt für Apps schon sehr übersättigt sei, wenn es um bestimmte Arten von Apps geht.

Andererseits wird die Technikindustrie als sehr einflussreich und als Gestalter der Zukunft gepriesen, die im Grunde die Bequemlichkeit der Menschen bedient.

7.7 Der Weg gesellschaftlicher Werte in die Technologie

Diese Frage ist eine der wichtigsten Forschungsfragen für diese Arbeit und hängt sehr stark mit der Frage, wie die Gesellschaft die Arbeit der Entwickler/innen beeinflusst zusammen, da die kontinuierlichen Feedbackloops im Userfeedback und in den Testing-Prozessen einen großen Anteil an der Einarbeitung der Werte in die Technologie hat. Weiters muss aber auch das Arbeitsumfeld und das Umfeld, in dem sich Programmierer/innen in ihrer Freizeit aufhalten, beachtet werden. Letzteres wurde in den Interviews weniger

angesprochen und kann hier deshalb nicht behandelt werden. Außerdem ist es nicht Sinn und Zweck dieser Arbeit, herauszufinden welche Werte ihren Eingang in die Apps finden, sondern wie und an welchen Stellen im Entwicklungsprozess dies nach den Erzählungen der Programmierer/innen möglich ist.

In den oben beantworteten Fragen wurde erörtert, wie die Verbindung zwischen Programmierer/innen und Gesellschaft aussieht. Nun gilt es klarzustellen, wie durch diese Verbindungen gesellschaftliche Werte ihren Weg in das Wirken der Technologie finden.

Die folgenden Punkte sind aus den Interviews und der Beantwortung der vorhergehenden Fragen als “Öffnungsstellen” hervorgegangen, in die Werte aus der Gesellschaft einfließen können.

Der grobe Ablauf wurde von allen relativ ähnlich beschrieben. Die einzelnen Unterschiede im Ablauf ergeben sich aus den diversen Zwecken der Apps. Je nach gesellschaftlichem Bereich, den eine App anspricht (Medizin, Bitcoin, Lern-App, Jobvermittlung, Computervision, etc.), ergeben sich unterschiedliche Anforderungen und so können auch die Entwicklungsabläufe variieren.

Vor der Durcharbeitung der einzelnen Punkte, sollte noch bedacht werden, in welchem Umfeld sich diese Personen bewegen. Zeta hat an einer Stelle angemerkt *“Ahm jo, na weil Entwickler sehr viel mit aundaren Entwicklern a beinaunda san. [...] Genau, oiso wenn jetzt so mein Freundeskreis aus Graz mir aunschau, do san a de meisten Studienkollegen oder Unikollegen. Genau, oiso a Techniker oder in an technischen Bereich.” (Zeta).* Damit ist er vermutlich nicht der einzige. Man könnte also annehmen, dass die jeweilige Persönlichkeit, die persönliche Geschichte und das Umfeld ein eigenes Set an Werten produzieren und für die App Technologie dann eine Art “Wertenährboden” bilden, auf dem unterschiedliche Ideen besser gedeihen als andere, bzw. manche Ideen gar nicht erst auftreten.

1. **Idee durch Problem in Umfeld**

Durch ihren “Problemlösungsdrang” nehmen Programmierer/innen immer wieder Situationen wahr, bei denen sie das Gefühl haben, das müsste besser, schneller, einfacher gehen, besonders wenn es dafür diese oder jene Technologie gibt. Anderen Menschen fallen diese Situationen nicht so markant auf, sie würden einfach weitermachen, wie bisher. Aufgrund der Probleme, die Programmierer/innen wahrnehmen, entspinnen sich dann konkrete Ideen, wie diese Probleme beseitigt werden können. Je nachdem in welchem Umfeld, das Problem aufgetreten ist, wird es unterschiedliche Werte ansprechen, die sich während des Entstehungsprozesses der Idee in dieser manifestieren.

2. **Feedback über Idee**

Ist die Idee soweit fertig gesponnen, wird mit Familie, Freund/innen, Kolleg/innen,

potenzielle Nutzer/innen, etc. darüber gesprochen, ob überhaupt Interesse daran besteht, wie sinnvoll die Idee ist und was man anders machen müsste, damit es sinnvoll wird oder damit Leute die App benutzen wollen würden. *"I hob a scho Ideen kobt, wo i ma docht hob, des is a wödklasse Idee. Und aundare haum daun gsogt, pfff na brauch ma ned, es gibt scho xyz. Und es gibt hoid a, oiso, es kuman a vü Leid zu mir und sogn, he i hob do voi de guade Idee, kaunst du des daun mochn? Und i sog daun so, naaa, i hoid ned vü davo oba loss mi moi drüber nochdenken und daun frog i hoid aundare Leid, wos de davo hoidn und je nochdem wie hoid daun des Feedback von denen is, entscheid i daun ob i des moch oda ned" (Beta).*

Als nächstes sollte man ein Konzept ausarbeiten, das den Grundstein legt und festlegt, welche Funktionen die App haben soll, wie das Design aussehen soll, welche Libraries verwendet werden sollen, für welches Betriebssystem programmiert werden soll, welche User Experience man erzeugen will, überhaupt was die User brauchen, aber auch einen Zeit- und Finanzplan.

3. **Feedback über Skizzen/Mock-ups**

Nach dem ersten Feedback von außen, der Erstellung eines Konzepts und der Planung der Roadmap kann man beginnen für einzelne Screens Skizzen zu zeichnen, sogenannte Mock-ups. Danach sollte man diese wieder mit anderen Leuten, vor allem der potentiellen Zielgruppe, besprechen und die Rückmeldungen einbauen. *"Ich würde empfehlen, diese Screens dann herzunehmen und durchzutesten mit Usern und nach und nach zu verbessern, solange bis völlig klar ist, wie das ausschaut und die User sagen, ja das passt mir gut, mir fehlt vielleicht noch das und das. Dann probiert ma das auch noch mal, ja jetzt find ichs super." (Delta).* Das ist also der zweite Einflusspunkt für Feedback von außen.

4. **technologische Design-/Funktions-/Codestandards als Werte**

Das Feedback über die Mock-ups wird dann bestenfalls in der Entwicklung des Prototypen eingearbeitet. An diesem Punkt könnte man auch die diversen aktuellen Standards, die für das Design, die Funktionen und den Code der App gelten, als "Werteträger" versstehen. Psilon hat an einer Stelle im Interview von den "Material Designrichtlinien" von Google gesprochen, als Beispiel für einen Rahmen in dem man sich bezüglich "gutem Design" bewegen soll. Je nachdem was das "gute Design" in dem Fall leisten soll, an welche Hauptzielgruppe (Allgemeinheit) es sich richtet und wer darin inkludiert ist (unterschiedliche Kulturen, beeinträchtigte Personen, Minderheiten einer Gesellschaft etc.), so tragen auch Designrichtlinien Werte in sich. *„Zum Beispiel hat Google dieses Material Design herausgebracht vor einigen Jahren, dos sind einfoch Designrichtlinien wie man eine Farbgebung macht, wie man Elemente macht, wie die Abstände sind und so weiter ja." (Psilon).*

5. **Feedback über Prototyp**

Nach der Fertigstellung des Prototypen, wird dieser wiederum an Bekannte oder potenzielle

Nutzer/innen usw. gegeben um ein Feedback vom ersten Eindruck einzuholen. Dabei wird über die Optik und die Operabilität der App als Gesamtes und einzelner Features und Funktionen diskutiert und wie sie verbessert werden können. *"Also quasi immer mit Feedbackloops, dass du halt wirklich weißt, obs 'Ding' ist, weil du könntest jetzt ein Jahr investieren, die geilste App überhaupt machen, und keiner kennt sich aus, weil du einfach nur von deiner Perspektive denkst und nicht von wie wirklich Nutzer denken. Weil wir sagen immer Nutzer sind dumm, aber im Endeffekt machst du einfach die UI schlecht, wenn Leute sich nicht auskennen." (Lambda).*

6. Feedback bzw. Wünsche über einzelne Features/Bugs in Beta/Testing

Ist das Feedback über den Prototypen eingearbeitet und ist das vorläufige Ergebnis halbwegs zufriedenstellend, kann man je nach Zielgruppe die App in einem *closed Beta*[14] testen mit einer Gruppe ausgewählter User, oder man stellt sie direkt in den App Store und entwickelt die App nach und nach weiter. Für das Testen allgemein gibt es eine große Vielzahl an Möglichkeiten. Wenn nach den ersten Tests alle oder fast alle Bugs gefixt wurden, kann sie gelauncht werden und dann beginnt die Wartung und Weiterentwicklung. Es werden laufend Support Anfragen kommen, die in Tickets aufgeteilt werden zur Abarbeitung. Außerdem werden Feedback und Nutzer/innen- oder Kund/innenwünsche auftauchen, die es in der Weiterentwicklung zu berücksichtigen gilt. So wird es über die Jahre hinweg weitergehen, außer es gibt einen Grund die Wartung und Weiterentwicklung zu beenden.

Meistens werden zu Beginn der Testphase Unittests durchgeführt, die kleinere, einzelne Features oder Funktionen der gesamten App automatisch testen. Dann gibt es manuelle sogenannte *Monkeytests*, bei denen die Programmierer/innen selber oder Nicht-Programmierer/innen die gesamte App durchtesten indem wild und durcheinander herumgedrückt und gewischt wird, um so Bugs zu entdecken oder Punkte/Situationen an denen die App crasht (crashen = sich aufhängen). Häufig werden dazu automatische Crash- / und Bugreports in die App eingebaut, die den Entwicklern genau anzeigt, wann wo eine App crasht. In der Firma von Alpha und Beta haben sie eine ganze Gruppe von Usern eingeladen um bei einem Frühstücksbuffet die *Monkeytests* durchzuführen, was zu sehr hilfreichen Erkenntnissen geführt hat laut Beta. Da eine andere Person ganz anders mit einer App umgeht als jemand der sie entwickelt. *"wir haum a amoi Leute ins Büro einglond. Ahm wie hod des ghoassn? Frühstückstesting? Usertesting? Do hods a Frühstück gebn, do hods verschiedene Stationen geben und de Leute haum daun einfoch gewisse Situationen durchspün müssn und von uns is imma wer dabei gsessn, der hod des daun genau dokumentiert, wo kummt er ned weida, wo is a Fehler drinnen, wo klicken de Leid wirklich hin und haum daun noch dem Essen, quasi noch dem Frühstück de ganzen Daten*

[14] Eine closed Beta-Phase ist eine Testphase bei der nur einem kleinen Teil der Nutzer/innen die vorläufige App zur Verfügung gestellt wird und sozusagen in einem „abgesicherten, geschlossenen Raum" die App getestet wird.

ausgewertet, und haum hoid daun dieses Produkt, oiso haum versuacht unser Produkt dadurch zu verbessern und do kummt ma hoid wirklich auf Sochn, wos sehr interessant eigentlich is, du mochst des en gaunzn Tog, du woast genau, waun i jetzt do hindruck daun passiert des und des, und waun a Externe des Produkt daun tested daun bist eigentlich sehr verwundert, wos eigentlich ois ned funktioniert oda wie de des probieren, de mochn des gaunz aundas eigentlich ois das du mochst." (Beta).

Bei Gamma war es ähnlich. Hier wurde ebenso einer Gruppe ausgewählter User ein Link zur *closed-Beta Version* gesendet, welche die *Monkeytests* dann aus der Ferne durchgeführt haben. Auch hier gab es sehr gutes *Userfeedback*. Schwierig wird es nur, wenn die Nutzer/innen, die Bugs nicht genau genug beschreiben, dann kann es eine Weile dauern, bis man den jeweiligen Fehler gefunden hat. Dennoch ist es nie möglich alle Fehler zu beheben, da es immer irgendwo eine/n Nutzer/in geben wird, die die App auf einem ganz bestimmten Smartphone mit alter Betriebssystemversion verwendet und dort dann etwas nicht funktioniert. In diesem Fall ist es unrealistisch, dass man sich um dieses Problem kümmert. *"Auf Android host den Spezialfall a nu, dasd hoid tausende von verschiedene Devices host, de olle irgendwie a bissi aundas funktionieren. Jo und daun funktionierts vielleicht nur grod bei dem ned, weil er . aus hundert verschiedenen Gründen auf diesem Device mit dieser Android-Version des und des passiert. Passiert hoid a oft, dasd des niemois reproduzieren kaunst und niemois herrichten, weil unmöglich." (Gamma).*

In der Bildverarbeitungsfirma und in der Medizin-App Firma gibt es auch eigene *Quality Assurance (QA)* Beauftragte, die den Testvorgang planen und kontrollieren, sich also rein auf das Testen konzentrieren können. In beiden Firmen wird auch nach dem Prinzip des agilen Projektmanagements gearbeitet, das nach ca. zwei Wochen Entwicklungsphase ("*Sprint*"), eine Testphase von ca. einer Woche vorsieht, nach der die neue Version der App oder eines Features freigegeben wird. Sie nennen das *Releaseweek*.

Zeta weist darauf hin, dass je nach Ziel und Zweck der App unterschiedliche Vorgehensweisen beim Testen mehr Sinn machen. *"Oiso im Endeffekt gibts zwoa Möglichkeiten a Betaphase zu mochn. Oanseits kau ma sogn, ma stellt des ana geschlossenen Gruppe zur Verfügung und losst des dort durchtesten. A aundare Möglichkeit is, dass ma sogt, ma geht scho moi an die Öffentlichkeit damit oba es is hoid quasi a Beta-Applikation und es is a ois Beta-Applikation gekennzeichnet [...] Oiso prinzipiell würd i sogn, des muas ma eigentlich von Fall zu Fall unterscheiden obs Sinn mocht oder nicht des gleich herauszugeben oder moi nur intern durchzutesten zum Beispü. Ahm vielleicht kummts auch auf die Art der Applikation drauf au, oiso waun ma jetzt sogt keine Ahnung, ma hod zum Beispü a App für a Bank, daun is woascheinlich gaud, waun mas zeast intern guad durchtestet und daun erst rausbringt. Im Vergleich dazu, waun des jetzt nur irgendwos, keine Ahnung, a Dating App is, kaun i woascheinlich a scho moi a public Beta mochn und waun do nu Fehler drin san, daun wirds ned so tragisch sein." (Zeta).*

In der Bildverarbeitungsfirma haben bisher alle gemeinsam die Use Cases (= Beispielfälle, Anwendungsfälle) auf verschiedenen Smartphones durchprobiert haben und dass das immer eher chaotisch war. Deshalb freuen sie sich, dass sie jetzt dezidiert eine bestimmte

Person als QA Beauftragte haben, die den Überblick behält.

Lambda erklärt auch genau, wie bei ihnen Tickets abgearbeitet werden. Dabei gibt es mehrere verschiedene Stadien: a) noch nicht begonnen, b) in progress, c) waiting/input needed, d) ready for testing, e) testing, f) done.

In der Medizin-App Firma wird manchmal auch die Usability durch einen *Codeswitch* getestet, das heißt bei manchen Usern wird auf "Knopfdruck" ein Feature anders dargestellt und es soll so herausgefunden werden, ob die User diese Ansicht oder Funktion angenehmer empfinden. *"Es kummt drauf aun, waun ma wos großes mochn, wir haum zuletzt unser Navigation neich gschriebn und des hauma so gmocht, dass mas hinter an feature switch versteckt haum. Des is quasi a Weiche im Code wo ma sogn kaun siagt der User des oder siagtas ned, und daun haumas für 4.000 Leid in Deitschlaund, Österreich frei gschoitn, damit ma quasi erst amoi an ana kloan Usergruppe seng, keman de damit zrecht, hauma irgendwöche groben Schnitzer drin, de ma beim Testn überseng haum? Ahm und daun hoach ma hoid quasi auf Feedback ausn Support, wir haum de Leid a vorab informiert, dass sie in der Gruppe drin san, damit ses wissen, dass se uns Feedback geben soin, und wauns des daun überstaundn hod, daun release mas quasi für de gaunze Wöd und für neie User. Des moch ma hoid, bei Features wo ma uns ned sicha san, wos für an impact hod des daun wirklich" (Omega).*

Usability Tests sind ein großer und wichtiger Brocken des Testens, durch welche Ideen und Werte der Zielgruppe sehr einfach in die Funktionsweise einer App einfließen können. Wie in der Literatur bei Hertzum (2010) beschrieben, werden sich diese Tests je nach Zielgruppe und Zweck der App unterschiedlich gestalten. Manche werden zum Beispiel mehr Wert auf Universal Usability legen und andere mehr auf *Hedonic Usability*.

7. Feedback in AppStore/PlayStore Bewertungen

Ist die App oder eine erste Version der App soweit fertig gestellt, geht sie im App Store oder im Google Play Store online, wo die App wiederum bewertet und Feedback abgegeben werden kann. Wie von Lambda angemerkt, muss man diese Bewertungen sehr genau filtern, da es immer wieder Personen gibt, die sich online unsachlich verhalten und an schlechten Tagen auch schon mal beleidigend sein können. Die Mehrheit der Nutzer/innen schreibt aber recht konstruktive Kritiken, die wiederum berücksichtigt werden können oder sollen, bei der Weiterentwicklung der App.

8. Feedback/Erweiterungswünsche von Nutzern/Kunden

Wie in den Interviews berichtet wurde, wird die Arbeit an einer App nicht versiegen, wenn sie am Markt gut ankommt. Deshalb wird es im "*Lifecycle*" einer App immer Ideen und Erweiterungs- oder Änderungswünsche von Nutzer/innen und Kund/innen geben, durch welche wiederum bestimmte Werte eingearbeitet werden. Dennoch liegt die Entscheidung, ob spezielle Wünsche angenommen werden, bei der Firma, da nicht alle Ideen und Vorstellungen von Nutzer/innen aus betriebswirtschaftlicher Sicht lohnend sind. *„Ahm also ich*

glaub, dass Kunden einen sehr großen Einfluss haben, vor allem bei Start-ups usw. Wir sind jetzt zwar kein Start-up mehr aber wir sind doch noch sehr abhängig von Kunden, die zahlen und ähm ich glaub, dass da ein großer Kunde, der sehr viel zahlt großen Einfluss auf uns hat und welche die hobbymäßig mit uns entwickeln eher weniger außer es ist guter Input. Also wenn sie sich das wünschen und sagen, wir hätten das urgern, das ist aber für uns keine Ahnung ein Monat Aufwand übertrieben gsagt, dann werdn mas wahrscheinlich nicht machen." Lambda).

9. Weiterentwicklungsentscheidungen des Teams (unter Einfluss der Marktsituation)

Letzten Endes gibt es noch die Möglichkeit, dass das Entwicklungsteam selber mit neuen besonderen Ideen die App umgestaltet auch unter Einbeziehung von Trends, die sich am App-Markt entwickelt haben. So können Werte, die den Markt beeinflussen, sich auch in der App manifestieren. Je nach Firma oder Start-up gestaltet sich die Wartung und Weiterentwicklung unterschiedlich. Für Start-ups scheint es üblich zu sein, iterativ zu arbeiten (das heißt Schritt für Schritt, ein Feature nach dem anderen), dass man also in kurzen Abständen von 2-3 Wochen den Code für eine Änderung oder ein neues Feature schreibt (*Sprint*), darauf folgt eine Woche testen und der Launch des neuen Features. Dieser Ablauf ist typisch für agiles Projektmanagement.

In seltenen Fällen kann es aber auch dazu kommen, dass keine Schritte zur Weiterentwicklung gesetzt werden und die Entwicklung an einer App abgeschlossen wird. Was den Abschluss betrifft, sind sich eigentlich alle einig, dass eine App im Grunde nie komplett fertig ist, weil es immer Verbesserungspotenzial gibt. Das kommt einfach schon daher, dass sich auch die Hardware verbessert und weiterentwickelt. Die Betriebssysteme werden laufend upgedated und somit müssen sich auch Apps immer wieder an die neue Technologie an die neuen Updates anpassen, ansonsten wäre eine App irgendwann nicht mehr benutzbar auf gewissen Smartphones mit bestimmten Betriebssystemen. *„Nie [lacht], jo na oiso im Normalfall sogt ma a Software is eigentlich nie fertig, weil sie kaun eigentlich immer weiter entwickelt werden. Ahm und es is auch so, dass wenn ma a App oder jede Softwäre eigentlich nicht weitentwickelt, is sie irgendwaun amoi nicht mehr wirklich verwendbar, weil si afoch olles rund herum verändert und daun des Ding irgendwaun afoch moi nimma funktionieren wird." (Zeta).*

Ein weiterer Grund sind natürlich Kund/innen- und Nutzer/innenwünsche, die wie schon erwähnt in die Weiterentwicklung miteinbezogen werden müssen. Je mehr zufriedene Nutzer/innen, desto länger wird die App "leben". Manche Programmierer/innen haben von sich aus schon ausreichend neue Ideen für eine App, die sie umsetzen möchten, oder Features, die nicht so gut funktionieren müssen abgeändert werden. *„Never [lacht]. Yes, I say never, because here society is coming and people which are testing or which are implementing your project in their projects, they find problems or new features every time. So we are here to solve every problem and to*

implement new features. So if you have a great product, it will never end." (Kappa).
"Nie, oiso a App is daun fertig waun mas, genau wie bei ana Website. A Website is daun fertig waun mas offline nimmt und bei ana App is des gleiche, es wird imma irgendwos gebn, wos ma verbessern kau und es gibt imma Sochn, Neuerungen bei de Geräte auf die ma reagieren muas und Userwünsche hean a nie auf, oiso genau. Die Entscheidung is nur wie mas priorisiert und konkret daun umsetzt" (Omega).

Gewisse Ausnahmen gibt es aber: Wenn die App grundsätzlich ihre Aufgaben fehlerfrei erfüllt (Epsilon) und man von vornherein keine Wartung geplant hat (Alpha) kann man die App theoretisch als fertig betrachten. In Realität wird jeder seine App, wenn sie gut läuft und nachgefragt wird, warten müssen.

Beta und Zeta erwähnen, die Möglichkeit finanzieller Schwierigkeiten oder direkt Konkurs.

„Oiso i hob a sicha scho einige Apps obgeschlossn. Natürlich is es eher, na oiso ma kaun a App obschließen, jo durchaus. Jo, entweder du bist fertig und host gsogt, du bist mit dem zfriedn, oiso mei Idee is umgsetzt, des woas, mehr gibts ned. Du gehst in Konkurs des is durchaus möglich, oba sunst gibts eigentlich imma wieder irgend a Soch wos ma verbessern, optimieren, wo ma Fehler ausbessern kau, oiso dass a App amoi fertig is, würd i ned sogn. Oiso es kummt imma, aufs Projekt drauf au. Oiso bei uns is imma so, wir haum nu nie gsogt, unser App is fertig, unser Ziel is imma, unser App is fehlerfrei, des wos ma nu nie erreicht haum. [lacht]" (Beta).

Iota ergänzt, dass auch Zeitmangel zum Abschluss führen kann. In seinem Fall, war es aber eher ein Abbruch, da es bei einem seiner privaten Projekte zu starken Erschwernissen gekommen ist und diese aus dem Weg zu räumen einfach zu lange gedauert hätte.

Allein schon die Tatsache, dass ein Produkt immer gewartet und weiterentwickelt werden muss, zeigt dass ihre Rolle als Verbindungsstück auch in Zukunft erhalten bleiben wird. Des Weiteren wird der anhaltende Fachkräftemangel und die wachsende Anerkennung des Berufes in der Gesellschaft sicher dazu beitragen, dass Entwickler/innen auch weiterhin eine zentrale Stellung in der Wirtschaft einnehmen werden. Ein Beispiel dafür wäre der Umgang mit Algorithmen, der in Zukunft reflektierter hinsichtlich möglicher Diskriminierung geregelt werden muss (Berger & Schöggl 2019).

In all diesen Schritten aber besonders durch die Feedbackloops mit Menschen von außerhalb des Start-ups, oder des Entwickler/innenteams, ergeben sich Punkte an denen gesellschaftliche Werte ihren Eingang in die App-Technologie finden.

7.8 „Wie nehmen Programmiererinnen und Programmierer ihre Rolle als Verbindungsstück zwischen Technik und Gesellschaft wahr?"

Diese Frage soll nun mithilfe der Analysekategorie "Forschungsfrage" beantwortet werden. Die erste Reaktion auf die Forschungsfrage war meist Schweigen, da sie überlegen mussten. Sie meinten es sei eine gute Frage und überlegten weiter. Hier sind die Ansichten individuell recht unterschiedlich, im Großen und Ganzen nähern sich die Aussagen aber an. Alpha musste lange überlegen, bis ihm einfiel, dass Informatiker/innen an sich, also jeglicher

EDV Beruf, als Verbindungsstück gesehen werden kann, denn ohne Computertechniker, Hardware sowie Software, *“wäre das Chaos perfekt [...] Natürlich ja. Oiso vor allem auch die EDV, wenn die Kollegen nicht da wäre, wär alles dahin.” (Alpha).*

Beta reagiert im ersten Moment verneinend, er sieht sich selbst nicht als Vermittler von Technologie außer jemand hat Interesse an der Funktion und Beta wird um Erklärung gebeten, dann würde er das machen. Grundsätzlich sieht er eine Trennung von Technologie schaffen und Technologie verwenden. Auf meine Frage, wie er es sieht, wenn man die Lage umdreht und er auf die Wünsche der Gesellschaft eingeht, diese umzusetzen, sieht er sich schon als Vermittler: *“dass wir daun doch die Vermittler san, de wos sogn, he du wüst, he i moch” (Beta)*. Er hat meine Frage so verstanden, als würden Programmierer/innen der Gesellschaft Technologie beibringen müssen (was schwierig ist, wenn die Gesellschaft kein Verständnis und kein Interesse dafür hat). Deshalb seine anfangs negative Reaktion.

Frau Gamma sieht sich generell als Verbindungsstück, da sie durch ihre Teamlead-Position und ihre Projektmanagement Aufgaben nicht nur zwischen Gesellschaft und Technologie vermittelt, sondern auch zwischen ihrem Team und den anderen Abteilungen oder dem Geschäftsführer. Sie sieht es als Teil ihrer Aufgaben, die Technologie, die sie entwickelt auch Nicht-Technikern, wie ihrem Geschäftsführer, zu erklären. Damit diese Leute auch verstehen, welcher Arbeitsaufwand hinter einem Feature steckt und warum etwas länger dauert etc. Das ist, ihrer Ansicht nach, nicht für jeden etwas, und spielt damit auf Teammitglieder an, die sie eher dem Klischeeprogrammierertyp zuordnen würde. Diese wollen nur wissen, wie sie etwas programmieren müssen und das wars. *“I muas sogn, i siach mi hoid persönlich jetzt ned ois den Hardcore Programmierer sondern eher die Person, wei sunst wa i jetzt ned in dieser Teamlead-Funktion, wo i a vü Organisatorische Sochn übernimm und wo i a min Chef drüber reden muas und eam hoid versuche näher zu bringen, wie schaut mei Sicht, mei technische Sicht drauf aus. Oiso i siach mi do scho sehr irgendwie in der Zwischenposition a.” (Gamma).*

Delta ist der Meinung, dass die Gesellschaft eine Trennung zwischen Technologie und der restlichen Welt sieht, was quasi darauf hinausläuft, dass die Gesellschaft sagt, was sie haben will und Programmierer/innen sollen das dann umsetzen. Er selbst sieht sich schon als Verbindungsstück und sieht diese klare Trennung nicht. Userfeedback und die Zufriedenheit der Nutzer sind ihm als Verbindungsstück sehr wichtig. Des Weiteren sieht er generell eine große Verantwortung auf Programmierer/innen lasten, was Datenschutz, Sicherheit sowie den Sinn und Zweck einer App betrifft, weil sie die Risiken einschätzen können und danach handeln müssen. *“Ahm ich denke, dass man als Entwickler auf jeden Fall auch eine Mitverantwortung hat für das, was man tut, das betrifft einerseits Dinge wie Datenschutz und die Sicherheit aber auch ahm irgendwo auch eine Form der Ethik.” (Delta).* Er erkennt auch an, dass das

Klischeebild von Programmierer/innen immer schwächer wird.

Epsilon sieht Programmierer/innen vorrangig als Problemlöser und Optimierer, die den Alltag vereinfachen wollen und ergänzt, dass die meisten wahrscheinlich nicht bewusst darüber nachdenken, sich diese Sicht der Vermittlung und Verantwortung aber zumindest unterbewusst doch irgendwie verankert hat. Er erinnert sich dabei an 2-3 Lehrveranstaltungen im Bachelorstudium die von gesellschaftlichen Grundlagen der Informatik handelten und wiederholt, dass er den Beruf wie jeden anderen Beruf empfindet, bei dem es leider auch schwarze Schafe, nämlich Hacker, gibt. *"Es ist halt schwierig. Ich glaub viele denken gar nicht drüber nach. Also es wird an der Uni auch im Bachelorstudium, vor allem 2 oder 3 Vorlesungen gibt es Richtung so gesellschaftliche Grundlagen der Informatik zum Beispiel oder gesellschaftswissenschaftliche Irgendwas. Und da soll man ja, da lernt man wirklich, dass man eigentlich irgendwas machen soll was der Gesellschaft hilft, mit seiner Arbeit und das wird im Prinzip in den ersten Bachelorsemestern gelehrt an der Uni. Und ich glaub das verankert sich schon irgendwie und oftmals denkt man nicht wirklich drüber nach." (Epsilon).*

Auch Zeta ist der Meinung, dass man sich unterbewusst schon als Verbindungsstück sieht und dass das Wissen über Technologie, das man all die Jahre angesammelt hat, schon so stark internalisiert wurde, dass es selbstverständlich geworden ist und man sich ein Leben ohne dieses Wissen nicht mehr vorstellen könnte. *"Oiso i glaub des geht a stoak in de Richtung, dass des Wissen, des ma darüber hod, eigentlich scho so tief in einem verankert is, oiso waun mas scho so laung mocht, dass ma si goa nimma vorstön kau, wie des is waun ma des ned woas." (Zeta)*. Gleichzeitig ist Technologie schon so ausdifferenziert, dass man trotzdem von vielen Bereichen keine Ahnung hat. Er selbst, sieht sich auch als Verbindungsstück und versucht, wie Gamma, immer wieder die technischen Hintergründe so zu erklären, dass Nicht-Techniker auch verstehen, warum etwas so gemacht wird und nicht anders oder warum etwas nicht funktioniert.

Wie schon erwähnt möchte Iota den Menschen etwas Nützliches, interessantes bieten, das wenn möglich auch noch die Gesellschaft voranbringt. Auch er ist der Meinung, dass viele Kolleg/innen nicht an die Gesellschaft denken, wenn sie programmieren, sondern einfach Spaß im Vordergrund steht. Es ist also nicht immer ein direkter Zusammenhang zur Gesellschaft vorhanden und viele würden den Beruf auch sehen, wie jede/r andere/r. *"... man macht das manchmal um sich selber herauszufordern und es ist nicht immer dieser große Zusammenhang würd ich sagen vorhanden. Auf jeden Fall nicht. (lacht) Ähm ja, also ich denke, dass wenn man sich große Ziele setzt, dann möchte man das auf jeden Fall, wenn man ambitioniert ist usw. Für viele ist es wahrscheinlich auch einfach nur ein Beruf wie jeder andere" (Iota).*

Frau Kappa will auch etwas Nützliches, sinnvolles für die Gesellschaft schaffen oder zumindest für den Teil der Gesellschaft, der das SDK der Firma brauchen kann. Sie fügt noch hinzu, dass wahrscheinlich alle leidenschaftlichen Programmierer/innen ein Produkt

schaffen wollen, das die Leute glücklich macht.

Für Lambda ist es so, dass er die Firma als Schnittstelle zwischen analogen Daten und digitalen Daten sieht und ihr Produkt, das SDK, die Gesellschaft in Richtung Digitalisierung vorantreibt. Sie wollen der Industrie helfen, dass Leute weniger manuell arbeiten müssen und somit die Effizienz gesteigert wird. Programmierer/innen sind seiner Ansicht nach da um Probleme zu lösen und für die Allgemeinheit zu vereinfachen und Dinge einfacher nutzbar zu machen und am Ende Technologie der Gesellschaft zugänglich und verständlich zu machen.

In Sigmas Augen sind Programmierer/innen die Personen mit der Kontrolle über Technologie, sie verwandeln menschliche Inputs in das was Maschinen tun sollen. Er versteht Technologie als menschengemacht, als eine menschliche Sache. *"Naja also ich glaub Programmierer sehen sich als die, die die Kontrolle über die Technologie haben. Die quasi menschliche Inputs verwandeln in das was Maschinen halt tun sollen im Endeffekt [...] Die Technologie wird von Menschen gemacht und bedient, das heißt es ist ultimativ eine menschliche Sache." (Sigma).* Jedes Verhalten oder Versagen ist menschlich, darum hat auch der/die Programmierer/in als Mensch eine gesellschaftliche Verantwortung dafür, dass alles reibungslos funktioniert und nicht missbraucht wird. Menschen wollen sich ihr Leben einfacher und interessanter machen und miteinander kommunizieren, Programmierer/innen setzen das dann für Geld um. Er hat Medizinische Informatik studiert, weil er nach dem Zivildienst etwas Gutes tun wollte und das mit seiner Technikaffinität kombinieren wollte. Sein Ziel war es Menschen zu helfen egal ob im Gesundheitswesen oder anderswo. Er hat zum Beispiel in einem Projekt ein Spiel entwickelt, das es Menschen erleichtern soll, nach einem schweren Unfall oder einer schweren Krankheit bestimmte Fähigkeiten wieder zu erlernen.

Psilon sieht sich insofern als Mittelstück, da er durch die Entwicklung einer Sprachlern-App es Migrant/innen und Flüchtlingen ermöglicht, einfacher Deutsch zu lernen und sich somit besser zu integrieren. Was für die Gesellschaft als auch für die jeweilige Person von Vorteil ist. *"I kauns nur für mich beauntworten. Jo. Wie ich mich ois Programmierer ois Mittelstück zwischen Gesellschaft und Technologie sehe. Okay, durch die Wahl des Themas durch diese App und durch einen starken Fokus auf Usability, glaube ich dass dos für die Gesellschaft wertvoll ist, was ich mache. Einfoch indem ich probiere Minderheiten zu unterstützen um Deutsch zu lernen. " (Psilon).*

Omega verweist darauf, dass Programmierer/innen nur ein kleiner Teil eines größeren Teams sind, ohne die die Arbeit auch nicht funktionieren würde. Das heißt alle tragen dazu bei Technologie den Nicht-Technikern zugänglicher zu machen. Usability ist in diesem Fall extrem wichtig, da die Medizin-App Menschen aus allen Gesellschaftsschichten ansprechen soll und auch für weniger begabte Personen hilfreich sein soll, muss man das in der Technologie natürlich entsprechend berücksichtigen. Oft sind es nämlich genau diese

Personen, die von guter unterstützender Technologie am meisten profitieren. *“Hmmm oiso i glaub du kaunstas aussa lesn, oiso i hoff du hostas aussa kriagt aus meine Auntworten oba im Grund siag is so, dass ma jo oiso da Programmierer aun sich, is jo in unserer Firma ned vü, oiso ohne an Team rundherum funktioniert des jo ned, oba ois quasi eingebettet in an guadn Team trogn ma auf jedn Foi dazua bei, dass ma Technologie für Leid, de a ned technologieaffin san, zugänglicher mochn. Und dafür moch ma eben de Usertests, dafür san ma sehr User fokussiert um des a zu gewährleisten. “ (Omega).*

Wenn sich die interviewten Personen zwar nicht direkt oder zumindest nicht bewusst als Verbindungsstück wahrnehmen, so sehen sie sich doch als solches, wenn sie direkt danach gefragt werden. Jede/r zwar in einem anderen Ausmaß aber subjektive Wahrnehmung ist ja immerhin etwas sehr Individuelles.

Kurz zusammengefasst ist durch die Interviews ist also klar geworden, dass sie sich nicht bewusst als Verbindungsstück zwischen Gesellschaft und Technologie wahrnehmen oder es jedenfalls noch nie selber so ausgesprochen haben. Besonders bei den Themen Begeisterung und Ziele ist jedenfalls hervorgegangen, dass sie sich schon als Verbindungsstück fühlen. Ansonsten würden sie wahrscheinlich nicht ansprechen, dass sie eine sinnvolle, nützliche App für ihre Mitmenschen entwickeln wollen.

8 Diskussion

Um die Forschungsfragen hier noch einmal kurz zusammenfassend zu beantworten, kann gesagt werden, dass die Verbindung zur Gesellschaft hauptsächlich durch die Kommunikation mit Außenstehenden, wie Nutzer/innen, Kund/innen, Familie, Freund/innen oder allgemein Nicht-Techniker ausschlaggebend ist. Zum einen hat sich die Anerkennung der Entwickler/innen seit einiger Zeit gebessert, da die Wichtigkeit von moderner Technologie von einer breiteren Gesellschaftsschicht erkannt wird. Zum anderen findet der Austausch zwischen Gesellschaft und Technologie durch die Programmierer/innen vorrangig in den Feedbackschleifen im Entwicklungsprozess statt, was gleichzeitig den zentralen Einfluss der Gesellschaft auf die Programmierarbeit darstellt und in weiterer Folge den Weg der gesellschaftlichen Werte in die Technologie darstellt. Anders herum sind die Feedbackschleifen in den meisten Fällen ein Erfolgsgarant für das Ziel der Entwickler/innen, mit ihrer Technologie Einfluss auf die Gesellschaft zu nehmen, indem sie Produkte schaffen, die einen Sinn oder einen Nutzen für die User haben.

Dazu kann noch erwähnt werden, dass sich die Art und Weise, wie sie etwas Nützliches zur Gesellschaft beitragen wollen, sehr unterscheiden kann und sich daraus verschiedene Kategorien entwickeln lassen. Von „den Nutzer/innen Freude und Spaß bereiten“, über „(hilfsbedürftige) Menschen im Alltag mit Technik unterstützen“, oder „gleichgesinnte Menschen über Social Media Apps zu verbinden“, sowie „den Usern schöne und reibungslos funktionierende Techniklösungen zu bieten“, bis hin zu „an der Weiterentwicklung der Gesellschaft durch Technologie beteiligt sein“ ergeben sich einige erste Bereiche in der Hinsicht. Dementsprechend könnte diese Kategorisierung auch so weitergeführt werden, dass sich verschiedene Typen von Beziehungen zur Gesellschaft einteilen lassen, die aber nach meinem Verständnis auf einer breiteren Ebene angesiedelt wären. Zwei Beispiele dazu könnten sein, „aktiv Einfluss nehmende Beziehung“ (was auf die oben angeführten „Nützlichkeits-Kategorien“ anwendbar wäre) oder eine „Beziehung mit zurückhaltendem/unbewussten Einfluss“ (wenn es den Programmierer/innen rein ums Ausführen ihrer Aufgaben geht).

Die Verbindung zur Technologie teilt sich auf in die Begeisterung am spielerischen Tüfteln und in die professionelle Arbeitsweise. Der Blick in die Zukunft der eigenen Rolle als Verbindungsstück ist zuversichtlich, da mit der steigenden Digitalisierung auch die Anerkennung des Berufs und die Wichtigkeit verantwortungsbewusster Entwickler/innen weiterhin zunehmen wird. Es wäre jedenfalls ein möglicher Anknüpfungspunkt für weitere Untersuchungen, zu fragen, wie sehr das Verantwortungsbewusstsein für ihre Arbeit eine

Rolle spielt.

Um die Ergebnisse auch mit der Literatur in Verbindung zu setzen, werden hier ein paar Anknüpfungspunkte erläutert.

Obwohl nicht viel Literatur aufzufinden war, die sich spezifisch mit Entwickler/innen beschäftigt, so können die oben angeführten Artikel doch mit gewissen Einzelheiten aufwarten, die sich in den Ergebnissen wiederfinden. Zum Beispiel ist da die Tatsache, dass Computertechnologie und besonders Programmierer/innen noch verhältnismäßig selten in der Literatur vorkommen (Felt et al. 2017; Downey & Lucena 2004), was sich in den Aussagen über die Gesellschaft und deren Wahrnehmung von Entwickler/innen widerspiegelt. Wäre diese Thematik schon in der Gesellschaft angekommen, so wäre sie auch in der Wissenschaft häufiger vertreten. Beziehungsweise hat mit der zunehmenden Anerkennung dieses Berufes auch die Rezeption in der Wissenschaft zugelegt und umgekehrt wahrscheinlich auch.

Ein Verknüpfungspunkt, der ganz klar sichtbar ist, ist die “Magie” der Technik. Für die Programmierer/innen aus den Interviews, sind es die Nicht-Techniker aus der Gesellschaft, die Technik nicht ausreichend verstehen und sie deshalb als etwas Magisches, Übernatürliches sehen. Doch wenn die Entwickler/innen selbst in die Situation kommen, dass die Technik etwas Unerwartetes tut, das nicht ihren Vorstellungen entsprach, so befinden sie sich selbst in der Situation “von Voodoo-Magie umgeben zu sein”, wie es bei Whitson (2018) deutlich wurde.

Der Aspekt, dass die Smartphone Technologie als eine Art täglicher Begleiter oder Kamerad gesehen werden kann, kommt auch sowohl in der Literatur als auch in den Interviews vor. In der Literatur (Carolus et al. 2019) ist es zwar das Smartphone als Gesamtes und nicht nur eine spezielle App, wie Iota es ansprach, aber Apps sind ja ein wesentlicher Bestandteil des Smartphones und es müsste geklärt werden inwiefern die Nutzer das trennen, wenn es um diese Mensch-App-Beziehung geht. Es ist jedoch nachvollziehbar, das Smartphone als Begleiter zu betrachten, wenn man bedenkt, wie jede/r Nutzer/in sein Gerät individuell gestalten kann mit einer Hülle und einem Hintergrundbild. Man könnte auch der Frage nachgehen, wie die Auswahl diverser Apps, die heruntergeladen wurden, die Identität des/der Nutzers/Nutzerin widerspiegelt.

Was die Unternehmenskultur betrifft, war auch interessant zu sehen, was die Firmen tun, um ihre Entwickler/innen glücklich und bei Laune zu halten. Laut Graziotin et al. (2018) sind Programmierer/innen eine tendenziell glückliche Berufsgruppe, was im Fall von Start-ups verständlich ist, wenn diese, wie aus den Interviews hervorgeht, eine Reihe von diversen

Team-Events veranstalten oder mit einem großen Angebot von Rückzugsorten und sportlichen, kulinarischen Aktivitäten oder gar Kinderspielecken in den Büroräumlichkeiten aufwarten können. Außerdem spielt dabei mit Sicherheit auch die Projektkultur eine Rolle, da mit der Agile-Methodologie (Bjorn et al. 2019) ein Arbeitsklima von Transparenz, flachen Hierarchien, Empowerment und offener Kommunikation (Iden & Bygstad 2018) geschaffen wird, das sich bestimmt nicht nur Programmierer/innen wünschen.

Ein Punkt der nicht direkt angesprochen wurde aber in gewisser Weise mit den Interviews verbunden werden kann, ist das Programmieren für Menschengruppen, die aufgrund von Alter oder Beeinträchtigungen einen anderen Zugang bzw. keinen Zugang zu Technologie haben, wie es im Artikel von Giakoumis et al. (2013) behandelt wird. Von den Interviewpartnern für die vorliegende Arbeit, hat dies noch niemand gemacht, abgesehen von Sigma, der in seinem Studium der Medizin-Informatik ein Interface für ältere Menschen programmiert hat, damit sie den Touchscreen besser bedienen können. Auf ähnliche Weise engagiert sich Omega im Rahmen seiner Arbeit für Personen, die an Diabetes Typ 1 erkrankt sind. Im Gegensatz zu den “Körper-Hackern” aus Kaziunas et al.’s Bericht (2018), macht er das in seinem geregelten Arbeitsverhältnis, ohne jemanden in ein gesundheitliches Risiko zu bringen.

Was den Habitus und die Identität der Entwickler/innen betrifft, stimmen auch hier die Aussagen aus den Interviews mit der Literatur (Strübing 1992; Schachtner 1993; Dzida, Spittel, Sylla 1983) Großteils überein, wenn es um die Arbeitsweise geht zum Beispiel. Alle versuchen es, einen klar strukturierten, leserlichen, guten Code zu schreiben um eine gute Zusammenarbeit mit anderen Programmier-Kolleg/innen zu gewährleisten. Anders als in dieser Literatur, sehen sie sich aber nicht als Nerds oder Geeks, sondern als ganz normale Menschen, die einen bestimmten Beruf ausführen. Wie in den Ergebnissen bereits angeführt, ist das hauptsächlich auf die höhere Verbreitung dieses Berufs in der Arbeitswelt, die hohe Nachfrage an qualifizierten Fachkräften in dieser Branchen und der damit einhergehenden gestiegenen Anerkennung des Berufs in der Gesellschaft seit 20-30 Jahren zurückzuführen.

Ein wichtiger Punkt, der noch Potential zur Verknüpfung mit der Literatur hat, ist das Usability Thema. In den Interviews wird Usability oder meist eher *User Experience* als etwas ganz wesentlich Wichtiges angesprochen, das jedoch in vielen Unternehmen noch niemand verstanden hat und dem kein Platz für Diskussionen darüber eingeräumt wird. In fast allen Start-ups der interviewten Personen wurden User-Tests durchgeführt, mit dem Ziel, ihr Produkt dadurch zu verbessern, so wie es auch in der Literatur (Reeves 2018) erwähnt wurde. In einer Quelle (Tractinsky 2018) wurde sehr stark auf Usability in der akademischen

Debatte eingegangen und wie in den Kommentaren dazu darauf hingewiesen wurde, ist das für die Praxis in der Realität nur geringfügig relevant. So verschieden Usability in der Literatur (Hertzum 2010, 2018; Bertelsen 2018; Reeves 2018, Stage 2018) dargestellt wurde und wie diese Vielfalt immer wieder betont wurde, so unterschiedlich sind die Nutzer Tests tatsächlich wie aus den Interviews hervorgegangen ist. Da jede Firma ein völlig anderes Produkt, für eine komplett unterschiedlicher Zielgruppe entwickelt, gestaltet sich auch Usability und die Tests dazu immer unterschiedlich. Da Usability als ein Teil menschlicher Beziehungen gesehen werden kann und sich jede Beziehung mit anderen Menschen oder gesellschaftlichen Gruppen von anderen solcher Beziehungen unterscheidet, sind auch die Anforderungen an Usability sehr divers.

Da Usability aber nicht nur in den dafür entwickelten Tests definiert wird, sondern während des ganzen Entwicklungsprozesses, insbesondere in den Feedbackschleifen mit Usern und Kund/innen, geformt wird, kann gute Usability als Ergebnisse der kontinuierlichen Aushandlungsprozesse zwischen den relevanten Akteuren (Vinck 2003) verstanden werden. Wie in der Literatur bei Vinck (ebd.), sind auch im Entwicklungsprozess einer App die Techniker/innen gezwungen, sich mit den Vorstellungen und Wünschen anderer Mitarbeiter/innen, oder den Vorgesetzten, den Kund/innen, den Usern u.a. auseinanderzusetzen. In den Interviews wurde deutlich wie ausgedehnt solche Feedbackloops als Aushandlungsprozesse sein können. Am Ende steht dann im Idealfall ein Produkt mit guter Usability.

9 Fazit/Ausblick

Zu Beginn dieser Arbeit wurde aufgezählt, was sich alles in den vergangenen 30 Jahren in unserem Alltag durch Technologie geändert hat. Ein großer Teil der aktuellsten Entwicklungen, betreffen das Smartphone, das mit den darauf installierten Apps unser Leben erleichtert (oder erleichtern soll). Mit der zunehmenden Verbreitung von Smartphones steigt auch die Anerkennung und Wertschätzung des Berufs der Entwickler/innen, sowie auch deren Einflussnahme auf unser tägliches Leben und deren Verantwortung, dass diese Einflussmacht nicht missbraucht wird. Alles in allem kann nun die Schlussfolgerung gezogen werden, dass sich Programmierer/innen unterbewusst sehr wohl als eine Art Verbindungsstück (auf unterschiedliche Weisen) zwischen Technik und Gesellschaft fühlen, sie aber bisher noch nicht darauf angesprochen wurden. Nachdem die Forschungsfrage am Ende der Interviews gestellt wurde, konnten alle zustimmen, dass man Programmierer/innen diese Rolle zuschreiben kann, wobei die Art der Verbindung im ersten Moment unterschiedlich verstanden wurde: Manche denken eher an eine Übersetzungsfunktion, andere wiederum an eine Unterrichtsfunktion über Technologie, oder aber an die Umsetzungsfunktion der Nutzer/innenwünsche. Alle dieser Ansichten über die Verknüpfungsrolle sind richtig und Teil dieser Rolle, wie sie in dieser Arbeit verstanden wird. Ihre Verbindung zur Technologie drückt sich vor allem durch das Knowhow über die Technik aus, sowie die Fähigkeit, Vorstellungen, Ideen, Wünsche, Design der Menschen mittels Programmiersprachen dem Computer verständlich zu machen und ihn damit dazu zu bringen, diese Vorstellungen etc. auszuführen.

Die Verbindung zur Gesellschaft gestaltet sich wiederum so, dass sie erstens selbst Teil[15] dieser sind und zweitens hauptsächlich in Feedbackschleifen mit den Individuen darin über die Technik kommunizieren.

Nun zu der zentralen Frage, wie im Entwicklungsprozess gesellschaftliche Werte in die App gelangen? Vorrangig durch die bereits erwähnten Feedbackschleifen mit den Nutzer/innen oder mit Familie und Freund/innen. Eine weitere Möglichkeit ist die Weltsicht der Entwicklerin/des Entwicklers selbst, aber auch die Entwicklungen am Technikmarkt nehmen Einfluss darauf, wie sich die App gestalten wird. Gleichzeitig sind diese Feedback Loops, die Punkte, an denen die Gesellschaft bzw. die Nutzergruppen darin am meisten Einfluss nehmen kann.

Umgekehrt sind sich die Programmierer/innen sehr bewusst darüber, welchen Einfluss die

[15] Ansicht der Autorin.

Technik hat, die sie entwickeln. Besonders deutlich wird das in Aussagen über die Verantwortung, die sie gegenüber der Gesellschaft tragen oder über ihr Ziel, einen sinnvollen Beitrag zu leisten.
Der Zukunft blicken sie zuversichtlich entgegen, da ja die Anerkennung ihres Berufsstandes immer noch zunimmt und auch, weil sie selbst schon Erfahrungen mit dem Fachkräftemangel gemacht haben, sodass sie sich kaum um ihre Stelle sorgen müssen. Durch die eingangs erwähnte Ausbreitung der Digitalisierung und der nun zentraleren Rolle als Programmierer/in, sehen sie sich selbst als mehr in die Mitte der Gesellschaft gerückt, da sie sich auch als normale Menschen bezeichnen, im Gegensatz zu Nerds, die eher am Rande der Gesellschaft verortet werden könnten.

Für weitere Studien in diesem Bereich, wäre eine erste Möglichkeit zu untersuchen, inwiefern unterschiedliche Software Unternehmen oder Start-ups (große und kleine etc.), unterschiedliche Typen von Programmierer/innen unter ihren Angestellten haben? Bzw. ob sich Freelancer von den angestellten Entwickler/innen in irgendeiner Weise unterscheiden? Weiters gilt es genau herauszufinden, inwieweit Usability wirklich vernachlässigt wird in Österreich und ob es schon Gegentrends gibt? Wie veraltet sind die technologischen Systeme tatsächlich, die in Unternehmen diverser Branchen im Einsatz sind?
Eine mögliche Frage wäre auch, welche Dimension von Usability bei welchen Apps zum Einsatz kommt? Oder welche der oben genannten „Nützlichkeits-Kategorien“ bei welchen Apps am stärksten vertreten ist?
Natürlich gibt es die Möglichkeit von unzähligen Fallstudien darüber welche Werte in unterschiedliche Apps einfließen? Und welche der oben identifizierten “Einflusspunkte” in diesen Fällen eine zentrale Stellung einnehmen?
Die Möglichkeiten an weiterführenden Studien sind schier unendlich, was nicht weiter verwunderlich ist, bei dem bisherigen Ausmaß an Untersuchungen und bei der rasanten Weiterentwicklung dieses Forschungsfeldes. So hoffe ich, mit dieser Arbeit einen Anknüpfungspunkt in diese Richtung zu bieten.

10 Literaturverzeichnis

Achleitner, Ann-Kristin. 2017. Start-up-Unternehmen. In: *Gabler Wirtschaftslexikon.* Springer Gabler Verlag. https://wirtschaftslexikon.gabler.de/definition/start-unternehmen-42136/version-189893 [letzter Zugriff: 03.08.2019].

Atteslander, Peter. 2010. *Methoden der empirischen Sozialforschung.* 13., neu bearbeitete und erweiterte Auflage. Berlin: Erich Schmidt.

Beer, David. 2017. The social power of algorithms. *Information, Communication & Society*, Vol. 20(1) 1–13.

Bendel, Oliver. 2011. *Crowdfunding*. Gabler Wirtschaftslexikon. https://wirtschaftslexikon.gabler.de/definition/crowdfunding-53556 [letzter Zugriff: 11.09.2019].

Berger, Christian, Astrid Schöggl. 2019. “Hard coded”: Algorithmische Diskriminierung. *Arbeit-Wirtschaft-Blog*, 23. Juli 2019. https://awblog.at/algorithmische-diskriminierung/ [letzter Zugriff: 17.09.2019].

Bertelsen, Olav W. 2018. Commentary: Usability and the Primacy of Practice, *Human–Computer Interaction*, 33:2, 182-185.

Bijker, Wiebe E., John Law. 1992. *Shaping Technology/Building Society*. Cambridge: MIT Press.

Bijker, Wiebe E., Thomas P. Hughes, and Trevor. J. Pinch. 2012. *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*. Anniversary ed. Cambridge, Mass.: MIT Press.

Bjorn, Pernille. Anne-Marie Soderberg, S. Krishna. 2019. Translocality in Global Software Development: the Dark Side of Global Agile. *Human-Computer Interaction*, Vol. 34, 174-203.

Blumer, Herbert. 1954. What is wrong with social theory? *American Sociological Review*, Vol. 19(1), 3–10.

Boren, Ted. & Judith Ramey. 2000. Thinking aloud: Reconciling theory and practice. *IEEE Transactions on Professional Communication*, Vol. 43(3), 261–278.

Das, Mitra, Shirly Kolack. 1990. T*echnology, Values and Society. Social Forces in Technological Change*. New York; Bern; Frankfurt am Main; Paris: Peter Lang Publishing.

Downey, Gary Lee. 1998. *The machine in me: an anthropologist sits among computer engineers*. New York: Routledge.

Dzida, Wolfgang, Angela Spittel und Karl-Heinz Sylla. 1983. Einsatz eines "Message System" bei der Software-Entwicklung - Ein Erfahrungsbericht. In: *Psychologie des Programmierens*. Hrsg. Helmut Schauer, Michael Tauber. 139-169, Wien/München:

Oldenbourg.

Hübsch, Mohammed. 2014. *UI und UX Design – Kenne den Unterschied*. https://entwickler.de/online/web/ui-und-ux-design-kenne-den-unterschied-143641.html [letzter Zugriff: 16.09.2019].

Flanagin, Andrew J., Craig Flanagin, Jon Flanagin. 2010. Technical code and the social construction of the internet. *New Media and Society* 12(2), S.179-196.

Flórez, Francisco Buitrago, Rubby Casallas, Marcela Hernández, Alejandro Reyes, Silvia Restrepo, and Giovanna Danies. 2017. Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, Vol. 87 (4), 834 –860.

Fogg, B.J. 2003. *Persuasive Technology: Using Computers to Change What We Think and Do*. San Francisco, CA: Morgan Kaufmann.

Froschauer, Ulrike und Lueger, Manfred. 2003. *Das qualitative Interview. Zur Praxis interpretativer Analyse sozialer Systeme*. Wien: WUV-Univ.-Verlag.

Giakoumis, Dimitris, Nikolaos Kaklanis, Konstantinos Votis, Dimitrios Tzovaras. 2013. *Enabling user interface developers to experience accessibility limitations through visual, hearing, physical and cognitive impairment simulation*. Berlin, Heidelberg: Springer - Verlag.

Graziotin, Daniel, Fabian Fagerholm, Xiaofeng Wang, & Pekka Abrahamsson. 2018. What happens when software developers are (un)happy . *Journal of Systems and Software*, Vol.140, 32-47.

Gründerszene Lexikon: *Start-up* https://www.gruenderszene.de/lexikon/begriffe/startup?interstitial [letzter Zugriff: 03.08.2019].

Gründerszene Lexikon: *App* https://www.gruenderszene.de/lexikon/begriffe/app?interstitial [letzter Zugriff: 11.09.2019].

Gründerszene Lexikon: *Crowdfunding* https://www.gruenderszene.de/lexikon/begriffe/crowdfunding?interstitial [letzter Zugriff: 11.09.2019].

Gründerszene Lexikon: *Venture-Capital* https://www.gruenderszene.de/lexikon/begriffe/venture-capital-vc?interstitial [letzter Zugriff: 11.09.2019].

Gründerszene Lexikon: *Usability* https://www.gruenderszene.de/lexikon/begriffe/usability?interstitial [letzter Zugriff: 13.09.2019].

Häußling, Roger. 2019. *Techniksoziologie. Eine Einführung*. 2. Auflage. Opladen, Toronto: Verlag Barbara Budrich.

Heim, Joachim. 1974. *Programmierer und Systemanalytiker. Berufssoziologische Aspekte*.

Aachen: Wirtschafts- und Sozialwissenschaftliche Fakultät der Universität Köln.

Helfferich, Cornelia. 2011. *Die Qualität qualitativer Daten: Manual für die Durchführung qualitativer Interviews.* 4. Aufl. Wiesbaden: VS Verlag für Sozialwissenschaften.

Hertzum, Morten. 2010. Images of Usability. *International Journal of Human–Computer Interaction*, Vol. 26(6), 567-600.

Hertzum, Morten. 2018. Commentary - Usability a Sensitizing Concept. *Human-Computer Interaction*, Vol. 33, 178-181.

Iden, Jon. Bendik Bygstad. 2018. The social interaction of developers and IT operations staff in software development projects. *International Journal of Project Management*, Vol. 36, 485-497.

Innerhofer, Judith E. 2019. *Frauen an die Codes*. ZEIT Österreich Nr. 23/2019, 29. Mai 2019. https://www.zeit.de/2019/23/programmiererinnen-frauen-karriere-it-branche-maennerdomaene/komplettansicht [letzter Zugriff: 16.09.2019].

Kaplan, Daniel.2014. *Coder vs Programmer vs Software Engineer vs Architect*. http://www.sleepeasysoftware.com/coder-vs-programmer-vs-software-engineer-vs-architect-vs/ [letzter Zugriff: 11.09.2019].

Karriere.de. *Wo Hacker ihr unheimliches Handwerk lernen*. https://www.karriere.de/it-sicherheit-wo-hacker-ihr-unheimliches-handwerk-lernen/23042746.html [letzter Zugriff: 12.09.2019].

Kaziunas, Elizabet, Silvia Lindtner, Mark S. Ackerman & Joyce M. Lee. 2018. Lived Data: Tinkering With Bodies, Code, and Care Work, *Human–Computer Interaction*, Vol. 33(1), 49-92.

Kupka, Ingbert. 1983. Paradigmen des Programmierens. In: *Psychologie des Programmierens*. Hrsg. Helmut Schauer, Michael Tauber. 11-34, Wien/München: Oldenbourg.

Latour, Bruno, Steeve Woolgar. 1979. *Laboratory life: the social construction of scientific facts*. Beverly Hills, CA: Sage.

Lee, Eun Ju, Clifford Nass and Scott Brave. 2000. Can computer-generated speech have gender: an experimental test of gender stereotype. In: *Proceeding on CHI'00 extended abstracts on human factors in computing systems* (eds Lee, Eun Ju, Clifford Nass and Scott Brave), The Hague, 1–6 April, pp. 289–290. New York: ACM.

Lee, Eun Ju, Clifford Nass and Scott Brave. 2000. *Proceeding on CHI'00 extended abstracts on human factors in computing systems*, New York: ACM.

Licoppe, Christian. Julien Figeac. 2018. Gaze Patterns and the Temporal Organization of Multiple Activities in Mobile Smartphone Uses. *Human-Computer Interaction*, Vol. 33, 311-334.

Lueger, Manfred. 2010. *Interpretative Sozialforschung: Die Methoden.* Wien: Facultas.

Mackenzie, Adrian 2006. *Cutting Code. Software and Sociality*. New York et al.: Peter Lang.

Mackenzie, Adrian, Theo Vurdubakis. 2011. Codes and Codings in Crisis. Signification, Performativity and Excess. *Theory, Culture & Society,* Vol.28(6), 3–23.

MacKenzie, Donald, Judy Wajcman. 1985. *The Social Shaping of Technology: How the Refrigerator Got Its Hum*. Philadelphia: Open University Press.

McFarland, Daniel A., Kevin Lewis, Amir Goldberg. 2016. Sociology in the Era of Big Data: The Ascent of Forensic Social Science. *The American Sociologist,* Vol.47(1), 12–35.

Nass, Clifford, Jonathan Steuer and Ellen R. Tauber. 1994. Computers are social actors. In: *Proceedings of the SIGCHI conference on human factors in computing systems celebrating interdependence* (CHI 94) (eds Nass, Clifford, Jonathan Steuer and Ellen R. Tauber), Boston, MA, 24–28 April, pp. 72–78. New York: ACM.

Nass, Clifford, Jonathan Steuer and Ellen R. Tauber. 1994. *Proceedings of the SIGCHI conference on human factors in computing systems celebrating interdependence* (CHI 94). New York: ACM.

Norton Internet Security. *What is the Difference between Black, White and Grey Hat Hackers?* https://us.norton.com/internetsecurity-emerging-threats-what-is-the-difference-between-black-white-and-grey-hat-hackers.html [letzter Zugriff: 12.09.2019].

Nørgaard, Mie. & Kasper Hornbaek. 2006. What do usability evaluators do in practice? *Proceedings of the DIS Conference on Designing Interactive Systems*. New York, NY: ACM.

Petersen, Jennifer. 2015. Is code speech? Law and the expressivity of machine language. N*ew media & society.* 17(3), S.415 –431.

Reeves, Stuart. 2018. Commentary: Usability in Vivo. *Human–Computer Interaction*, Vol. 33:2, 190-194.

Rehbein, Boike. 2006. *Die Soziologie Pierre Bourdieus*. Konstanz: UVK Verlagsgesellschaft.

Schachtner, Christel. 1993. *Geistmaschine. Faszination und Provokation am Computer.* Frankfurt am Main: Suhrkamp.

Schauer, Helmut, Michael Tauber. 1983. *Psychologie des Programmierens*. Wien/München: Oldenbourg.

Schultz, Eva. 2019. *Statistiken zur Smartphone-Nutzung in Österreich*. statista.com. https://de.statista.com/themen/3654/smartphone-nutzung-in-oesterreich/ [letzter Zugriff: 11.09.2019].

SEO-Analyse.com. *APP Begriffserklärung und Definition*. https://www.seo-analyse.com/seo-lexikon/a/app/ [letzter Zugriff: 11.09.2019].

Simon, Herbert A. 1994. *Die Wissenschaften vom Künstlichen*. Wien/New York: Springer.

Skorkin, Alan. 2010. *The Difference Between A Developer, A Programmer And A Computer Scientist*. https://skorks.com/2010/03/the-difference-between-a-developer-a-programmer-and-a-computer-scientist/ [letzter Zugriff: 11.09.2019].

Stage, Jan. 2018. Commentary: Usability in Theory and Practice. *Human–Computer Interaction*, Vol. 33:2, 195-197.

Strübing, Jörg. 1992. *Arbeitsstil und Habitus. Zur Bedeutung kultureller Phänomene in der Programmierarbeit. Werkstattberichte* - Band 34. Kassel: Wissenschaftliches Zentrum für Berufs- und Hochschulforschung der Gesamthochschule Kassel.

Tractinsky, Noam. 2018. The Usability Construct: A Dead End?, *Human–Computer Interaction*, 33:2, 131-177.

Turkle, Sherry. 2005. *The Second Self: Computers and the Human Spirit*. Cambridge, MA: The MIT Press.

Vinck, Dominique. 2003. *Everyday Engineering. An Ethnography of Design and Innovation*. Cambridge Massachusetts: The MIT Press.

Weddehage, Jan. 2015. *Coder, Programmierer, Entwickler: Wo liegt der Unterschied?* entwickler.de. https://entwickler.de/online/development/coder-programmierer-entwickler-unterschied-152349.html [letzter Zugriff: 03.08.2019].

Whitson, Jennifer R. 2018. Voodoo software and boundary objects in game development: How developers collaborate and conflict with game engines and art tools. *new media & society*, Vol. 20(7) 2315 –2332.

Wiener, Norbert. 1961. *Cybernetics or Control and Communication in the Animal and the Machine*. 2. Auflage. Cambridge (Mass.): MIT-Press.

Winograd, Terry, Fernando Flores. 1989. *Erkenntnis – Maschinen – Verstehen. Zur Neugestaltung von Computersystemen*. Berlin: Rotbuch.

Witzel, Andreas. 2000. Das problemzentrierte Interview. *Forum Qualitative Sozialforschung*, Vol.1(1), Art. 22.

WKO Gründerservice. 2019. *Business Angels*. https://www.gruenderservice.at/site/gruenderservice/planung/Business_Angels.html [letzter Zugriff].

Zoric, Tamara. Zeljko Stojanov. 2018. Software developers' perceptions of soft skills in software requirements engineering. *Journal of Engineering Management and Competitiveness* (JEMC), Vol.8, No.1, 54-64.

11 Anhang

11.1 Abstract

Trotz der massiven Verbreitung von Smartphones und dem nach wie vor steigenden Grad an Digitalisierung, wurden Programmierer/innen in der sozialwissenschaftlichen Forschung bisher noch recht spärlich untersucht. Diese Arbeit stellt die Frage, wie sich Entwickler/innen als Verbindungsstück zwischen Technik und Gesellschaft sehen und wie gesellschaftliche Werte ihren Weg in die Technologie finden. Zur Beantwortung dieser Fragen wurden 12 mobile App Programmierer/innen aus Wiener Start-ups interviewt. Die angewandten Methoden waren das problemzentrierte Interview nach Witzel (2000) und die Themenanalyse nach Froschauer und Lueger (2003). Die zentralen Ergebnisse zeigen, dass sich Entwickler/innen zwar noch nicht direkt mit dieser Rollenzuschreibung auseinandergesetzt haben, sich aber unterbewusst schon als Verbindungsstück oder Vermittler/in wahrnehmen. Die gesellschaftlichen Werte gelangen in die App Technologie hauptsächlich während des Entwicklungsprozesses in den wiederkehrenden Feedbackschleifen mit Nutzer/innen, Kund/innen, Familie, oder Freund/innen etc. Diese Feedbackschleifen können als Aushandlungsprozesse zwischen den einzelnen, im Entwicklungsprozess involvierten, Akteuren verstanden werden und sind ein wichtiger Erfolgsfaktor für gute Usability bzw. User Experience.

Despite the widespread usage of smartphones and the continuous growth of digitalisation, programmers have rather been neglected as a research topic in the social sciences. This thesis asks the question, whether developers see themselves as a link between technology and society and how social values are incorporated in technology. To answer these questions, 12 qualitative interviews with mobile app programmers from Viennese start-ups were conducted. The used methods were the problem-focused interview according to Witzel (2000) and the “theme-analysis” by Froschauer and Lueger (2003). The essential outcome is, that developers have not yet directly thought of themselves in this particular role, but subconsciously they seem to feel that way. The way how social values are being transported and incorporated into technology, mainly happens during the development process via recurring feedback loops with users, customers, family, or friends etc. These feedback loops can be viewed as negotiation processes between all the involved actors in the development process. Furthermore, it is an important factor for the successful design of usability or user experience, which was of great concern to the interviewed programmers.

11.2 Glossar

Algorithmus
Ein Algorithmus ist eine Vorschrift, wie ein Problem oder eine bestimmte Gruppe von Problemen gelöst werden kann. Er setzt sich zusammen aus mehreren einzelnen Anweisungen, die sowohl in Programmiersprachen aber auch in menschlichen Sprachen ausgedrückt werden können.

App
Der Begriff App kommt vom englischen application software, was wörtlich übersetzt Anwendungsprogramm bedeutet. Eine App ist also ein Computerprogramm (meist auf einem Smartphone oder Tablet), das in einem bestimmten Lebensbereich oder zu einem bestimmten Thema spezifische Aufgaben erfüllt und dabei den Nutzer/innen hilft, Probleme zu lösen oder einfach Spaß bereiten soll (SEO-Analyse.com https://www.seo-analyse.com/seo-lexikon/a/app/; gruenderszene.de https://www.gruenderszene.de/lexikon/begriffe/app?interstitial).

black box
Eigentlich ein Flugdatenschreiber, der Aufzeichnet, was im Flugzeug passiert. Der Begriff wird häufig als Analogie verwendet für Geräte, die mit Daten gefüttert werden.

Bug
Ein Fehler im Code.

Bugfix
Die Korrektur eines Fehlers.

Bugreport
Die Dokumentation eines Fehlers. Wann tritt welcher Fehler wo in der App, wie auf?

Business Angel
Ein Business Angel ist eine Person, die ein junges Unternehmen mit Eigenkapital unterstützt, sowie mit Know-How aus jahrelanger Erfahrung und Geschäftskontakten als Berater/in zur Seite steht (WKO Gründerservice 2019).

Closed Beta
Eine ausgewählte Gruppe von Nutzern, bekommt die App als Testversion und meldet sämtliche Fehler an die Entwicklerfirma, die die Fehler korrigiert, bevor die fertige App für die ganze Welt oder die gesamte Zielgruppe zur Verfügung gestellt wird.
Eine closed Beta-Phase ist eine Testphase bei der nur einem kleinen Teil der Nutzer/innen die vorläufige App zur Verfügung gestellt wird und sozusagen in einem „abgesicherten, geschlossenen Raum“ die App getestet wird.

Code
In Programmiersprache geschriebene Anweisungen für den Computer, das Smartphone etc.

Crash
Die App (der Computer) stürzt ab.

Crashreport
Die Dokumentation eines Absturzes. Wann tritt der Absturz wo in der App, wie auf?

Crowdfunding
Wörtlich übersetzt heißt Crowdfunding Gruppenfinanzierung, ein im Deutschen bekannterer Ausdruck ist Schwarmfinanzierung. Es geht dabei darum, Internetnutzer (eine große Menge davon), die sich für das eigene Projekt interessieren könnten, um eine Spende zu bitten. Es ist also eine Art, das eigene Projekt zu finanzieren, wenn man selbst nicht die Mittel hat. Der Aufruf zur Spende, also zum Crowdfunding findet meist über eine bestimmte Webseite statt und in vielen Fällen im Zusammenhang mit der Unternehmensgründung (Bendel, Gabler Wirtschaftslexikon; gründerszene.de).

Geek
Eine Person, die sich obsessiv und enthusiastisch für ein intellektuelles Thema interessiert oder sich sehr intensiv mit einem Hobby beschäftigt.

Habitus
Der Habitus nach Bourdieu ist grob erklärt die Gesamtheit aus Erscheinung und Auftreten einer Person, dazu können unter anderem Lebensstil, Kleidung, Sprache und Geschmack gezählt werden. So kann, laut Bourdieu, der Habitus die Zugehörigkeit zu einer gesellschaftlichen Gruppe oder Schicht wiederspiegeln, in diesem Fall zur Gruppe der Programmierer/innen (Rehbein 2006).

High Level Code
Programmiersprache mit hohem Abstraktionsgrad, die verhältnismäßig einfach zu verstehen ist.

Internet der Dinge
Internet, das mit oben genannten elektronischen Geräten verbunden ist, welche darüber von der Ferne gesteuert werden können.

Kommentieren
Ein Kommentar im Code schreiben, das vom Programm nicht ausgelesen wird und daher für Nutzer nicht sichtbar ist. Es wird im Code angezeigt, bleibt aber ohne Funktionen und wird häufig angewendet als Notiz an sich selbst oder an Kollegen, wenn ein Feature noch nicht fertig ist zum Beispiel.

Low Level Code
Programmiersprache mit niedrigem Abstraktionsgrad, die komplexer zu verstehen ist, dafür aber Befehle effizienter ausführen kann.

Nerd
Meist verstanden als introvertierte, intellektuell obsessive Person, mit mangelnden sozialen Kompetenzen, die sich hauptsächlich mit unpopulären, wenig bekannten Aktivitäten beschäftigt.

Nudging
Nudging bedeutet übersetzt so viel wie lenken, anregen oder in eine Richtung steuern. Es wird häufig als neuere Marketingstrategie für Produkte oder Ansichten eingesetzt, dabei werden den jeweiligen Personen Informationen zu ihren Auswahl- oder Handlungsmöglichkeiten dargelegt, welche sie unterbewusst zu einer bestimmten Entscheidung führen.
Bei Hyper-Nudging wird den Personen eigentlich keine andere Wahl mehr gelassen, als sich für die durch Nudging „empfohlene“ Möglichkeit zu entscheiden. Entscheidungen

werden also vorweggenommen und wird nur ein Anschein von Entscheidungsmöglichkeiten erhalten.

prototypen
Eine erste Versuchsversion einer App schreiben.

to refactor
Den Code im Nachhinein korrigieren und nachbessern, wenn während des Schreibens zu wenig Zeit war.

Release
Wenn die App oder ein Feature für die Nutzer freigegeben wird.

Request
Eine Anfrage, ein Wunsch. Meist für eine bestimmte Eigenschaft oder Funktion, die sich Nutzer oder Kunden wünschen.

SDK = software development kit
Meist Code eines einzelnen Features, der in individuelle Apps integriert werden kann.
Ein Software Development Kit ist im Grunde ein vollständig funktionsfähiges Feature für ein Programm oder eine App, das relativ einfach in die eigene App integriert werden kann. Im Fall der oben genannten Firma, wird an die Programmierer/innen anderer Firmen, die das SDK integrieren sollen, auch eine Anleitung mitgeliefert, wie das gemacht werden kann.

Smart devices
Elektronische Geräte, die mit dem Internet verbunden sind und dadurch über das Smartphone oder Tablet gesteuert werden können, wie die Kaffeemaschine, der Fernseher.

Venture Capital
Venture Capital bedeutet zu Deutsch Risikokapital/Wagniskapital. Es handelt sich um Investment, das häufig an relativ junge Unternehmen vergeben wird als Unterstützung oder Entwicklungshilfe für die erste Zeit der Etablierung. Den Investor/innen ist das Risiko dabei bewusst, dass sie beim Scheitern der jungen Firma, ihr Kapital nicht wieder bekommen (gründerszene.de).

11.3 Leitfaden

Einstiegsfrage:

Erzählen Sie mir bitte einfach mal alles über Ihre Programmierlaufbahn, von Anfang an, wie Sie dazu gekommen sind, bis heute?

Nachfragen:

1. Was begeistert Sie so an der Technologie/am Programmieren?
2. Wie sehen Sie Ihre Programmiertätigkeit bezüglich ihres Stellenwerts in der Gesellschaft?
3. Was sind Ihre Ziele, wenn Sie programmieren? (bzgl. Tech/Gesellschaft)
4. Wie kommt es zu einer Idee für eine App?
5. Wie läuft die Entwicklung der Mobile App von der Idee bis zum Launch ab?
6. Welche äußeren Bedingungen beeinflussen die Arbeit? (Finanzieller Rahmen, Wünsche der AuftraggeberInnen/KundInnen/KollegInnen etc.) + Umgang damit?
7. Wie funktioniert die Arbeit im Team? Wer trifft Entscheidungen?
8. Wie gestaltet sich die Beta-/Testphase?
9. Wann ist die Entwicklung einer App abgeschlossen?
10. Wie würden Sie Ihren Code beschreiben?
11. Welche Aufgaben soll dieser erfüllen?
12. Wie muss ein Code Ihrer Meinung nach sein?

"Statistische" Daten:

Alter, Herkunft, wie lange wird schon programmiert, in welchem Bereich wird programmiert (iOS vs. Android).

Als letzte wurde die Forschungsfrage gestellt, mit dem Hinweis, es sei ein Versuch, weil es normalerweise nicht üblich ist.

11.4 Themenkodierung

Kategorien:

- erster Kontakt mit Computertechnologie
- Ausbildungslaufbahn
- magisch/Black Box
- learning by doing
- erste Jobs
- Begeisterung
- Arbeitsweise
- Stellenwert
- Ziele
- Idee
- Ablauf Entwicklung
- Teamarbeit
- Einflüsse
- Code
- Beta/Testing
- Abschluss
- Klischee
- Userfeedback
- social benefit
- Forschungsfrage (FF)

11.5 Interviewübersicht

Firma	Jobvermittlung		Bitcoin	Lern-App	Agentur	Bilder scannen (Computervision)						Medizin-App
Pseudonym	Alpha	Beta	Frau Gamma	Delta	Zeta	Epsilon	Iota	Frau Kappa	Lambda	Sigma	Psilon	Omega
Alter	34	31	28	29	30	32	27	24	27	30	59	37
Herkunft	Wien	OÖ	OÖ	Wien	Salzburg	Slowakei/ Wien	Serbien/Berlin	Rumänien	NÖ	Wien	Kärnten	OÖ
programmiert seit	16	14	19	14	11	15	20	19	14	8	15	14
Sprache	Android	iOS	Android	Android	Android	Android	Android	Android	iOS	Xamarin	Android	iOS
Datum	30.1.	30.1.	7.2.	15.2.	7.3.	20.2.	8.3.	8.3.	19.3.	19.3.	19.3.	26.4.
Dauer	1h 7	1h 15	43 min	1h 4	1h 2	1h 8	47 min	1h 6	54 min	37 min	1h 13	1h 5
Ort	Firmenbüro	Firmenbüro	mein Wohnzimmer	Café	Firmenbüro	Firmenbüro	Firmenbüro	Firmenbüro	Firmenbüro	Firmenbüro	Firmenbüro	Firmenbüro
Kontaktaufnahme	durch E-Mail an Firma	durch E-Mail an Firma	durch meinen Arbeitskollegen (ist eine Bekannte von ihm)	durch E-Mail an Firma	über anderen Interviewkandidaten	durch E-Mail an Firma	durch Teamlead, der auch Interviewkandidat war	durch Teamlead, der auch Interviewkandidat war	durch seinen Arbeitskollegen	durch Teamlead, der auch Interviewkandidat war	durch Teamlead, der auch Interviewkandidat war	durch E-Mail an Firma
Schulform	Gymnasium	HAK	HBLA	HTL	HTL	Gymnasium	Realschule/Gymnasium	Gymnasium	HTL	HTL	HTL	HTL
Uni	FH Technikum	TU versucht	FH OÖ Hagenberg	TU Wien	TU Graz	TU Wien	TU Berlin	Uni in Rumänien	FH Technikum	Uni Wien	Uni in Kärnten	FH Oberösterreich