TECHNISCHE UNIVERSITÄT
CHEMNITZ

Der geistige Deadlock: Kognitive Blockaden beim Programmieren

Masterarbeit

zur Erlangung
des akademischen Grades
M.Sc.

Fakultät für Informatik
Professur Softwaretechnik

Eingereicht von: Belinda Schantong
Matrikel Nr.: [redacted]
Einreichungsdatum: 24.02.2023

Betreuerin: Prof. Dr.-Ing. Janet Siegmund
Dominik Gorgosch M.Sc.

Danksagung

Ich danke meiner Betreuerin Janet Siegmund und meinem Betreuer Dominik Gorgosch für die Unterstützung, Beratung und die motivierende Rückmeldung während dieser Arbeit.

Ferner danke ich Cube, Fae, Felis, Fenken, Fleggi, Fuchs, Micha, Rezam, Serenade und Sona für ihre Zeit, die Gespräche und die wertvollen Einsichten, die diese gebracht haben.

Abstract

Hintergrund: Programmieren und wissenschaftliches Schreiben gelten jeweils als Kernkompetenzen ihres Fachs und ähneln sich in der Problematik, dass sie als schwer zu vermitteln gelten. Ein Zweig der Schreibprozessforschung befasst sich deswegen mit der Erforschung von Schreibblockaden.

Ziele: Ziel der Arbeit ist es, herauszufinden, ob Schreibblockaden auch beim Programmieren existieren.

Methodik: Zu diesem Zweck wurden Interviews mit erfahrenen Programmierenden geführt, um ihre Programmierprozesse und Probleme in den Prozessen zu erfassen. Anschließend wurden die beschriebenen Prozesse und Probleme mit den Prozessen und Blockaden des wissenschaftlichen Schreibens verglichen.

Ergebnisse: Der Vergleich zeigt, dass Prozesse und Strategien des Schreibens auch beim Programmieren vorkommen und dass Blockaden bei beiden Prozessen auf die gleiche Weise entstehen können. Einige Arten von Schreibblockaden kommen daher auch beim Programmieren vor.

Implikationen: Wenn einige Probleme beim Programmieren wie Schreibblockaden funktionieren, bedeutet das, dass sie womöglich auch wie Schreibblockaden behandelt werden können. Strategien der prozessorientierten Schreibdidaktik zur Überwindung von Blockaden könnten daher auch in der Programmierdidaktik eingesetzt werden.

Keywords: Informatik-Didaktik, Schreibblockade, Programmierprozess, Interne Repräsentation, Kognition

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Software-Systeme sind aus unserem Alltag nicht mehr wegzudenken und ihre Relevanz scheint nur weiter zu steigen. Entsprechend wichtig ist es, Menschen auszubilden, die diese Software-Systeme programmieren und warten können. Aber auch Endnutzerprogrammierung wird immer wichtiger, weshalb auch außerhalb der Informatik viele Studiengänge Programmierkurse beinhalten, da die Fähigkeit in verschiedensten Disziplinen zum Beispiel für die Datenanalyse benötigt wird. Ausgerechnet Programmierung gilt aber als Kompetenz, die besonders schwer zu lehren ist. Programmierkurse an den Universitäten leiden unter hohen Durchfallquoten und die Studierenden leiden unter Pflichtkursen, welche aus ihrer Sicht nur ihr Studium behindern. In der Folge davon laufen Studierende Gefahr, eine regelrechte Abneigung gegen das Programmieren zu entwickeln und in Zukunft jedes Modul, welches Programmierung beinhaltet, zu vermeiden.

Solches Verhalten ist auch in Bezug auf wissenschaftliche Hausarbeiten bekannt. Wissenschaftliches Schreiben stellt für viele Studierende eine große Herausforderung dar, welche von vielfältigen Problemen begleitet sein kann [21, S.IX]. Misserfolge bei Hausarbeiten können mitunter zu langfristigen Dispositionen zum Vermeiden des Schreibens führen [23, S.66]. Nun ist Schreiben (insbesondere kontextangemessenes Schreiben, wovon wissenschaftliches Schreiben eine nennenswerte Art ist) aber eine Kompetenz, welche nicht nur ein Ziel höherer Bildung darstellt, sondern auch Voraussetzung für das Lernen in diesem Bereich ist [30, S.107]. Dies gilt, mindestens im Bereich der Informatik, auch für das Programmieren. An diesen Kompetenzen zu scheitern oder sie notgedrungen mit Kompetenzen zu ersetzen, die dem Überspielen fehlender Kompetenz dienen [30, S.107], kann daher fatal sein. Unglücklicherweise gibt es einige Studien in Bezug auf Programmierkurse, welche suggerieren, dass Lernende auch in Aufbaukursen noch oft Probleme haben, da sie die Kompetenzen, welche mit Abschluss der Grundkurse erwartet werden, nicht mitbringen [41].

In Bezug auf das Schreiben hat sich seit den 1980er Jahren eine prozessorientierte Didaktik entwickelt. Diese betrachtet Schreiben als kontinuierlichen Problemlösungsprozess, welcher nicht nur dem Wiedergeben von Wissen dient, sondern auch ein Instrument ist, mit dessen Hilfe sich Wissen angeeignet und neues Wissen produziert werden kann [45, S.17]. In diesem Prozess kann es zu Problemen kommen, welche den Schreibfluss behindern und die Ideen versiegen lassen: Sogenannte Schreibblockaden.

Schreibblockaden entstehen oft aufgrund falscher Vorstellungen über das Schreiben

und den Schreibprozess an sich [43]. Solche falschen Vorstellungen über den Prozess des Programmierens und die Prozesse, die ablaufen, wenn Maschinen die Programme ausführen, sind ein Problem, das unter unerfahrenen Programmierenden ebenfalls sehr verbreitet ist [41]. Es stellt sich daher die Frage, ob diese falschen Vorstellungen sich bei Programmierenden ebenfalls zu Blockaden entwickeln können, welche das Programmierenlernen behindern.

Dieser Frage werden wir in dieser Arbeit nachgehen. Dafür nutzen wir eine Methodik, welche von einer der wichtigsten Arbeiten im deutschsprachigen Raum zum Thema Schreibblockaden und deren Überwindung inspiriert ist: Gisbert Keseling sammelte im Rahmen seiner Arbeit als Schreibberater diverse Fallbeispiele von blockierten Schreibenden und kontrastierte diese mit den Schreibprozessen von erfolgreichen Wissenschaftlerinnen und Wissenschaftlern, welche er zu diesem Zweck zu ihrem Vorgehen beim Schreiben interviewte [28].

In dieser Arbeit nehmen wir das Thema von der anderen Seite in Angriff: Wir interviewen sieben erfahrene Programmierende, um Einblicke in ihre Arbeitsprozesse und die Probleme, die dabei auftreten können, zu erhalten. Anschließend vergleichen wir die Prozesse und Probleme des Programmierens mit den Prozessen und Blockaden des wissenschaftlichen Schreibens, um zu überprüfen, ob wir von dem Wissen über Schreibblockaden Rückschlüsse auf Probleme beim Programmieren ziehen können. Sollte dies der Fall sein, können wir womöglich das Wissen und die Techniken aus der prozessorientierten Schreibdidaktik nutzen, um einige der Probleme in der Programmierdidaktik besser zu verstehen und schlussendlich zu überwinden.

Die Arbeit ist folgendermaßen aufgebaut: In Kapitel 2 widmen wir uns zunächst dem Schreiben. Um Schreibblockaden zu verstehen, ist es wichtig, Schreibprozesse zu verstehen. Deswegen stellen wir in diesem Kapitel die wichtigsten Schreibprozessmodelle vor und betrachten anschließend, wie unterschiedlich individuelle Schreibprozesse sein können. Schließlich definieren wir, was Schreibblockaden eigentlich sind und erklären die häufigsten Blockaden beim wissenschaftlichen Schreiben.

In Kapitel 3 wenden wir uns dem Programmieren zu. Wir stellen den Programmierprozess anhand der Prozesse des Software Engineerings nach Ian Sommerville vor. Anschließend betrachten wir, was wir über kognitive Prozesse während des Programmierens wissen. Als letztes ziehen wir ein Zwischenfazit und klären einige Begriffe, welche in den beiden Forschungsfeldern Schreiben und Programmieren teilweise unterschiedliche Bedeutungen haben.

Die Idee, Schreib- und Programmierprozesse zu vergleichen, ist nicht neu, weswegen wir in Kapitel 4 bisherige Vergleiche betrachten. In Kapitel 5 wird die Theorie anschließend zusammengefasst und einige Begriffe werden geklärt.

In Kapitel 6 beschreiben wir den Aufbau und den Ablauf der Interviews und ihrer Analyse. Dazu werden zunächst noch einmal die Forschungsfragen klar definiert, die Kriterien zur Auswahl der Teilnehmenden werden erklärt und der genutzte Fragebogen wird vorgestellt. Anschließend wird das Vorgehen bei der Durchführung der

Interviews und bei deren Transkription erläutert. Es folgt eine genaue Beschreibung der Teilnehmenden basierend auf deren Angaben in den Interviews. Schließlich wird die Methode der thematischen Analyse, welche wir angewandt haben, erklärt.

Die Diskussion der Ergebnisse wird nach den beiden Forschungsfragen aufgeteilt. In Kapitel 7 betrachten wir zunächst die Programmierprozesse der Interviewten und vergleichen sie sowohl mit den Prozessen als auch mit den Strategien des wissenschaftlichen Schreibens. Am Ende wird die erste Forschungsfrage beantwortet.

In Kapitel 8 betrachten wir die Probleme, welche die Interviewten beschrieben haben, und vergleichen sie mit den Blockaden beim wissenschaftlichen Schreiben. Anschließend wird die zweite Forschungsfrage beantwortet.

Nachdem die Forschungsfragen beantwortet sind, diskutieren wir in Kapitel 9 die Implikationen der Ergebnisse. Wir machen konkrete Handlungsvorschläge basierend auf den Ergebnissen und diskutieren mögliche weitere Forschungsansätze.

In Kapitel 10 diskutieren wir mögliche Einschränkungen der Validität, bevor wir in Kapitel 11 die wichtigsten Erkenntnisse der Arbeit noch einmal zusammenfassen und einen Ausblick darauf geben, wie die Forschung weitergeführt werden kann.

2 Einführung in die Schreibprozessforschung

Die Schreibprozessforschung ist ein interdisziplinäres Forschungsfeld, welches sich mit den physischen, kognitiven und psychologischen Prozessen beim Schreiben von Texten auseinandersetzt. Sie ist Teil des weiteren Feldes der Schreibforschung. In ihrer heutigen Form entstand die Schreibprozessforschung in den 1970er Jahren infolge eines 'Paradigmenwechsels' in der Erforschung und Didaktik des Schreibens, der zunächst von Nordamerika ausging [20, S.11].

Bis Anfang der 1970er Jahre stand bei der Schreibforschung das Produkt des Schreibens, nämlich der Text im Vordergrund. Die vorherrschende Annahme war, dass ein guter Text durch möglichst komplexe Sätze gekennzeichnet wird [20, S.11], weshalb die Didaktik sich auf das Vermitteln von Textmustern und korrekter Grammatik fokussierte [8, S.31-57]. Ein Grund für die Neuausrichtung der Schreibdidaktik war die 'kognitive Wende' in der Linguistik, die dafür sorgte, dass die Linguistik als Teil der kognitiven Psychologie betrachtet wurde. Dies brachte eine Generation von Forschenden und Lehrenden hervor, die sowohl gesprochene als auch geschriebene Sprache als kognitiven Prozess betrachteten und diesen in den Mittelpunkt der Erforschung und Vermittlung des Schreibens stellten. Damit rückte das schreibende Individuum ins Zentrum des Interesses. Befeuert wurde diese Neuausrichtung durch die wachsende Furcht, dass Lernende immer schlechter im Schreiben wurden. [20, S.13] Diese 'Schreibkrise' scheint im Nachhinein betrachtet weniger das Resultat tatsächlich sinkender Schreibkompetenzen in der Bevölkerung gewesen zu sein, sondern ist vielmehr der Tatsache geschuldet gewesen, dass die Anforderungen an Schreibende durch Veränderungen in der Medienlandschaft gestiegen waren [51, S.2]. Nichtsdestotrotz sorgte die Krise für großes Interesse an der Schreibprozessforschung und führte zur Finanzierung zahlreicher Projekte. [20, S.14]

In den 1980er Jahren kam der Paradigmenwechsel in der Schreibforschung auch in Deutschland an [20, S.15], die Notwendigkeit einer systematischen Vermittlung von Schreibkompetenzen an der Hochschule allerdings erst im Zuge der Umsetzung der Bologna-Reform [45, S.16]. Und während die Schreibprozessforschung im Laufe der Zeit auch das Schreiben am Arbeitsplatz oder literarisches Schreiben betrachtete [20, S.15], so ist insbesondere im deutschsprachigen Raum weiterhin ein großer Fokus auf dem akademischen Schreiben und der Didaktik davon geblieben. Die treibende Kraft hinter der Schreibprozessforschung sind an Universitäten angegliederte Schreibzentren, für die neben der Forschung vor allem die Ausbildung von Studierenden in

Form von Workshops, Seminaren und Einzelberatungen wichtig ist [46].

Zu den wichtigsten Aufgaben der Schreibprozessforschung zählt also die Untersuchung von Schreibprozessen, wobei sich insbesondere zu Beginn vor allem auf die kognitiven Prozesse beim Schreiben konzentriert wurde, bevor nach und nach auch soziale Aspekte mehr Beachtung fanden und Schreiben als sozial eingebettetes Handeln begriffen wurde [45, S.18-24]. Daneben werden weitere Aspekte des Schreibens untersucht. Zum Beispiel wie sich Schreibkompetenz definieren lässt, wie sie erworben und weiterentwickelt wird, welche Probleme beim Schreiben auftreten und wie sie überwunden werden können, welche Funktionen das Schreiben hat und mit welchen Strategien Schreibende vorgehen [20, S.25-42].

Dazu wurde im Rahmen der Schreibprozessforschung ein breites Repertoire von Methoden entwickelt, um den Schreibprozess und das Schreibenlernen zu beobachten und zu untersuchen. Dazu zählen unter anderem Think-Aloud-Protokolle [45, S.20], diverse Interviewformen [10], das Aufzeichnen des physischen Schreibprozesses mittels Screencapturing und Key-Logging [45, S.20], die Analyse der entstandenen Texte, sowie besondere Textformen wie die Schreiblernbiographie [10]. Für einen Überblick siehe auch Girgensohn und Sennewald [20, S.51-57].

Die Schreibprozessforschung entstand also aus dem Bedürfnis nach einer modernen, lernendenzentrierten Schreibdidaktik heraus und wurde in ihren Anfängen stark von der kognitiven Wende beeinflusst. Im Folgenden werden die wichtigsten Erkenntnisse über den Schreibprozess zusammengefasst, wobei zunächst einige Modelle des Schreibprozesses betrachtet werden, bevor individuelle Unterschiede in Form von Schreibstrategien in unseren Fokus rücken. Zuletzt befassen wir uns mit Schreibblockaden.

2.1 Phasenmodell des Schreibprozesses

Der Schreibprozess ist ein Prozess im zweifachen Sinne: Zum einen ist er ein kognitiver Problemlösungsprozess, wobei das Problem an dieser Stelle die Schreibaufgabe ist. Zum anderen ist er ein zeitlich ausgedehnter Prozess, der von der Konzeption der Idee bis zum fertigen Text als Endprodukt eine Reihe verschiedener Tätigkeiten umfasst [50]. Dieser Abschnitt befasst sich zunächst mit der zweiteren Betrachtungsweise, dem Prozess zur Herstellung eines Textes von seiner Konzeption bis zu seiner Fertigstellung.

Der Schreibprozess lässt sich auf natürlich anmutende Weise in verschiedene Phasen gliedern. Dabei werden meistens mindestens drei unterschieden: Die Vorbereitung, das Schreiben der Rohfassung und das Überarbeiten.

Die Vorbereitung, oder auch 'prewriting activities' [45, S.24], umfasst das Planen und Generieren von Inhalten vor dem eigentlichen Verschriftlichen. Selbst bei simplen Schreibaufgaben, die wir automatisiert vornehmen, ist diese Phase erkennbar.

Für das schnelle Formulieren eines Einkaufszettels zum Beispiel besteht zunächst das Ziel, eine Merkhilfe zu schreiben. Für diese wird ein Schema abgerufen, das einen Plan für die Form vorgibt, in diesem Fall aneinandergereihte Stichworte. Der Inhalt wird gegebenenfalls noch weiter strukturiert, etwa indem eine sinnvolle Reihenfolge eingehalten wird, die dem Layout des Ladens entspricht (z.B. frische Zutaten – Nahrungsmittel – Non-Food im Falle eines klassischen deutschen Discounters).

Es folgt das Versprachlichen und Verschriftlichen der generierten Pläne und Inhalte, das sogenannte 'drafting' [45, S.24] oder eben das Schreiben der Rohfassung. Dies ist die Phase, in der der Stift in die Hand genommen wird und der Text, oder im Falle des Einkaufszettels die Stichworte, niedergeschrieben werden.

Zuletzt steht das Überarbeiten des generierten Textes, auch als 'revising' [45, S.24] bezeichnet. In dieser Phase wird der Text aufgrund eigener Vorstellungen oder wegen Feedbacks von außen verändert. Im Falle des Einkaufszettels ist diese Phase nicht unbedingt notwendig, doch es könnte beispielsweise noch eine Besprechung in der Wohngemeinschaft stattfinden, bei der sich herausstellt, dass manche Posten auf dem Zettel doch nicht benötigt werden und dafür andere hinzugefügt werden müssen.

Während sich diese drei Phasen in den meisten Schreibprozessen wiederfinden lassen, werden sie für komplexere Schreibaufgaben oft noch weiter unterteilt. Beim wissenschaftlichen Schreiben zum Beispiel gehört zum Schreibprozess zusätzlich eine umfangreiche Recherche dazu. Diese Recherche wiederum ist abhängig von gesetzten Zielen und Fragestellungen und führt wahrscheinlich zu einer Phase der Strukturierung, in der das gesammelte Material eingeordnet und die Arbeit gegliedert wird [21, S.58-67]. Die Vorbereitung lässt sich so in drei weitere Phasen gliedern. Dieses erweiterte Phasenmodell ist in Abbildung 2.1 abgebildet. Neben der genannten feineren Einteilung der Vorbereitung wird hier nach dem Überarbeiten noch eine weitere Phase des Korrigierens und Editierens hinzugefügt. Das Ziel dieser Phase ist es, den Text fertig zur Abgabe, beziehungsweise zur Veröffentlichung zu machen [21, S.71]. In der Phase des Überarbeitens geht es darum, die inhaltlichen Aspekte des Texts zu überprüfen und seine kommunikative Klarheit sicherzustellen [45, S.25], wohingegen beim Korrigieren der Text auf formale Kriterien überarbeitet wird.

Bereits das Minimalbeispiel des Einkaufszettels zeigt allerdings einige Schwächen des Phasenmodells auf. Die Phasen sind nicht klar voneinander abgegrenzt, sie überlappen sich: Ich sehe in den Kühlschrank, erkenne die Lücken, notiere mir erste Punkte auf die Liste. Dann gehe ich weiter zum Vorratsschrank und wiederhole diese Tätigkeiten. Sind dies nun mehrere Schreibprozesse oder ist jeder weitere Schrank nur noch in die Phase des Überarbeitens einzuordnen?

Ähnlich beim wissenschaftlichen Schreiben. Die Phase des Orientierens, in der ein Thema gefunden und eingegrenzt werden soll, ist zwar häufig der Ausgangspunkt des Schreibprozesses, doch kann dem Formulieren der Fragestellung bereits umfang-

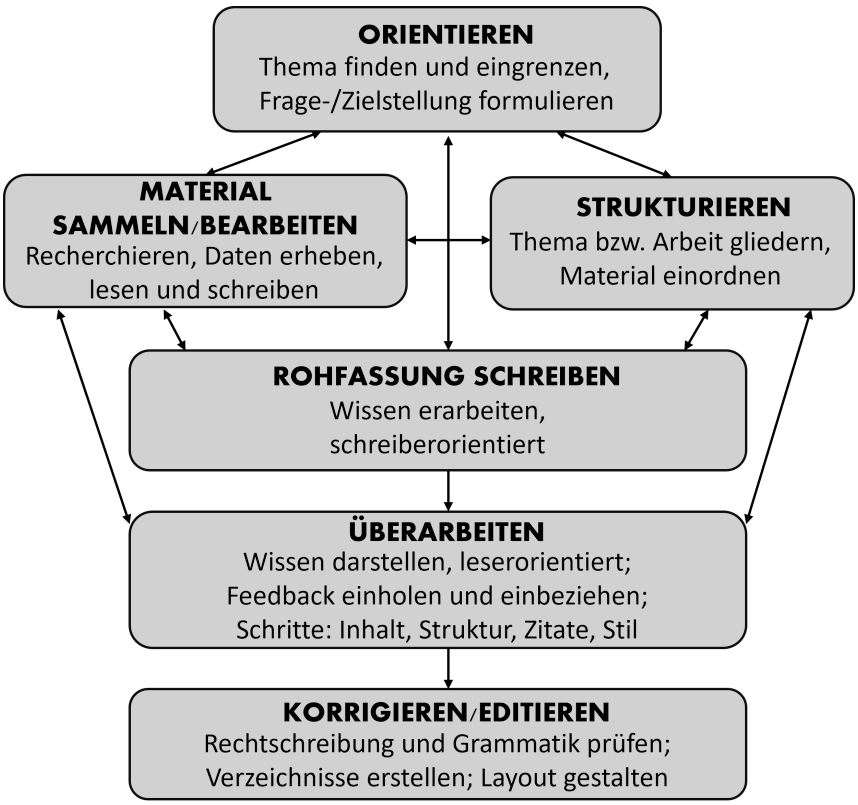

Abbildung 2.1: Phasenmodell des wissenschaftlichen Schreibprozesses nach [21]

reiche Recherche vorausgehen; beim Überarbeiten des Textes können Wissenslücken auffallen, die neue Recherche erfordern, die neue Erkenntnisse zu Tage fördern kann, welche eine Umstrukturierung des Textes auslösen können. Diese Beispiele zeigen, dass der Schreibprozess nicht linear ist, sondern sowohl iterativ als auch rekursiv [45, S.17].

Iterativ bedeutet in diesem Fall, dass die genannten Phasen wiederholt durchlaufen werden, wie beim Beispiel des Einkaufszettels. Wird produktzerlegend gearbeitet, der Text also nicht in einem Zug runtergeschrieben, werden die Phasen für die verschiedenen Abschnitte durchlaufen. Ein einzelner Abschnitt kann immer wieder überarbeitet werden und die Phasen können sich immer wieder gegenseitig anstoßen.

Rekursiv bedeutet, dass innerhalb des Prozesses der gesamte Schreibprozess erneut eingebettet sein kann. Eine neue Idee, die beim Schreiben entspringt, und der zum Beispiel ein zusätzliches Kapitel gewidmet werden soll, kann den Schreibprozess von vorne auslösen.

Die Pfeile im Modell von Abbildung 2.1 zwischen den Phasen sollen diesen Umstand zwar bereits andeuten, dennoch stellt das Phasenmodell den Schreibprozess annähernd linear dar, was nicht der Realität entspricht. Das Phasenmodell zeigt also keinen realistischen, sondern einen idealisierten Schreibprozess und geht nicht auf in-

dividuelle Unterschiede ein, die beim Schreiben aber sehr groß sein können [45, S.25]. Das Modell ist außerdem nicht in der Lage, zu zeigen, was während des Schreibens in der schreibenden Person vorgeht [17, S.37] oder welchen externen Einflüssen der Schreibprozess unterworfen ist.

Warum dann überhaupt noch Phasenmodelle? Den Schreibprozess in Phasen zu unterteilen, hat vor allem in der Schreibdidaktik seinen Sinn: Anhand des Modells lassen sich verschiedene Aspekte des Schreibprozesses erklären und es kann als Grundlage genutzt werden, um den eigenen Schreibprozess zu reflektieren und auf Stärken und Schwächen hin zu analysieren [45, S.25]. Vielen Schreibenden ist auch gar nicht klar, dass ihr Prozess aus vielen Teilprozessen besteht und die Zerlegung kann helfen, überforderte Schreibende zu entlasten. Mit der Einteilung in Phasen wird ihnen der Druck genommen, sofort einen perfekten Text schreiben zu müssen [21, S.59]. Sie können sich so erreichbare Zwischenziele setzen.

Das Phasenmodell ist also in erster Linie ein didaktisches Instrument, das vor dem Hintergrund verwendet werden sollte, dass die individuellen Arbeitsprozesse von Schreibenden sich stark unterscheiden können. Es ist keine Darstellung der 'Norm'. [45, S.25]

2.2 Kognitive Modelle des Schreibprozesses

Das einflussreichste Modell der Schreibprozessforschung ist das Modell des Schreibens als kognitiver Prozess von Linda Flower und John R. Hayes aus dem Jahr 1980 [23, S.58]. Das Modell stach damals dadurch heraus, dass es im Gegensatz zu vielen anderen Modellen zu der Zeit auf empirischen Daten basierte und kein rein theoretisches Modell war [20, S.16]. Es war außerdem das erste Modell dieser Art, das die soziale und physische Aufgabenumgebung miteinbezog [20, S.17]. In den Jahren danach haben die Autoren das Modell mehrfach überarbeitet. Die Version aus dem Jahr 1996 ist in Abbildung 2.2 zu sehen. Das Modell "versteht Schreiben in seinem Wesen als ein fortlaufendes Lösen von Problemen mit einem konstanten Grundmuster"[45, S.20]. Es ist, besonders in seinen frühen Versionen, stark von der kognitiven Psychologie geprägt [23, S.58].

Die zentrale Frage, die das Modell zu beantworten versucht, ist, wodurch die Entscheidungen der Schreibenden beim Schreibprozess gesteuert werden [17, S.35]. Das Modell zeigt die große Menge interner und externer Einflussfaktoren auf, wobei den internen besonders viel Aufmerksamkeit gewidmet wird. Hayes betont aber: "Schreiben ist ein kommunikativer Akt, der einen sozialen Kontext und ein Medium benötigt. Es ist auch eine schöpferische Aktivität, die Motivation erfordert, und eine intellektuelle Aktivität, die kognitive Prozesse und Gedächtnisleistung beansprucht."[23, S.60]

Entsprechend teilt sich das Modell in zwei Hauptkomponenten: Die Aufgabenumge-

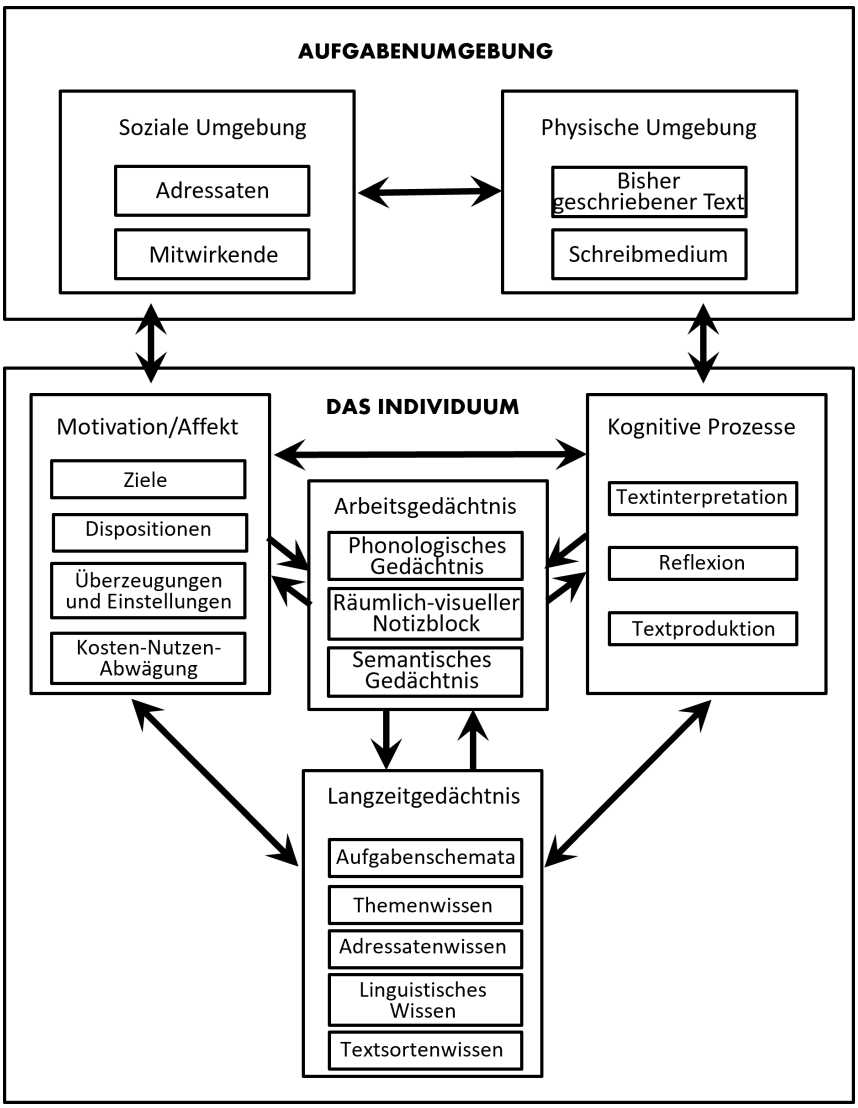

Abbildung 2.2: Modell des Schreibens als kognitiver Prozess [23, S.61]

bung und das (schreibende) Individuum.

Die Aufgabenumgebung besteht aus der sozialen Umgebung, in die die Schreibaufgabe eingebettet ist, und der physischen Umgebung. Schreiben ist in einen weiten sozialen Kontext eingebettet, beim wissenschaftlichen Schreiben beispielsweise in den institutionellen Kontext der Universität und in die wissenschaftliche Diskursgemeinschaft. Adressaten können die Leserschaft, aber auch Lehrende sein, die den Text bewerten werden. Insbesondere wenn es sich bei dem Text um eine Prüfungsleistung handelt, wird der große Einfluss der sozialen Umgebung sichtbar, wie wir in Kapitel 2.4 noch ausführlicher besprechen werden.

Die physische Umgebung ist die direkte Umgebung, in der geschrieben wird. Ihr wird großen Einfluss auf die kognitiven Prozesse beim Schreiben beigemessen. Abgesehen

von offensichtlichen Störfaktoren wie zum Beispiel Lärm, die in der Umgebung zu finden sein können, wird sie von zwei wichtigen Faktoren beeinflusst: Zum einen ist da der bisher geschriebene Text, der im Laufe des Prozesses immer wieder gelesen wird und als wichtige Inspirationsquelle für den unmittelbar anschließenden Text gilt. So gestaltet der Schreibprozess seine eigene Umgebung laufend neu, was er mit anderen kreativen Prozessen wie Malerei oder Programmierung gemeinsam hat [23, S.63].

Zum anderen hat auch das Schreibmedium einen wesentlichen Einfluss. Verschiedene Schreibmedien stellen verschiedene Werkzeuge für Schreibende zur Verfügung, die die Teilprozesse beim Schreiben unterschiedlich gut unterstützen. So weist Hayes etwa darauf hin, dass es beim Schreiben mit Stift und Papier einfacher ist, Skizzen, Pfeile oder Notizen einzubauen, während es mit dem Texteditor am Computer einfacher ist, Textblöcke zu verschieben oder mit Schriftarten und Formatierungen zu experimentieren [23, S.63].

Auf Seiten des Individuums spielen Motivation und Affekt, das Langzeitgedächtnis, das Arbeitsgedächtnis und die kognitiven Prozesse beim Schreiben eine Rolle. Das Langzeitgedächtnis und das Arbeitsgedächtnis stellen dabei kognitive Ressourcen dar. Aus dem Langzeitgedächtnis rufen Schreibende Wissen über eine Vielzahl beim Schreiben relevanter Themen ab. Dazu gehört linguistisches Wissen, zum Beispiel über Wortschatz, Grammatik und Textsorten; Wissen über die beschriebenen Inhalte und Themen; sowie Wissen über das Zielpublikum. Ist letzteres den Schreibenden bekannt, können sie die Gestaltung des Texts an dessen Vorlieben und Erwartungen anpassen, die den Schreibenden durch vergangene Interaktionen bekannt sind. Schreiben sie für ein unbekanntes Zielpublikum, sind sie gezwungen, selbst die Position der Adressaten einzunehmen und den Text so zu lesen, wie es die Lesenden tun würden. Dieser Perspektivenwechsel ist ein komplexer Akt, der selten bewusst vollzogen wird [23, S.81].

Das Arbeitsgedächtnis dagegen führt alle nicht automatisierten Handlungen aus, koordiniert und überwacht sie, und wird deswegen von allen Teilprozessen beim Schreiben beeinflusst. Das Parallele-Prozesse-Modell von Robert de Beaugrande [1] in Abbildung 2.3 veranschaulicht dabei, mit welcher kognitiven Belastung Schreibende umgehen müssen, was insbesondere bei ungeübten Schreibenden auch zu Überlastungen führen kann [20, S.21-24].

Ein weiterer wichtiger Faktor im Modell ist die Motivation. Die Motivation von Schreibenden ist höher, wenn sie an einer Aufgabe arbeiten, die sie interessiert oder herausfordert. Auch extrinsische Motivation spielt eine Rolle, etwa wenn der Text benötigt wird, um einen Kurs an der Universität zu bestehen. Schreibende verfolgen mit dem Schreiben daher häufig eine Vielzahl von Zielen. Diese können sich auf den Inhalt des Textes beziehen, etwa, dass sie ihre Gedanken mit dem Text, oder mit einer bestimmten Stelle, besonders gut ausdrücken möchten. Aber auch die Aufgabenumgebung kann die Ziele und Motivation beeinflussen, etwa wenn Schreibende einen bestimmten Adressatenkreis zufriedenstellen möchten. [23, S.66]

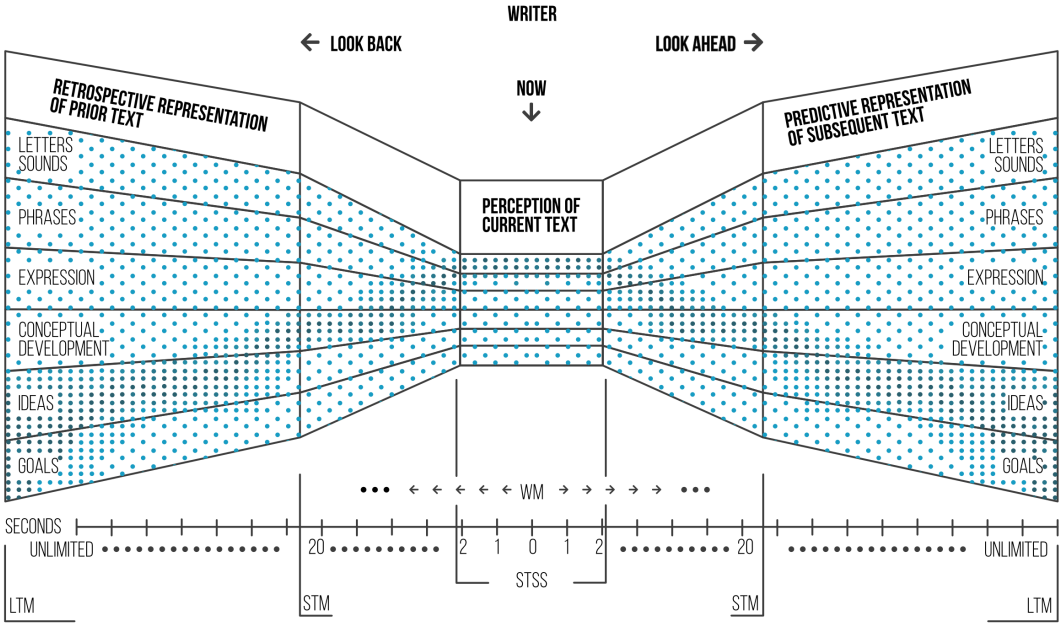

Abbildung 2.3: Parallele-Prozesse-Modell des Schreibprozesses [20, S.22]

Diese Ziele sowie bewusste oder unterbewusste innere Haltungen nehmen direkten Einfluss darauf, welche Strategien und Prozesse die Schreibenden für ihre Arbeit auswählen. So tendieren viele Schreibende zu einer Art 'Schreibtyp' und bevorzugen es etwa, sofort draufloszuschreiben, oder aber ihre Texte vorher sorgfältig zu planen (siehe dazu auch Kapitel 2.3). Aber nicht nur kurzfristige Reaktionen auf Ziele oder die Auswahl von Strategien sind in diesem Bereich relevant. Schreibende können langfristige Dispositionen und Einstellungen zum Schreiben entwickelt haben, die ihre Prozesse gravierend beeinflussen. So wird davon ausgegangen, dass Studierende, die Schreiben als Talent betrachten oder generell glauben, dass Leistung auf angeborenen und unveränderbaren Kompetenzen beruht, öfters Schreibangst entwickeln oder zu Verhalten neigen, mit dem sie möglichst vielen Schreibaufgaben aus dem Weg gehen wollen [23, S.66].

Als letztes finden sich beim Individuum die kognitiven Prozesse. In dem ursprünglichen Modell von 1980 zählen Hayes und Flower hier noch 'Planen', 'Übersetzen' (im Sinne des Übersetzens von Gedanken in Text) und 'Überarbeiten' auf. Im Modell von 1996 hat Hayes die Begriffe verallgemeinert: 'Textinterpretation' sind Prozesse, welche ausgehend von externem Input, wie zum Beispiel anderen Texten oder Bildern, eine interne Repräsentation dieses Inputs erzeugen. Zur Textinterpretation gehören also weitere Subprozesse, wie zum Beispiel Lesen, Zuhören oder das Erfassen von Grafiken. Dazu gehört auch das Lesen des bereits geschriebenen Textes.

Der Prozess der 'Reflexion' erzeugt ausgehend von einer vorhandenen internen Repräsentation weitere interne Repräsentationen. Reflektierende Prozesse sind zum Beispiel Problemlösung, Entscheidungsfindung und Schlussfolgern. Planung lässt sich ebenfalls hier einordnen und zwar als eine Methode zur Problemlösung, bei der Schritte entwickelt werden, um ein Ziel zu erreichen.

Der Prozess der 'Textproduktion' schließlich umfasst alle Prozesse, welche die internen Repräsentationen in wahrnehmbaren Output umwandeln, welcher in die Aufgabenumgebung transportiert wird. Der Output kann geschrieben oder auch gesprochen sein, etwa wenn der Text mit Hilfe eines Diktiergeräts produziert wird.

Zur Veranschaulichung der kognitiven Prozesse beim Schreiben kann hier wieder das Parallele-Prozesse-Modell von de Beaugrande hinzugezogen werden (Abbildung 2.3). Wie de Beaugrande betont auch Hayes die Gleichzeitigkeit der Prozesse: Sie laufen nicht hintereinander ab, sondern stehen den Schreibenden wie Werkzeuge zur Verfügung, die sie jederzeit benutzen und auch miteinander kombinieren können. Verschiedene Tätigkeiten beim Schreiben nutzen unterschiedliche kognitive Prozesse und diese unterschiedlich stark. So hat Hayes zum Beispiel gesonderte kognitive Modelle für die Tätigkeiten des Überarbeitens [23, S.73] und des Lesens [23, S.72] entworfen. Insbesondere das Lesen ist ein zentraler Prozess beim Schreiben, der oft unterschätzt wird. Beim Lesen konstruieren die Lesenden eine mentale Repräsentation des Textes, die nicht nur sprachliche, sondern auch räumliche Eigenschaften des Textes berücksichtigt. [23, S.74-76]

Während je nach Schreibaufgabe etliche fremde Texte gelesen und verstanden werden müssen, müssen Schreibende auch ganz unabhängig davon interne Repräsentationen ihres bereits geschriebenen Textes, der aktuell relevanten Textstelle und des weiterhin geplanten Textes konstruieren und handhaben. Das verlangt laut dem Modell von de Beaugrande insbesondere dem sensorischen Kurzzeitgedächtnis viel ab, dessen Ressourcen begrenzt sind, und ist ein weiterer Grund, weshalb es beim Schreiben zu kognitiver Überlastung kommen kann. [20, S.23f]

2.3 Individuelle Schreibprozesse

Wie bereits angesprochen können individuelle Schreibprozesse sehr unterschiedlich aussehen. Hayes und Flower beobachteten zwei Extreme bei den gewählten Strategien: Von Schreibenden, die möglichst schnell drauflosschreiben, zu solchen, die möglichst viel im Voraus planen, bevor sie den tatsächlichen Text ausformulieren. [17] Die Existenz dieses Spektrums scheint unter Autoren weitgehend bekannt zu sein und wird auch in nicht-wissenschaftlichen Kontexten rege diskutiert. Der Autor George R.R. Martin prägte zum Beispiel die Metapher von 'Gärtnern' und 'Architekten': Gärtner seien Autoren, die eine Idee 'anpflanzen' und schauen, wohin sie sich entwickelt, wohingegen Architekten den Aufbau des Texts schon ganz genau kennen, bevor sie das erste Wort setzen [34].

In der Schreibwissenschaft sind für diese beiden Extreme der Schreibstrategien auch als 'strukturschaffend' (die Struktur ergibt sich beim Schreiben) und 'strukturfolgend' (beim Schreiben wird einer vorher festgelegten Struktur gefolgt) bezeichnet [21, S.30]. Die wenigsten Autoren sind aber zu 100% in eines der Extreme einzuordnen und es ist üblich, dass Strategien kombiniert und vermischt werden.

Die Strategien, zu denen Schreibende sich hingezogen fühlen, werden manchmal auch als deren 'Schreibtyp' bezeichnet. Das sind jene Strategien, die sich bei den bisherigen Schreibprojekten der Schreibenden bewährt und vielleicht sogar automatisiert haben [21, S.31]. Um zu verdeutlichen, dass es sich dabei um erlernte und nicht etwa angeborene Verhaltensweisen handelt, wird in der Schreibwissenschaft häufig der Begriff 'Schreibstrategie' bevorzugt. [37, S.346-355]

Umfassende Arbeit zu den Schreibstrategien wurde von Hanspeter Ortner geleistet [37], der gleichzeitig ein scharfer Kritiker der bisher besprochenen Modelle ist. Neben der Reduzierung auf einen idealisierten Schreibprozess durch das Phasenmodell, ist Ortners größter Kritikpunkt die Trennung von Schreiben und Denken in den Modellen. Hayes' und Flowers kognitiver Prozess des 'Übersetzens' suggeriert in Ortners Augen, dass das Formen der Gedanken und das Niederschreiben derselben zwei grundsätzlich voneinander getrennte Prozesse seien [37, S.94-110]. Tatsächlich ist das Schreiben für erfahrene Schreibende auch eine Technik des Denkens. Sie entwickeln durch den Schreibprozess Gedanken und Wissen aktiv weiter.

Ortner stützt sich hierbei auf das Knowledge-Telling und Knowledge-Transforming Modell von Bereiter und Scardamalia [4]. Diese haben grundlegende Unterschiede in den Schreibstrategien von erfahrenen und unerfahrenen Schreibenden festgestellt: Unerfahrene Schreibende generieren Text, indem sie anhand von Stichworten aus der Aufgabenstellung oder dem bereits vorhandenen Text ihr Wissen dazu abrufen und es niederschreiben. Dieser Prozess ist das sogenannte Knowledge-Telling. [5, S.87-90]

Bei erfahrenen Schreibenden wird das Knowledge-Telling Teil eines umfassenderen Prozesses, des Knowledge-Transformings. Dieser Prozess generiert neues Wissen, indem unfertige Ideen immer wieder neu überdacht und umformuliert werden. Schreibende überprüfen beim Prozess des Knowledge-Transformings stetig, ob der Text das aussagt, was sie tatsächlich aussagen wollen, wodurch sie gleichzeitig diese Aussage reflektieren. Es entstehen zwei Problemräume: Einer für den Inhalt und einer dafür, wie der Inhalt transportiert wird. Im Spannungsfeld dieser beiden Problemräume wird neues Wissen geschaffen. [5, S.90-92]

Um solche erfahrenen Schreibenden, die das Knowledge-Transforming bewusst oder unbewusst betreiben, geht es Ortner in seiner Arbeit zu den Schreibstrategien [37, S.1-4]. Er untersuchte Selbstaussagen von professionellen Schreibenden zu ihren Prozessen und arbeitete anhand dieser zehn Schreibstrategien heraus. Die Strategien unterscheiden sich danach, in welcher Dimension und wie stark sie den Schreibprozess zerlegen. Im Folgenden werden die zehn Strategien kurz vorgestellt:

1. Flow-Schreiben: Es wird assoziativ und ohne Nachdenken drauflosgeschrieben, ohne einer festen Idee oder einem Thema zu folgen. [37, S.356-378]
2. Ein Text zu einer Idee: Es wird ebenfalls drauflosgeschrieben, doch bei dieser Strategie gibt es eine leitende Idee oder ein Thema, zu dem die Aufmerksamkeit immer wieder zurückkehrt. [37, S.391-408]
3. Mehrversionen-Schreiben: Hierbei wird wie bei der zweiten Strategie zunächst ein Text zu einer Idee geschrieben. Anschließend setzen die Schreibenden aber neu an und schreiben einen neuen Text zu derselben Idee. Das wird mehrfach wiederholt und der Text über die verschiedenen Versionen hinweg weiterentwickelt. [37, S.408-428]
4. Redaktionelle Arbeit am Text: Auch hier wird eine erste Version zügig runtergeschrieben. Zur finalen Version des Textes gelangen die Schreibenden dann aber durch sukzessives Überarbeiten jener ersten Version. [37, S.428-440]
5. Planendes Schreiben: Vor dem Schreiben des eigentlichen Textes wird ein organisierter Plan der Makrostruktur des Textes erarbeitet, zum Beispiel in einer Gliederung, einer Mind Map oder anderen Formen von Notizen. [37, S.440-462]
6. Ausarbeiten im Kopf: Diese Schreibenden entwickeln mehr als nur die Makrostruktur des Textes vorab, sondern sie formulieren ganze Sätze, Abschnitte oder sogar ganze Texte im Kopf, sodass das Schreiben oft nur noch ein 'Aufschreiben' dieser Gedanken ist. [37, S.462-484]
7. Schrittweises Vorgehen: Diese Schreibstrategie existiert nur rein in der Theorie und entspricht dem linearen Abarbeiten der Phasen im Schreibprozess wie in Kapitel 2.1 vorgestellt. [37, S.484-491]
8. Synkretistisches Schreiben: Chaotisches Vorgehen, das keiner erkennbaren Logik folgt. Diese Schreibenden beginnen den Schreibprozess an einer beliebigen Stelle des Textes und integrieren dabei alle bisher genannten Strategien. [37, S.491-540]
9. Textteil-Schreiben: Bei dieser Strategie werden zunächst einzelne Textteile wie Szenen oder Kapitel geschrieben, die noch nicht notwendigerweise zusammenhängen. Erst im Laufe des Prozesses wird die Handlung entwickelt, die diese einzelnen Textteile verbindet. [37, S.540-543]
10. Extreme Zerlegung: Eine Zuspitzung der beiden vorhergehenden Strategien. Die Schreibenden entwickeln einzelne Ideen zu unzusammenhängenden Textteilen, die sich oft nicht mehr zu einem kohärenten Text verbinden lassen. Was entsteht sind eher 'Gedankensplitter', einzelne Puzzleteile, die sich aber nicht zu einem fertigen Bild zusammenfügen. [37, S.543-564]

Ortners zehn Strategien sind die kleinteiligste Systematisierung dieser Art und erfassen daher auch Schreibtypen, die bei anderen Einteilungen keine Beachtung finden.

Ortner macht so die große Vielfalt an Strategien deutlich, was didaktisch sehr wertvoll sein kann. [47, S.187]

Je nach Kontext können andere Systematisierungen aber nützlicher sein. So hat sich in der Schreibberatung an Universitäten eine Einteilung in nur fünf Strategietypen bewährt [21, S.30]. Dies ist nicht weiter verwunderlich, wenn wir bedenken, dass die Strategie Nr. 7, das schrittweise Vorgehen, in der Praxis gar nicht existiert und einige weitere von Ortners Strategien für wissenschaftliche Texte, wie sie von Studierenden verlangt werden, grundsätzlich nicht geeignet sind (insbesondere Strategien 1 und 10, das Flow-Schreiben und die extreme Zerlegung). Die fünf relevantesten Schreibstrategien des wissenschaftlichen Schreibens sind nach Grieshammer et al. [21, S.30]:

- Spontanes Schreiben: Das klassische strukturschaffende Schreiben, bei dem drauflosgeschrieben und die Struktur des Textes beim Schreiben erarbeitet wird.
- Planendes Schreiben: Das klassische strukturfolgende Schreiben, bei dem zunächst ein Plan erarbeitet und dann befolgt wird.
- Mehrversionen-Schreiben: Entspricht Ortners Schreibstrategie Nr. 3. Dem finalen Text wird sich über viele Versionen angenähert.
- Redaktionelles Schreiben: Entspricht Ortners Schreibstrategie Nr. 4. Dem finalen Text wird sich durch sukzessives Überarbeiten eines ersten Entwurfs angenähert.
- Puzzle-Schreiben: Geht nach Lust und Laune vor und schreibt, plant oder überarbeitet Textteile nach Belieben. Entspricht einer Mischung aus Ortners Schreibstrategien Nr. 8 und 9.

In verschiedenen Kontexten und für verschiedene Schreibaufgaben sind einzelne Schreibstrategien also unterschiedlich gut geeignet. Schreibende bevorzugen aber individuell vor allem die Strategien, die sich im Laufe ihres Lebens in der Praxis bewährt haben. Die Frage ist nun, was passiert, wenn diese bewährten Strategien versagen.

2.4 Schreibblockaden

2.4.1 Definition von Schreibblockaden

Schreibblockaden entstehen dann, wenn die gewählten Strategien nicht zur gestellten Schreibaufgabe passen. Dieses heutige Verständnis von Schreibblockaden beruht zu wesentlichen Teilen auf der Arbeit von Mike Rose [43]. Er definiert Schreibblockaden als "die Unfähigkeit, mit dem Schreiben zu beginnen oder es fortzuführen, und zwar aus anderen Gründen als mangelnder Schreibkompetenz oder fehlendem Einsatz" [44, S.194].

Blockierte Schreibende haben also grundsätzlich die Motivation und die nötigen Kompetenzen, um einen zufriedenstellenden Text zu verfassen. Wenn sie es schaffen, eine Schreibaufgabe zu beenden, sind den fertigen Texten die Probleme bei der Entstehung nicht mehr anzusehen. Bei einer Blockade sind diese Kompetenzen nur aus verschiedenen Gründen nicht verfügbar. [44, S.193]

Das Ausmaß einer Blockade wird außerdem nicht danach bemessen, wie lange kein Text zu Papier gebracht wird, sondern "nach der Dauer der Zeit, in der die Produktivität beim Schreiben eingeschränkt ist" [44, S.194]. Eine Schreibblockade zeichnet sich also nicht dadurch aus, dass kein Text entsteht. Im Gegenteil: Wie Keith Hjortshojs Fallbeispiele anschaulich zeigen, produzieren blockierte Schreibende mitunter sehr viel Text, ohne dabei aber wesentliche Fortschritte bei ihrer Schreibaufgabe zu machen [26, S.220-222]. Umgekehrt können Schreibende auch viel Zeit damit verbringen, Ideen gedanklich zu bearbeiten, ohne dabei sofort Text zu Papier zu bringen, sind deswegen aber nicht blockiert (siehe auch Ortners Schreibstrategie Nr. 6 in Kapitel 2.3).

Charakteristisch für Schreibblockaden sind außerdem negative Gefühle wie Frustration, Wut oder Verwirrung, sowie unproduktive Verhaltensweisen wie das Verpassen von Abgabeterminen [44, S.194]. Blockaden können sich verfestigen, sodass sie nicht nur eine Schreibaufgabe betreffen, sondern sich durch das generelle Schreibverhalten der Schreibenden ziehen[1]. Oft bestehen sie daher zu dem Zeitpunkt, an dem die Betroffenen sich Hilfe suchen, schon seit längerer Zeit [26, S.217].

Für seine Untersuchung von Schreibblockaden wendet Rose eine mehrschichtige Methodik an. Er erarbeitet zunächst einen Fragebogen, nach dessen Auswertung er zehn Studierende auswählt, die er aufgrund ihrer Antworten entweder als sehr stark oder sehr geringfügig beziehungsweise nicht blockiert einstuft. Anschließend filmt er die ausgewählten Studierenden beim Schreiben eines Essays und sammelt Selbstaussagen zu ihrem Schreibprozess mit der stimulated Recall-Methode. Bei dieser führt er ihnen das aufgenommene Video vor und hält an markanten Stellen an, um sie gezielt zu ihrem Verhalten zu befragen.

Mit Hilfe dieser Methodik arbeitet Rose sechs Gründe für Schreibblockaden heraus:

1. Zu starre, falsche oder nicht angemessen angewandte Regeln,
2. irreführende Vorannahmen über das Schreiben,
3. zu frühes Überarbeiten,

[1] Je nach Schwere der Blockade werden manchmal unterschiedliche Begriffe verwendet. So unterscheiden Grieshammer et al. zum Beispiel in 'Schreibprobleme', 'Schreibhemmungen' und 'Schreibblockaden', wobei sie mit 'Blockaden' ausschließlich ein Stadium des Problems meinen, in dem die Betroffenen zwingend psychologische Hilfe benötigen [21, S.74]. Auf eine solche Unterscheidung soll im Rahmen dieser Arbeit verzichtet werden, wodurch auch weniger schwere Probleme als 'Blockaden' bezeichnet werden, sofern sie der oben genannten Definition von Mike Rose entsprechen.

4. fehlende, unflexible oder unpassende Planungs- und Argumentationsstrategien,
5. widersprüchliche Regeln, Vorstellungen oder Planungsstrategien,
6. unangemessene oder falsch verstanden Kriterien zur Bewertung des eigenen Schreibens. [44, S.194f]

Alle Schreibenden haben eine eigene Vorstellung davon, wie ein Schreibprozess abläuft. Als Orientierungspunkte dienen ihnen dafür aber oft fertige, veröffentlichte Texte, was die Vorstellung vom Schreibprozess verzerren kann: "Wenn sie fertiggestellt sind, erzeugt der beste Text (wie bei jeder gelungenen Vorführung) eine Illusion von Leichtigkeit und spontaner Brillanz. Wir stellen uns vor, dass brillantes Schreiben direkt aus brillanten Gedanken und Köpfen geflossen ist." [26, S.224]

Hinzu kommt, dass Schreibregeln oft als absolut verstanden und auch vermittelt werden. Dabei kommt jede Regel mit ihrem eigenen historischen und soziologischen Hintergrund und ist daher in manchen Situationen angemessen, in anderen aber nicht. [44, S.201f] Wenn dieser Kontext nicht mit vermittelt wird, können Schreibende im Versuch scheitern, Regeln in unangemessene Situationen anzuwenden, oder es sammeln sich Regeln zu einem Thema an, die sich aber widersprechen.

Ein Beispiel: Beim wissenschaftlichen Schreiben gibt es viele stilistische Regeln, welche die Möglichkeiten, sich auszudrücken, deutlich einschränken können. So soll häufig sowohl 'ich' als auch 'man' vermieden werden, ebenso wie Passivkonstruktionen. Anthropomorphisierung, also menschliche Eigenschaften und Handlungen nicht-menschlichen Objekten zuzuschreiben, gilt aber ebenso als schlechter Stil. Damit sind bereits folgende Ausdrücke ausgeschlossen oder 'schlecht', um den Inhalt einer Arbeit zu erklären:

- In dieser Arbeit untersuche ich X.
- In dieser Arbeit erfährt man X.
- In dieser Arbeit wird X untersucht.
- Diese Arbeit untersucht X.

Versuchen Schreibende sich an all diese Regeln gleichzeitig zu halten, ohne zu wissen, warum diese Regeln existieren, beziehungsweise was diese Regeln eigentlich bezwecken sollen, können sie schnell zu dem Glauben gelangen, dass in wissenschaftlichen Arbeiten eigentlich gar nichts erlaubt ist, was sich zu einer langfristigen negativen Einstellung entwickeln kann.

Aus Roses Theorie lässt sich eine Verfahrensweise zum Auflösen von Blockaden ableiten [26, S.218]:

1. Die nicht funktionierenden Regeln und Strategien identizifieren,

2. die Gründe dafür diskutieren, weshalb die Strategien nicht (mehr) funktionieren,
3. alternative Ansätze empfehlen.

Oder kurz: Die ungeeigneten Strategien der Schreibenden durch bessere ersetzen [28, S.27].

2.4.2 Blockaden beim wissenschaftlichen Schreiben

Gisbert Keseling [28] erarbeitet mit Hilfe von gesammelten Daten aus der Schreibberatung, dass es nicht eine universelle Form der Schreibblockade gibt. Jede Diagnose ist anders, allerdings lassen sich gewisse Gruppen von Blockaden erkennen. Keseling arbeitet im Kontext des wissenschaftlichen Schreibens von Studierenden fünf Gruppen von Blockaden heraus, die er in zwei gröbere Kategorien einteilt: Blockaden bei der Konzeptbildung und adressatenbezogene Blockaden. Ein Überblick über die beiden Kategorien und die dazugehörigen Blockadengruppen findet sich in Abbildung 2.4.

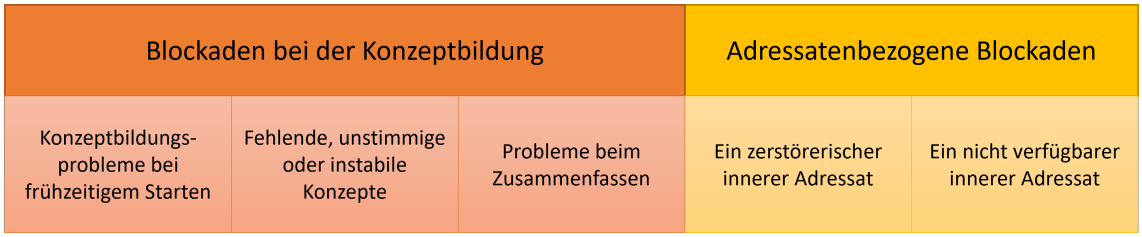

Abbildung 2.4: Die fünf häufigsten Blockaden beim wissenschaftlichen Schreiben nach Keseling [28]

Blockaden bei der Konzeptbildung

Bei diesen Blockaden gelingt es den Schreibenden nicht, vor oder während des Schreibens ein angemessenes Konzept für den Text zu finden. Keseling schlägt vor, dass sich diese Art von Blockaden auf einen Nenner bringen lassen, indem wir Texte aus der Sicht der Gestalttheorie betrachten [28, S.103-107]. Keseling geht davon aus, dass einem guten Text, ergo einem Text mit einer guten Gestalt, sogenannte 'Vorgestalten' vorausgehen müssen und dass diese wiederum gut oder schlecht sein können. Schlechte Vorgestalten führen zu Blockaden bei der Konzeptbildung. Diese Schreibenden haben also eine mentale Repräsentation des zu schreibenden Texts entwickelt, mit der etwas nicht stimmt: Sie ist entweder unvollständig oder einzelne Teile des Bildes passen nicht zusammen. Zu diesen Blockaden zählt Keseling drei Subgruppen:

- **Konzeptbildungsprobleme bei frühzeitigem Starten:** Diese Blockaden betreffen typischerweise Schreibende, die an der Tätigkeit des Schreibens sehr

viel Freude empfinden, sich aber weniger mit notwendigen Vorarbeiten anfreunden können. Deswegen beginnen sie sofort mit dem Schreiben, brechen den Prozess aber irgendwann ab, entweder aus Ideenmangel oder weil ihnen die Schwächen im eigenen Text auffallen, sie aber nicht die Geduld zum Ändern oder Neuschreiben aufbringen können. [28, S.54-69]

- **Fehlende, unstimmige oder instabile Konzepte:** Diese Blockaden sind oft damit verbunden, dass erst sehr spät mit dem Schreiben der Rohfassung begonnen wird. Den Betroffenen gelingt es nicht, ihr Thema angemessen einzugrenzen oder eine passende Fragestellung zu finden. Sie wenden unpassende Textmuster für die Aufgabe an oder haben damit zu kämpfen, dass ihre ursprünglichen Konzepte nicht zu späteren Erkenntnissen passen, was große Änderungen nötig machen würde. [28, S.70-96]
- **Probleme beim Zusammenfassen:** Symptomatisch für diese Blockaden sind Texte, die Zitatesammlungen gleichen, da die Schreibenden sich nicht vom originalen Wortlaut ihrer Quellen lösen können. Oft scheitern sie aber bereits daran, Texte auf ihre wesentlichen Inhalte zusammenzufassen oder scheuen komplett den Prozess des Lesens. [28, S.96-103]

Adressatenbezogene Blockaden

Diese beiden Gruppen von Blockaden treten nicht bei der Konzeptbildung, sondern beim Formulieren auf:

- **Ein zerstörerischer innerer Adressat:** Schreibende, die von dieser Art Blockade betroffen sind, haben häufig große Probleme damit, überhaupt Text zu Papier zu bringen, was von der Angst begleitet wird, irgendetwas falsch zu machen. Keseling führt diese Symptome darauf zurück, dass die Schreibenden nicht in der Lage sind, sich vorzustellen, dass sie für eine interessierte Leserschaft schreiben. Stattdessen haben sie ihre Dozenten vor Augen, die in ihren Augen ja aber bereits alles zu dem beschriebenen Thema wissen und die Arbeit nur lesen, um sie zu bewerten. [28, S.108-120]
- **Ein nicht verfügbarer innerer Adressat:** Diese Blockadengruppe nimmt eine etwas ungewöhnliche Stellung in Keselings Kategorisierung ein, denn es handelt sich hier eigentlich nicht um kognitive Blockaden. Die betroffenen Schreibenden waren im Kontext einer Schreibgruppe häufig ohne Probleme in der Lage, Text zu produzieren, konnten dieselbe Produktivität aber nicht zu Hause ohne unmittelbar anwesende Leserschaft abrufen. Im Gegensatz zu den Blockaden mit dem zerstörerischen inneren Adressaten, bei denen die Betroffenen nicht in der Lage sind, einen wohlwollenden inneren Adressaten zu konstruieren, scheinen diese Schreibenden überhaupt keinen inneren Adressaten konstruieren zu können. Keseling stellt außerdem fest, dass die Betroffenen oft durch Kindheitserfahrungen geprägt waren, bei denen Bezugspersonen in

> entscheidenden Momenten nicht für sie verfügbar waren. Es scheint sich daher eher um eine psychische als um eine kognitive Blockade zu handeln, die sich durch Änderungen am Schreibverhalten auch nur sehr schwer auflösen lässt. [28, S.120-132]

Aus den Beschreibungen dieser Blockadegruppen wird ersichtlich, dass sie eng an den Kontext des wissenschaftlichen Schreibens von Studierenden gebunden sind, da Keseling nur aus diesem seine Daten bezog. Es ist daher davon auszugehen, dass sie sich nicht eins zu eins auf andere Kontexte übertragen lassen. Schon das literarische Schreiben könnte deutlich andere Blockaden hervorbringen, auch wenn sich sicherlich einige in den hier beschriebenen Gruppen einordnen lassen dürften.

In einer weiteren Studie vergleicht Keseling das Vorgehen der Studierenden mit dem von erfahrenen Wissenschaftlern und Wissenschaftlerinnen. Dabei stellt er fest, dass in deren Schreibprozessen durchaus ähnliche Probleme auftreten können, wie zum Beispiel instabile Konzepte oder dass die Vorarbeiten nicht ausreichend sind. Im Gegensatz zu den Studierenden sind sie aber in der Lage, angemessen auf diese Probleme zu reagieren, weshalb diese sich nicht zu Blockaden entwickeln. [28]

3 Der Programmierprozess

Die Erforschung des Programmierprozesses ist im Gebiet des Empirical Software Engineerings anzusiedeln. In diesem Forschungsfeld ist das Ziel, die Konstruktion von Software besser verstehen zu lernen. Zu diesem Zweck werden sowohl die Produkte des Software Engineerings, wie Code und Dokumentationen, als auch die Prozesse, die zu deren Entstehung führen, untersucht. Gerade letztere entziehen sich aber oft dem Zugriff der Forschenden, weshalb der Diskurs in diesem Bereich weniger schnell Fortschritte macht. [35]

Aus einer industriellen Perspektive verspricht die Erforschung der Prozesse beim Programmieren eine Verbesserung und Beschleunigung besagter Prozesse und damit der Produktivität bei der Entwicklung [18, 31]. Aus didaktischer Perspektive stellt sich oft die Frage, wie sich Programmieren am besten lehren lässt, da insbesondere Anfängerkurse eine hohe Durchfallrate aufweisen und Programmieren daher als sehr schwierig zu lernen gilt [41, S.330]. Einige Stimmen fordern daher mehr Prozessorientierung bei der Lehre des Programmierens [3, 12, 14, 19].

Dadurch wurden einige Modelle von Programmierprozessen entwickelt, die explizit als Anleitungen für Lernende zu verstehen sind. So zum Beispiel die 'Seven Steps', welche eine Strategie zum Übersetzen eines Problems in Code darstellen [25] oder 'STREAM', eine vereinfachte Kombination aus mehreren Methoden des Software Engineerings zum Programmieren einzelner Klassen [12]. Während diese modellierten Prozesse zwar die Schritte, die zur Erstellung eines Programms notwendig sind (Verstehen des Problems, Entwurf und Implementierung einer Lösung, Überprüfen der Lösung), grob aufzeigen, so sind sie für die Anwendung in einem streng limitierten Rahmen gedacht und daher nicht in der Lage, die Prozesse, die hinter der Entstehung komplexer Software stehen, zu beschreiben.

3.1 Prozesse des Software Engineerings nach Sommerville

Bei der Entwicklung komplexer Software ist der Prozess der Programmierung in die umfassenderen Prozesse des Software Engineerings eingebunden. Jedes Softwareprojekt muss dabei einige grundlegende Prozesse durchlaufen. Ian Sommerville nennt hier die Softwarespezifikation, die Softwareentwicklung, die Softwarevalidierung und Softwareevolution [49, S.31]. Ein Überblick über diese vier Prozesse und ihre Sub-

prozesse, welche wir in diesem Kapitel erläutern werden, findet sich in Abbildung 3.1.

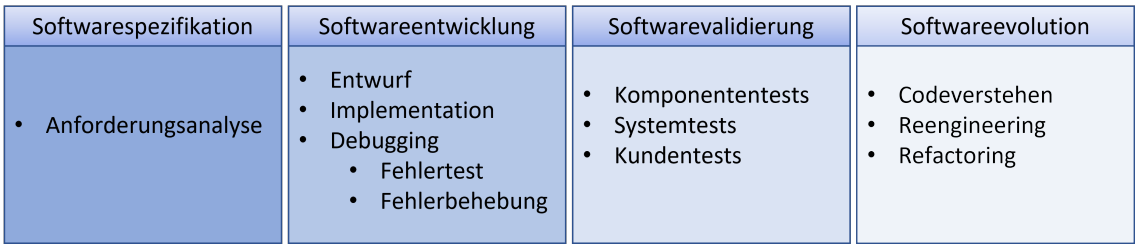

Abbildung 3.1: Übersicht der Softwareprozesse nach Sommerville [49]

Abhängig von der Art der Software, die damit gebaut wird, können diese Prozesse in unterschiedliche Vorgehensmodelle integriert sein. Eines der bekanntesten Vorgehensmodelle ist das Wasserfallmodell, bei dem die Prozesse in möglichst strikt voneinander getrennten Phasen linear abgearbeitet werden [49, S.57-60]. Bei anderen Vorgehensmodellen, wie der inkrementellen oder der agilen Softwareentwicklung, werden die Prozesse mehr vermischt oder parallel zueinander durchlaufen [49, S.60-62].

Die Softwarespezifikation ist ein Prozess, bei dem die von Kunden und System verlangten Funktionen und Beschränkungen verstanden und definiert werden sollen. Der Prozess wird daher auch als Anforderungsanalyse oder als Requirements-Engineering bezeichnet. Entstehen sollen dabei detaillierte Anweisungen, welche Funktionen umgesetzt werden müssen. Dazu können auch bereits Prototypen und Modelle des geplanten Systems entstehen. [49, S.65-67] Bei der agilen Softwareentwicklung hat sich das Konzept der User-Storys bewährt, bei denen die Bedürfnisse der Kunden in Form von kurzen, hypothetischen Nutzungsszenarien erklärt werden [49, S.93-95].

Der Prozess der Softwareentwicklung umfasst Entwurf und Implementierung der Software. Der Entwurfsprozess kann je nach Aufgabe und entwickelnder Person sehr unterschiedlich ausgeprägt sein. Es können detaillierte Entwurfsdokumentationen zur Architektur des Systems, zu Datenbanken, Schnittstellen oder einzelnen Komponenten entstehen. Oftmals bestehen Entwürfe aber auch nur aus informellen Notizen im Code, auf Whiteboards oder im Kopf der entwickelnden Personen.

Die Implementierung teilt sich weiter auf in die Prozesse der Programmierung und des Debuggings. Zur Programmierung verliert Sommerville nicht viele Worte, sie "ist eine persönlich gestaltete Aktivität [...] Manche Programmierer fangen mit Komponenten an, die sie bereits vollständig verstehen [...] Andere gehen den entgegengesetzten Weg" [49, S.69].

Während der Programmierung testen Entwickler ihren Code für gewöhnlich bereits auf dessen Funktionsfähigkeit und beheben Fehler, die sie finden. Die kombinierten

Prozesse des Fehlertestens und der Fehlerbehebung nennt man Debugging. [49, S.67-69]

Bei der Softwarevalidierung soll sichergestellt werden, dass die entwickelten Komponenten allen Anforderungen entsprechen. Dazu werden sie für gewöhnlich separat getestet, bevor sie in das System integriert und ihre Funktionstüchtigkeit dort überprüft wird. Schließlich prüfen die Kunden, ob alles zu ihrer Zufriedenheit ist und das System in der geplanten Einsatzumgebung funktioniert. [49, S.70-71]

Die Softwareevolution umfasst Prozesse, die den Fortbestand und die Funktionstüchtigkeit des Softwaresystems langfristig sicherstellen sollen. Das kann die Weiterentwicklung der Software durch die Implementation neuer oder die Änderung bestehender Funktionen bedeuten. Am Anfang steht daher bei jedem Prozess der Softwareevolution ein umfassender Prozess des Codeverstehens. Software-Reengineering ist ein Prozess, bei dem alte Software modernisiert wird, zum Beispiel durch Überarbeiten der Systemarchitektur, Übersetzen in eine neue Programmiersprache, oder Neuerstellung der Systemdokumentation. Dazu gibt es den Prozess des Refactorings, mit dem der Degradierung des Systems durch Änderungen vorgebeugt werden soll. Beim Refactoring wird das System überarbeitet, um bessere Verständlichkeit, geringere Komplexität und verbesserte Strukturen zu erhalten, ohne aber an der Funktionalität etwas zu verändern. [49, S.292-315] Insbesondere bei agiler Softwareentwicklung ist Refactoring während der Implementierung bereits ein relevanter Prozess, bei dem mögliche Verbesserungen am Code sofort umgesetzt werden sollen, sobald sie entdeckt werden. [49, S.95-96]

3.2 Kognitive Prozesse beim Programmieren

Die Umsetzung der Prozesse beim Software Engineering durch die Menschen, die die Software entwickeln, beansprucht bei diesen eine Vielzahl kognitiver Ressourcen. Basierend auf Wangs Ebenenmodell [53] und der Taxonomie von Bloom [6] haben Renumol et al. [40] insgesamt 42 kognitive Prozesse identifiziert, die während des Programmierens relevant sind. Dabei wurde festgestellt, dass es keinen Unterschied macht, ob es sich bei den Programmierenden um effektive oder ineffektive Programmierende handelt, die Zahl der relevanten Prozesse bleibt gleich. Eine Vielzahl dieser Prozesse konnte von Darnstaedt et al. bestätigt werden [13]. Die beiden Studien unterscheiden sich deutlich in ihrem Kontext (Programmierung in C vs. Programmierung eines Industrieroboters), was auf eine gewisse Allgemeingültigkeit der Ergebnisse schließen lässt, trotz der geringen Zahl an Teilnehmenden.

Der Großteil der Forschung zu kognitiven Prozessen beim Programmieren konzentriert sich aber auf das Programmverstehen, wie eine Metastudie von Begum zeigt [2]. Grundsätzlich wird davon ausgegangen, dass Lesen und Schreiben von Code zwei unterschiedliche Tätigkeiten sind, welche sich bei Lernenden weitgehend unabhängig voneinander entwickeln [56]. Der höhere Fokus auf dem Programmverstehen (also

dem Lesen von Code) lässt sich dadurch erklären, dass es auch in der Praxis mehr Zeit einnimmt. Eine Untersuchung von Minelli et al. [36] ergab, dass professionelle Software-Entwickler bis zu 70% ihrer Zeit mit Programmverstehen verbringen.

Neben dem Fokus auf Programmverstehen offenbart die Metastudie einige weitere interessante Aspekte der Kognition beim Programmieren. Zum einen geht sie auf die internen und externen Repräsentationen von Software beim Programmieren ein. Wie auch beim Schreiben von Texten müssen Menschen, die Software entwickeln, auch eine mentale Repräsentation dieser Software konstruieren. Wie genau diese Repräsentationen aussehen, wurde bisher in nur wenigen Studien untersucht. [2] Petre und Blackwell befragten Programmierexperten zu ihren mentalen Repräsentationen, nachdem sie gedanklich eine Lösung für ein Programmierproblem designen sollten, ohne sich dabei an die Beschränkungen durch Code zu halten. Die mentalen Repräsentationen, die dabei beschrieben wurden, waren mehrdimensional, beinhalteten neben visuellen auch auditive und andere Sinneswahrnehmungen, waren dynamisch und konnten von den Entwicklern gedanklich frei bewegt und modifiziert werden [39].

Das Konzept von internen oder mentalen Repräsentationen wird auch im Bezug auf Lernende beim Programmieren angewendet, wobei hier häufig auch von Modellen statt Repräsentationen gesprochen wird. Lernende müssen eine interne Repräsentation davon entwickeln, wie ein Computer Programme ausführt, um das Verhalten von Programmen nachvollziehen zu können. Tun sie das nicht, kann es im Lernprozess zu Problemen kommen. Auch das Handhaben mehrerer interner Repräsentationen beim Schreiben von Programmen, nämlich der, wie das Programm derzeit aussieht, und der, wie das Programm eigentlich aussehen sollte, kann Lernende vor große Schwierigkeiten stellen. [41]

Bei der Konstruktion mentaler Repräsentationen werden Programmierende aller Fähigkeitsstufen von Werkzeugen wie Debuggern unterstützt. Petre stellte in einer Folgestudie zu den mentalen Repräsentationen bei Experten allerdings fest, dass bestehende Werkzeuge zur Visualisierung von Software oft nicht in der Lage sind, die mentalen Repräsentationen adäquat wiederzugeben, weshalb sie von Experten nicht genutzt werden [38].

Ein weiterer interessanter Aspekt, den die Metastudie von Begum zum Vorschein bringt, ist das Konzept von verteilter Kognition innerhalb eines Entwicklerteams, welches in einer anderen Meta-Studie von Lavallée et al. [32] näher beleuchtet wird. Hierbei wird davon ausgegangen, dass die kognitiven Prozesse, die zum Lösen eines Problems notwendig sind, auf eine Gruppe von Menschen aufgeteilt werden können. Die Meta-Kognition eines Teams, also zu wissen, was die anderen Teammitglieder wissen, ist essenziell für die Zusammenarbeit und die Konstruktion einer teamübergreifenden mentalen Repräsentation der zu bearbeitenden Aufgaben und infolgedessen für das Gelingen der Aufgaben. Während es einige Fallstudien und Vorschläge für Tools zur Verbesserung der Teamarbeit gibt, ist insbesondere die Aufteilung der kognitiven Belastung im Team noch nicht ausreichend untersucht.

4 Related Work: Vergleiche von Schreiben und Programmieren

Schreiben und Programmieren können auf mehr als einer Ebene miteinander verglichen werden. Annette Vee betrachtet Schreiben und Programmieren als verwandte Technologien: Beides sind symbolbasierte Systeme zur Codierung von Information, welche mit Schrift arbeiten und einer fixen Grammatik folgen. Sie sind außerdem beides Kulturtechniken, welche durch eine soziale Umgebung geformt und interpretiert werden und mit deren Hilfe neue Ideen konstruiert werden können. [51, S.95-98]

Beides sind kreative, kompositorische Prozesse, bei welchen eine endliche Menge von Symbolen auf unendlich viele Arten und Weisen zusammengesetzt wird, um damit Ideen auf einem höheren Level zu kommunizieren, und die beide auf ihre Art das menschliche Denken formen. [16, 24, 54]

Auch der Vergleich auf der Ebene des Arbeitsprozesses ist keine neue Idee. Bereits 1989 argumentierte Rex E. Gantenbein für eine Herangehensweise in der Lehre des Programmierens, welche von der prozessorientierten Schreibdidaktik und ihrem Prinzip, den Schreibprozess in Phasen zu unterteilen, inspiriert ist. [19]

Hermans und Aldewereld [24] vergleichen die beiden Prozesse ebenfalls mit Hilfe von Phasenmodellen, beschränken sich dabei allerdings auf eine Perspektive, die beide Prozesse als linear darstellt. Ziel ihres Vergleichs ist es, Ideen zu generieren, wie Schreiben und Programmieren von dem Wissen aus dem jeweils anderen Feld profitieren können. Sie zeigen dabei mehrere Punkte auf, bei denen das der Fall sein könnte:

- Das Phasenmodell des Schreibprozesses, welches die Autoren nutzen, beinhaltet die beiden Phasen 'Stilisieren' und 'Formatieren' des Textes, für welche es in dem zum Vergleich herangezogenen Phasenmodell des Programmierens kein Äquivalent gibt. Die Autoren argumentieren, dass klare Definitionen für die Begriffe Stil und Formatierung in Bezug auf Code unter anderem die Lehre von Code Conventions vereinfachen könnte.
- Programmierende erhalten früher Feedback von ihrer Umgebung, ergo dem Compiler und dem Computer, während Schreibende oft lange Zeit nur sich selbst als Leserschaft haben. Schreibende könnten daher von Technologien, die mehr die Position eines Lesenden einnehmen, profitieren, oder auch von

Konzepten wie Pull Requests und formalen Code Reviews.

- Die Einteilung von Schreibenden in 'Plotter' und 'Pantser' (ähnlich der Einteilung in 'Architekten' und 'Gärtner' wie in Kapitel 2.3 besprochen) empfinden die Autoren auch für das Programmieren als naheliegend und werfen den Gedanken auf, dass das klassische Software Engineering deutlich mehr auf strukturfolgende 'Plotter' ausgelegt ist, als auf strukturschaffende 'Pantser', und welche Konsequenzen das haben könnte.

Der Vergleich von Hermans und Aldereweld hat einige Schwächen. Zum einen, wie von den Autoren selbst bereits angesprochen wurde, nutzen sie für den Vergleich ein Phasenmodell, welches sie auch explizit nur linear betrachten. Die Probleme von Phasenmodellen wurden in Kapitel 2.1 bereits besprochen, nämlich dass sie einen idealisierten und realitätsfernen Prozess abbilden. Der Vergleich bleibt dadurch sehr oberflächlich. Zum anderen wirkt auch die Auswahl der zum Vergleich herangezogenen Modelle recht willkürlich, die Begründung scheint sich darauf zu beschränken, dass beide Modelle genau sieben Schritte beinhalten. Schreibprozesse können sich je nach Aufgabe aber drastisch unterscheiden, wie ebenfalls bereits in Kapitel 2.1 angesprochen. Für Programmierprozesse gilt dasselbe, wie die verschiedenen angesprochenen Modelle in Kapitel 3 zeigen. So ist das Fehlen von Schritten im Programmierprozessmodell, die dem 'Stilisieren' und 'Formatieren' im Schreibprozessmodell entsprechen, zwar auffällig, liegt aber auch darin begründet, dass etwa der Prozess des Refactorings im gewählten Programmierprozessmodell keine Beachtung findet.

Trotzdem sind die Punkte, die Hermans und Aldereweld herausarbeiten, interessant. Der erste Punkt zu Stil und Formatierung beleuchtet das Problem, dass es für Programmcode mindestens zwei verschiedene Adressaten gibt, nämlich die Maschine und die Menschen. Für die Maschine ist es grundsätzlich egal, wie der Code aussieht, solange sie ihn ausführen kann. Deswegen argumentiert Alvaro Videla [52] in seinem Paper, in dem er die Frage stellt, was wir als Programmierende von Literaturtheorie lernen können, dass die beiden Adressaten Maschine und Mensch dem von Umberto Eco [15] umrissenen Konzept von 'Lesenden auf zwei Ebenen' entsprechen. Lesende auf der ersten Ebene sind zunächst nur an der Geschichte interessiert, also daran, *was* passiert und wie die Geschichte ausgeht. Auf dieser Ebene befindet sich auch die Maschine[1] im Bezug auf Code. Auf der zweiten Ebene dagegen befinden sich Lesende, welche daran interessiert sind, zu verstehen, wie das, was in der Geschichte passiert, präsentiert wird. Diese Lesenden bezeichnet Eco als 'semiotische' oder 'ästhetische' Lesende. Videla argumentiert, dass es sich bei den menschlichen Adressaten von Programmcode um Lesende auf der zweiten Ebene handelt, da auch sie daran interessiert sind, zu verstehen, *wie* das, was im Code passiert, funktioniert. Ein konkreter Code ist im Endeffekt nur *eine* mögliche Repräsentation der Lösung, die der Autor des Codes gefunden hat, weswegen Lesende des Codes nicht nur diese

[1] Wir nutzen hier den allgemeinen Begriff 'Maschine' anstatt Compiler oder Computer, da Videla in seinem Text auch anspricht, dass die Bedürfnisse von Compiler und Computer sich unterscheiden können.

Lösung, sondern auch ihre Präsentation rekonstruieren müssen. Allerspätestens dann, wenn es die Aufgabe dieser Lesenden ist, diese Präsentation durch Refactoring zu verbessern.

Das führt uns auch zum zweiten Punkt von Hermans und Aldereweld, nämlich dem Feedback, das Programmierende durch die Maschine schon viel früher erhalten, als Schreibende von Texten, die an Menschen gerichtet sind. Das ist zum Beispiel auch, was Hassenfeld und Bers für den Grund dafür halten, weswegen Kinder den Prozess des Debuggings von Code deutlich lieber angehen, als den Prozess des Überarbeitens von Text [22]. In ihrer Studie betrachten sie die Fallbeispiele von zwei Schülerinnen, die zur gleichen Zeit kreatives Schreiben und Programmieren in ScratchJr lernen, um mit Hilfe der beiden Medien Geschichten zu erzählen. Die Schülerinnen verbringen im Schnitt nur 5,7% ihres Schreibprozesses mit dem Überarbeiten ihrer Texte, während Debugging bis zu 20% ihres Programmierprozesses ausmacht. Zudem nehmen sie häufig gar keine Änderungen an ihren Texten vor, selbst wenn sie von den Lehrerinnen zum lauten Vorlesen ermutigt werden. Dagegen scheinen sie keine Probleme damit zu haben, von sich aus Debuggingprozesse für ihre Programme einzuleiten.

Während es also bereits einige Vergleiche zwischen Programmieren und Schreiben, auch auf Prozessebene, gibt, sind sie bisher eher oberflächlich oder fokussieren sich auf einzelne Aspekte, bei denen der Gesamtprozess eine untergeordnete Rolle spielt. Ziel dieser Arbeit ist also zunächst, diese Defizite aufzuholen und einen umfassenden Vergleich zwischen Schreib- und Programmierprozessen zu ziehen, der als Grundlage dienen kann für die Frage, ob es kognitive Blockaden beim Programmieren gibt.

5 Zwischenfazit

5.1 Zusammenfassung der Theorie

In Kapitel 2 haben wir die Prozesse des Schreibens anhand von Phasen- und kognitiven Modellen kennengelernt. Ferner haben wir besprochen, dass Schreibprozesse sich individuell stark unterscheiden können, weswegen auch von verschiedenen Schreibtypen gesprochen wird. Das zeigt sich auch in den Blockaden, zu denen es beim wissenschaftlichen Schreiben kommen kann: So kann es bei der Konzeptbildung zu unterschiedlichen Blockaden kommen, je nachdem, ob es sich bei den Schreibenden um 'Pantser' oder 'Plotter' handelt. Blockaden einer zweiten Kategorie können bei der Konstruktion eines inneren Adressaten entstehen. Experten sind in der Lage, solche Blockaden zu umgehen, da sie mit angemessenen Strategien auf Probleme reagieren.

Die in Kapitel 3 vorgestellten Prozesse beim Programmieren zeigen einige Ähnlichkeiten zum Schreiben: Beides macht von kognitiven Prozessen Gebrauch, bei denen mentale Repräsentationen des Textes, beziehungsweise der Software, konstruiert werden. Diese mentalen Repräsentationen müssen im Laufe weiterer Prozesse externalisiert werden.

Es gibt aber auch einige wesentliche Unterschiede zwischen Schreiben und Programmieren, welche in der vergleichenden Literatur, die wir in Kapitel 4 besprochen haben, deutlich werden. Diese liegen vor allem in der Unterscheidung zwischen menschlichen und maschinellen Adressaten und dem direkten Feedback, welches letztere zur Verfügung stellen.

5.2 Begriffserklärungen

Die Begriffe aus der Schreibprozessforschung neigen dazu, etwas ungenau zu sein, da sie sich überlappen oder verschiedene Konzepte mit demselben Wort bezeichnen. Innerhalb des Feldes mag das noch funktionieren, doch spätestens wenn zwei verschiedene Forschungsfelder aufeinandertreffen, ist ein wenig Aufräumarbeit nötig. An dieser Stelle soll daher geklärt werden, welche Begriffe im Zuge der späteren Analyse benutzt werden und was genau sie in diesem Kontext bedeuten sollen.

- **Prozess** – Ein Prozess meint an dieser Stelle eine zeitlich ausgedehnte Tätigkeit. Ein Prozess hat in der Regel eine oder mehrere Eingaben und Ausgaben, er

bewirkt also Veränderung in irgendeiner Form. Als Prozess werden sowohl die einzelnen Teilprozesse des Software Engineerings wie in Kapitel 3 bezeichnet, als auch die Phasen im Phasenmodell des Schreibprozesses. Der Begriff 'Prozess' wird an dieser Stelle über 'Phase' bevorzugt, da er keine zeitliche Abfolge impliziert. Er ist abzugrenzen von den Begriffen:

- **Schreibprozess** beziehungsweise **Programmierprozess** – Damit ist jeweils die Menge aller Prozesse gemeint, welche zur Erstellung eines Texts, beziehungsweise einer Software notwendig sind, von der Konzeption bis zu der Fertigstellung.
- **Strategie** – Eine Strategie ist eine Tätigkeit, welche vorgenommen wird, um ein Ziel zu erreichen, zum Beispiel um ein Problem zu lösen. Das kann die Anwendung bestimmter Techniken, Methoden oder Werkzeuge beinhalten. Strategien und Prozesse beeinflussen sich gegenseitig: Zum Beispiel kann eine Strategie die Ausführung eines oder mehrerer Prozesse beinhalten (ebenso wie umgekehrt), während unvorhergesehene Änderungen im Prozess auch Änderungen von Strategien nach sich ziehen.
- **Schreibstrategie** – Eine Schreibstrategie ist die übergeordnete Strategie, mit der ein Schreibprozess oder auch ein Programmierprozess angegangen wird (Autoren fühlen sich häufig zu einer Schreibstrategie hingezogen, was auch als ihr 'Schreibtyp' bezeichnet wird). Schreibstrategien können aber auch gezielt als Strategien für Teilprozesse eingesetzt werden.
- **Problem** – Ein Problem kann alles sein, was einen Prozess negativ beeinflusst, indem es ihn zum Beispiel verzögert oder erschwert.
- **Blockade** – Meint in diesem Fall explizit *kognitive* Blockaden nach der Definition von Mike Rose, also Probleme im Prozess, welche nicht auf mangelnde Kompetenz oder Motivation zurückgeführt werden können. Sie entstehen, wenn die gewählten Strategien nicht zu den zu erledigenden Aufgaben passen, beziehungsweise wenn mit unpassenden Strategien auf auftretende Probleme reagiert wird. Neben den kognitiven gibt es auch psychische Blockaden, welche für diese Arbeit aber nicht von Relevanz sind.
- **Interne** beziehungsweise **Mentale Repräsentation** – Damit ist eine Vorstellung der Software oder des Texts gemeint, welche nur im Kopf existiert. Durch Prozesse wie Schreiben oder Programmieren kann diese externalisiert, also zu einer externen Repräsentation umgewandelt werden.

6 Methodik

Dieses Kapitel beschreibt den Aufbau der Interviewstudie. Die Methodik ist angelehnt an diejenige von Gisbert Keseling, der Interviews mit 15 Forschenden zu ihrem Vorgehen beim Schreiben von wissenschaftlichen Texten führte [28, S.145-160]. Von den Interviews erhoffte Keseling sich drei Dinge: Erstens ein Korrektiv zu den Verfahren der blockierten Schreibenden, die er im Rahmen seiner schreibberaterischen Tätigkeit untersuchte. Zweitens Einsicht in die Schwierigkeiten bei der Textproduktion, mit denen geübte Schreibende zu kämpfen haben. Und drittens Aufschluss darüber, wie es den geübten Schreibenden gelingt, mit diesen Schwierigkeiten fertig zu werden. [28, S.134]

6.1 Forschungsfragen

In Anlehnung an Keseling lauten die Forschungsfragen dieser Arbeit folgendermaßen:

- **Forschungsfrage 1**: Wie sehen gesunde Programmierprozesse aus und sind sie mit Schreibprozessen vergleichbar?
- **Forschungsfrage 2**: Gibt es beim Programmieren ähnliche Blockaden wie beim Schreiben?

Mit einem 'gesunden Programmierprozess' ist in diesem Fall ein Programmierprozess gemeint, bei dem keine Blockaden auftreten. Das bedeutet, dass der Programmierprozess nicht durch unpassende Reaktionen auf Probleme längerfristig unterbrochen oder abgebrochen wird, sondern dass er am Ende zu einem zufriedenstellenden Produkt führt. Auch in einem gesunden Programmierprozess treten Probleme auf, diese sind aber nur vorübergehend und werden durch angemessene Reaktionen der Programmierenden gelöst.

6.2 Teilnehmende

Für die Studie wurden Programmierer und Programmiererinnen mit professioneller Erfahrung in der Softwareentwicklung interviewt. Das bedeutet, dass sie zum Zeitpunkt des Interviews mindestens ein Jahr lang für eine Firma oder einen anderen Arbeitgeber als Entwickler an Software mitgearbeitet haben. Diese Einschränkung sollte sicherstellen, dass die Interviewten über ausreichend Erfahrung verfügen, um

zu wissen, wie ein gesunder Programmierprozess aussieht, und diesen Prozess auch artikulieren zu können.

6.3 Materialien

Das Interview ist als halboffenes Interview konzipiert. Das bedeutet, dass nicht notwendigerweise alle Fragen bei allen Teilnehmenden immer gleich gestellt werden, sondern dass der genaue Ablauf der Befragung von den Antworten der Interviewten abhängt. Es können spontan Nachfragen zu interessanten Punkten gestellt werden, die die Interviewten anbringen, und es können Fragen aus dem Fragenkatalog weggelassen werden, falls die Interviewten diese bereits im Rahmen einer anderen Antwort ausreichend besprochen haben. [7, S.308-321]

Grundlage für das Interview ist ein Fragenkatalog, der aus drei Frageblöcken besteht. Der erste Block dient der Erfassung der Programmiererfahrung, der zweite Block konzentriert sich auf den Programmierprozess der Interviewten und der dritte Block auf Probleme, die im Prozess auftreten können. Im Folgenden werden die Inhalte der einzelnen Blöcke kurz erläutert. Der vollständige Fragenkatalog ist in Anhang A zu finden.

6.3.1 Fragebogen zur Erfassung der Programmiererfahrung

Der erste Block des Fragenkatalogs basiert auf dem Fragebogen zur Erfassung der Programmiererfahrung nach Siegmund et al. [48]. Der Fragebogen wurde dem restlichen Interview vorangestellt, um sicherzustellen, dass die Interviewten über ausreichend Programmiererfahrung verfügen. Ferner sollte dadurch eine Grundlage zur Vergleichbarkeit der Daten für mögliche Folgestudien geschaffen werden.

Die Fragen wurden auf Deutsch übersetzt und teilweise abgeändert, um sie der neuen Probandengruppe anzupassen. Insbesondere die Fragen zur Ausbildung wurden angepasst, um zu erfassen, wie viele Jahre die Interviewten in einer formalen Ausbildung zur Softwareentwicklung verbracht haben (y.Edu) und um welche Art der Ausbildung (Berufsausbildung oder Studium) es sich dabei gehandelt hat (kind.Edu). Ferner wurde die Frage nach der Selbsteinschätzung der Programmiererfahrung im Vergleich zu Kommilitonen abgeändert zu einem Vergleich mit Arbeitskollegen (s.CoWorkers). Die Fragen nach der Erfahrung im Umgang mit den konkreten Programmiersprachen Java/C/Haskell und Prolog wurden entfernt. Die Folgefrage, wie viele zusätzliche Programmiersprachen die Interviewten zu einem mittleren Grad beherrschen, wurde dagegen verallgemeinert (num.Languages). Schließlich wurden die Fragen nach dem Alter der Interviewten (y.Age) und nach der üblichen Größe ihrer professionellen Programmierprojekte (size.Proj) an andere Positionen im Fragebogen verschoben. Damit enthält der erste Fragenblock insgesamt 15 Frage-Items.

6.3.2 Fragenblock Programmierprozess

Der zweite Block des Fragenkatalogs enthält 11 Frage-Items. Er beginnt mit der Bitte, dass die Interviewten von einem ihrer aktuellen Programmierprojekte erzählen sollen (P1). Diese offene Frage zu Beginn soll die Interviewten dazu animieren, möglichst viel über ihren Programmierprozess von sich aus zu erzählen, ohne dass sie dabei durch die Formulierungen der Fragen beeinflusst werden können. Erst, wenn die Interviewten zu dieser ursprünglichen Frage nichts mehr weiter zu sagen haben, sollen Folgefragen zu den Punkten gestellt werden, die die Interviewten nicht von sich aus angesprochen haben.

Die Fragen P2 bis P4 haben zum Ziel, das Programmierprojekt besser zu verstehen, um die beschriebenen Prozesse in Kontext setzen zu können. Die Fragen P5 und P6 sollen Details des Programmierprozesses erfassen und die Fragen P8 bis P10 eventuelle Probleme, die speziell bei dem beschriebenen Projekt vorgekommen sind. Das Ziel all dieser Fragen ist nicht, dass sie explizit gestellt werden müssen, sondern sie sollen als Orientierungspunkte dienen, um das Gespräch zurück auf Kurs zu bringen, sollte es sich zwischendurch verlieren.

Die Frage P7 soll erfassen, ob es sich bei dem Vorgehen zu dem beschriebenen Projekt um den üblichen Prozess der Interviewten handelt. Sollte die interviewte Person als Folge davon von weiteren Projekten erzählen, bei denen das Vorgehen abweicht, können die oben genannten Fragen erneut zur Orientierung genutzt werden.

Eine besondere Stellung nimmt die Frage P11 ein. Der Gedanke hinter dieser Frage ist, zu erfassen, ob die Interviewten beim Programmierprozess einen Adressaten vor Augen haben, ähnlich wie es beim Schreibprozess der Fall ist. Die Frage entstand explizit vor dem Hintergrund, dass einige Schreibblockaden adressatenbezogen sein können (siehe Kapitel 2.4).

6.3.3 Fragenblock Blockaden

Der dritte und letzte Block des Fragenkatalogs enthält 8 Frage-Items zu Problemen, die während des Programmierprozesses auftreten können. Der Gedanke hinter diesen Fragen ist nicht unbedingt, dass es sich bei den Interviewten um blockierte Entwickler handelt, sondern sie dienen eher dazu, Probleme zu erfassen, die sich bei weniger erfahrenen Programmierern zu Blockaden entwickeln könnten. Sie sollen außerdem Erfahrungsberichte über gescheiterte Programmierprozesse sammeln, um zu untersuchen, ob Blockaden dabei eine Rolle gespielt haben könnten.

Die erste Frage B1 fragt demnach gezielt nach abgebrochenen Programmierprojekten, was quasi den Extremfall darstellt: Das schlechtmöglichste Resultat einer Blockade. Die Fragen B2 bis B4 sollen moderatere Fälle erfassen, auch unter der Annahme, dass insbesondere im Kontext der professionellen Arbeit als Softwareentwickler Projekte eher selten komplett abgebrochen werden. Die Fragen B5 und

B6 dienen wieder zur Orientierung als Folgefragen. Die Frage B7 schließlich soll die Lösungsstrategien der Interviewten in solchen Fällen erfassen.

Als letzte Frage dieses Fragenblocks wirft Frage B8 dann konkret den Begriff 'Programmierblockade' in den Raum. Die Frage zielt darauf ab, spontane Assoziationen der Interviewten zu dem Begriff zu erfassen. So sollen noch weitere Probleme, die die Interviewten zu dem Zeitpunkt vielleicht noch nicht genannt haben, zum Vorschein kommen.

6.4 Ablauf der Interviews

In diesem Kapitel wird der allgemeine Ablauf der Interviews beschrieben. Da das Interview als halboffenes Interview konzipiert war, kann der exakte Ablauf der Befragung unter den verschiedenen Teilnehmenden etwas variieren.

Zu dem Interview wurden Teilnehmende aus dem Bekanntenkreis der Autorin persönlich eingeladen. Sie wurden in der Einladung darüber informiert, dass das Interview ungefähr 30 Minuten in Anspruch nimmt, aufgezeichnet und anschließend transkribiert wird, und dass die Fragen sich um ihren persönlichen Programmierprozess drehen. Wenn sie die Einladung annahmen, wurden ihnen die Datenschutzdokumente zugesandt, in denen der Ablauf und die weitere Verarbeitung der erhobenen Daten detailliert beschrieben wurden. Stimmten sie der Datenschutzerklärung und den Teilnahmebedingungen zu, wurde ein Termin für das Interview ausgemacht. Die Datenschutzdokumente können in Anhang B eingesehen werden.

Das Interview fand online über das Konferenzsystem BigBlueButton des Universitätsrechenzentrums der Technischen Universität Chemnitz statt[1]. Für die Interviews wurde ein gesonderter Webroom erstellt, der mit einem Zugangscode geschützt war. Dieser Zugangscode war nur den Teilnehmenden und der Versuchsleitung bekannt und wurde für alle Teilnehmenden jeweils neu generiert.

Vor den Interviews wurde den Teilnehmenden noch einmal der Zweck der Audioaufnahme erläutert und noch einmal nachgefragt, ob sie mit der Aufnahme einverstanden seien. Erst nach ausdrücklicher Zustimmung wurde die Aufnahme des Gesprächs gestartet, wobei der genaue Start der Aufnahme laut verkündet wurde.

Den Teilnehmenden wurde erklärt, dass das Interview aus drei Teilen besteht, aber nicht, wie diese drei Teile jeweils inhaltlich aussehen. Das Interview begann dann mit dem Fragebogen zur Erfassung der Programmiererfahrung. Alle 15 Fragen des Fragebogens wurden nacheinander gestellt, wobei es den Interviewten möglich war, ihre Antworten noch weiter auszuführen, wenn sie etwas erklären wollten. Nach Abschluss des Fragebogens wurde der erste Teil des Interviews für beendet erklärt.

Der zweite Teil des Interviews begann mit der Erklärung, dass diese Fragen sich um

[1] `https://www.tu-chemnitz.de/urz/vidcon/bbb/`

den persönlichen Programmierprozess der Befragten drehen. Als erstes wurden die Teilnehmenden immer darum gebeten, von einem aktuellen Programmierprojekt zu erzählen. Anschließend wurde das Interview in einem halboffenen Stil weitergeführt.

Der Übergang zum dritten Frageblock konnte fließend geschehen, falls die interviewten Personen von sich aus auf Probleme im Programmierprozess zu sprechen kamen. Meistens wurde jedoch der zweite Fragenblock ebenfalls explizit für beendet erklärt und der dritte Fragenblock mit der Erklärung eingeleitet, dass nun Probleme im Prozess das Hauptthema seien.

Nach Abschluss des dritten Teils des Fragenkatalogs wurden die Teilnehmenden gefragt, ob sie dem bisher Gesagten noch etwas hinzuzufügen hätten. Anschließend wurden sie über die Hintergründe der Studie aufgeklärt und dass es sich dabei um einen Vergleich zwischen Schreiben und Programmieren handelt. Danach wurden sie nochmal gefragt, ob sie aufgrund dieses neuen Gesichtspunktes noch etwas hinzufügen wollen. Anschließend wurde das Interview für beendet erklärt und die Aufnahme gestoppt.

6.5 Transkription der Interviews

Die Aufnahmen der Interviews wurden von BigBlueButton im MP4 Format gespeichert und konnten von dort heruntergeladen werden. Unter Zuhilfenahme der Web-App oTranscribe[2] wurden sie manuell transkribiert. Bei der Transkription wurde besonders Wert auf die Verständlichkeit des Inhalts gelegt. Non- und paraverbale Merkmale des Gesagten wurden zum größten Teil ausgeklammert. Lediglich wenn es zum Verständnis des Inhalts notwendig war, wurden Notizen zu nonverbalen Äußerungen hinzugefügt (etwa um Ironie kenntlich zu machen). Füllwörter, Wortwiederholungen und abgebrochene oder wiederholte Sätze wurden entfernt, sofern sie nicht zum Verständnis des Inhalts beitrugen. Ebenso wurden nur besonders markante Pausen markiert, die für das Verständnis des Inhalts potentiell relevant sein könnten. Für eine Übersicht der verwendeten Transkriptionszeichen, siehe Tabelle 6.1. Abschließend wurden Namensnennungen oder Beschreibungen, die zur Identifikation von Personen oder Organisationen führen könnten, aus den Transkripten entfernt. Nachdem die Transkription abgeschlossen war, wurden die MP4 Dateien gemäß der Datenschutzvereinbarung gelöscht. Die so bearbeiteten Transkripte befinden sich in Anhang C.

[2] `https://otranscribe.com`

Transkriptionszeichen	Bedeutung
*	Markante Pause
/	Abgebrochener Satz
(?)	Das Gesagte wurde nicht verstanden
(?text)	Das Gesagte wurde nicht verstanden. Der Text ist, was vermutlich gesagt wurde
(text)	Non-verbale Handlung. Zum Beispiel (lachen)

Tabelle 6.1: Übersicht der verwendeten Transkriptionszeichen

6.6 Beschreibung der Teilnehmenden

Für die Interviews wurden insgesamt sieben Teilnehmende rekrutiert, davon eine weiblich und sechs männlich. Für die folgende Beschreibung werden ihre Angaben aus dem Fragebogen zur Programmiererfahrung genutzt, welche in Tabelle 6.2 zusammengefasst sind. Die Teilnehmenden sind zwischen 21 und 31 Jahre alt (im Durchschnitt 25,57 Jahre). Sie verfügen im Durchschnitt über 10,14 Jahre Programmiererfahrung, davon haben sie im Schnitt 4 Jahre in Ausbildung verbracht und programmieren seit 6,71 Jahren in professionellem Kontext, wobei die Größe der professionellen Projekte, an denen sie arbeiten, mindestens mittelgroß ist, also zwischen 900 und 40'000 Codezeilen oder darüber. Sie beherrschen außerdem im Schnitt 6,14 Programmiersprachen auf einem fortgeschrittenen Niveau[3]. Eine Aufschlüsselung der Programmiersprachen pro Person ist in Tabelle 6.3 zu finden.

Auf einer Skala von 1 bis 10 schätzen die Teilnehmenden ihre persönliche Programmiererfahrung im Schnitt bei 7,71 ein. Im Vergleich mit Experten, die über 20 Jahre professionelle Programmiererfahrung verfügen, schätzen sie sich auf einer Skala von 1 bis 5 im Durchschnitt bei 3 ein. Im Vergleich mit ihren Arbeitskollegen schätzen sie sich im Durchschnitt dagegen bei 4,29 ein. Niedrigere Selbsteinschätzungen werden teilweise damit begründet, dass die Person zuletzt nicht viel Zeit in Weiterbildung investieren konnte (ID1). Hohe Selbsteinschätzungen werden dagegen teilweise damit begründet, dass die Personen zusätzlich zum Programmieren im beruflichen Kontext auch noch viel privat programmieren (ID2). Höhere Selbsteinschätzung spezifisch gegenüber den Arbeitskollegen wird teilweise damit begründet, dass die Personen sich vorwiegend mit Studierenden vergleichen (ID4, ID7).

Bezüglich der Programmierparadigmen, bei denen die Befragten ihre Erfahrung ebenfalls auf einer Skala von 1 bis 5 einordnen sollten, beträgt die durchschnittliche Programmiererfahrung für funktionale Programmierung, imperative Program-

[3]Basierend auf Selbsteinschätzung; die Interviewten sollten nur Programmiersprachen aufzählen, bei denen sie ihre Erfahrung auf einer Skala von 1 bis 5 bei mindestens einer 3 einordnen.

ID	y.Age (Jahre)	y.Prog (Jahre)	y.Edu (Jahre)	y.Prof (Jahre)	size.Proj	s.PE (1-10)	s.Experts (1-5)	s.CoWorkers (1-5)	num.Lang	s.Funct (1-5)	s.Imp (1-5)	s.Logic (1-5)	s.OOP (1-5)
ID1	29	16	5	11	mittel	6	2	4	3	1	3	1	5
ID2	25	7	7	3	mittel	5	2	3	3	1	2	1	4
ID3	23	6	3	5	groß	9	3	5	4	4	1	1	5
ID4	22	5	4	1	mittel	9	4	4	11	5	5	2	5
ID5	28	14	3	12	groß	9	4	4	11	3	5	1	5
ID6	31	17	3	12	groß	9	5	5	5	3	5	3	5
ID7	21	6	3	3	mittel	7	3	4	6	1	4	1	4
Ø	25,57	10,14	4	6,71		7,71	3	4,29	6,14	2,57	3,57	1,43	4,71

Tabelle 6.2: Programmiererfahrung der Teilnehmenden

Sprache	Assembler	C	C#	C++	Go	Haskell	Java	JavaScript	Kotlin	Perl	PHP	Python	Rust	Scala	TypeScript
ID1			+					+			+				
ID2					+		+	+							
ID3			+	+				+							+
ID4		+	+	+		+	+	+	+		+	+	+		+
ID5	+	+		+	+		+	+		+	+	+		+	+
ID6		+		+	+		+		+						
ID7		+		+			+	+	+						+

Tabelle 6.3: Programmiersprachen, welche die Teilnehmenden auf fortgeschrittenem Niveau beherrschen

mierung, logische Programmierung und objektorientierte Programmierung jeweils 2,57, 3,57, 1,43 und 4,71. Zu bemerken ist, dass, mit Ausnahme von ID3, all diejenigen, die ihre Programmiererfahrung insgesamt auf 9 eingeschätzt haben, ihre Erfahrung auch in mindestens drei der vier abgefragten Programmierparadigmen jeweils auf 3 oder höher einschätzen.

6.7 Analyse der Interviews

Die Transkripte wurden mit Hilfe der thematischen Analyse ausgewertet. Bei dieser Methodik werden wiederkehrende Themen in den Daten ausfindig gemacht, indem interessante Stellen mit sogenannten Codes markiert werden. In einem nächsten Schritt werden gefundene Codes zu potentiellen Themen vereint. Durch mehrere Iterationen sollen die gefundenen Themen überprüft und präzisiert werden. [9]

In dieser Arbeit wurde für die Analyse ein theoriegetriebener, realistischer Ansatz mit Konzentration auf die semantische Ebene genutzt, bei dem das ganze Datenset gleichermaßen betrachtet wird. Theoriegetrieben bedeutet, dass die Codes bereits durch den theoretischen Hintergrund und die Forschungsfragen vorgegeben sind. In diesem Fall werden die Daten explizit aus einer schreibwissenschaftlichen Perspektive betrachtet. Realistisch ist hier im Gegensatz zu konstruktivistisch gemeint und bedeutet, dass die Erfahrungen der Interviewten für sich betrachtet werden und nicht etwa als das Ergebnis sozialer Strukturen [9].

Aufgrund der beiden Forschungsfragen waren Codes für den Prozess (blaue Markierung) und für Probleme (rote Markierung) gegeben. Die Aussagen zum Ablauf des Programmierprozesses wurden anschließend weiter aufgeschlüsselt: Grün wurden Passagen markiert, die Aussagen zur Vorbereitung oder Planung des Prozesses enthalten. Gelb wurden Passagen markiert, die Aussagen zur Implementierung enthalten. Orange wurden Passagen markiert, die das Testen und Debuggen zum Inhalt haben. Diese drei Codes entsprechen etwa der Dreiteilung des Schreibprozesses in Pre-Writing, Drafting und Revising (siehe Kapitel 2.1).

Die Aussagen zu Problemen im Programmierprozess wurden um eine graue Markierung ergänzt, falls sie Gegenstrategien zu den Problemen zum Inhalt haben. Schließlich erhielten interessante Passagen, die sich keinem der anderen Codes zuordnen lassen, eine violette Markierung.

Die so bearbeiteten Transkripte sowie die Notizen dazu befinden sich in Anhang D.

Die hier beschriebene Codierung stellte die Grundlage für die weitere Analyse dar. Insgesamt wurden mit den gegebenen Daten vier voneinander mehrheitlich unabhängige thematische Analysen durchgeführt. Für die erste Forschungsfrage wurden die Daten auf die Prozesse des Software Engineerings nach Sommerville, auf die Prozesse des wissenschaftlichen Schreibens und auf die Schreibstrategien hin untersucht. Die Ergebnisse werden im nachfolgenden Kapitel besprochen. Für die zweite

Forschungsfrage wurde nach der ersten Codierung auf eine Mischung aus dem theoriegetriebenem und einem eher induktiven Ansatz gewechselt, bei dem die Themen sich aus den Daten ergeben. Die Ergebnisse dafür werden in Kapitel 8 besprochen.

7 Forschungsfrage 1: Ergebnisse und Diskussion

Um die erste Forschungsfrage zu beantworten, führen wir mehrere Analysen des codierten Datenmaterials durch. In Abschnitt 7.1 untersuchen wir die Daten zunächst darauf, ob die Programmierprozesse der Interviewten die Prozesse des Software Engineerings nach Sommerville enthalten. Das soll die Vergleichbarkeit der Programmierprozesse untereinander und mit anderen Studien gewährleisten. Anschließend untersuchen wir die Programmierprozesse der Interviewten in Abschnitt 7.2 daraufhin, ob sie auch die Prozesse des wissenschaftlichen Schreibens enthalten, welche in Kapitel 2.1 vorgestellt worden sind. Dadurch stellen wir die Vergleichbarkeit der Programmierprozesse mit Schreibprozessen fest. Schließlich vergleichen wir in Abschnitt 7.3 das Vorgehen der Interviewten mit den Schreibstrategien, welche in Kapitel 2.3 vorgestellt worden sind. Dadurch zeigt sich, dass Schreib- und Programmierprozesse nicht nur auf der idealisierten Ebene der Phasenmodelle vergleichbar sind, sondern auch auf individueller Ebene. Außerdem zeigt der Vergleich, dass Programmierprozesse genauso individuell sein können wie Schreibprozesse. In Abschnitt 7.4 diskutieren wir schließlich die Ergebnisse und beantworten die erste Forschungsfrage.

7.1 Die Prozesse der sieben Interviewten

Im ersten Schritt soll zunächst untersucht werden, wie die Programmierprozesse der Interviewten aussehen. Zu diesem Zweck sollen die Prozesse des Software Engineerings nach Sommerville (siehe Kapitel 3.1) als theoretischer Rahmen hinzugezogen werden. In Tabelle 7.1 befindet sich eine Übersicht, welche der von Sommerville definierten Prozesse in den Interviews explizit angesprochen werden. Der Prozess 'Softwareentwicklung' wurde dabei in die Teilprozesse 'Entwurf', 'Programmierung' sowie 'Fehlertest und Debugging' aufgesplittet. Der Prozess der 'Softwareevolution' wurde zusätzlich in die Teilprozesse 'Programmverstehen', 'Refactoring' und 'Reengineering' aufgeteilt.

Die Aufteilung des Prozesses 'Softwareentwicklung' in die drei Teilprozesse entspricht der bei der Analyse verwendeten Codierung von 'Planung', 'Implementierung' und 'Testen' (siehe Kapitel 6.7). Dabei konnte schnell festgestellt werden, dass die Bezeichnung 'Planung' für die vorbereitenden Prozesse nicht ausreichend ist: Es schien

zweierlei Arten von Planung zu geben. So berichtet ID2 in der ursprünglichen Beschreibung seines Programmierprozesses davon, wie er sich gedanklich einen Plan von der zu implementierenden Logik macht: "Ich denk mir halt meinen Teil im Kopf, wie das aussehen könnte." Im späteren Verlauf des Interviews kommen wir auf den Unterschied zwischen großen (sprich aufwendigen) und kleinen Aufgaben zu sprechen und er erklärt: "Es hat auch viel mit der Planung zu tun, wie viel man dafür geplant hat." Auf Nachfrage, ob denn hier wieder von der Planung im Kopf die Rede sei, kommt ein irritiertes: "Nee? Die größeren Sachen bei uns werden halt erst mal im Team besprochen. Welche Anforderungen wir haben. Was es dann können soll."

Prozess	ID1	ID2	ID3	ID4	ID5	ID6	ID7
Softwarespezifikation	+	+	+	∘	+	+	-
Entwurf	∘	+	+	+	+	+	+
Programmierung	+	+	+	+	+	+	+
Fehlertest und Debugging	+	+	+	+	+	+	+
Softwarevalidierung	+	-	+	-	+	+	-
Programmverstehen	+	+	+	+	+	+	+
Refactoring	-	+	+	-	+	+	∘
Reengineering	-	+	-	+	+	-	+

Tabelle 7.1: Prozesse der Interviewten in Anlehnung an Sommerville. Ein + bedeutet, dass der Prozess in dem Interview explizit angesprochen wurde. Ein ∘ bedeutet, dass das Vorhandensein des Prozesses aus dem Gesagten implizit erschlossen werden kann. Ein - bedeutet, dass es nicht erwähnt wurde.

Diese zwei Arten der Planung scheinen dem Unterschied von Sommervilles Prozessen der 'Softwarespezifikation' und des 'Softwareentwurfs' zu entsprechen, wobei die Übergänge fließend sein können, da auch während der Softwarespezifikation bereits Entwürfe entstehen können. Aussagen, die sich eher der Softwarespezifikation zuordnen lassen, kommen, genau wie bei ID2, häufiger erst auf Nachfrage zum Vorschein, so auch bei ID5 und ID6. Das könnte daran liegen, dass die Softwarespezifikation bei allen genannten Interviewten im Team stattfindet, während die Interviewfragen sich explizit auf ihren 'persönlichen' Programmierprozess bezogen.

Insgesamt konnten eindeutige Aussagen zur Softwarespezifikation bei fünf der sieben Interviewten festgestellt werden. Die Aussagen beinhalten die Stichworte 'User-Stories' (ID3, ID6) , 'Anforderungen' (ID2, ID3, ID5, ID6) und 'Kunden' (im Kontext von Kundenwünschen) (ID1, ID2, ID3, ID5, ID6), oder es geht darum 'das Ticket auszuarbeiten' (ID5). ID4 berichtet als einziger von einem privaten Projekt, um seinen Prozess zu erklären, bei dem genaue Spezifikationen keine so große Rolle spielen. Allerdings ließe sich auch hier argumentieren, dass Aussagen wie "[ich]

denke, hey, zum Beispiel ein Authentifikations-Service wär jetzt ganz interessant" durchaus auf einen Spezifikationsprozess schließen lassen, da er in diesem Fall selbst den Platz des Kunden einnimmt, der gerne eine bestimmte Funktionalität hätte.

Zu den drei Teilprozessen 'Entwurf', 'Programmierung' sowie 'Fehlertest und Debugging' ließen sich bei allen Interviewten eindeutige Aussagen finden. Einzig ID1 macht keine Aussage, durch die sich ein Entwurfsprozess explizit von der Spezifikation hätte trennen lassen. Er berichtet davon, wie in Meetings grafische Notizen zum späteren Aussehen des gewünschten Features gemacht werden, was sowohl als Entwurf, als auch als Modell oder Prototyp interpretiert werden könnte, welche laut Sommerville noch der Spezifikation zuzuordnen wären [49, S.66].

Die Softwarevalidierung wurde nur von vier der sieben Interviewten eindeutig angesprochen. Das könnte sich aber ebenfalls darauf zurückführen lassen, dass dafür oft andere Teammitglieder oder sogar Personen außerhalb des Teams zuständig sind. So berichtet ID1, der zur Zeit des Interviews der einzige Entwickler seines Teams ist, dass seine Mitarbeitenden oft trotzdem "auch nochmal testen, um quasi ne komplette Funktionalität vorweisen zu können." ID3 erklärt zum einen ausführlich den Prozess des 'Reviewens' von Code durch andere Mitarbeitende und erzählt zum anderen auch von Tests, die die Kunden durchführen. Das Konzept des Codereviews wird auch von ID5 angesprochen. ID6 berichtet von "standardisierte[n] Vorgehen zur Qualitätssicherung", allerdings erst, als es um die Unterschiede zwischen privatem Programmieren und Programmieren für den Beruf geht. Es kann also davon ausgegangen werden, dass das Fehlen von Aussagen zur Softwarevalidierung bei den anderen drei Interviewten weniger darauf schließen lässt, dass keine Validierung stattfindet, sondern mehr, dass diese nicht unbedingt in ihren Zuständigkeitsbereich fällt oder dass sie nicht als Teil des Programmierprozesses angesehen wird.

Als letztes betrachten wir Sommervilles Prozesse der Softwareevolution, wobei hier anzumerken ist, dass alle Interviewten an bereits bestehenden Produkten gearbeitet und diese weiterentwickelt haben, wodurch die Prozesse der Evolution nicht wirklich sinnvoll von Entwurf- und Implementierung zu trennen sind.

Programmverstehen ist daher auch bei allen Interviewten eine Grundvoraussetzung für ihre Arbeit, was auch mit den Erkenntnissen von Minelli et al. [36] übereinstimmt, laut denen Software-Entwickler bis zu 70% ihrer Zeit mit Programmverstehen verbringen. Daher wird die Notwendigkeit des Programmverstehens auch von allen Interviewten erwähnt.

Refactoring, also das Verbessern von Code durch Überarbeiten, ohne dabei die Funktionalität zu verändern, wird von vier der sieben Interviewten explizit angesprochen. Allerdings ist nicht immer ganz eindeutig, dass tatsächlich keine Funktionalität verändert wird, weshalb die Grenzen zum Reengineering ebenfalls fließend sind. So erwähnt ID2, dass manchmal nur Code 'überarbeitet' werden muss, im Gegensatz zu Aufgaben, bei denen Code neu geschrieben werden muss. ID3 spricht davon, den selbst geschriebenen Code zu 'optimieren', ID5 erzählt detailliert von einem Projekt,

das hauptsächlich aus Refactoring bestand. ID6 nennt Refactoring ebenfalls explizit und ID7 erzählt von Fällen, in denen Code 'schöner gemacht' werden könnte.

Explizite Reengineeringprozesse, bei denen alte Systeme grundlegend überarbeitet werden, werden bei vier der sieben Interviewten angesprochen, allerdings nicht um ihre üblichen Programmierprozesse zu erklären, sondern immer im Kontext von spezifischen Beispielen (meist für Probleme). Das liegt vermutlich daran, dass das Reengineering im Endeffekt eine besondere Art Programmierprozess ist, der von den gleichen Teilprozessen Gebrauch macht, und daher nicht selbst als Teilprozess zu betrachten ist.

Was bei Sommerville zunächst keine größere Erwähnung findet, in den Interviews aber ein paar Mal angesprochen wird, sind Prozesse, die dazu dienen, den Code abgabefertig zu machen. ID3 nennt diesen Prozess "Committen", meint damit aber nicht nur den Befehl aus dem Git-Workflow, von dem der Begriff wahrscheinlich kommt. Bevor besagter Befehl zum Einsatz kommt, sei es zum Beispiel wichtig, jede Datei und jede Änderung auf überflüssig gewordenen Code zu überprüfen. Etwa Code, der nur zum persönlichen Testen genutzt wurde und mit der eigentlichen Funktionalität nichts zu tun hat. ID5 erwähnt das Schreiben der Dokumentation als Teil des Prozesses nach dem eigentlichen Programmieren, zusätzlich dazu, dass sie "rundrum noch ein bisschen was geschrieben" habe. Im Fall von ID3 ließe sich vielleicht argumentieren, dass sich der beschriebene Prozess dem Refactoring zuordnen lässt. Allerdings scheint dabei aber nicht mehr unbedingt Code gezielt und großflächig überarbeitet zu werden, es handelt sich eher um eine Art Aufräumen. Diese kleinen Korrekturen, das letzte Überprüfen, dass alles richtig und ordentlich ist, scheint sich durchaus noch einmal vom Refactoring zu unterscheiden.

Aus den Aussagen lässt sich schließen, dass, zusätzlich zu den von Sommerville erwähnten Prozessen, der Prozess 'Committen' relevant ist. Außerdem scheint es nicht sinnvoll zu sein, die Prozesse der Softwareevolution von der Softwareentwicklung zu trennen, weshalb die Prozesse 'Codeverstehen' und 'Refactoring' für den weiteren Verlauf der Analyse letzterem zugeordnet werden. Der Prozess des 'Reengineerings' wird dagegen ausgeklammert werden, da er meist als separater Auftrag erteilt wird.

7.2 Vergleich mit den Prozessen des wissenschaftlichen Schreibens

Während wir im ersten Schritt gezeigt haben, dass die Teilprozesse des Software Engineerings in den persönlichen Programmierprozessen aller Interviewten vorhanden sind, werden wir in diesem Abschnitt argumentieren, dass die Interviewten beim Programmieren zusätzlich auch sämtliche Prozesse des Schreibens durchlaufen. Zum Vergleich werden die Prozesse des wissenschaftlichen Schreibens, wie im Phasen-

modell von Abbildung 2.1 in Kapitel 2.1 aufgelistet, herangezogen. Es geht dabei weniger um eine Eins-zu-eins-Gegenüberstellung der Prozesse, wie sie zum Beispiel von Hermans und Aldereweld vorgenommen wurde (siehe Kapitel 4), auch wenn sich eine solche zugunsten des Überblicks in Abbildung 7.1 finden lässt. Stattdessen werden wir die Prozesse des Schreibens nacheinander betrachten und untersuchen, inwiefern sie sich in den Interviews wiederfinden lassen.

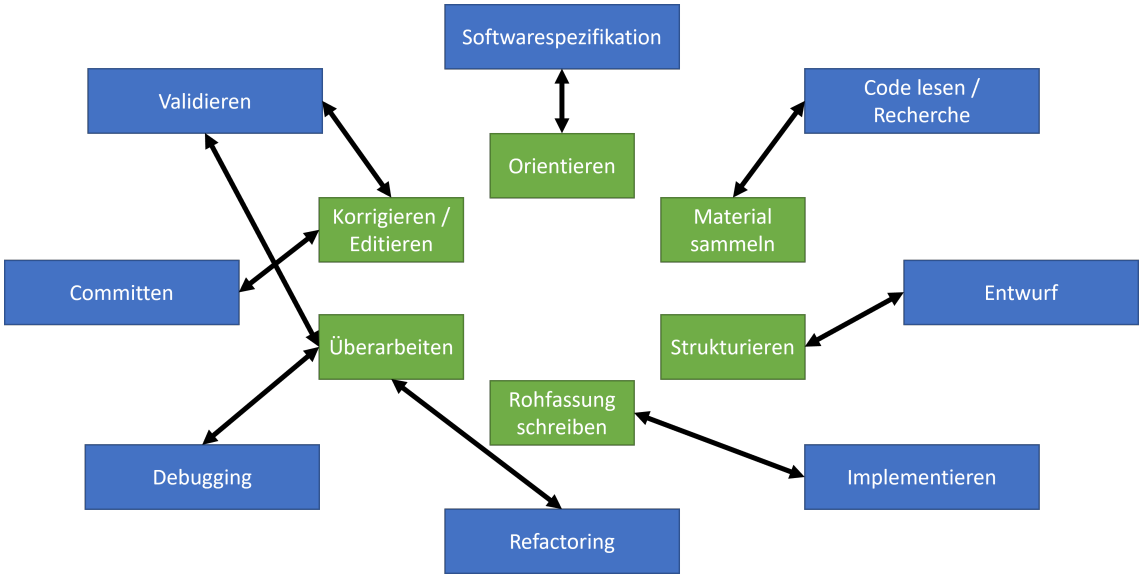

Abbildung 7.1: Vergleich der Prozesse des wissenschaftlichen Schreibens (grün, innen) und des Software Engineerings (blau, außen).

Orientierung

Beim Prozess der Orientierung wird die Aufgabenstellung geprüft, die verfügbare Zeit wird eingeteilt, zentrale Parameter des Texts wie zum Beispiel Thema, Fragestellung und Ziele werden festgelegt. [45, S.24]

Jeder Schreib- oder Programmierprozess nimmt irgendwo seinen Anfang, weswegen der Prozess der Orientierung viel mit dem Prozess der Anforderungsanalyse gemeinsam hat. So beschreibt ID3 den Beginn seines typischen Programmierprozesses im Rahmen des Scrum-Workflows folgendermaßen: "Die Person, die die Anforderungen stellt [...] kommt mit der Anforderung, erklärt uns ungefähr, was er möchte, und wir versuchen als Experten, eben als die Entwickler, klar [...] zu definieren: Was soll gemacht werden, was sind die Problemstellungen?" Hier wird die Aufgabenstellung geprüft und Ziele werden definiert. Diese können, insbesondere bei agilen Prozessen, immer wieder angepasst werden, wie neben ID3 auch ID5, ID6 und ID7 in ihren Interviews bestätigen. Beim Schreiben ist das nicht anders, es ist immer möglich, dass neue Erkenntnisse und Ideen den Prozess der Orientierung neu anstoßen [21, S.58].

Zur Orientierung gehört auch das Planen des weiteren Schreibprozesses und das

Einteilen der dafür zur Verfügung stehenden Zeit. Bei den Interviewten, die mit dem Scrum-Workflow arbeiten, spielen hier mehrere Teilprozesse eine Rolle, nämlich das 'Refinement' und das 'Planning': "In dem Planning planen wir die [User-]Storys ein, je nachdem, wie viele Ressourcen wir haben."(ID3) Diese User-Storys "hat man sich vorher mal in einem Prozess, der heißt Refinement, schon mal angeschaut als Team und geschätzt. Das heißt, man weiß ungefähr grob, welcher Aufwand dranhängt."(ID6) Der Prozess der Orientierung scheint also etwas weiter gefasst zu sein, als der der Anforderungsanalyse, wobei das auch von der individuellen Umsetzung abhängt.

Material sammeln und bearbeiten

Bei diesem Prozess wird eine theoretische Basis für den Text geschaffen. Bei wissenschaftlichen Texten gehört dazu die Literaturrecherche, aber auch das Erheben von eigenen Daten und die Aufbereitung derselben. Die genauen Details dieses Prozesses hängen stark von der Art des benötigten Materials ab. [21, S.64]

Auf den ersten Blick scheint das nicht viel mit Programmierung zu tun zu haben. Zwar ist es einleuchtend, dass bereits vorhandener Code gelesen werden muss und auch mal im Internet nach neuen Komponenten gesucht wird. Diese beiden Tätigkeiten werden auch bei Sommerville erwähnt, bleiben aber eher eine Randnotiz [49, S.69, S.293]. In den Interviews wird dagegen schnell deutlich, dass Recherche eine sehr viel größere Rolle spielt, als zunächst erwartet. ID7 berichtet etwa, dass "[du] meistens einen recht großen Anteil deiner Zeit damit verbringst, überhaupt erst mal zu gucken, wie existenter Code strukturiert ist. Also an welchen Stellen muss neu geschriebener Code eingreifen?" ID3 ergänzt das um weitere Fragen, die geklärt werden müssen: "Wenn es bestehenden Code gibt, den musst du dir ganz genau durchlesen, damit man sehen kann, welche Stellen kritisch sind, worauf sollte man achten?" Auch ID5 berichtet davon, dass ein großer Teil ihrer Vorbereitung für den Softwareentwurf daraus besteht, Code zu lesen und eventuell die ursprünglichen Autoren des Codes auch noch zu befragen, sofern diese noch ausfindig gemacht werden können.

Damit ist es aber nicht getan, denn auch Projekte, die nicht auf einer bestehenden Codebasis aufbauen, benötigen einiges an "Research Zeit", wie ID7 es nennt. Anstelle des Lesens von bestehendem Code rückt dann stattdessen "so ein Spaß wie API Definition. Das nimmt definitiv bei von Grund auf angefangenen Projekten mehr Zeit ein, weil da hast du halt auch noch die Auswahl, welche Frameworks, welche Kommunikationsprotokolle und so weiter und so fort, den ganzen Tech-Stack zusammenzusuchen." ID4 berichtet hier detailliert davon, wie dieser Prozess bei seinem privaten Projekt aussieht: "[Ich] hab halt einfach angefangen zu programmieren, indem ich mir angeguckt hab, wie erstellt man ein App-Frontend, wie kann man damit was machen. Das heißt, da habe ich erst mal so ein bisschen geguckt und ein bisschen rumgeklickt, rumgeschaut, bis ich dann irgendwo [ein] funktionales,

gut aussehendes Designprinzip hatte." Was sich in diesem 'Rumschauen' und 'Rumklicken' zu verbergen scheint, ist das, was bei Hayes als Textinterpretation bezeichnet wird: Das Erstellen interner Repräsentationen durch Input, welcher aus weit mehr als nur aus Texten bestehen kann (siehe Kapitel 2.2).

Bei ID4 kommen wir im Laufe des Interviews noch öfter auf diese internen Repräsentationen zu sprechen, die auch deutlich mehr umfassen können, als nur das unmittelbar relevante Codestück. So erzählt ID4 von dem Fall eines Videos, das ihm die zündenden Ideen zum Verständnis der Technologie geliefert hat, für die er zu der Zeit programmieren sollte, obwohl das Video nur am Rande mit seiner Arbeit zu tun hatte. Doch durch die neue Perspektive verbesserte sich die interne Repräsentation, die ID4 von der Technologie hatte, wodurch sich der Programmierprozess einfacher gestaltete.

Strukturieren

Der Prozess des Strukturierens wird bei Ruhmann und Kruse nicht extra von dem Prozess des Materialsammelns abgetrennt [45, S.24], bei Grieshammer et al. aber schon [21, S.66], jedoch betonen auch sie, wie eng die beiden Prozesse miteinander verzahnt sind. Beim Strukturieren werden die Informationen aus der Materialsammlung geordnet und dabei die Gliederung des zu schreibenden Texts entwickelt. Der Prozess spiegelt so gesehen die kognitiven Prozesse der 'Reflexion' nach Hayes wider: Aus den internen Repräsentationen, die aus dem Input generiert wurden, werden weitere interne Repräsentationen geschaffen, zu denen auch die des zu schreibenden Textes gehört.

Diese enge Verzahnung lässt sich auch in den Interviews wiederfinden. In den oben zitierten Aussagen von ID4 und ID5 gehen die Prozesse des Codelesens beziehungsweise der Recherche nahtlos in das Erstellen eines Plans (ID5), beziehungsweise eines "Designprinzips" (ID4) über. Auch ID3 fängt beim genauen Durchlesen der Anforderungen schon an, über die gesamte Implementation nachzudenken. Generell findet hier sehr viel im Kopf statt: Auch ID2 berichtet, dass er sich 'seinen Teil im Kopf denkt', wie die Logik der Implementation aussehen könnte. ID7 erzählt zunächst generell von vielen Aspekten der Planung. Auf Nachfrage hin, ob er sich denn viel aufschreibe oder das mehr in seinem Kopf abliefe, meint er, dass sehr, sehr viel davon in seinem Kopf sei. Erst wenn "viele kleinere Baustellen" auftauchen, ziehe das auch digitale Notizen nach sich. Das ist auch der Grund, weshalb Sommerville die Prozesse Entwurf und Programmierung nicht explizit trennt [49, S.67-69].

Beim Strukturieren entstehen nicht zwingend sichtbare Ausgaben, trotzdem ist der Prozess essenziell wichtig. Hier entstehen unter anderem Datenmodelle, wie bei ID3: "[Sobald] die Daten komplizierter werden, musst du [...] die ganzen Beziehungen in den Daten, wie die Daten zusammenarbeiten noch richtig gut strukturieren und da muss man sich vorher Gedanken machen." Oder Algorithmen, wie ID4 beschreibt:

"Wenn ich mir jetzt denke, man kann ihn [den Code] ja auch nicht komplett unstrukturieren. Man kann ja nicht sagen, ich suche zuerst in dem Feld und danach sortiere ich es. [...] Also wenn ich ganz häufig über Algorithmen nachdenke, dann denke ich schon in so einer Art Pseudocode nach."

Externalisiert werden die internen Entwürfe und Repräsentationen oft dann, wenn es für die Zusammenarbeit im Team unerlässlich ist. So berichtet ID2, dass bei ihnen dann im Team geplant und Dinge aufgeschrieben werden, wenn es 'etwas Größeres' ist, "damit man nicht sagen kann: 'nee, das [haben wir] aber so nicht besprochen.'" Auch ID7 sagt, dass bei Projekten, die aus mehreren Komponenten bestehen, wie zum Beispiel Frontend und Backend, "vorher schon mal eine API-Dokumentation oder so was im besten Falle stehen [sollte], damit man auch einfach nicht aneinander vorbei entwickelt."

Die beim Prozess des Strukturierens entstehenden Entwürfe sind sowohl beim Schreiben [21, S.66] als auch beim Programmieren [49, S.67] zu Beginn oft grob und werden im Laufe der Arbeit durch immer neue Iterationen weiter verfeinert.

Rohfassung schreiben

In diesem Prozess geht es darum, das leere Blatt Papier, beziehungsweise den leeren Editor mit Text zu füllen. Die während der Materialsammlung und der Strukturierung generierten internen Repräsentationen müssen in wahrnehmbaren, schriftlichen Output umgewandelt werden. Beim wissenschaftlichen Schreiben besteht oft die Schwierigkeit, diesen Prozess überhaupt als eigenständigen Prozess zu erkennen [21, S.68]. Das liegt daran, dass vielen unerfahrenen Schreibenden nicht bewusst ist, dass Texte sich bei ihrer Entstehung nicht sofort in druckreifem Zustand befinden müssen.

Bei den Programmierprozessen lässt sich beobachten, dass der Prozess der Programmierung, also des Niederschreibens von Code, im Gesamtprozess oft kaum Erwähnung findet. Sommerville widmet der Programmierung ganze vier Sätze [49, S.69] und auch in den Interviews wurde der Prozess ohne explizite Nachfrage gerne mit Floskeln wie "und dann wurde das umgesetzt"(ID1) oder "und dann habe ich losgelegt"(ID6) zusammengefasst. ID5 verknüpft das zumindest mit der Tatsache, dass sie durch ihre Planung bereits genau weiß, was sie machen will, "womit's dann halt unterm Strich halt bei der Umsetzung wirklich nur noch das reine, okay, ich muss an der Stelle ein bisschen Code hinschreiben, ich muss dort ein bisschen Code schreiben, mehr ist es dann halt schon nicht mehr." Im Gegensatz zum wissenschaftlichen Schreiben scheint es beim Programmieren also nur wenig Hemmung beim Formulieren zu geben.

Überarbeiten

Für wissenschaftliche Texte bedeutet Überarbeiten, aus der Rohfassung einen leserorientierten Text zu schaffen, der alle Anforderungen an wissenschaftliche Texte erfüllt. Mit Leserorientierung ist dabei gemeint, dass die Inhalte des Textes klar kommuniziert und die Lesenden überzeugend durch die Argumentation geführt werden. Die Rohfassung wird im Gegensatz dazu auch als 'schreiberorientiert' bezeichnet, da es in ihr eher darum geht, die Gedanken und das Wissen der Schreibenden zu ordnen und zwar in einer Weise, die erst mal nur sie verstehen müssen. Um den Prozess des Überarbeitens erfolgreich zu gestalten, ist es hilfreich, sich Feedback von Lesenden zu holen. [21, S.68-71]

Im Kontext des Programmierens gibt es hier zwei Prozesse, die einer näheren Betrachtung wert sind. Das ist zum einen das Debugging und zum anderen das Refactoring. Die beiden Prozesse entsprechen der in Kapitel 4 angesprochenen doppelten Leserschaft von Code, nämlich Maschine und Mensch: Während Debugging zum Ziel hat, dass die Maschine alle Anweisungen versteht, wie von den Programmierenden vorgesehen, hat Refactoring zum Ziel, anderen Programmierenden das Verständnis zu erleichtern.

Der Prozess des Debuggings teilt sich auf in Fehlertest und Fehlerbehebung [49, S.69] und ist sehr eng mit dem Schreiben von Code verknüpft. In den Interviews werden die Tätigkeiten oft in einem Zug erwähnt, es wird 'während' der Entwicklung oder 'nebenbei die ganze Zeit' getestet (ID2, ID3). Einige Interviewte betrachten das Testen bei der anfänglichen Beschreibung ihrer Prozesse zunächst gar nicht näher und bringen es erst in späteren Kontexten zur Sprache (ID1, ID6) oder sogar nur, wenn es darum geht, dass Probleme spezifisch beim Debugging auftreten (ID4). Erwähnt wird es schlussendlich von allen Interviewten, doch dass der Prozess so selten explizit aus dem übergeordneten Programmierprozess gelöst wird, zeigt, wie selbstverständlich er für die erfahrenen Programmierenden dazugehört. Einzig ID7 trennt die Prozesse deutlicher voneinander ab und erzählt, dass er Programmierung und Tests bevorzugt nacheinander angeht: "Ich implementiere meistens einen Teil, schreibe dann Tests dagegen, schau, ob die durchlaufen und wiederhole das so lange, bis das komplette Stückchen Software sozusagen ergänzt ist." Bei kleineren Projekten, die unter 300 Codezeilen bleiben, schreibe er sie auch zu großen Teilen direkt runter, bevor er sich mit größeren Tests befasst. Aber auch diese Aussage ergänzt er damit, dass er während das Programmierens trotzdem noch 'hier und da' kleinere Sachen testet.

Die 'ganze Zeit nebenbei zu testen', wie es von ID2 beschrieben wird, bedeutet, sich die ganze Zeit nebenbei Feedback von der Entwicklungsumgebung zu holen, eine Möglichkeit, die beim Schreiben nicht besteht. "Schreibende müssen sich bei der Textproduktion laufend in die Position des jeweils vorgestellten Publikums versetzen" [45, S.26], ein Akt, der unerfahrenen Schreibenden noch gar nicht möglich ist und der auch erfahrenen Schreibenden noch einiges an Schwierigkeiten berei-

ten kann. Technische Hilfen wie Rechtschreib- und Stilkorrekturen existieren zwar, können Schreibenden aber letzten Endes auch nicht die Frage beantworten, ob sie von dem menschlichen Zielpublikum verstanden werden. So gesehen ähnelt der Prozess des Debuggings mehr einem Gespräch mit der Maschine, als dem Verfassen eines Textes für jene. Beim Sprechen erhalten wir laufend Feedback von unserem Gegenüber, ob die Verständigung gerade gelingt oder nicht [45, S.26] und es stehen uns deutlich mehr Möglichkeiten zur Verfügung, uns auszudrücken und auch uns zu korrigieren: Ein fehlgeschlagener Kommunikationsversuch kann wiederholt werden, eine missverstandene Äußerung berichtigt [28, S.254]. Bei Texten muss dies a priori geschehen, die Möglichkeit von Missverständnissen muss möglichst ausgeräumt werden, bevor die Leserschaft den Text zu Gesicht bekommt, da Berichtigungen nur sehr begrenzt, wenn überhaupt, möglich sind. Hierin liegt also einer der wesentlichen Unterschiede zwischen Schreib- und Programmierprozessen.

Wie sieht es aber bei Leserorientierung in Bezug auf eine menschliche Leserschaft aus? Am ehesten kommt das Thema im Kontext der Frage zur Sprache, wen die Interviewten beim Programmieren 'im Kopf' haben. Diese Frage führte bei ID3, ID5 und ID7 zu Ausführungen zum Thema 'Leserlichkeit des Codes'. "Im Endeffekt muss jeder Code so geschrieben werden, dass, egal welcher Entwickler sich ransetzt, er muss anhand der Worte, die er liest, nicht anhand der Logik, die er sieht, verstehen können, was da passiert", erklärt zum Beispiel ID3. ID7 drückt es ein wenig salopper aus: "Es gibt ja da diesen schönen Ausspruch mit: Entwickle immer so oder schreibe deinen Code immer so, als wäre der Kollege, der ihn nach dir kriegt, ein 'psychopathischer Mörder'." An dieser Aussage wird deutlich, dass Programmierende ebenfalls den inneren Perspektivenwechsel vollziehen müssen, wie Schreibende es tun, um gut verständlichen Code zu schreiben.

Allerdings haben sie auch dafür technische Hilfsmittel zur Verfügung. ID3 erklärt das sogenannte 'Linting': "Linting sind Regeln, die vom Team selber innerhalb des Projekts definiert werden." Mit Hilfe von Plug-ins für die integrierte Entwicklungsumgebung (als Beispiel wird von ID3 ESLint genannt) können die Programmierenden dann wiederum direktes Feedback erhalten, ob sie diese Regeln einhalten. ID3 nennt hier Beispiele wie "hier sollte nicht 'Test' stehen" oder "hier hast du eine Methode, die geht über 100 Zeilen lang, das ist unleserlich".

Neben solchen Tools ist es für viele der Interviewten auch normal, sich Feedback aus dem Team zu holen. ID6 übergibt seinen Code direkt nach der Grundarbeit an zwei, drei Kollegen, die dabei "die Architektur quasi kennenlernen und Feedback geben". ID1, ID3 und ID5 berichten von Reviews oder Tests durch andere Mitarbeiter. Bei ID2 und ID3 wird bei Schwierigkeiten auf Pair-Programming zurückgegriffen. All das sind Möglichkeiten, Feedback zu erhalten, die bei Schreibprozessen prinzipiell auch gegeben sind, bei Programmierprozessen aber mit größerer Selbstverständlichkeit dazugehören. So erwähnen Ruhmann und Kruse, dass Co-Autorenschaft und kooperative Textproduktion zwar auch in vielen wissenschaftlichen Domänen üblich sind, sich dies aber kaum in schreibdidaktischen Angeboten in deutschen Hochschulen

niederschlägt [45, S.27].

Leserorientierung spielt im Programmieralltag also durchaus eine Rolle, weshalb wir auch davon ausgehen können, dass in den Prozessen der Interviewten mehr Refactoring stattfindet, als sie explizit erwähnen. Jedenfalls wenn wir Refactoring als 'Überarbeiten im Hinblick auf menschliche Leserschaft' verstehen. Wie dieser Prozess aktiv genutzt werden kann, erklärt ID6 sehr anschaulich: "Refactoring heißt in dem Fall, Architektur-Entscheidungen nochmal zu verändern im Nachhinein: Layer anders zu gestalten, Klassen, Objekte woandershin zu verschieben. Vielleicht auch den Scope zu verändern, also zu sagen, ne Klasse hat jetzt ein paar mehr oder weniger Aufgaben." Auch er betont, wie nützlich dabei die Werkzeuge sind, die einem integrierte Entwicklungsumgebungen zur Verfügung stellen: "Das ist ja quasi mit der modernen Programmiersoftware, IDEs, kostet das ja alles nichts." Der besondere Wert, den diese Werkzeuge, beziehungsweise die Vereinfachung des Prozesses, den sie mit sich bringen, in den Augen von ID6 haben, ist, dass sie Programmierenden die Angst vor Veränderung nehmen: "Ich denke, es ist halt auch wichtig, zu sehen, dass man quasi dieses Wissen, keine Angst vor Veränderung und vor nochmal ne Entscheidung zu revidieren und zu reviewen und daraus Schlüsse zu ziehen, dass man davor keine Angst hat."

Korrigieren und Editieren

Beim Schreibprozess wird dieser Prozess meist als Endphase angesehen, bei der es darum geht, den Text abgabefertig zu machen. Der Text wird auf Rechtschreibung und Grammatik korrigiert, es werden Verzeichnisse und Anhänge angelegt und das Layout wird gestaltet. [21, S.71] Der Text wird also in die Form gebracht, die er haben muss, um zu seinem angedachten Zweck weiterverwendet werden zu können.

Wie am Ende von Kapitel 7.1 angesprochen, sind auch beim Programmieren abschließende Prozesse notwendig, damit der Code abgegeben, in diesem Fall ausgeliefert oder in das bestehende Softwaresystem eingefügt werden kann. Wie beim Schreiben auch kann 'Abgeben' mitunter noch Feedback und dadurch doch noch weitere Überarbeitungsprozesse nach sich ziehen, wie zum Beispiel ID3 berichtet: "Wenn [der Mitarbeiter] reviewt hat und alles cool ist, dann merged er's in den master-Branch. Wenn nicht, dann schreibt er sein Feedback in Kommentaren, sagt: 'Hey, das muss [geändert] werden, das funktioniert gar nicht.'" Die Prozesse sind also nicht so endgültig, wie das Phasenmodell des Schreibprozesses suggeriert, auch wenn natürlich irgendwann einmal ein letztes Mal committed wird.

Im Laufe der Analyse waren wir also in der Lage, sämtliche Prozesse des wissenschaftlichen Schreibens in den Programmierprozessen der Interviewten nachzuweisen. Inklusive der Prozesse, bei denen der Vergleich etwas weniger naheliegt, wie die Materialsammlung oder das Korrigieren und Editieren. Interessant ist an dieser

Stelle besonders die klare Zweiteilung der Überarbeitungsprozesse beim Programmieren. Auch beim Schreiben wird empfohlen, das Überarbeiten in mehrere Phasen zu gliedern, in welchen unterschiedliche Ziele gesetzt werden sollen [45, S.26]; es fehlt aber die klare Zweiteilung der Adressaten.

Da die Analyse zeigt, dass auch die Prozesse des Schreibens in einem Programmierprozess durchlaufen werden, stellt sich nun im nächsten Schritt die Frage, ob die Umsetzung dieser Prozesse auch den Schreibstrategien entspricht.

7.3 Schreibstrategien beim Programmieren

Nachdem wir die Programmierprozesse der Interviewten im letzten Abschnitt dahingehend untersucht haben, ob sie die Prozesse des wissenschaftlichen Schreibens enthalten, werden wir in diesem Abschnitt auf die Schreibstrategien eingehen, die während der Interviews zum Vorschein kamen. Zu diesem Zweck ziehen wir die Strategietypen beim wissenschaftlichen Schreiben nach Grieshammer et al. heran, die in Kapitel 2.3 besprochen wurden. Ziel dieses Abschnitts ist es, zu zeigen, dass alle fünf Schreibstrategien, nämlich spontanes Schreiben, planendes Schreiben, Mehrversionen-Schreiben, redaktionelles Schreiben und Puzzle-Schreiben, in den Programmierprozessen der Interviewten zur Anwendung kommen. Es geht allerdings nicht darum, die Interviewten mit einem bestimmten Typen zu 'diagnostizieren', denn wir gehen davon aus, dass erfahrene Programmierende genau wie erfahrene Schreibende zwar potentiell zu einer Strategie tendieren, aber grundsätzlich flexibel bei der Wahl ihrer Strategien sind und diese der Aufgabe angemessen anpassen können. Zu Beginn jedes Unterabschnitts wird die jeweilige Schreibstrategie noch einmal in wenigen Worten zusammengefasst, bevor ihr Auftreten in den Interviews diskutiert wird.

Spontanes Schreiben

Das klassisch strukturschaffende Vorgehen, bei dem spontan und assoziativ drauflos geschrieben wird. Neue Ideen, Gedanken und Zusammenhänge werden beim Schreiben entdeckt, wodurch nach und nach ein Bild des Textes entsteht: Die Struktur des Textes wird erst während des Schreibens geschaffen. [21, S.32f]

Diese Strategie scheint bei mindestens drei der Interviewten zum Einsatz zu kommen. ID3 berichtet von ihr im Rahmen von privaten Projekten, bei denen er deutlich mehr Freiheiten genießt als beim Programmieren für die Arbeit. Er bezeichnet sich in diesem Kontext lachend als den 'undiszipliniertesten Entwickler der Welt' und beschreibt sein Vorgehen folgendermaßen:

> "Ich fang viele kleine Projekte an und kleine Projekte haben meistens keine Foundation, also keinen Boden. Und wenn ich noch gar nicht weiß, dass, was ich hier mache, überhaupt weiterentwickelt wird, da

> denk ich mir einfach, ich mach's jetzt quick und dirty, Hauptsache es funktioniert. Im Fall von Spielehacks oder sowas denke ich: Ist das überhaupt möglich, was ich machen möchte? Ist das überhaupt ein Bot, den ich entwickeln kann? Und dann lege ich erst mal drauflos und schreib wirklich so schmutzig wie's geht, aber auch so schnell und produktiv wie es nur geht den Code runter."

Hier lassen sich bereits die meisten Merkmale des spontanen Schreibens erkennen: Die einzigen Fragen, die vorab geklärt werden, beziehen sich darauf, ob die Idee überhaupt grundsätzlich umsetzbar ist. Danach wird 'drauflos' geschrieben, es geht 'schnell' und der entstehende Code ist 'schmutzig'. Beim spontanen Schreiben werden die Gedanken, die aufs Papier gebracht werden, kaum hinterfragt, es findet keine Selbstzensur statt, was eine der größten Stärken dieser Strategie ist und der Grund, weshalb mit ihr zügig viel Text produziert werden kann [21, S.32]. Dafür ist der entstehende Text sehr schreiberorientiert und benötigt viel Überarbeitung, damit er von anderen Lesenden verstanden wird.

Die beiden anderen Beispiele für die Strategie des spontanen Schreibens finden sich bei ID2 und ID6. Allerdings erzählen diese beide im Rahmen ihrer Projekte bei der Arbeit davon, weshalb ihnen ein paar mehr Einschränkungen gegeben sind als bei ID3. ID6 berichtet, dass er 'einfach loslegt', nachdem die nötigen Entscheidungen im Team getroffen wurden. Auf Nachfrage, ob es denn keine Planungsphase gebe, sagt er, dass die 'an der Stelle ja schon abgeschlossen war', er scheint sie also nicht als zentral für seinen persönlichen Programmierprozess zu betrachten. Er ergänzt noch, dass das Definieren der User-Story, also "dass man sich quasi mit diesem Satz hinsetzt und anfängt was zu bauen" als Planung interpretiert werden könnte, hält das aber im Konjunktiv. Später erklärt er auch, dass Architektur-Entscheidungen sich bei ihm während des Schreibens ergeben, was typisch für das strukturschaffende Vorgehen ist. Die Frage, ob er bei privaten Projekten auch so spontan vorgehe, bejaht er: "Fast eigentlich noch mehr."

Interessant ist auch die Art und Weise, wie ID6 von Problemen bei dieser Strategie berichtet. Er erzählt, dass ihm oft intuitiv Lösungen einfallen, die im Kopf 'super cool' klingen. Bei der Umsetzung kann es dann aber vorkommen, dass er bemerkt: "Das fühlt sich einfach falsch an." Dieses Gefühl versucht er noch näher zu beschreiben: "Irgendwie stößt man dann auf Details und verzettelt sich in den Details und die versucht man zu lösen, an die man halt zuerst nicht gedacht hat. Und dann hat man dann halt eher so das Gefühl, dass die ganze Lösung recht in die falsche Richtung geht." Auch das ähnelt der Art und Weise, wie spontan Schreibende sich inhaltlich 'verrennen' können: "Er [der spontan Schreibende] verfolgt einen Gedanken nach dem anderen, schweift vom Thema ab, sieht das Ende nicht." [21, S.32]

Ganz ähnlich scheint es auch ID2 zu ergehen, wenn er auf die Frage, was zu Verzögerungen bei Projekten führen kann, antwortet: "Sachen, wo man geplant hat, es könnte so aussehen, und im Nachhinein merkt man halt, dass es halt doch nicht

so aussehen muss, weil es doch eigentlich komplett nicht dem entspricht, was man sich dabei gedacht hat." Hier fällt zwar das Stichwort 'geplant', jedoch scheint das Vorgehen von ID2 allgemein eher einem Herantasten zu ähneln. Er erklärt dazu, dass er sich zu Beginn eines Projekts 'im Kopf denkt', wie die Logik aussehen könnte und "dann probier ich das halt aus". Nur wenn er sich mit einem 'wirklich' komplizierten Problem konfrontiert sieht, "dann malt man sich halt schon mal so ein bisschen so stiefmütterlich so auf", wie die Logik aussehen könnte. Die Verwendung der Beschreibung 'stiefmütterlich' weist allerdings bereits darauf hin, dass das nicht unbedingt die bevorzugte Strategie von ID2 ist. Lieber geht er gegen Probleme mit 'viel rumprobieren' an: Lösungen werden grob entworfen, dann getestet und wieder verworfen, wenn sie nicht funktionieren.

Planendes Schreiben

Das Gegenstück zum spontanen Schreiben: Nichts geschieht ohne reifliche Überlegung, die Schreibenden arbeiten mit Plänen und Gliederungen oder lassen ihre Gedanken ausgiebig im Kopf reifen. Die Struktur des Texts wird so vorher sehr genau festgelegt und ändert sich während des Schreibprozesses nur noch geringfügig. [21, S.34f]

Dieses Vorgehen lässt sich vermutlich bei allen Interviewten finden, denn wie auch Hermans und Aldereweld festgestellt haben, ist das die bevorzugte Strategie des klassischen Software Engineerings [24]. Dazu kommt, dass das Stichwort 'Planung' auch öfter bereits in den Fragen während des Interviews genutzt wurde, wodurch die Interviewten womöglich eher geneigt waren, dieses Thema noch weiter auszuführen. ID2 wendet Planung, in diesem Fall das 'stiefmütterliche Aufmalen', zur Problemlösung bei schwierigen Fällen an. ID3 berichtet im Bezug auf Projekte bei der Arbeit immer wieder, dass er sich zu allem sehr ausführlich Gedanken machen muss und nicht einfach drauflosprogrammieren darf, was er später explizit mit seinem Vorgehen bei privaten Projekten kontrastiert. Am charakteristischsten ist hier wohl das Vorgehen von ID5:

> "Der ganz grobe Plan, wie ist das zu machen, steht schon an der Stelle, wo ich damit anfange, es umzusetzen. Womit's dann halt unterm Strich halt bei der Umsetzung wirklich nur noch das reine, okay, ich muss an der Stelle ein bisschen Code hinschreiben, ich muss dort ein bisschen Code schreiben, mehr ist es dann halt schon nicht mehr."

Einen Plan auszuarbeiten bis zu dem Punkt, an dem das eigentliche Schreiben nur noch ein 'Herunterschreiben' ist, ist typisch für Schreibende, die das planende Vorgehen bevorzugen und konnte auch von Gisbert Keseling bei seinen Interviews zur Erfassung der Schreibstrategien von erfahrenen Wissenschaftlern gefunden werden [28, S.158].

Auch ID7 macht sich vorher Pläne und besonders bei Projekten für die Arbeit

versucht er Design und Implementierung strikt voneinander zu trennen. Bei privaten Projekten plane er die Dinge nicht bis ins letzte Detail durch, hat aber trotzdem eine "ganz, ganz grobe Architekturskizze", bei der die Details dann während der Implementierung ergänzt werden.

Mehrversionen-Schreiben

Wie beim spontanen Schreiben wird bei dieser Strategie ein erster Entwurf ohne viel Planung zügig runtergeschrieben. Statt mit diesem Text weiterzuarbeiten, wird dieser dann aber weggelegt und stattdessen eine neue Version geschrieben. Dem finalen Text wird sich über die immer neuen Versionen langsam angenähert. [21, S.36f]

Diese Strategie scheint manchmal von ID3 angewendet zu werden, spezifisch im Kontext von privaten Projekten. Wie bereits besprochen, schreibt er bei Projektideen, von denen er sich noch nicht sicher ist, wohin sie führen, einen ersten Entwurf 'schnell und schmutzig' herunter, was dem spontanen Schreiben entspricht. "Und sobald ich sehe, es hat Fuß und es könnte ein cooles Projekt werden, was ich weiterentwickeln möchte, restarte ich nochmal komplett." Dadurch entstehen zumindest zwei Versionen und ID3 ist in der neuen Version in der Lage, gezielt Probleme anzugehen, die er in der ersten Version ignoriert hat. In der Schreibdidaktik wird diese Strategie deswegen auch empfohlen, um Schreibblockaden aufzulösen [21, S.37].

Bei den anderen Interviewten taucht diese Strategie in ihren üblichen Prozessen weniger auf, jedoch scheint manchmal auf 'neu schreiben' als Lösungsstrategie bei besonders verzwickten Problemen zurückgegriffen zu werden. Von so einem Fall berichtet etwa ID4. ID7 erwähnt eine ähnliche Situation, bei der eine Neuauflage ein gescheitertes Projekt eventuell noch hätte retten können. Auch die herantastende Vorgehensweise von ID2, die im Abschnitt zum spontanen Schreiben beschrieben wurde, könnte als Mehrversionen-Schreiben interpretiert werden. Allerdings geht aus dem Interview nicht hervor, ob dabei eher verschiedene Versionen von Code und dessen Entwürfen entstehen, oder ob bestehende Entwürfe modifiziert werden. Letzteres würde eher zu der nächsten Strategie passen.

Redaktionelles Schreiben

Bei dieser Strategie wird sich der finalen Textversion über viele Verbesserungen nach und nach angenähert. Die Schreibenden formulieren ein Stück des Textes, gehen dann einen Schritt zurück, lesen das Geschriebene und überarbeiten es. Die erste Version wird insgesamt sehr zügig geschrieben, jedoch entsteht selten die ganze Rohfassung auf einmal. [21, S.38]

Diese Schreibstrategie scheint der Idee der inkrementellen Entwicklung sehr nahe zu kommen [49, S.60-62], weshalb es auch hier nicht verwunderlich ist, dass wir diese Strategie in den Interviews wiederfinden können. Das übliche Vorgehen von ID7 für

die Implementierung, nachdem alle Planungsfragen geklärt sind, sieht etwa folgendermaßen aus: "Ich implementiere meistens einen Teil, schreibe dann Tests dagegen, schau, ob die durchlaufen und wiederhole das so lange, bis das komplette Stückchen Software sozusagen ergänzt ist." Das Vorgehen von ID6, das wir grundsätzlich bereits beim spontanen Schreiben eingeordnet haben, kann in ein Vorgehen übergehen, das dem redaktionellen Schreiben ähnelt, wenn wir seine Vorliebe für das Refactoring miteinbeziehen, das wir im Rahmen von Kapitel 7.2 bereits ausführlich besprochen haben.

ID5 berichtet von so einer redaktionellen Strategie im Rahmen eines Projekts, das sie explizit als Ausnahme nennt, bei dem ihr Vorgehen also nicht ihrem üblichen Programmierprozess entsprochen hat. Es ging um ein Ticket, bei dem sie Code ergänzen sollte, der ursprünglich von einem anderen Autoren stammt "und der Code sah aus wie Kraut und Rüben". Deswegen sei zunächst ein umfangreiches Refactoring nötig gewesen, "wo dann halt sehr sehr viel neuer Code entstanden ist, wo es auch vorher nicht wirklich den Plan gab, wie es aussehen soll, wo es halt wirklich Stück für Stück entstanden ist". Dieses Stück-für-Stück-Arbeiten beschreibt sie auch noch näher: "Also ich hab halt zu einem Teilbereich den Code geschrieben, so wie es im Kopf halt für mich sofort schlüssig war. Und hab dann geschaut, wie ich das nächste Stückchen, was dazu muss, wie ich das dort reinintegriert bekomme." Hier lässt sich deutlich das Wechselspiel aus spontanem Schreiben und Überarbeiten erkennen. Diese Arbeit habe sie auch sehr viel Zeit gekostet, gerade weil es vorher keinen Plan gab.

Puzzle-Schreiben

Beim Puzzle-Schreiben wird an verschiedenen Teilen des Textes 'gleichzeitig' gearbeitet, zwischen denen laufend hin und her gesprungen wird. Die Auswahl erfolgt je nach Lust und Laune. Die Strategie kann Elemente von jeder anderen Schreibstrategie integrieren, wodurch das Puzzle-Schreiben beim Vermeiden von Blockaden helfen kann. [21, S.40f]

Diese Strategie entspricht dem Vorgehen, das ID4 beschreibt. Er bezeichnet es zunächst als 'hin und her programmieren'. Auf Nachfrage, was das bedeute, erklärt er:

> "Dass ich halt nicht wirklich einen Design-Prozess durchlebt habe in dem Projekt für mich, sondern einfach denke: 'Hey, zum Beispiel ein Authentifikations-Service wär jetzt ganz interessant.' Und dass ich dann halt gucke, ja, wo mache ich das jetzt. [...] Und dann habe ich erst mal wieder geguckt, aha, Backend, schön und gut, und hab das dann erst mal im Frontend implementiert, also in der App tatsächlich [...] Und dann habe ich ein bisschen weiter im Frontend gemacht um zu schauen, was wäre denn cool so zu haben, also wie soll es dann am Ende aussehen."

Hier lässt sich das Springen zwischen den verschiedenen Teilbereichen gut nachvollziehen. Auch dass ID4 damit Blockaden ausweichen und seine Motivation oben halten kann, erzählt er explizit: "Wenn ich mir denke, hey, hier ist grad ein Fehler oder eine Blockade, dann denke ich mir halt an der Stelle: Okay, dann machst du an einer anderen Stelle kurz weiter. Weil irgendwann findet sich die Motivation schon wieder, an der Stelle mehr zu lesen." Problematisch wird es nur, wenn es keine anderen Baustellen mehr gibt, auf die er ausweichen kann.

Zusammenfassend können wir sagen, dass wir bei den Interviewten alle Schreibstrategien des wissenschaftlichen Schreibens vorgefunden haben. Ein Überblick über die Verteilung der Strategien bei den Interviewten ist in der Tabelle 7.2 zu sehen.

Strategie	ID1	ID2	ID3	ID4	ID5	ID6	ID7
Spontanes Schreiben		+	+			+	
Planendes Schreiben		+	+	+	+		+
Mehrversionen-Schreiben		+	+	+			
Redaktionelles Schreiben			+		+	+	+
Puzzle-Schreiben				+			

Tabelle 7.2: Schreibstrategien der Interviewten

In der Tabelle fällt auf, dass wir ID1 keine der Schreibstrategien zugeordnet haben. Das liegt daran, dass wir aus den Aussagen von ID1 nicht eindeutig genug auf das Vorkommen einzelner Strategien schließen können. Er berichtet davon, dass für ein Projekt im Team geplant und das dann "ganz normal umgesetzt" wird, was eher für das planende Schreiben spricht, aber durch Nachfragen während des Interviews nicht unbedingt bestätigt werden konnte. Das liegt möglicherweise an der besonderen Situation von ID1, dass er als einziger Entwickler an einer Codebasis arbeitet, mit der er sehr vertraut ist. ID1 sagt dazu auch, dass er "viele versteckte Dinge" im Code kennt. Seine mentale Repräsentation dieser Codebasis wird daher sehr ausgeprägt sein, was dafür sorgen könnte, dass weniger explizite Planungsprozesse notwendig sind.

7.4 Beantwortung von Forschungsfrage 1

In diesem Abschnitt widmen wir uns der Beantwortung der ersten Forschungsfrage, welche lautet:

> **Wie sehen gesunde Programmierprozesse aus und sind sie mit Schreibprozessen vergleichbar?**

Einen gesunden Programmierprozess haben wir im Rahmen dieser Forschungsfrage definiert als einen Prozess, in dem keine Blockaden auftreten. Daraus lässt sich schließen, dass ein Programmierprozess dann gesund ist, wenn die Programmierenden in der Lage sind, beim Auftauchen von Problemen flexibel zu reagieren und angemessene Schritte einzuleiten.

Diese Flexibilität zeigt sich in den Programmierprozessen, die in den Interviews beschrieben werden. In drei Schritten haben wir diese genauer untersucht. In Kapitel 7.1 haben wir festgestellt, dass die Programmierprozesse der Interviewten alle Prozesse des Software Engineerings nach Sommerville (Kapitel 3.1) durchlaufen. Die Prozesse der Softwarespezifikation und Softwarevalidierung werden nicht immer explizit erwähnt, was vermutlich daran liegt, dass andere Teammitglieder als die Interviewten selbst dafür zuständig sind. Wir haben außerdem festgestellt, dass es nicht sinnvoll ist, die Prozesse der Softwareevolution von denen der Softwareentwicklung zu trennen, da die Interviewten in den meisten der beschriebenen Projekte an bereits bestehendem Code arbeiten. Schließlich haben wir den von Sommerville beschriebenen Prozessen zusätzlich den Prozess 'Committen' hinzugefügt, womit alle abschließenden Tätigkeiten gemeint sind, die erledigt werden müssen, bevor ein funktionierender Code ins System eingespeist werden kann.

In Kapitel 7.2 haben wir in einem zweiten Schritt festgestellt, dass sich zusätzlich zu den Prozessen des Software Engineerings auch alle Prozesse des wissenschaftlichen Schreibens in den Programmierprozessen der Interviewten wiederfinden lassen. Die Interviewten berichten von ausführlichen Recherchetätigkeiten, welche der Materialsammlung beim wissenschaftlichen Schreiben ähnelt: Durch sie konstruieren die Programmierenden mentale Repräsentationen des zu ergänzenden Codes, was zu Entwürfen des neu zu schreibenden Codes führt, was wiederum dem Prozess des Strukturierens beim wissenschaftlichen Schreiben entspricht. Das Schreiben der Rohfassung entspricht der Implementierung des Codes, welcher anschließend auf die Bedürfnisse der doppelten Leserschaft von Code überarbeitet wird: Bei den Prozessen des Testens und Debuggens wird der Code für die maschinellen Adressaten überarbeitet, während der Prozess des Refactorings einer Überarbeitung für die menschlichen Adressaten des Codes entspricht.

Schließlich haben wir in Kapitel 7.3 die Programmierprozesse der Interviewten hinsichtlich der dabei verwendeten Schreibstrategien untersucht. Dabei konnten wir feststellen, dass die häufigsten Strategien des wissenschaftlichen Schreibens nach Grieshammer et al. (siehe Kapitel 2.3) alle in den beschriebenen Programmierprozessen zum Einsatz kamen. Die Interviewten nutzen unterschiedliche Strategien für unterschiedliche Projekte und können flexibel auf auftretende Probleme reagieren, indem sie die Strategie wechseln.

Dadurch, dass nachweislich Prozesse und Strategien des Schreibens in den Programmierprozessen zum Einsatz kommen, sehen wir die Vergleichbarkeit zwischen Schreib- und Programmierprozessen definitiv als gegeben an. Die Prozesse sind sich ähnlich genug, dass wir davon ausgehen können, dass sie auch auf kognitiver Ebene

von ähnlichen Prozessen Gebrauch machen und entsprechend auch von ähnlichen kognitiven Problemen geplagt werden können. Um das zu bestätigen, werden wir Probleme von Programmierprozessen im Rahmen der zweiten Forschungsfrage näher untersuchen.

Es gibt aber auch einige beachtenswerte Unterschiede zwischen Schreib- und Programmierprozessen, auf die wir zuvor noch kurz eingehen möchten. Zum einen funktioniert Code im Gegensatz zu Text sozusagen auf zwei Ebenen: Er wird auf der einen Seite gelesen, gleichzeitig aber auch ausgeführt. Diese beiden Funktionsebenen passen zu den beiden Adressatengruppen von Code, die wir bereits in Kapitel 4 angesprochen haben: Die maschinellen Adressaten einerseits, für die das *Was* des Codes relevant ist und die für dessen Ausführung zuständig sind. Und andererseits die menschlichen Adressaten, die daran interessiert sind, *wie* der Code funktioniert. Wie diese beiden unterschiedlichen Adressatengruppen im Programmierprozess relevant werden, haben wir beim Prozess des Überarbeitens in Kapitel 7.2 ausführlich besprochen. Durch die verschiedenen Bedürfnisse der beiden Adressatengruppen sind beim Programmieren mehrere unterschiedliche Überarbeitungsprozesse notwendig, nämlich Debugging auf der einen und Refactoring auf der anderen Seite. Die reine Präsenz von mehr unterscheidbaren Prozessen halten wir aber nicht für eine wesentliche Einschränkung bei der Vergleichbarkeit von Schreib- und Programmierprozessen.

Ein zweiter wesentlicher Unterschied zwischen Schreiben und Programmieren ist die Tatsache, dass Texte nach ihrer Veröffentlichung für gewöhnlich als 'fertig' gelten und nicht wieder verändert werden. Auch hier gibt es Ausnahmen, etwa bei neuen Auflagen, doch auch diese verändern den Text selten so grundlegend, wie das bei der Softwareevolution der Fall ist. Damit Software relevant bleibt und weiterhin funktionieren kann, muss sie ständig verändert und neuer Technologie oder neuen Gesetzen und Geschäftsprozessen angepasst werden, wodurch der größte Teil der Softwareentwicklung tatsächlich aus Softwareevolution besteht [49, S.288f].

Die Frage ist nun, ob das die Vergleichbarkeit von Schreib- und Programmierprozessen einschränkt. Auch hier denken wir, dass das nicht der Fall ist. Wie die Analyse der Programmierprozesse in den Interviews gezeigt hat, lassen sich die Prozesse der Softwareevolution ohne Probleme in den Entwicklungsprozess integrieren. Der wesentliche Unterschied zu Programmierprozessen, bei denen komplett neue Software entwickelt wird, ist der große Prozess des Codeverstehens, der notwendig ist, um sich in das bestehende Softwaresystem einzuarbeiten. Genau aus diesem Grund halten wir aber den Prozess spezifisch des wissenschaftlichen Schreibens für einen besseren Vergleichsgegenstand als zum Beispiel das Schreiben eines persönlichen Briefes. Denn auch beim wissenschaftlichen Schreiben ist ein umfangreicher Verstehensprozess sehr gut sichtbar, der in diesem Fall primär aus Literaturrecherche besteht. Ein wissenschaftlicher Text ist stets Teil eines Diskurses, von dem die Schreibenden sich zunächst ein genaues Bild machen müssen, um ihren Text in diesen Diskurs einbetten zu können.

Abschließend lässt sich sagen, dass zwar Unterschiede zwischen Schreiben und Programmieren vorhanden sind, diese aber für die Vergleichbarkeit im Rahmen dieser Arbeit vernachlässigbar sind. Deswegen sehen wir die Vergleichbarkeit zwischen Schreib- und Programmierprozessen als gegeben an.

8 Forschungsfrage 2: Ergebnisse und Diskussion

Um die zweite Forschungsfrage zu beantworten und zu zeigen, dass auch Programmierprozesse von Blockaden betroffen sein können, analysieren wir die Probleme, von denen die Interviewten berichten, genauer. In Kapitel 8.1 ordnen wir die Probleme nach Themen, wobei wir besondere Aufmerksamkeit auf die kognitiven Probleme legen. Diese vergleichen wir anschließend in Kapitel 8.2 mit den Blockaden beim wissenschaftlichen Schreiben. In Kapitel 8.3 fassen wir die Ergebnisse zusammen und beantworten die zweite Forschungsfrage.

8.1 Probleme in den Prozessen

Die Probleme, die die Interviewten schildern, sind vielfältig und eng mit den Prozessen verwoben. Sie berichten von zwischenmenschlichen Problemen bei der Arbeit im Team, von dem Frust über Entscheidungen von Seiten des Managements, von der Erschöpfung durch die kognitive Belastung und von Bugs und Logikproblemen, für die sich scheinbar keine Lösungen finden lassen. Schnell wird deutlich, dass Probleme von allen möglichen Einflussfaktoren abhängig sein können. An dieser Stelle scheint es hilfreich, das Modell des Schreibprozesses von Hayes wieder aufzugreifen, welches in Kapitel 2.2 besprochen wurde. Das Modell gilt in der Schreibberatung als besonders nützlich, da es einen guten Überblick über die Faktoren, welche den Schreibprozess beeinflussen, liefert und dadurch aufzeigt, wo Ursachen von Problemen überall liegen können [21, S.21]. Diesen Überblick wollen wir uns nun auch für den Programmierprozess zunutze machen.

In diesem Abschnitt wollen wir daher die von den Interviewten geschilderten Probleme betrachten und überlegen, wie sie in dem Prozessmodell von Hayes einzuordnen wären. Dem Modell folgend betrachten wir also die physische und soziale Aufgabenumgebung, Motivation und Affekt, die kognitiven Ressourcen des Arbeits- und Langzeitgedächtnisses und schließlich die kognitiven Prozesse. Da für diese Arbeit vor allem letztere bedeutsam sind, soll der Überblick über die anderen Bereiche einigermaßen oberflächlich gehalten werden. Dabei ist aber zu beachten, dass die verschiedenen Bereiche alle miteinander verbunden sind und sich gegenseitig beeinflussen, weshalb Probleme sich auch mehreren Bereichen zuordnen lassen oder Probleme aus einem Bereich sich auf andere Bereiche ausweiten können.

<table>
<tr><th colspan="2">Aufgabenumgebung</th><th colspan="2">Motivation und Affekt</th><th colspan="2">Kognitive Prozesse</th></tr>
<tr><td>Ausscheiden von Kollegen</td><td>Technische Probleme</td><td>Kosten-Nutzen-Rechnung geht nicht auf</td><td>Perfektionismus</td><td>Unklare/ ändernde Anforderungen</td><td>Lösungsansatz hat nicht funktioniert</td></tr>
<tr><td>Persönliche Konflikte</td><td>Veraltete Technologie</td><td>„Das ist gar kein Programmieren“</td><td>Lernziel erreicht</td><td>Zu große / komplexe Anforderungen</td><td>Nicht vorhandener Lösungsansatz</td></tr>
<tr><td>Ablenkung</td><td>Unangemessene Technologie</td><td colspan="2">Langzeit- und Arbeitsgedächtnis</td><td>Projekt wird größer als erwartet</td><td>Bugs / Fehler implementiert</td></tr>
<tr><td>Unsauberes Arbeiten</td><td>Schlechte Qualität existierenden Codes</td><td>Kognitive Überlastung</td><td>Mangel an Wissen</td><td>Probleme ‚verästeln‘ sich</td><td>Henne-Ei-Problem</td></tr>
</table>

Abbildung 8.1: Arten von Problemen sortiert nach Bereichen

Insgesamt wurden in den Interviews 41 voneinander unabhängige Aussagen zu Problemen identifiziert. Diese wurden zu 22 Arten von Problemen zusammengefasst, die dann wiederum einem der genannten Bereiche zugeordnet wurden. Ein Überblick ist in Abbildung 8.1 zu sehen.

Aufgabenumgebung

Bei der sozialen Aufgabenumgebung stehen vor allem die Teammitglieder und die Interaktion mit diesen im Vordergrund. Besonders beim Gespräch mit ID3 wird deutlich, wie viel zwischenmenschliche Arbeit Teil des Alltags ist. Er erzählt, wie frustrierend es sein kann, wenn andere Programmierende Abkürzungen suchen, weil sie keine Lust haben, sich mit einem fehlgeschlagenen Unit Test herumzuschlagen, oder wenn sie Code schreiben, der in den eigenen Augen nicht gut genug ist, und wie das zu Konflikten im Team führen kann. Aber ”der Mensch ist nicht der Code, den er schreibt”, betont ID3, das sei wichtig zu verinnerlichen. Er ergänzt später, dass oft auch Anweisungen von Vorgesetzten der Grund sein können, dass nicht genug Zeit in Refactoring gesteckt wird.

Auf der anderen Seite liegt das Problem, von dem ID1 und ID5 berichten, nämlich wenn kein Team vorhanden ist. So ist ID1 derzeit der einzige Entwickler in seinem Team, wodurch auch einige größere Programmierprojekte in seiner Firma auf Eis gelegt werden mussten. ID5 erzählt von Schwierigkeiten bei einem Projekt, bei dem aufgrund von langsamer Kommunikation die Teammitglieder, die ursprünglich das Konzept für das Projekt erarbeitet hatten, in der Zwischenzeit die Firma verlassen hatten.

Unsauberes Arbeiten und daraus resultierender schlechter Code können dann wiederum zum Problem anderer Programmierender als Teil ihrer physischen Aufgabenumgebung werden. ID5 erzählt bei einem Projekt von Code, der ”wie Kraut und

Rüben" aussah, während die Kollegen von ID2 über das Altsystem, mit dessen Reengineering sie beschäftigt sind, sprechen wie über ein mystisches Monster: "Da gibt's halt so Sachen so, jah, das fassen wir lieber nicht an[...]. Wenn wir das machen würden, dann wüssten wir nicht, was danach passiert." Noch drastischer ist ein Fall von ID7, bei dem die Qualität eines übernommenen Codes "eher so Minus Fünf" war, weshalb dieses Projekt schließlich komplett aufgegeben werden musste.

Ansonsten finden sich im Bereich der physischen Aufgabenumgebung diverse technische Probleme. Abgestürzte PCs (ID3) und veraltete Technologie (ID5) gehören hier dazu. ID7 erzählt auch, dass er schon manchmal versucht habe, Technologien auf ein Problemfeld anzuwenden, für das sie nicht geeignet waren, wobei es sich dabei bis zu einem gewissen Grad um kalkuliertes Risiko gehandelt habe, bei dem er gewusst habe, dass es vielleicht nicht funktionieren würde.

Das letzte Problemfeld, das im Bereich der Aufgabenumgebung angesprochen wird, sind diverse Ablenkungen. ID1 muss sich aufgrund seiner besonderen Situation als alleiniger Entwickler in seinem Team oft um alle möglichen technischen Probleme kümmern. Aber auch ID3 lamentiert über viele E-Mails, die eigentlich nicht in seinen Zuständigkeitsbereich fallen. Zudem kann es schnell passieren, dass die Zeit in den regelmäßigen Meetings, die bei ID3 zum Workflow gehören, nicht optimal genutzt wird.

Motivation und Affekt

Aufgrund der Tatsache, dass Programmieren für die Interviewten in erster Linie ihr Beruf ist, berichten sie eher weniger von Problemen, die sich dem Bereich der Motivation zuordnen lassen. Bei denjenigen, bei denen solche doch zur Sprache kommen, geschieht das auch meistens im Kontext von privaten Projekten. Diese werden aus diversen Gründen nicht zu Ende geführt. Projekte, die in erster Linie aus der Motivation heraus gestartet werden, sich weiterzubilden, werden oft dann für beendet erklärt, wenn der gewünschte Lerneffekt eingetreten ist, so zum Beispiel bei ID1. ID3 bricht Projekte ab, wenn sie ihm zu repetitiv werden, nachdem die anfängliche Neuheit des Projekts nachlässt. Das empfindet er aber nicht als problematisch, sondern einfach als Teil des Lernprozesses. Sowohl ID3 als auch ID4 brechen Projekte ab, wenn sie merken, dass diese ihre Programmierfähigkeiten nicht ausreichend herausfordern. Für ID4 sind das Aufgaben, bei denen er das Gefühl hat, nur von einer Programmiersprache in eine andere zu übersetzen, anstatt etwas Neues zu schaffen. ID3 nennt an dieser Stelle das Beispiel eines Videospiels, welches er gerne programmiert hätte, nur um dann festzustellen, dass er den Großteil seiner Zeit mit der künstlerischen Gestaltung der Spieleumgebung verbrachte, statt mit Programmieren. In solchen Momenten wären andere zusätzliche Motivationsfaktoren notwendig gewesen, die die Kosten-Nutzen-Abwägungen der Programmierenden ausgleichen würden, damit diese das Interesse an den Projekten nicht verlieren.

Kosten-Nutzen-Abwägungen spielen auch für größere Institutionen als das Indi-

viduum eine Rolle, weshalb diese auch für die wenigen Fälle, in denen tatsächlich berufliche Projekte abgebrochen wurden, als Grund genannt werden. Solche Fälle werden von ID2 und ID7 genannt. Die Kosten-Nutzen-Abwägung findet hier jeweils auf der Ebene des Managements statt. Die eigentlichen Probleme, die die Kosten in die Höhe treiben, liegen auf anderen Ebenen auch in anderen Bereichen, wie in dem bereits besprochenen Fall mit dem schlechten Vorgängercode, von dem ID7 berichtet.

Schließlich ordnet Hayes auch den Einfluss von Zielen in den Bereich der Motivation ein, weswegen an dieser Stelle noch das Problem des Perfektionismus' erwähnt werden soll. Dieses wird von ID6 angesprochen: "Manchmal kommt man halt einfach nicht auf die eine Lösung für ein Problem, wo ich jetzt sage, da ist alles perfekt." In diesem Fall habe man die Wahl, das zu einem echten Problem werden zu lassen, oder Kompromisse zu schließen.

Langzeit- und Arbeitsgedächtnis

Das Langzeitgedächtnis ist im Schreibprozess dafür zuständig, unser Wissen bereitzustellen. Aus diesem Grund ordnen wir das Problem von fehlendem Wissen in diesem Bereich ein. Auch kompetente Programmierende können immer wieder auf Problemfelder stoßen, für die ihr bisheriges Wissen nicht ausreichend ist. ID3 ist auf so einen Fall im Kontext von Spielehacks gestoßen. Dort sei "alles sehr maschinennah und wer entwickeln kann, versteht nicht plötzlich, was der PC mit Nullen und Einsen macht". Auch ID4 weiß von solchen Problemen zu berichten. Als er zum ersten Mal für ein EEG programmieren soll, weiß er zwar, wie die Daten aussehen, die verarbeitet werden müssen, und mit welchen Bibliotheken er sie visualisieren kann, aber ihm fehlt das Wissen darüber, wie das EEG an sich funktioniert, was ihm Probleme bei der Programmierung bereitet. Er erklärt: "Wenn das [Domänen-Wissen] nicht da ist, bringt natürlich jede Library-Dokumentation nichts, wenn man einfach nicht weiß, *was* man machen möchte."

In Bezug auf das Arbeitsgedächtnis wollen wir an dieser Stelle kurz auf das Problem der kognitiven Überlastung eingehen. Während des Programmierens müssen Programmierende sehr viele Informationen gleichzeitig handhaben, was das Arbeitsgedächtnis stark beansprucht [42]. Bei ID1 äußert sich das in einem Zustand, den er selbst als "Codeblind" bezeichnet: "Weil ich gefühlt die Hälfte der Dinge, die ich da gerade lese, gar nicht mehr wahrnehme, also im Code." Möglicherweise lassen sich hier auch die Aussagen von ID2 einordnen, der von einem sehr komplexen Projekt erzählt, welches "das Gehirn schon krass gefordert hat" und wo er "viel Gehirnschmalz reinstecken" musste.

Kognitive Prozesse

> "Ja, natürlich. Den gedanklichen Deadlock, ich glaube, den kennt jeder Entwickler."
>
> *ID7*

Schließlich wenden wir uns den Problemen zu, die sich im Bereich der kognitiven Prozesse einordnen lassen. Die hier genannten Probleme lassen sich grob in drei Kategorien ordnen: Probleme mit den Anforderungen, Probleme mit den Lösungen und Fehler bei der Implementierung.

Die Probleme mit den Anforderungen sind die umfangreichste Kategorie. Ein erstes Problem besteht, wenn die Anforderungen von Anfang an nicht klar genug sind oder sich häufig ändern. Wie ID3 es ausdrückt: "Leute, die die Anforderungen stellen, haben keine Ahnung, was sie wollen." Das könne bei schlechtem Management zu einem Workflow führen, bei dem Kunden quasi täglich "irgendwelche Änderungen reinzuschieben" versuchen und dann "kann es sein, dass du alles wieder vom Tisch werfen musst und neu anfangen musst". Die Interviewten arbeiten zwar meistens in Umgebungen, in denen solche Probleme durch gutes Projektmanagement bereits vermieden werden sollen. Was aber passieren kann, wenn dieses versagt, zeigt ein Fall, von dem ID5 berichtet:

> "Der Kunde hatte am laufenden Band ändernde Anforderungen. [...] Das ist irgendwie so zwei, drei Mal passiert, dass ich halt zwei, drei Wochen Arbeit versenkt habe und der Kunde dann sagte: 'Nee, eigentlich will ich das doch anders.' Und letztlich bin ich darüber in nen Burnout gerannt und hab letzten Endes die Firma verlassen."

Ein anderes Problem, das auftreten kann, ist, dass die Anforderungen schlicht zu groß oder zu komplex sind. ID3 erzählt davon, dass das manchmal vorkommt, auch bei privaten Projekten in Form von "richtig riesigen" Ideen, bei denen es ihm dann aber nicht möglich war, herauszufinden, wie er das in Code darstellen könnte: "Es kam auch schon vor, dass ich deswegen ein Projekt verworfen hab. Weil ich mir einfach nicht in meinem Kopf definieren konnte, wie setz ich das in Realität um." Auch ID6 sagt, dass es manchmal einfach Dinge gebe, die zu komplex seien.

Nicht immer lässt sich ein komplexes Problem schon im Vorhinein erkennen. ID7 berichtet davon, dass Probleme sich unerwartet sehr weit auffächern können, insbesondere wenn an bereits existentem Code etwas geändert werden muss: "Wo du halt irgendwelche eigentlich sehr, sehr kleinen Änderungen hast, die aber einfach durch Abhängigkeiten im Code an sehr, sehr vielen Stellen plötzlich [für] Veränderungen sorgen." In solchen Momenten könne es sehr schwierig sein, den Überblick zurückzugewinnen. Auch ID2 berichtet von Verzögerungen in Projekten, die dadurch entstehen, dass mit Veränderungen am Code "Sachen wieder auftreten, von denen man vorher nicht wusste, dass die dadurch dann auftreten".

Eine andere Form von Komplexität nennt ID7 das "Henne-Ei-Problem". Dieses entsteht, wenn mehrere Problemstellungen vorhanden sind, welche sich gegenseitig bedingen. Dadurch kann es manchmal schwierig sein, überhaupt einen Anfang zu finden.

Probleme mit den Lösungen werden auf zwei Arten angesprochen: Zum einen, wenn Ideen für eine Lösung einfach ausbleiben, zum anderen wenn die angedachten Lösungen nicht funktionieren. Ersteres haben wir bereits bei ID3 im Fall der zu großen Ideen gesehen. Auch ID6 berichtet von solchen Momenten, welche er als "kreative Blockaden" empfindet. Auf Nachfrage, wie eine kreative Blockade beim Programmieren aussehen würde, erklärt er nämlich: "Dass du halt einfach vor einem Problem sitzt und dir dazu nichts einfällt."

Probleme, bei denen die ursprünglich angedachte Lösung nicht funktioniert, beschreibt er stattdessen eher als schleichendes Bauchgefühl, welches wir auch schon im Rahmen von Kapitel 7.3 näher betrachtet haben. Da könne es vorkommen, "dass man zum Beispiel mal nen Sprint lang an einem bestimmten Problem arbeitet und danach feststellt: Okay, das, was wir uns überlegt haben als Lösung für dieses Pro-blem jetzt, was unser erstes Bauchgefühl ist, hat nicht funktioniert." ID6 erklärt, dass es dann nötig ist, einen Schritt zurückzugehen und mit den gesammelten Erfahrungen das Problem noch einmal von vorne zu betrachten. Dass Programmierende sich in solchen Situationen aber auch regelrecht an einem Problem aufhängen können, zeigt die Beschreibung von ID2: "Und dann muss man das wieder umdenken und dann wieder umdenken und dann wieder neu versuchen und wieder neu versuchen, da kommt man halt nicht vorwärts." Auch bei ID3 klingt das ähnlich: "Logischerweise kommt es mir auch gerne mal vor, dass ich dann versuche, Sachen zu implementieren, es geht nicht, es geht nicht, es geht nicht, es geht nicht und dann habe ich irgendwann einen Fehler implementiert."

Womit wir auch bei der dritten Kategorie in diesem Bereich wären: Fehler bei der Implementierung. Tatsächlich wird grundsätzlich das Vorkommen von Bugs und Fehlern noch nicht unbedingt als Problem für den Programmierprozess empfunden. ID6 erklärt, dass er sogar sehr viel Spaß daran hat, sich tief in die Details des Codes einzudenken, um einen Bug zu beheben, und meint: "Mit den richtigen Werkzeugen wirst du den früher oder später finden." Problematisch wird es erst dann, wenn der Fehler sich eben doch nicht aufspüren lässt. Von solchen verzwickten Fällen erzählen ID4 und ID5. ID4 hatte das Problem, dass ein Fehler bei nebenläufigen Prozessen aufgetreten ist, er aber nicht in der Lage war, zu bestimmen, welcher Prozess genau das Problem verursacht. Die üblichen Testverfahren haben nicht funktioniert, da der Fehler mehr oder weniger zufällig nur aufgetreten ist. Auch bei ID5 funktionieren die üblichen Lösungsstrategien nicht, da ihr Fehler in einer Phase des Boot-Prozesses auftritt, in dem sie kein Feedback von anderer Software erhalten kann.

8.2 Vergleich mit den Blockaden beim wissenschaftlichen Schreiben

In diesem Abschnitt wollen wir die beschriebenen kognitiven Probleme mit den Blockaden beim wissenschaftlichen Schreiben nach Gisbert Keseling vergleichen, welche wir in Kapitel 2.4.2 vorgestellt haben. Wie in besagtem Kapitel bereits angemerkt wurde, sind die konkreten Diagnosen von Keseling sehr individuell und auch auf der Ebene der fünf Subgruppen, in die er die Blockaden einteilt, beziehen sie sich noch sehr spezifisch auf den Prozess des wissenschaftlichen Schreibens. Deswegen wird der Vergleich primär auf der Ebene der übergeordneten Kategorien der Blockaden bei der Konzeptbildung und der adressatenbezogenen Blockaden stattfinden. Eine ähnlich detaillierte Gruppierung für Blockaden beim Programmieren zu erstellen, wie Keseling das für die Blockaden beim wissenschaftlichen Schreiben getan hat, liegt nicht im Interesse dieser Arbeit, denn dafür ist die Datenlage nicht ausreichend. Wir beschränken uns an dieser Stelle auf die Frage, ob überhaupt die Möglichkeit für Blockaden in den genannten Kategorien besteht.

8.2.1 Blockaden bei der Konzeptbildung

Es spricht vieles dafür, dass Blockaden dieser Kategorie auch bei Programmierprozessen auftreten können. Keseling vermutet, dass diese Blockaden die Gemeinsamkeit haben, dass mit den 'Vorgestalten', welche Schreibende von ihrem Text entwickeln, etwas nicht stimmt; dass ihre mentale Repräsentation des Texts also unvollständig oder fehlerhaft ist [28, S.103-107]. Dass auch Programmierende eine mentale Repräsentation des Programms entwickeln, haben wir in Kapitel 3.2 feststellen können. Die meisten der in den Interviews beschriebenen kognitiven Probleme lassen sich ebenfalls auf dieses Konzept der fehlerhaften Vorgestalten zurückführen:

Um die gewünschten Anforderungen erfolgreich umzusetzen oder Fehler finden zu können, müssen Programmierende eine möglichst gute Vorgestalt von der zu schreibenden Software haben. Unklare Anforderungen oder ständige Änderungen bei den Anforderungen erschweren es, diese gute Vorgestalt überhaupt erst zu bilden. Ebenso verhält es sich mit Anforderungen, welche zu umfangreich oder zu komplex sind. Das Beispiel von ID3, welcher bei "riesigen Ideen" nicht in der Lage war, eine Vorstellung zu entwickeln, wie diese Idee in Code umgesetzt aussehen könnte, ähnelt hier sehr der von Keseling identifizierten ersten Variante in der Gruppe von Blockaden, welche mit fehlenden, unstimmigen oder instabilen Konzepten zusammenhängen: Bei dieser Variante der Blockade waren die Schreibenden nicht in der Lage, ihr Thema einzugrenzen und zu einer angemessenen Fragestellung zu gelangen, weshalb sie sich unüberwindbaren Bergen von Arbeit gegenüber sahen [28, S.70-73]. Im Endeffekt war auch hier die ursprünglich angenommene Schreibaufgabe zu groß, um zu einem Konzept zu gelangen, so wie es auch ID3 bei seinem Programmierprojekt schildert.

In den Fällen, in denen sich Probleme unerwartet auffächern, wie von ID2 und ID7 geschildert, kann es sein, dass die Vorgestalt nicht detailliert genug ist, um die plötzlich auftretenden Veränderungen nachvollziehen zu können, was den beschriebenen Verlust des Überblicks zur Folge hat. Möglicherweise kommt es dadurch auch zu einer Art kognitiver Überlastung, bei der das Arbeitsgedächtnis nicht in der Lage ist, die hohe Anzahl plötzlicher Änderungen an der mentalen Repräsentation zu verarbeiten, welche nötig wären, um den Überblick zu behalten.

Das "Henne-Ei-Problem" scheint ähnlicher Natur zu sein. Die Vorgestalt ist hier womöglich nicht ausgereift genug, weshalb kein Ansatz für die Problemlösung gefunden werden kann. Laut ID7 hat sich bei diesem Problem die Gegenstrategie des "Rubberduck-Debuggings" bewährt, also das Problem einem Teammitglied zu schildern, "weil man dann halt schnell mal auf neue Gedanken kommt". Das Gespräch hilft also möglicherweise dabei, die Vorgestalt zu überarbeiten.

Auch die Probleme mit den Lösungen lassen sich mit der Idee der fehlerhaften Vorgestalt verknüpfen. Beim Ausbleiben einer Idee für die Lösung gelingt es scheinbar gar nicht, eine Vorgestalt zu bilden. Keseling identifiziert Ideenmangel aber als *Symptom* von Schreibblockaden, nicht als Ursache davon [28, S.52f]. Wie sich das beim Programmieren verhält, lässt sich aus den kurzen Berichten im Rahmen der Interviews nicht entnehmen.

Bei Lösungen, welche nicht funktionieren, könnte es sein, dass entweder die mentale Repräsentation der Lösung oder die des Programms, in welches sie eingefügt werden soll, fehlerhaft ist, beziehungsweise dass die beiden Repräsentationen aus irgendeinem Grund nicht zusammenpassen.

Schließlich gibt es noch die Probleme mit Fehlern bei der Implementation beziehungsweise Bugs. Hier ist auffällig, dass diese erst wirklich als Problem wahrgenommen werden, wenn die bewährten Strategien zum Finden von Fehlern versagen, wie in den Fällen von ID4 und ID5. Das deckt sich mit der Definition von Blockaden und spricht daher dafür, dass es sich in diesen Fällen auch wirklich um Blockaden handelt. Auch dieses Problem lässt sich mit der fehlerhaften Vorgestalt erklären. Wie bei den ausufernden Seiteneffekten bei Änderungen in bereits bestehenden Programmen, könnte auch hier die Vorgestalt schlicht nicht detailliert genug sein, um alle möglichen Fehlerquellen zu erfassen.

Sogar ein Problem, das wir außerhalb der kognitiven Prozesse angesiedelt haben, passt in dieses Bild: Bei dem Projekt mit dem EEG von ID4, bei dem er die Probleme schlicht mit fehlendem Wissen über die Funktionsweise des EEGs erklärt, hatten wir im Rahmen von Kapitel 7.2 bereits darüber gesprochen, inwiefern ihm die Erweiterung des besagten Wissens beim Programmieren geholfen hat. An dieser Stelle möchten wir uns den genauen Wortlaut während des Interviews noch einmal genauer ansehen:

> "Beziehungsweise habe ich dort auch mal eine Sache, die ich gebraucht hab, nachts in einem Youtube-Video gefunden, was gar nichts mit EEGs

> eigentlich zu tun hatte. [...] Das war, falls das aus Marvel bekannt ist, es gibt da [die Figur Doctor Octopus[1]], der solche mechanisch steuerbare Arme hat. Und das hat jemand in einem Hobbyprojekt versucht nachzubauen mit nem EEG, wo es dann darum ging, diese Arme irgendwie mit Gedanken zu steuern. Das war alles sehr stümperhaft gemacht, hat aber so ein paar Ideen gebracht, was man machen könnte. Das heißt, wie man dann zum Beispiel mit den Hirnwellen oder mit den einzelnen Frequenzbändern agieren könnte [...] Also, dass ich dann eine wesentlich klarere Vorstellung hatte, wie das tatsächlich abläuft und dementsprechend intern einfach irgendwie eine bessere Repräsentation hatte."

ID4 scheint hier eher zufällig, also nicht mit dem eindeutigen Ziel, sein Wissen im Bereich EEG zu erweitern, auf ein Video gestoßen zu sein, welches auch gar nicht hauptsächlich zum Ziel hat, Wissen über das EEG zu vermitteln. Trotzdem war die neue Perspektive für ID4 hilfreich beim Konstruieren der mentalen Repräsentation. Das zeigt zum einen, dass es sich an dieser Stelle nicht um ein Problem gehandelt hat, welches auf mangelnde Programmierkompetenzen zurückzuführen ist, was ein kognitives Problem wahrscheinlicher macht. Zum anderen zeigt dieser Fall, dass zu einer guten mentalen Repräsentation mehr gehört, als nur den Code zu kennen.

8.2.2 Adressatenbezogene Blockaden

Im Gegensatz zu den Blockaden, die bei der Konzeptbildung auftreten, liefern die Interviews nur wenige Hinweise auf adressatenbezogene Blockaden. Die in den Interviews beschriebenen kognitiven Probleme ließen sich alle eher dem Bereich der Konzeptbildung zuordnen. Das tatsächliche Formulieren des Codes scheint für die Interviewten kein Problem zu sein. Die einzige Aussage, die in diese Richtung mögliche Probleme andeutet, stammt von ID6 in Bezug auf den Prozess des Refactorings: "Es ist halt auch wichtig, zu sehen, dass man quasi dieses Wissen, keine Angst vor Veränderung und vor nochmal ne Entscheidung zu revidieren und zu reviewen und daraus Schlüsse zu ziehen, dass man davor keine Angst hat." Es ist laut ID6 also wichtig, keine Angst vor Veränderungen im Code zu haben. Im Umkehrschluss impliziert die Aussage, dass diese Angst durchaus existieren kann. ID6 kennt sie vielleicht selbst oder hat sie möglicherweise bei anderen erlebt. Angst davor, beim Schreiben etwas falsch zu machen, ist laut Keseling eines der Symptome für Blockaden, welche sich auf einen zerstörerischen inneren Adressaten zurückführen lassen [28, S.113].

Ein Grund dafür, dass diese sonst in den Interviews nicht auftauchen, könnte in dem begrenzten Vokabular oder der begrenzten Ausdruckskraft von Programmiersprachen liegen. Oder wie ID5 es ausdrückt: "Du hast nicht so viele Wörter zur Auswahl, um eine Schleife zu machen."

[1] `https://en.wikipedia.org/wiki/Doctor_Octopus`

Im Rahmen seiner Untersuchungen zu Schreibblockaden führte Gisbert Keseling auch eine Studie durch, die sich mit der Frage befasste, wie Schreibende zu ihren finalen Formulierungen gelangen. Dabei stellte er fest, dass beim Schreiben eines förmlichen Briefes der häufigste Grund für Neuformulierungen in der Textsorte lag: Die Schreibenden wollten einen bestimmten Jargon, nämlich Amts- oder Geschäftsdeutsch, treffen [28, S.258]. Zu Neuformulierungen wird also gegriffen, wenn es um die Ästhetik des Textes geht, um die Frage, *wie* etwas ausgedrückt werden soll. Wie wir in Kapitel 4 angesprochen haben, ist es den primären Adressaten des Codes, nämlich den maschinellen, grundsätzlich egal, *wie* die Anweisungen im Programm ausgedrückt werden. Fragen der Ästhetik können sich also höchstens in Bezug auf die menschlichen Adressaten des Codes stellen.

Die maschinellen Adressaten an sich sind möglicherweise auch ein weiterer Grund für das Fehlen dieser Blockaden: Durch das sofortige Feedback von Computer und Compiler müssen die Programmierenden sich nicht bei jedem Wort selbst überlegen, ob dieses passen könnte oder nicht. Entsprechend besteht quasi gar keine Gelegenheit für die Konstruktion eines zerstörerischen inneren Adressaten.

Den menschlichen Adressaten scheint es allerdings nicht egal zu sein, wie der Code, mit dem sie arbeiten müssen, aussieht, worauf zumindest zahlreiche Aussagen in den Interviews schließen lassen. Auf die Frage, welche Gedanken ID2 durch den Kopf gehen, wenn er programmiert, antwortet er zunächst scherzhaft: "Was ich mir dabei so denke? Erst mal natürlich: 'Wer hat das denn angestellt?'" ID3 erzählt, dass er Code schreiben möchte, mit dem er "selbst zufrieden" ist. Darunter versteht er, dass sein Code nicht nur funktioniert, sondern auch qualitativ hochwertig ist, was für ihn wiederum bedeutet, dass andere Programmierende ihn ohne Probleme verstehen sollen. Er bringt an dieser Stelle auch das Argument der Wirtschaftlichkeit ein: Wenn künftige Programmierende sich erst fünf Stunden in einen Code einlesen müssen, geht dadurch wertvolle Arbeitszeit verloren. Bei ID5 erinnern wir uns an die Aussage, dass der Code, den sie für ein Projekt überarbeiten sollte, "wie Kraut und Rüben" aussah und ID7 berichtet davon, dass "manchmal Schönheit für Geschwindigkeit draufgeht". Im letzteren Fall werden bei der Ästhetik also bewusst Abstriche gemacht, was impliziert, dass Überlegungen zur Ästhetik durchaus angestellt werden. Es findet also trotz des eingeschränkten Vokabulars so etwas wie Formulierung statt und es scheint durchaus üblich zu sein, dass diese auch zum Ziel von Kritik durch andere Programmierende wird. Trotzdem scheint das Formulieren zumindest den interviewten Programmierenden keine Probleme zu bereiten.

Ein Grund dafür wiederum könnte in der Erfahrung der Interviewten liegen. Blockaden entstehen durch falsche und falsch angewandte Regeln und Vorannahmen. Es ist davon auszugehen, dass diese falschen Vorstellungen mit wachsender Erfahrung korrigiert und dadurch abgebaut werden können. Es wäre daher interessant zu vergleichen, ob weniger erfahrene Programmierende vielleicht mehr Probleme mit dem Formulieren haben.

Möglicherweise greift an dieser Stelle aber auch die angewandte Methodik zu kurz.

Wie wir in Kapitel 7.2 bereits feststellen konnten, haben die Interviewten den Prozess des Code-Schreibens kaum näher beschrieben. An dieser Stelle wären womöglich mehr Nachfragen vonnöten gewesen. Zentral ist hier auch die Frage P11, welche in das Interview aufgenommen wurde, spezifisch um nach der Beziehung zwischen den Programmierenden und den Adressaten zu fragen. Die Frage wurde aber oft missverstanden und musste genauer erklärt werden, wodurch mögliche Adressaten wie andere Programmierende oder Kunden genannt wurden und dadurch den Interviewten bereits vorgegeben waren. Die Frage hätte wohl sorgfältiger formuliert werden sollen, um Missverständnissen vorzubeugen und detailliertere Auskünfte zu erhalten.

8.3 Beantwortung von Forschungsfrage 2

In diesem Abschnitt soll nun die zweite Forschungsfrage beantwortet werden, welche lautet:

> **Gibt es beim Programmieren ähnliche Blockaden wie beim Schreiben?**

Das grundsätzliche Vorhandensein von Blockaden lässt sich bereits aus deren Definition erschließen: Sie entstehen dann, wenn für die Aufgabe unpassende Strategien angewendet oder mit unpassenden Strategien auf auftretende Probleme reagiert wird. Nach dieser Definition können kognitive Blockaden in jedem kognitiv gesteuerten Prozess auftreten, weswegen die Forschungsfrage den Vergleich mit den Schreibblockaden miteinbezieht.

In Kapitel 8.1 haben wir die Interviews nach Aussagen, welche Probleme im Programmierprozess beschreiben, durchsucht und diese analysiert, um sie einem Bereich von Einflussfaktoren zuzordnen. Zur Bestimmung möglicher Problembereiche haben wir uns am kognitiven Modell des Schreibprozesses nach Hayes (Abbildung 2.2) orientiert. Probleme wurden entweder der Aufgabenumgebung, der Motivation, den kognitiven Ressourcen oder den kognitiven Prozessen zugeordnet.

Die Probleme, die sich den kognitiven Prozessen zuordnen ließen, haben wir in Kapitel 8.2 mit den Blockaden beim wissenschaftlichen Schreiben nach Keseling verglichen. Dabei ließ sich feststellen, dass alle genannten kognitiven Probleme sich mit unvollständigen oder fehlerhaften mentalen Repräsentationen erklären lassen. Das legt den Schluss nahe, dass Blockaden bei der Konzeptbildung, wie es sie beim wissenschaftlichen Schreiben geben kann, auch bei Programmierprozessen auftreten können.

Die zweite Kategorie von Schreibblockaden, die adressatenbezogenen Blockaden, ließen sich in den Interviews nicht eindeutig wiederfinden, bis auf eine indirekte Erwähnung. Das könnte verschiedene Gründe haben, welche wir in Kapitel 8.2.2

diskutieren, und bedeutet nicht, dass es diese Blockaden beim Programmieren nicht geben kann.

Insgesamt lässt sich also feststellen, dass kognitive Blockaden beim Programmieren sehr wahrscheinlich sind. Weitere Untersuchungen sind notwendig, um festzustellen, wie häufig sie sind und mit welchen Symptomen sie sich äußern, um eine sinnvolle Kategorisierung durchzuführen.

9 Implikationen

Da die Ergebnisse die Annahme nahelegen, dass kognitive Blockaden beim Programmieren nicht nur vorkommen, sondern auch ähnliche Ursachen wie beim wissenschaftlichen Schreiben haben, stellt sich die Frage, welchen Nutzen wir aus diesem Wissen gewinnen können. In diesem Kapitel sollen daher Implikationen basierend auf dieser Annahme herausgearbeitet werden.

9.1 Blockaden vorbeugen

Über irreführende Vorannahmen aufklären

Betrachten wir die Ursachen für Schreibblockaden nach Rose wie in Kapitel 2.4 erklärt, müssen wir uns die Frage stellen, welche falschen oder irreführenden Vorannahmen, Regeln und Strategien insbesondere Lernende über das Programmieren haben – und auch, welche wir ihnen vermitteln.

Mythen über das Programmieren, wie zum Beispiel, dass Programmieren ein angeborenes Talent ist (es wird von dem sogenannten 'Geek-Gen' gesprochen), beeinflussen die Wahrnehmung der Öffentlichkeit zum Thema Programmieren [33] und prägen Lernende womöglich weit über ihre Zeiten als Programmieranfänger hinaus. Ähnliche Vorstellungen, die Schreiben als angeborenes Talent deklarieren, sind auch als Auslöser von Schreibangst bekannt [23, S.66], es wäre also denkbar, dass sie auch zu Angst vor dem Programmieren führen können. Da Blockaden oft erst in fortgeschrittenen Stadien zum Vorschein kommen und Programmieranfänger sich ohnehin zunächst die notwendigen Kompetenzen aneignen müssen, um funktionierende Programme zu schreiben, bleiben solche irreführende Vorannahmen vermutlich lange Zeit unentdeckt, wenn sie nicht angesprochen werden.

Lehrpersonen sollten sich daher die Zeit nehmen, gängige falsche Vorannahmen gezielt anzusprechen und zu zerstreuen. Ferner sollte Zeit darin investiert werden, zu zeigen, wie der Alltag beim Programmieren aussieht, damit irreführende Annahmen durch realistischere ersetzt werden. Um herauszufinden, welche Vorannahmen gängig sind, könnten Fragebögen verwendet werden, wie diejenigen, die auch schon Mike Rose zur Erforschung von Schreibblockaden genutzt hat [43].

Lehrpersonen sollten sich außerdem bewusst machen, dass sie Lernenden nicht nur Fakten über Syntax und Datenstrukturen vermitteln, sondern auch ganz automa-

tisch Ideen und Vorstellungen darüber, was es bedeutet, zu programmieren.

Prozessorientierung im Unterricht

Damit Lernende eine realistische Vorstellung vom Prozess des Programmierens entwickeln, ist es wichtig, diesen Prozess auch zu zeigen und nicht nur seine Endprodukte in Form von fertigen Programmen. Hierfür schlagen Bennedsen und Caspersen vor, Aufnahmen von Programmierprozessen zu nutzen [3]. Eine andere Möglichkeit, den Prozess zu zeigen, ist Live-Coding [11], was ad hoc Fragen und Reaktionen von den Lernenden ermöglicht.

Lernende sollten aber nicht nur durch die Umsetzung in Code Schritt für Schritt geführt werden, sondern auch durch die vorbereitenden und die Überarbeitungsprozesse. Es sollte also auch gezeigt werden, wie ein Problem überhaupt erst analysiert und mit welchen Schritten es anschließend gelöst wird. Probleme zunächst natürlichsprachlich zu formulieren und in kleinere Probleme aufzubrechen, sind beispielsweise bekannte Lösungsstrategien, welche bewusst aufgezeigt werden sollten. Dasselbe gilt für Strategien zum Debugging, die auch als solche benannt werden sollten, wie zum Beispiel das Überprüfen von Ausgaben auf der Konsole. Auch Refactoring sollte bei Live-Demonstrationen bewusst praktiziert werden, um zu zeigen, dass es normal ist, Code zu überarbeiten und Entscheidungen zu revidieren.

Bei der Vermittlung von Arbeits- und Lösungsstrategien sollte berücksichtigt werden, dass es beim Programmieren wie beim Schreiben vermutlich verschiedene Strategietypen gibt, wofür die Tatsache spricht, dass wir in Kapitel 7.3 verschiedene Schreibstrategien bei den erfahrenen Programmierenden wiederfinden konnten. Wenn Lernende sich zu einer bestimmten Schreibstrategie hingezogen fühlen, kann das ein Grund dafür sein, dass sie einzelne Strategien und Techniken unterschiedlich gut annehmen [21, S.31]. Es sollte daher auch vermittelt werden, dass es nicht nur *eine* mögliche Strategie beim Lösen von Programmierproblemen gibt. Es sollten sowohl planende, als auch eher 'drauflosschreibende' Ansätze gezeigt werden.

Die erhobenen Daten in dieser Studie lassen noch keine Rückschlüsse darauf zu, welche Strategietypen beim Programmieren verbreitet sind oder mit welcher Häufigkeit unterschiedliche Strategietypen auftreten, weswegen hier weitere Studien vonnöten wären. Auch interessant wäre die Frage, ob Menschen bei unterschiedlichen Prozessen wie Schreiben und Programmieren zu demselben Strategietypen tendieren und wir daher anhand des Schreibtypen einer Person ihren Programmiertypen vorhersagen könnten.

Prozessorientierte Prüfungsformen

Eine Idee, die aus der Schreibdidaktik stammt und auch für die Didaktik des Programmierens interessant sein könnte, ist das Portfolio als Prüfungsform. Bei einer Klausur oder bei einer Projektarbeit wird oft nur das (möglichst) fertige Produkt

betrachtet, was im Fall von Programmierkursen oft ein funktionierendes Programm ist. Ein Portfolio dagegen ist eine Art Mappe, in der allgemein Arbeitsergebnisse gesammelt werden. "Diese können aus verschiedenen Texten oder Visualisierungen bestehen oder, im Fall von elektronischen Portfolios, auch aus Audio-, Video- und sonstigen Dateien." [20, S.97]

Laut Girgensohn und Sennewald [20, S.97-100] werden zwei wesentliche Formen des Portfolios unterschieden: Das Prozess-Portfolio und das Produkt-Portfolio. Bei ersterem werden neben fertigen Texten auch Entwürfe, Rohfassungen, Literaturlisten, Exzerpte et cetera gesammelt – alles, was den eigenen Entwicklungs- und Lernprozess dokumentiert. Beim Produkt-Portfolio dagegen werden Texte gesammelt, die die Lernenden als besonders gelungen empfinden. Bei beiden Variationen sind außerdem Reflexionen wichtig: Es soll darüber geschrieben werden, inwiefern die präsentierten Objekte im Portfolio den eigenen Erkenntnisprozess unterstützt haben oder weshalb die gewählten Texte als besonders gut empfunden werden.

Auf das Programmieren bezogen könnten Lernende beispielsweise ein GitHub Repository anlegen, in dem neben fertigen Programmen auch Zwischenschritte hinterlegt werden: Zum Beispiel Entwürfe und Pläne, Analysen von Beispielprogrammen, Unit Tests, Dokumentationen oder auch verschiedene Versionen eines Programms vor und nach einem Refactoring. Ein reflektierender Text kann die Lehrperson durch das Portfolio führen. Bewertet wird nicht die Qualität einzelner Produkte, insbesondere, wenn es sich dabei um Zwischenschritte handelt, sondern der Gesamteindruck des Portfolios und die Tiefe der Reflexion.

Solche prozessorientierten Prüfungsformen könnten dafür sorgen, dass nicht nur die Lehrpersonen, sondern auch die Lernenden den verschiedenen Prozessen des Programmierens vermehrt Aufmerksamkeit schenken und sich nicht nur auf das Endprodukt konzentrieren.

9.2 Blockaden diagnostizieren und auflösen

Falls Lernende in ihrer Entwicklung als Programmierende 'steckenbleiben' und Probleme haben, funktionierende Programme zu entwickeln, welche sich nicht nur auf mangelnde Beherrschung von formalen Kriterien wie der Syntax zurückführen lassen, sollte hinterfragt werden, ob es sich dabei um eine Blockade handeln könnte. Der einfachste Weg, um Blockaden zu diagnostizieren, ist das persönliche Beratungsgespräch. In diesem soll identifiziert werden, welche Vorannahmen oder welche falsch angewandten Regeln der Blockade zugrunde liegen. [21, 26, 28, 44]

Alternativ können auch hier Fragebögen hinzugezogen werden. So hat Keseling basierend auf seiner Kategorisierung von Schreibblockaden beispielsweise einen Fragebogen zur Selbstdiagnose entwickelt [29]. Diese können aber natürlich nicht dieselbe Tiefe erreichen wie das persönliche Gespräch.

Peer-Tutoring zur Unterstützung der Lehrpersonen

Da für persönliche Gespräche, insbesondere bei großen Kursen, nur bedingt Platz ist, ist ein weiteres Konzept, das in der Schreibdidaktik an Hochschulen zum Einsatz kommt, das Peer-Tutoring. Peer-Tutoren sind Studierende, welche speziell dafür fortgebildet werden, andere Studierende bei ihren Schreibproblemen zu beraten und zu unterstützen. Sie dürfen dabei aber nicht mit einem Lektorat verwechselt werden, denn ihr Ziel ist es nicht, Texte zu bewerten oder zu korrigieren, sondern sie leisten Hilfe zur Selbsthilfe; unterstützen die Lernenden also dabei, ihre Probleme selbst zu erkennen und Lösungen zu entwickeln. Studierende empfinden diese Art der Hilfe oft als zugänglicher, da die Beziehung zu Peer-Tutoren nicht von einem hierarchischen Gefälle geprägt ist. [20, S.90-93] Die Peer-Tutoren profitieren dabei selbst von der Tätigkeit, indem sie dabei ihre eigenen Schreibkomptenzen, aber auch analytische und kommunikative Kompetenzen weiterentwickeln [27].

Brown [11] empfiehlt bereits, Peer-Instruction und Pair-Programming im Klassenzimmer zu nutzen und fortgeschrittenere Lernende mit weniger fortgeschrittenen Lernenden zusammenzubringen, damit diese sich in der Gruppenarbeit gegenseitig helfen. Die fortgeschrittenen Lernenden festigen dabei ihr Wissen und üben sich im Codeverstehen, während die weniger fortgeschrittenen von jenem Wissen profitieren können. Diese Idee ließe sich zu Peer-Tutoring Programmen erweitern, bei denen die Peer-Tutoren gezielt geschult werden, um Lernende zu unterstützen und Lehrpersonen zu entlasten. Zusätzlich können den Tutoren ihre Arbeit und die dadurch erworbenen Schlüsselkompetenzen dann auch bescheinigt werden, was ein zusätzlicher Anreiz sein kann.

Blockaden systematisch auflösen

Keseling beschreibt das Vorgehen bei der Beratung im Schreiblabor Marburg, wie mit Hilfe von gezielten Schreibaufgaben Blockaden diagnostiziert und aufgelöst wurden [28, 29-36]. So erhielten Ratsuchende etwa den Auftrag, ein Exposé zu ihrer Arbeit zu schreiben. Fingen sie damit direkt an, wurde bei ihnen zum Beispiel eher eine Frühstarter-Blockade vermutet. Waren sich die Beratenden sicher, dass sie es mit einer Blockade zu tun hatten, luden sie die Studierenden zu einer Schreibgruppe ein. In dieser arbeiteten die Studierenden an ihren Schreibaufträgen, wobei die Beratenden ihnen wenn nötig ausgewählte Aufgabentypen vorgaben, die ihnen helfen sollten, ihre aktuellen Schwierigkeiten zu überwinden. Die Entwicklung geeigneter Aufgabentypen nahm dabei besonders viel Zeit in Anspruch.

Voraussetzung für die Entwicklung gezielter Beratungsstrategien, um Blockaden beim Programmieren zu diagnostizieren und aufzulösen, ist also zunächst die Bestimmung möglicher Blockadetypen und die Entwicklung von Aufgaben, die diese gezielt angehen. Die Experten-Interviews in dieser Arbeit können uns nur erste Hinweise darauf liefern, wie diese Blockaden bei unerfahrenen Programmierenden aussehen könnten, weshalb weitere Studien zu diesem Thema unerlässlich sind.

9.3 Blockaden jenseits der Lehre

Außerhalb der Lehre bestärken die Ergebnisse dieser Arbeit die Notwendigkeit für angemessene Werkzeuge zur Externalisierung interner Repräsentationen von Software, welche auch schon Petre mit ihren Studien festgestellt hat [38]. Wie wir in Kapitel 8.2 gezeigt haben, können viele der Probleme, mit denen auch erfahrene Programmierende noch regelmäßig konfrontiert werden, auf Probleme mit der internen Repräsentation der Software zurückgeführt werden. Es wäre daher empfehlenswert, weiter Ressourcen in die Entwicklung von Hilfsmitteln zu investieren, welche die Handhabung dieser internen Repräsentationen unterstützen. Die bestehenden Programmierprozesse könnten dadurch weiter verbessert werden.

Weitere Möglichkeiten zur Externalisierung und Visualisierung von Software könnten auch die Arbeit im Team und die Kommunikation über ein Softwareprodukt, wenn es etwa Fachfremden Personen vorgestellt werden soll, verbessern. Es wäre nämlich möglich, dass sich die Ideen von verteilter Kognition innerhalb von Entwicklungsteams, wie in Kapitel 3.2 angesprochen, und kognitiver Blockaden zusammenbringen lassen. Wenn innerhalb von Entwicklungsteams eine gemeinsame mentale Repräsentation der Aufgabe konstruiert wird, könnte das Team auch von gemeinsamen Blockaden betroffen werden. Das gilt ebenso für noch größere Systeme. Einige der Fälle der Interviewten, in denen Projekte explizit an Entscheidungen von Vorgesetzten gescheitert sind, welche die flexible Reaktion auf Probleme verhinderten, bestärken diese Idee.

10 Einschränkung der Validität

Die Konstruktvalidität ist dann gegeben, wenn mit der Methodik tatsächlich das gemessen wird, was gemessen werden soll. In dieser Arbeit haben wir zur Datenerhebung die qualitative Methode eines halboffenen Experten-Interviews genutzt, um Daten über den Ablauf von gesunden Programmierprozessen sowie über mögliche Probleme in den Prozessen zu erhalten. Um sicherzustellen, dass die Antworten wirklich ein zuverlässiges Bild von gesunden Programmierprozessen zeichnen, haben wir ausschließlich Teilnehmende gewählt, welche über mindestens ein Jahr professionelle Programmiererfahrung verfügen. Wir haben die Programmiererfahrung zusätzlich durch die Einbindung des Fragebogens zur Erfassung der Programmiererfahrung verifiziert. Die Frage-Items wurden in direkter Anlehnung an Keselings Interviews [28, S.146f] konstruiert und wurden so formuliert, dass sie die Befragten zu ausführlichen Berichten animieren sollten. Durch die halboffene Struktur des Interviews waren im Zweifelsfall auch Nachfragen möglich.

Die interne Validität kann durch Störfaktoren wie falsche Angaben der Interviewten oder eine Beeinflussung durch die Versuchsleitung gefährdet sein. Interviewte könnten zum Beispiel gehemmt sein, über Probleme zu sprechen, wenn sie denken, dass dies negativ auf sie zurückfallen könnte. Um diesen Problemen entgegenzuwirken, wurde dem Datenschutz eine besonders hohe Bedeutung beigemessen. Die Konferenzräume, in denen die Interviews stattgefunden haben, wurden durch zufällige Passwörter geschützt, welche für jeden Teilnehmenden neu generiert wurden und nur diesem und der Versuchsleitung bekannt waren. Die Transkripte wurden sorgfältig anonymisiert und Namensnennungen von Personen oder Institutionen wurden entfernt, sodass keine Rückschlüsse auf die Identität der Teilnehmenden oder ihrer Arbeitgeber möglich sind.

Ferner könnten Interviewte dazu neigen, ihre Programmierprozesse zu beschönigen. Aus diesem Grund enthielt der Fragenblock zum Thema Probleme mehrere Fragen, welche alle indirekt auf Probleme abzielen. Sie fragen zunächst nach Beispielen von Programmierprozessen, welche entweder abgebrochen oder verzögert wurden. Dadurch sollten die Interviewten zunächst von Erlebnissen berichten und nicht gezwungen sein, womöglich eigene Schwächen in Worte zu fassen. Erst danach wurde zusätzlich konkreter nach Problemen gefragt.

Schließlich wurde darauf geachtet, dass die Interviewten im Vorfeld nichts vom eigentlichen Thema der Arbeit erfuhren, damit sie dadurch nicht beeinflusst werden konnten. Ihnen wurde lediglich mitgeteilt, dass sie im Interview zu ihren Program-

mierprozessen befragt würden, nicht wozu genau diese Daten erhoben wurden. Erst nachdem alle Frage-Items zufriedenstellend beantwortet waren, wurden die Interviewten über das Thema der Arbeit aufgeklärt.

Die externe Validität schließlich soll messen, inwiefern sich die Ergebnisse auf andere Kontexte übertragen lassen. Dabei ist festzustellen, dass wir bewusst möglichst erfahrene Programmierende interviewt haben. Dadurch ist fraglich, ob die Ergebnisse sich auf wenig erfahrene Programmierende übertragen lassen. Außerdem haben trotz der Betonung der Erfahrung vier von sieben Teilnehmenden weniger als zehn Jahre Programmiererfahrung insgesamt. Es wird davon ausgegangen, dass mindestens zehn Jahre Erfahrung notwendig sind, damit Programmierende als Experten beziehungsweise Expertinnen gelten [55]. Entsprechend ist auch die Übertragbarkeit auf erfahrenere Programmierende eine offene Frage.

Schließlich konnte unsere Befragung nur eine Handvoll Domänen der Softwareentwicklung abdecken. So gab es zum Beispiel nur im Rahmen von privaten Projekten Aussagen über Programmierprozesse, bei denen nicht mit einer bestehenden Codebasis gearbeitet wurde, nicht aber in professionellen Kontexten. Künftige Arbeiten könnten diese Lücken schließen, indem sie mehr Kriterien als nur die professionelle Programmiererfahrung in die Auswahl der Teilnehmenden einfließen lassen.

11 Fazit

Ziel dieser Arbeit war es, herauszufinden, ob Schreibblockaden auch Programmierprozesse betreffen können. Die Analyse der Interviews von sieben erfahrenen Programmierenden hat ergeben, dass diese beim Programmieren nicht nur die Prozesse des Software Engineerings, sondern auch die Prozesse des wissenschaftlichen Schreibens durchlaufen. Zudem ließen sich in dem beschriebenen Vorgehen bei verschiedenen Programmierprojekten auch die wichtigsten Schreibstrategien des wissenschaftlichen Schreibens finden. Das legt die Vermutung nahe, dass es in den Programmierprozessen auch zu Schreibblockaden kommen kann. Die weitere Analyse der Probleme, von denen die Interviewten berichteten, hat dies bestätigt: Einige der Probleme lassen sich im Bereich der kognitiven Prozesse beim Programmieren ansiedeln und können mit unvollständigen oder fehlerhaften internen Repräsentationen erklärt werden. Diesen Ursprung teilen sie sich mit einer Kategorie von Schreibblockaden, den Blockaden bei der Konzeptbildung. Die zweite Kategorie, die adressatenbezogenen Blockaden, ließen sich in den Problemen, von denen die Interviewten berichteten, nicht wiederfinden.

Obwohl wir in Kapitel 9 einige Ideen anstoßen, wie wir die Didaktik des Programmierens aufgrund dieser Ergebnisse verbessern können, können die Ergebnisse an sich nur der Anfang einer Erforschung von Blockaden beim Programmieren sein. Aufbauende Arbeiten sollten sich zunächst dringend mit der Frage beschäftigen, ob sich von den Problemen erfahrener Programmierender überhaupt auf die Probleme von Lernenden schließen lässt. Auch die Definition von Schreibblockaden, welche in dieser Arbeit genutzt wurde, wirft in dieser Hinsicht einige Fragen auf: Ab wann sind Programmierende überhaupt kompetent genug, dass sie als 'blockiert' gelten können? Die Definitionen des Kompetenzbegriffs in Bezug aufs Schreiben in den genutzten Quellen sind wenig eindeutig und lassen keine Schlüsse aufs Programmieren zu.

Eine weiterführende Studie, bei der die Interviews mit Programmieranfängern wiederholt werden, befindet sich bereits in Arbeit. Weitere Studien, bei denen die Ergebnisse mit Hilfe von Fragebögen quantitiv untersucht werden sollen, sind geplant. Der Aufbau einer Datengrundlage wie derjenigen aus Keselings Beratungsprotokollen wäre wünschenswert, die Realisierbarkeit dieser Idee muss aber noch überprüft werden.

Um das Wissen aus der prozessorientierten Schreibdidaktik also gewinnbringend nutzen zu können, wird eine enge Zusammenarbeit mit den Lernenden notwendig

sein. Eine Aufgabe, die nicht unbedingt leicht umzusetzen sein wird. Doch die Programmierdidaktik hat der Didaktik wissenschaftlichen Schreibens gegenüber einen enormen Vorteil: Es ist unwahrscheinlich, dass ihr Subjekt für eine Nebensächlichkeit gehalten wird, welche die Lernenden schon von selbst aufschnappen. Entsprechend sehen wir gute Chancen, dass neue Methoden in dem Fach ausprobiert und Fuß fassen werden, sollten sie nun direkt aus der Schreibdidaktik übernommen werden können oder nicht.

Literaturverzeichnis

[1] de Beaugrande, R.: Text Production: Toward a Science of Composition. Norwood (1984)

[2] Begum, M.: Cognition and Distributed Cognition in Software Engineering Research [WIP]. In: Psychology of Programming Interest Group (PPIG) (2021)

[3] Bennedsen, J., Caspersen, M.E.: Revealing the Programming Process. In: Proceedings of the 36th SIGCSE technical symposium on Computer science education. pp. 186–190. ACM (2005)

[4] Bereiter, C., Scardamalia, M.: The Psychology of Written Composition. Hillsdale (NJ) (1987)

[5] Bereiter, C., Scardamalia, M.: Knowledge-telling und knowledge-transforming. Schreiben. Grundlagentexte zur Theorie, Didaktik und Beratung pp. 87–93 (2014)

[6] Bloom, B.S.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Longman (1956)

[7] Bortz, J., Döring, N.: Forschungsmethoden und Evaluation für Human-und Sozialwissenschaftler: Limitierte Sonderausgabe. Springer-Verlag (2006)

[8] Bräuer, G.: Warum schreiben? Schreiben in den USA: Aspekte, Verbindungen, Tendenzen. Peter Lang (1996)

[9] Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology 3(2), 77–101 (2006), https://doi.org/10.1191/1478088706qp063oa

[10] Brinkschulte, M., Kreitz, D.: Qualitative Methoden in der Schreibforschung. wbv (2017)

[11] Brown, N.C., Wilson, G.: Ten quick tips for teaching programming. PLoS Computational Biology 14(4), e1006023 (2018)

[12] Caspersen, M.E., Kolling, M.: STREAM: A First Programming Process. ACM Transactions on Computing Education (TOCE) 9(1), 1–29 (2009)

[13] Darnstaedt, D.A., Ahrens, A., Richter-Trummer, V., Todtermuschke, M., Bocklisch, F.: Vorgehen zur Beschreibung von menschlichem Expertenwissen und

kognitiven Prozessen beim Teach-in von Industrierobotern. Zeitschrift für Arbeitswissenschaft 76(1), 34–49 (2022)

[14] De Raadt, M., Watson, R., Toleman, M.: Teaching and Assessing Programming Strategies Explicitly. In: Proceedings of the 11th Australasian Computing Education Conference (ACE). vol. 95, pp. 45–54. Australian Computer Society Inc. (2009)

[15] Eco, U.: On Literature. Secker and Warburg (2005)

[16] Egri-Nagy, A.: |{Math, Philosophy, Programming, Writing}| = 1. arXiv:1803.05998 [cs.CY] (2018), https://doi.org/10.48550/arXiv.1803.05998

[17] Flower, L., Hayes, J.R.: Schreiben als kognitiver Prozess. Eine Theorie. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 35–56. Verlag Barbara Budrich (2014)

[18] Forsgren, N., Storey, M.A., Maddila, C., Zimmermann, T., Houck, B., Butler, J.: The SPACE of Developer Productivity: There's More to It than You Think. Queue 19(1), 20–48 (2021), https://doi.org/10.1145/3454122.3454124

[19] Gantenbein, R.E.: Programming as Process: A "Novel" Approach to Teaching Programming. ACM SIGCSE Bulletin 21(1), 22–26 (1989), https://doi.org/10.1145/65294.65297

[20] Girgensohn, K., Sennewald, N.: Schreiben lehren, Schreiben lernen: Eine Einführung. WBG (Wissenschaftliche Buchgesellschaft) (2012)

[21] Grieshammer, E., Liebetanz, F., Peters, N., Lohmann, B.: Zukunftsmodell Schreibberatung: Eine Anleitung zur Begleitung von Schreibenden im Studium. Schneider Verlag Hohengehren (2019)

[22] Hassenfeld, Z.R., Bers, M.U.: Debugging the Writing Process: Lessons From a Comparison of Students' Coding and Writing Practices. The Reading Teacher 73(6), 735–746 (2020), https://doi.org/10.1002/trtr.1885

[23] Hayes, J.R.: Kognition und affekt beim schreiben. ein neues konzept. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 57–86. Verlag Barbara Budrich (2014)

[24] Hermans, F., Aldewereld, M.: Programming is Writing is Programming. In: Companion to the first International Conference on the Art, Science and Engineering of Programming. pp. 1–8. ACM (2017), https://doi.org/10.1145/3079368.3079413

[25] Hilton, A.D., Lipp, G.M., Rodger, S.H.: Translation from Problem to Code in Seven Steps. In: Proceedings of the ACM Conference on Global Computing Education. pp. 78–84. ACM (2019), https://doi.org/10.1145/3300115.3309508

[26] Hjortshoj, K.: Schreibblockaden verstehen. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 213–233. Verlag Barbara Budrich (2014)

[27] Hughes, B., Gillespie, P., Kail, H.: Was sie mitnehmen. Das 'Peer Writing Tutor Alumni Project'. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 213–233. Verlag Barbara Budrich (2014)

[28] Keseling, G.: Die Einsamkeit des Schreibers: Wie Schreibblockaden entstehen und erfolgreich bearbeitet werden können. VS Verlag für Sozialwissenschaften (2004)

[29] Keseling, G.: Schreibblockaden überwinden. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 235–253. Verlag Barbara Budrich (2014)

[30] Krue, O., Chitez, M.: Schreibkompetenz im Studium. Komponenten, Modelle und Assessment. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 107–126. Verlag Barbara Budrich (2014)

[31] LaToza, T.D., Arab, M., Loksa, D., Ko, A.J.: Explicit programming strategies. Empirical Software Engineering 25(4), 2416–2449 (2020)

[32] Lavallée, M., Robillard, P.N., Paul, S.: Distributed Cognition in Software Engineering. In: eKNOW 2013: The Fifth International Conference on Information, Process, and Knowledge Management. pp. 57–62. IARIA (2013)

[33] Lewis, C.M., Shah, N., Falkner, K.: Equity and Diversity. In: Fincher, S.A., Robins, A.V. (eds.) The Cambridge Handbook of Computing Education Research, pp. 479–508. Cambridge University Press (2019)

[34] Martin, G.R.: A Winter Garden (Jul 2022), `https://georgerrmartin.com/notablog/2022/07/08/a-winter-garden/`, [Abgerufen am 26. Dezember 2022]

[35] Meyer, B., Gall, H., Harman, M., Succi, G.: Empirical Answers to Fundamental Software Engineering Problems (Panel). In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 14–18. ACM (2013), `http://doi.org/10.1145/2491411.2505430`

[36] Minelli, R., Mocci, A., Lanza, M.: I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In: 2015 IEEE 23rd International Conference on Program Comprehension. pp. 25–35. IEEE (2015), `https://doi.org/10.1109/ICPC.2015.12`

[37] Ortner, H.: Schreiben und Denken. Max Niemeyer Verlag (2000)

[38] Petre, M.: Mental imagery and software visualization in high-performance software development teams. Journal of Visual Languages & Computing 21(3), 171–183 (2010), https://doi.org/10.1016/j.jvlc.2009.11.001

[39] Petre, M., Blackwell, A.F.: A Glimpse of Expert Programmers' Mental Imagery. In: Papers presented at the seventh workshop on Empirical studies of programmers. pp. 109–123. ACM (1997), https://doi.org/10.1145/266399.266409

[40] Renumol, V., Janakiram, D., Jayaprakash, S.: Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming. ACM Transactions on Computing Education (TOCE) 10(3), 1–21 (2010), https://doi.org/10.1145/1821996.1821998

[41] Robins, A.V.: Novice Programmers and Introductory Programming. In: Fincher, S.A., Robins, A.V. (eds.) The Cambridge Handbook of Computing Education Research, pp. 327–376. Cambridge University Press (2019)

[42] Robins, A.V., Margulieux, L.E., Morrison, B.B.: Cognitive Sciences for Computing Education. Learning Sciences Faculty Publications 22 (2019), https://scholarworks.gsu.edu/ltd_facpub/22

[43] Rose, M.: Writer's Block: The Cognitive Dimension. Carbondale (Ill.) (1984)

[44] Rose, M.: Schreibblockaden. Eine kognitive Perspektive. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 193–211. Verlag Barbara Budrich (2014)

[45] Ruhmann, G., Kruse, O.: Prozessorientierte Schreibdidaktik: Grundlagen und Arbeitsformen. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 15–34. Verlag Barbara Budrich (2014)

[46] Scott, A.: Re-centering writing center studies: What US-based scholars can learn from their colleagues in Germany, Switzerland, and Austria. Zeitschrift Schreiben 28(2016), 1–10 (2016)

[47] Sennewald, N.: Schreibstrategien. Ein Überblick. In: Dreyfürst, S., Sennewald, N. (eds.) Schreiben: Grundlagentexte zur Theorie, Didaktik und Beratung, pp. 169–193. Verlag Barbara Budrich (2014)

[48] Siegmund, J., Kästner, C., Liebig, J., Apel, S., Hanenberg, S.: Measuring and Modeling Programming Experience. Empirical Software Engineering 19(5), 1299–1334 (2014)

[49] Sommerville, I.: Software Engineering. Pearson (2018), [10. Auflage]

[50] Vedral, J.: Forschung und Didaktik wissenschaftlichen Schreibens. Fachzeitschrift Psychologie in Österreich 8(5), 484–493 (2012)

[51] Vee, A.: Coding literacy: How computer programming is changing writing. MIT Press (2017)

[52] Videla, A.: Lector in Codigo or The Role of the Reader. In: Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming. pp. 180–186. ACM (2018), https://doi.org/10.1145/3191697.3214326

[53] Wang, Y., Wang, Y., Patel, S., Patel, D.: A layered reference model of the brain (LRMB). IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 36(2), 124–133 (2006)

[54] Wing, J.M.: Computational Thinking. Communications of the ACM 49(3), 33–35 (2006)

[55] Winslow, L.E.: Programming Pedagogy—a Psychological Overview. SIGCSE Bull. 28(3), 17–22 (1996), https://doi.org/10.1145/234867.234872

[56] Xie, B., Loksa, D., Nelson, G.L., Davidson, M.J., Dong, D., Kwik, H., Tan, A.H., Hwa, L., Li, M., Ko, A.J.: A theory of instruction for introductory programming skills. Computer Science Education 29(2-3), 205–253 (2019), https://doi.org/10.1080/08993408.2019.1565235

Anhang

https://github.com/schanlin/Master-Thesis