

**Wen kann funktionale Programmierung dabei
unterstützen, Entwicklungskompetenzen zu
erwerben?**

Bachelorarbeit

Autor: Anouk Martinez Wieczorek

(Matrikelnummer: 11154860)

Betreuung: Prof. Dr. Stefan Bente

Zweitbetreuung: Prof. Dr. Hoai Viet Nguyen

TH Köln, Campus Gummersbach

Steinmüllerallee 1

51643 Gummersbach

3. April 2025

Eidesstattliche Versicherung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

A handwritten signature in black ink, appearing to read 'Martinez W.' with a stylized flourish at the end.

Anouk Martinez Wiczorek

Gummersbach, den 03.04.2025

Inhaltsverzeichnis

1	Einleitung	1
2	Forschungsgrundlage	2
2.1	Aktuelle Lerninhalte an Universitäten und Hochschulen	2
2.2	Computational Thinking	4
2.3	Lerntypen	8
3	Forschungsteil	10
3.1	Lerntypen im Zusammenhang mit Computational Thinking	10
3.2	Eignung der Lerntypen für die Programmierparadigmen	12
3.3	Ausprägung der Computational Thinking Aspekte in den Programmierparadigmen	14
3.4	Zusammenfassung	17
4	Praktischer Selbstversuch	20
4.1	Lerntypenanalyse	20
4.2	Problembeschreibung	22
4.3	Zusammenhang mit Lerntypen	24
5	Fazit	26
5.1	Schlussfolgerungen aus Theorie und Praxis	26
5.2	Reflexion	27
5.3	Ausblick und Weiterführende Forschung	29
6	Anhang	33
6.1	Vorgehen bei der Untersuchung der Curricula	33
6.2	Ähnliche Forschungsergebnisse zum Forschungsteil	35
6.3	Auswahl des Problems für den Selbstversuch	36
6.4	Vorgehen im Selbstversuch	37
	Literaturverzeichnis	39

Abbildungsverzeichnis

1	Zahl der Programmiersprachen in Einführungskursen der Informatik	3
2	Zahl der Paradigmen in Einführungskursen der Informatik	4
3	Die Haupt- und Unterkategorien Computational Thinking aus verschiedenen Forschungen, grafisch dargestellt	7
4	Vorteile der Lerntypen in den Aspekten des Computational Thinking Zusammenfassung	12
5	Vorteile der Lerntypen in den betrachteten Paradigmen Zusammenfassung	14
6	Commits und Bugfixing Commits in verschiedenen Programmiersprachen (Berger, Hollenbeck, Maj, Vitek & Vitek, 2019)	16
7	Ausprägungen der CT Aspekte in den jeweiligen Programmierparadigmen Zusammenfassung	17
8	Ergebnisse des ILS Fragebogens	21
9	Ein Modellset der Türme von Hanoi mit 8 Holzplatten (Commons, 2024)	22
10	Verwendung von GHCi in der Windows PowerShell	38

Abkürzungsverzeichnis

CT Computational Thinking

FP Funktionale Programmierung

OO Objektorientierung

FSLSM Felder Silverman Lerntypen Modell

VPN Virtual Private Network

1 Einleitung

Obwohl sich seit Jahren bemüht wird, den Einstieg in die Programmierung einfacher zu gestalten, sind die Abbruchquoten in der Informatik in Deutschland im Verhältnis zu Anzahl der Studierenden¹ weiterhin hoch ((Destatis), 2024). Dies lässt sich besonders im Vergleich zu anderen Studienfeldern erkennen (Heublein, Hutzsch & Schmelzer, 2022). Es entstehen immer wieder ähnliche Probleme und Frustrationen, sowohl bei den Studierenden als auch beim Lehrpersonal (McDonald, 2018).

Hierbei ergibt sich die Frage, ob das Lernen von Programmieren individuelle Unterschiede je nach Person hat, die diese Problematiken besonders begünstigen. In dieser Arbeit soll untersucht werden, welche möglichen Stärken und Schwächen es gibt, die sich deutlicher in verschiedenen Personen auszeichnen, und mit welchen Programmierparadigmen diese jeweiligen Lerntypen dabei unterstützt werden können, Programmierung leichter zu erlernen. Speziell betrachtet hierbei wird das funktionale Paradigma. Es wurde sich für funktionale Programmierung entschieden, da das Paradigma ein nicht sehr weit verbreiteter Ansatz in den meisten Informatik-Einführungskursen ist (siehe Untersuchung Curricula). Es stellt sich hierbei die Frage, ob die Schwierigkeiten der Studierenden beim Lernen von Programmierung nicht durch herkömmliche Paradigmen verstärkt werden, und ob ein unkonventioneller Ansatz hierbei einen Unterschied machen kann.

Folgende Forschungsfragen werden betrachtet:

1. Fällt es jedem gleich leicht, Programmieren zu lernen, oder gibt es Hänge zu einem bestimmten Paradigma?
2. Welche Vor- und Nachteile haben gängige Paradigmen für die jeweiligen Lerntypen?
3. **Fokus:** Welchen Lerntypen würde funktionale Programmierung beim Lernen von Entwicklung helfen, und welchen Typen würde der Ansatz eher schaden?

¹In der Arbeit werden generell geschlechtsneutrale Begriffe verwendet, insofern sich diese nicht auf eine spezielle Person beziehen. Die verwendeten Personenbezeichnungen beziehen sich auf alle Geschlechter, und sollten allgemein zu verstehen sein.

2 Forschungsgrundlage

Um zu untersuchen, wem funktionale Programmierung Grundkenntnisse effektiver beibringen könnte als häufiger verwendete Paradigmen, muss zunächst die theoretische Grundlage gelegt werden. Zunächst wird analysiert, ob objektorientierte und prozedurale Programmierparadigmen tatsächlich häufiger in Einführungskursen der Informatik verwendet werden als funktionale. Dies soll eine Grundlage für die Frage schaffen, ob funktionale Programmierung tatsächlich als neuer Ansatz zum Lernen für neue Studierende untersucht werden kann. Anschließend wird untersucht, wie das Erlernen von Programmierkenntnissen definiert werden kann, und welche Kompetenzen relevant sind. Dies umfasst nicht nur das Schreiben von Code, sondern auch das Lernen der nötigen Denkweisen. Diese theoretischen Konzepte lassen sich in der Theorie des Computational Thinking (CT) zusammenfassen. Demnach müssen auch die Zusammenhänge von CT und den Lerntypen betrachtet werden. Dazu wird CT in diesem Abschnitt exakt für die Arbeit definiert und auf vier häufigste Aspekte reduziert. Zudem wird das Lerntypenmodell, das in dieser Arbeit verwendet wird, erläutert. Dieses dient zur Untersuchung der verschiedenen Lernstile der Programmieranfängenden.

2.1 Aktuelle Lerninhalte an Universitäten und Hochschulen

Umfragen unter Entwickelnden der letzten Jahre haben gezeigt, dass objektorientierte Sprachen noch immer zu den am meisten verwendeten zählen, sowohl im Berufsleben als auch unter Lernenden (Stackoverflow, 2024). Dieser Abschnitt untersucht, ob auch an deutschen Universitäten noch vermehrt objektorientierte Paradigmen gelehrt werden. Darauf aufbauend lässt sich bewerten, ob funktionale Programmierung überhaupt als Alternative in Erwägung gezogen werden kann.

2.1.1 Übersicht bekannter Paradigmen

Um die häufig behandelten Paradigmen untersuchen zu können, müssen zunächst die wichtigsten Kategorien definiert werden. Dazu werden die vier verbreitetsten Paradigmen betrachtet (Normak, 2017).

Imperative Programmierung Abfolge von Kommandos, die den Zustand des Programms inkrementell ändern. Beschreibungen von Aktionen in einer geordneten Abfolge, ähnlich wie bei einer Anleitung oder einem Rezept.

Funktionale Programmierung Orientierung an Funktionen aus mathematischer Sicht. Produzierte Werte sind unveränderlich, und das Programm hat

keinen Zustand. Alle Operationen im Programm werden ausschließlich von Funktionen gehandhabt. Die Funktionen sind Objekte erster Klasse und werden wie Daten behandelt.

Logische Programmierung Arbeitet mit Axiomen, Schlussregeln und Abfragen auf Basis gegebener Fakten.

Objektorientierte Programmierung Modellierung von echten Objekten im Code. Daten und Funktionen sind in Objekten gekapselt, die durch Klassen organisiert werden. Klassen sind hierarchisch organisiert, etwa durch Vererbung.

2.1.2 Untersuchung Curricula

Um zu untersuchen, wie verbreitet funktionale Programmierung momentan in Hochschulcurricula ist, wurden 121 Informatik-Studiengänge an verschiedenen Universitäten und Hochschulen in Deutschland untersucht². Die Analyse zeigt, dass die Mehrheit der Einführungskurse in der Informatik objektorientierte Programmierparadigmen behandelt³.

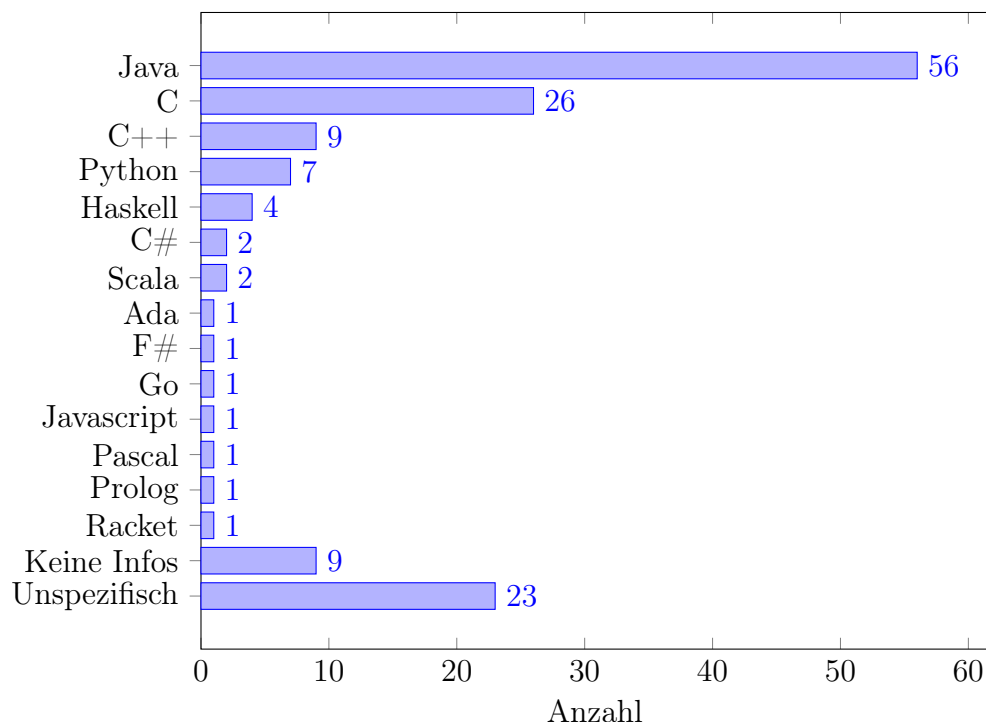


Abbildung 1: Zahl der Programmiersprachen in Einführungskursen der Informatik

²Eine detaillierte Beschreibung des Vorgehens befindet sich im Anhang der Arbeit unter Anhang.

³Der Unterschied der Datensätze „Unspezifisch“ und „Keine Infos“ wird im Abschnitt Unterschied der Datensätze Unspezifisch und Keine Infos näher erläutert.

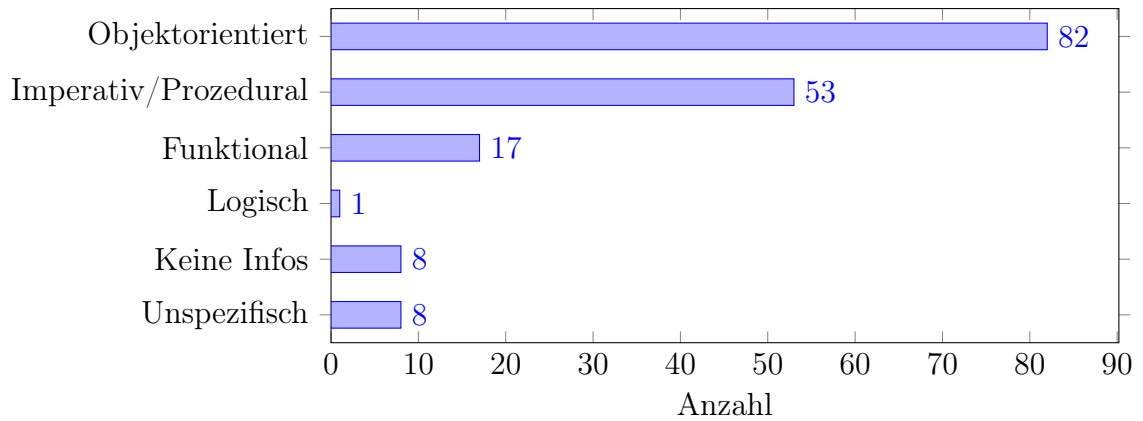


Abbildung 2: Zahl der Paradigmen in Einführungskursen der Informatik

Wenn das funktionale Paradigma in den Modulzielen gelistet war, wurde meist nicht spezifiziert, welche Sprache verwendet wird, um das Konzept näherzubringen, oder welche Ziele mit der Verwendung von funktionaler Programmierung speziell verfolgt werden. In den meisten Fällen wird an Institutionen Java als Einführung in die Programmierung verwendet, und damit größtenteils imperative und objektorientierte Ansätze. Ein funktionaler Ansatz für Programmierneinsteigende ist in Deutschland bislang kaum etabliert.

2.2 Computational Thinking

Um zu definieren, welche Aspekte genau zum Erlernen von Programmierkenntnissen nötig sind, müssen die einzelnen Aspekte von CT betrachtet und definiert werden. Anschließend können auf Basis der Recherche Vor- und Nachteile in der Anwendung und Erlernung von CT der einzelnen Lerntypen untersucht werden.

2.2.1 Definition

Der Begriff CT wurde erstmals 1980 verwendet (Papert, 1980), schlussendlich allerdings erst 2006 wieder weitläufig durch eine Kolumne in den öffentlichen Diskurs gebracht (J. Wing, 2006). Grund der erneuten Popularisierung des Begriffes war unter anderem die Diskussion von CT im Kontext von Schulcurricula und die Wichtigkeit des Konzeptes im Alltag. Da es keine einheitliche Definition von CT gibt, wurden im Kontext der Arbeit verschiedenste Studien betrachtet und verglichen. Es wurde sich zudem an einer Review-Studie aus dem Jahre 2017 orientiert, die bisherige Forschungsergebnisse abwägt und vergleicht (Shute, Chen & Asbell-Clarke, 2017).

In den frühesten Definitionen von CT wird das Konzept häufig als Grundlage für alle unsere Prozesse im Alltag beschrieben. CT ist demnach nicht nur eine

Fähigkeit, die nützlich für Entwicklerinnen ist, sondern begegnet uns überall im Leben (Khine, 2018). Besonders frühere Berichte stellen hervor, dass Informatik nicht nur bedeutet, dass man programmieren kann, sondern dass man Kompetenzen erwirbt, die sich auch in allen anderen Bereichen, sogar künstlerischen Feldern, anwenden lassen (J. Wing, 2006). Auch wird der Begriff im Kontrast zu mathematischem Denken gestellt, und besonders welche Unterschiede und Gemeinsamkeiten die beiden Konzepte haben.

2.2.2 Computational Thinking Aspekte

Aufgrund dessen, dass es keine einheitliche Definition von CT gibt, wurden im Rahmen der Arbeit verschiedenste Paper zum Thema analysiert. 2017 wurde eine einheitliche Definition der Aspekte des CT, sowie ein Vergleich aller bisherigen CT-Studien veröffentlicht, an der sich größtenteils in dieser Arbeit orientiert wird (Shute et al., 2017). Obwohl es viele verschiedene Definitionen von CT gibt, kommen diese letztendlich auf ähnliche Ergebnisse. Vier Aspekte des CT werden demnach immer wieder erwähnt (Shute et al., 2017).

Dekomposition Dieser Aspekt beschreibt die Kompetenz, die Zusammenhänge in einem größeren Problem zu verstehen und dieses in kleinere Teile herunterzubrechen. Durch iteratives Lösen dieser kleineren Teilprobleme kann eine größere Aufgabe oder ein komplexes System angegangen werden.

Abstraktion Im ursprünglichen Paper von Wing (J. Wing, 2006) wird Abstraktion als eine Kernbasis für CT genannt. Hierbei fließt ein weiterer Aspekt ein, die Verallgemeinerung. Demnach muss man in der Lage sein, ein komplexes Problem oder System abstrahieren zu können und auf wesentliche Aspekte zu reduzieren. Wing unterscheidet zwischen Abstraktion in jeweiligen Ebenen, Abstraktion als Ganzes und Verbindung zwischen den Ebenen (J. M. Wing, 2008).

Algorithmen Je nach Forscherin wird dieser Aspekt auch als algorithmisches Denken bezeichnet und soll sich so unter anderem von anderen Denkweisen, wie beispielsweise dem mathematischen Denken, abheben (Shute et al., 2017). Der Aspekt beschreibt die schrittweise Entwicklung eines Lösungsweges für ein Problem, sodass dieses allgemein und immer wieder lösbar durch einen Menschen oder einen Computer ist. Weitere Denkweisen, die mit CT verbunden werden können, sind wissenschaftliches sowie logisches Denken (Curzon & Curzon, 2018).

Debugging Dieser Aspekt wird auch teilweise als Evaluation (Curzon & Curzon, 2018) oder systematisches Testen (J. Wing, 2006) bezeichnet. Er beschreibt

die Analyse der Lösung. Debugging kann zudem dabei helfen, die Lösung zu verallgemeinern, damit diese wieder auf zukünftige Probleme angewandt werden kann.

Es wird zudem argumentiert, dass es noch weitere Aspekte gibt, die für CT allgemein relevant sind (Curzon & Curzon, 2018). Diese Aspekte können teilweise unter den bereits genannten einsortiert werden.

Menschliches Denken Neben algorithmischem, logischem und wissenschaftlichem Denken gibt es laut Curzon und McOwan (Curzon & Curzon, 2018) noch den abstrakteren, menschlichen Aspekt. Ein Algorithmus muss sich danach richten, was eine Nutzerin verstehen und realistisch im Alltag anwenden kann.

Heuristik Neben der Modellierung eines Algorithmus muss laut Curzon und McOwan noch abgewägt werden, ob sich die Lösung auch tatsächlich realistisch umsetzen lassen kann. Falls nötig, muss ein Problem weiter abstrahiert werden, um eine geeignete Lösung zu finden. Diese ist dann möglicherweise nicht perfekt, aber löst das Problem schneller und effizienter als eine Alternative, die gar nicht in einem realistischen Zeitraum entwickelt werden kann.

Kreativität Curzon und McOwan argumentieren, dass Kreativität trotz der scheinbar geringen Zusammenhänge zu Computerwissenschaften als wichtiger Aspekt des CT angesehen werden sollte. Demnach ist Kreativität beispielsweise bei dem Entwurf eines Algorithmus relevant, oder auch bei der Abstraktion. Verallgemeinerung und Mustererkennung erfordern laut den Autoren manchmal große Sprünge, um eine Lösung für ein Problem zu finden, das möglichst effizient und zufriedenstellend für die Nutzerinnen in der realen Welt ist. Dies ist besonders wichtig, wenn für ein spezielles Problem eine völlig neue Lösung entwickelt werden muss.

Modellierung Dieser Aspekt kann als Teil von Abstraktion eingeordnet werden. Modellierung eines komplexen Problems oder Systems kann demnach dabei helfen, dieses in einen größeren Kontext einzuordnen und zu verallgemeinern. Hierbei geht es nicht um grafische Modellierung, sondern um die Übertragung realer Probleme in ein virtuelles Modell.

Verallgemeinerung Als Unteraspekt der Mustererkennung und Abstraktion ist die Verallgemeinerung besonders relevant in der Wiederverwendung bereits

gefundenen Lösungen. Hierbei ist es wichtig zu erkennen, wann ein Sachverhalt zu einem bereits existierenden Problem abstrahiert werden kann. Mithilfe der Mustererkennung und Abstraktion kann anschließend entschieden werden, welche Lösungen sich wiederverwenden lassen und welche gegebenenfalls angepasst werden müssen.

Mustererkennung Ähnlich wie die Verallgemeinerung hilft dieser Aspekt dabei, Ähnlichkeiten in Problemen zu identifizieren und gegebenenfalls auf bereits existierende Lösungen zurückzugreifen. Allerdings unterscheiden sich die Kompetenzen darin, dass Mustererkennung eher dabei helfen kann, Probleme auf bekannte Darstellungen übertragen zu können. Beispielsweise fällt bei ähnlichen Problemen auf, dass diese beide durch einen Graphen dargestellt werden können, und so gegebenenfalls dieselben Algorithmen behandelt werden können.

Darstellung Dieser Aspekt kann auch als Unteraspekt der Abstraktion und Verallgemeinerung gesehen werden. Er beschreibt die Kompetenz, eine geeignete Darstellungsform für ein Problem oder Daten zu finden, die unter Umständen eine neue Sichtweise auf ein Problem geben kann. Hierbei wird ebenfalls hervorgehoben, wie wichtig eine selbst erstellte grafische Darstellung eines Problems beim gedanklichen „Durchbruch“ zu einem effizienteren Algorithmus helfen kann.



Abbildung 3: Die Haupt- und Unterkategorien Computational Thinking aus verschiedenen Forschungen, grafisch dargestellt

Die Arbeit konzentriert sich auf die vier CT Aspekte, die im ersten Abschnitt definiert wurden.

2.3 Lerntypen

Um genau zu definieren, welchen Studierenden FP helfen kann, müssen zunächst konkrete Kategorien für die Typen von Lernenden festgelegt werden. Um zu analysieren, welche Studierenden möglicherweise Vorteile beim Erlernen von Programmierung haben könnten, werden diese Kategorien von Lerntypen im Zusammenhang mit CT und den jeweiligen Paradigmen betrachtet.

Ebenso wie im Feld des CT gibt es für Lerntypen kein allgemein akzeptiertes Modell. Allerdings gibt es einige Theorien, die häufiger, besonders im Kontext der Informatik, angewandt werden. Ein verbreitetes Modell ist hierbei das Felder-Silverman Lerntypenmodell (FSLSM) (Felder, 1988). Das FSLSM wurde speziell im Kontext hoher Abbruchraten in Studiengängen der Ingenieurwissenschaften untersucht, wird allerdings auch vermehrt als Grundlage für Studien im Feld der Informatik verwendet (Kumar, 2017). Um die Lerntypen in der Forschungsgrundlage zu definieren, wird sich an diesem Lerntypenmodell orientiert.

2.3.1 Differenzierung im Vergleich zu anderen Modellen

Das FSLSM versteht sich nicht als völlig neues oder umfassendes Modell, und basiert auf bereits existierenden Modellen von etwa Jung (Jung, 1921), Kolb (Kolb, 1984) und Briggs (Myers, McCaulley, Quenk & Hammer, 1998). Zudem sind die Lerntypen nicht so eindeutig bestimmbar, wie es herkömmliche Modelle, zum Beispiel das weit verbreitete VARK Modell von N. Fleming suggerieren. Beispielsweise nutzen alle Menschen sowohl fühlende als auch intuitive Fähigkeiten, bevorzugen allerdings tendenziell eine der beiden Methoden. Das Lernmodell basiert nicht auf festen physischen Merkmalen, sondern auf differenzierteren Präferenzen der Subjekte, und betont, dass schwächere Lerntypendimensionen erlernt werden können.

Das FSLSM beschreibt vier verschiedene Bereiche, in denen sich die Lernstile größtenteils unterscheiden ⁴. Hierbei wird immer jeweils ein Gegensatzpaar beschrieben, in das sich Subjekte einsortieren lassen. Die meisten Personen präferieren demnach einen der beiden Stile jeder Kategorie.

⁴Ursprünglich als 5 Bereiche definiert, wobei die Dimension „Inductive Deductive“ nachträglich aus dem Modell entfernt wurde. Als Grund hierfür wurde angegeben, dass die meisten Schüler und Studierende einen schlussfolgernden Stil zwar präferieren, aber induktive Methoden in den meisten Fällen laut der Ansicht der Autoren objektiv besser für das Verständnis der Lernenden sind (Felder, 1988).

Sensorisch und Intuitiv Dieser Aspekt beschäftigt sich damit, wie Personen am besten die Welt um sich herum wahrnehmen können. Hierbei präferieren Personen gewöhnlich eine der beiden Methoden, auch wenn alle Menschen beide verwenden können. Sensorische Typen präferieren, Daten und Fakten auswendig zu lernen, und sich länger mit den Details eines Problems zu beschäftigen. Sie ziehen es vor, einen klar erkennbaren Bezug auf die echte Welt zu haben und Probleme mit etablierten Methoden zu lösen. Intuitive Typen hingegen können ihre Umgebung besser durch eigene Theorien, Vorstellungen und Vermutungen wahrnehmen und setzen auf Innovation und Unsicherheiten. Sie sind besser darin, Abstraktion anzuwenden, und lernen neue Konzepte normalerweise schneller als sensorisch veranlagte Personen.

Visuell und Verbal Dieser Aspekt beschäftigt sich mit der Art, wie Personen Input verarbeiten und Informationen am besten verstehen können. Visuelle Typen lernen am besten damit, was sie sehen können, etwa Diagramme, Bilder, Filme und Demonstrationen. Verbale Typen ziehen mehr aus Worten, sowohl geschriebenen als auch gesprochenen Erklärungen. Ihnen können besonders Gruppenarbeiten dabei helfen, das Material zu verstehen. Die Mehrheit der Lernenden hat typischerweise eine stärkere Ausprägung des visuellen Lerntypen, was im Konflikt mit den typischen Stilen von Vorlesungen in den Bereichen Informatik und Ingenieurwissenschaften steht. Hierbei wird meist ein frontaler Input gegeben, der im Großteil auf verbalen Vermittlungen basiert.

Aktiv und Reflexiv Dieser Aspekt beschäftigt sich mit der Verarbeitung von Informationen zum Wissen. Aktive Typen sind demnach besser darin, ihr Wissen direkt in der Anwendung zu vertiefen. Sie präferieren Gruppenarbeiten, in denen sie sich aktiv einbringen können, und lernen mit herkömmlichen Vorlesungsstilen nicht so effektiv. Reflexive Typen hingegen benötigen Zeit, um sich eigenständig mit Inhalten auseinanderzusetzen. Sie ziehen es vor, alleine zu arbeiten, und zuerst alle Fakten durchzudenken, bevor sie ein Problem angehen können.

Sequenziell und Global Dieser Aspekt beschäftigt sich mit damit, wie Lernende Zusammenhänge verstehen. Sequenzielle Typen können besser in einer festgelegten logischen Reihenfolge arbeiten, etwa chronologisch, mit einem Buch oder mit den Inhalten einer Vorlesung. Globale Typen hingegen lernen oft sprunghaft und nicht linear, und müssen zunächst eine große Menge an Informationen ansammeln, bevor sich für sie das Gesamtbild erschließt. Da viele herkömmliche Vorlesungsformen auf einer schrittweisen Einführung

der Konzepte basieren, haben globale Typen beim Erlernen neuer Sachinhalte normalerweise einen Nachteil gegenüber sequenziellen Lerntypen.

3 Forschungsteil

Um die Frage zu beantworten, ob es bestimmte Hänge zu einem oder dem anderen Paradigma gibt, muss zuerst beantwortet werden, wie die Paradigmen mit den Lerntypen zusammenhängen. Dazu wird untersucht, welche Lerntypen Vorteile beim Lernen der Computational Thinking (CT) Aspekte haben könnten. Außerdem wird geprüft, welche Empfehlungen Felder und Silverman für die Lerntypen geben und wie diese mit den Eigenschaften der Paradigmen zusammenhängen. Hierbei wird Objektorientierung (OO) mit funktionaler Programmierung (FP) verglichen. OO wurde hier als Vergleichspunkt gewählt, da es das häufigste Paradigma in Einführungskursen der Informatik ist. FP wird betrachtet, da es den Fokuspunkt der Arbeit darstellt. Andere Programmierparadigmen werden zunächst ausgelassen. Abschließend wird untersucht, welche Rolle die CT Aspekte in den unterschiedlichen Paradigmen spielen, um zu bestimmen, ob ein Paradigma besondere Vor- und Nachteile aufweist.

Es werden hierbei vermehrt Annahmen zu den Vorlieben der Lerntypen gemacht, aufgrund der Empfehlungen von Felder und Silvermann in Relation zu den Merkmalen der Paradigmen. Aufgrund der zeitlichen Beschränkung der Arbeit können diese Annahmen nicht empirisch überprüft werden (siehe Ähnliche Forschungsergebnisse zum Forschungsteil).

3.1 Lerntypen im Zusammenhang mit Computational Thinking

Das Erlernen von Programmierkenntnissen umfasst nicht nur das Schreiben von Code, sondern auch den Erwerb von CT Kompetenzen. Nicht jede Lerntypendimension zeigt Vor- oder Nachteile beim Lernen der CT Aspekte, allerdings lassen sich einige klare Zusammenhänge herausstellen. Es wurde zudem eine Studie betrachtet, die den Zusammenhang der von Felder und Silverman definierten Lerndimensionen speziell im Kontext des Erlernens von CT Aspekten untersucht (Chen, Su, Chen, Liao & Yuan, 2023).

3.1.1 Verbal Visuell

Visuelle Typen können einen Vorteil beim Erlernen von Abstraktion haben, da grafische Darstellungen eine sehr geeignete und beliebte Form sind, um das Konzept von Abstraktion zu erklären. Beispielsweise in der Objektorientierung werden hierbei öfters Bilder von Objekten wie Tieren oder Fahrzeugen verwendet, die dann auf Codeblöcke abstrahiert werden. In der Informatik werden oft Vorlesungen mit starkem verbalem Input gehalten, was für verbale Lerntypen einen

Vorteil bieten kann. Zudem können visuelle Lerntypen bei der Dekomposition möglicherweise einen Vorteil, je nach Vorlesungsform, haben. Ein Vorteil wäre hier möglich, wenn beispielsweise für das Aufteilen der Verantwortungen einer Klasse in der objektorientierten Programmierung grafische Darstellungen genutzt werden.

3.1.2 Reflexiv Aktiv

Reflexive Typen könnten einen klaren Vorteil im Aspekt Debugging haben. Sie sind eher dazu veranlagt, ihre gefundenen Lösungen zu evaluieren und zu hinterfragen. Aktive Typen hingegen sind risikobereiter und sind in ihrer Arbeit experimentierfreudiger. Ihr Ansatz ist es eher, schnell zu scheitern, aber auch schneller neue Lösungen auszuprobieren. Reflexive Typen haben zudem noch einen Vorteil im algorithmischen Denken, da diese Probleme oft eigenständig und ohne Diskussion mit Anderen lösen können (Chen et al., 2023).

3.1.3 Sensorisch Intuitiv

Intuitive Typen haben einen klaren Vorteil in der Abstraktion, und demnach ebenfalls in der Mustererkennung (Felder & Soloman, o. J.-a). Dies ist der Fall unter anderem, da sie sich eher mit Konzepten statt Auswendiglernen beschäftigen. Sie haben oft Schwierigkeiten in Kursen, die stark auf Formelanwendung und Wiederholung setzen, und bevorzugen stattdessen neue Theorien und Interpretationen. Dies kann ebenfalls durch einen praktischen Versuch bestätigt werden, in dem Vor- und Nachteile der Lerntypen hinsichtlich der CT Aspekte untersucht wurden (Chen et al., 2023). Auch im algorithmischen Denken werden Personen mit einem Hang zu intuitivem Lernen einen Vorteil haben, da sie normalerweise Konzepte schneller lernen als sensorische Typen. Sie ziehen Innovation den etablierten Methoden vor und meiden eher Repetition. Dies könnte im Aspekt des algorithmischen Denkens von Vorteil sein, um neue Ansätze für Probleme zu entwickeln. Besonders im Zusammenhang mit dem Aspekt des Debugging könnte diese Innovation zu schnelleren und effizienteren Lösungen führen.

3.1.4 Sequenziell Global

In dieser Lerndimension lässt sich ein Vorteil für sequenzielle Lerntypen im Aspekt der Dekomposition erkennen. Das Herunterbrechen auf Teilprobleme ist nicht nur ein wichtiger CT Aspekt, sondern auch eine Veranlagung des sequenziellen Lerntypen, der es vorzieht, Probleme in einem „Step by Step“ Ansatz anzugehen. Sie beschäftigen sich mit allen Teilproblemen, bevor sie die komplette Lösung entwickeln können. Sequenzielle Typen können die einzelnen Teile eines Problems

angehen, ohne das große Ganze zu kennen. Sie haben daher einen entscheidenden Vorteil im Herunterbrechen der Teilprobleme im Vergleich zu globalen Lerntypen.

	Dekomposition	Abstraktion	Algorithmen	Debugging
Visuell/Verbal	Visuell / Verbal (je nach Vorlesungsstil)	Visuell		
Reflexiv/Aktiv			Reflexiv	Reflexiv
Sensorisch/Intuitiv		Intuitiv	Intuitiv	
Sequentiell/Global	Sequentiell			

Abbildung 4: Vorteile der Lerntypen in den Aspekten des Computational Thinking Zusammenfassung

3.2 Eignung der Lerntypen für die Programmierparadigmen

Nachdem die Zusammenhänge zwischen den Lerntypen und CT Aspekten untersucht wurden, muss noch betrachtet werden, inwiefern sich die verschiedenen Lerntypen für die Programmierparadigmen eignen. Auch bei den Lerntypen des Felder Silverman Modelles lassen sich teilweise Hänge erkennen.

3.2.1 Verbal Visuell

Hinsichtlich der Unterscheidung zwischen visuellen und verbalen Typen lässt sich in beiden Paradigmen für einen visuellen Vorteil argumentieren. Beim Erlernen von objektorientierten Konzepten wie Vererbung bieten sich visuelle Darstellungen wie Graphen an, ähnlich wie im Aspekt der CT Abstraktion. Allerdings können visuelle Typen einfacher mathematische Konzepte und Formeln auf die Philosophien der funktionalen Programmierung übertragen und die Zusammenhänge besser verstehen. Da FP einen starken Zusammenhang mit der Mathematik hat, können durch gewohnte Strukturen von Formeln und Funktionen einfachere Parallelen gezogen werden. Ob verbale Lerntypen hier wirklich im Nachteil sind, hängt allerdings sehr von der Strukturierung der Vorlesung ab.

3.2.2 Reflexiv Aktiv

In der aktiven und reflexiven Dimension lässt sich in beiden betrachteten Paradigmen ein Vorteil für den reflexiven Typen erkennen. Die praktische Umsetzung von Programmierkenntnissen erfolgt meistens in Einzelarbeit und erfordert eine stetige Reflexion des Lernenden. Das Erlernen von Programmierkenntnissen könnte zwar auch in Gruppenarbeiten erfolgen, aber der typische Vorlesungsstil begünstigt eher den reflexiven Lerntypen.

3.2.3 Sensorisch Intuitiv

In dieser Dimension lassen sich klare Präferenzen für bestimmte Paradigmen erkennen. Sensorische Typen werden höchstwahrscheinlich OO bevorzugen. Die Klassen der OO modellieren Objekte der echten Welt und haben somit einen einfach verständlichen Bezug auf echte Probleme. Intuitive Typen hingegen sind besser darin, Abstraktion anzuwenden, und werden daher eher FP bevorzugen. Ihr Hang zu Innovation und abstrakteren Konzepten ohne direkten Bezug zur realen Welt bietet ihnen einen Vorteil im Verständnis und in der Umsetzung von FP.

3.2.4 Sequenziell Global

Auch in der sequenziell globalen Dimension lassen sich klare Zusammenhänge zu den Paradigmen erkennen. Globale Lerntypen werden es höchstwahrscheinlich vorziehen, zunächst eine objektorientierte Klasse als Ganzes zu betrachten, bevor sie sich mit der Implementierung von Details beschäftigen. Um eine sinnvolle Hierarchie zur Komposition der verschiedenen Klassen zu entwerfen, muss grundlegend ein globales Verständnis der Probleme vorhanden sein. Sequenzielle Typen hingegen werden eher FP bevorzugen, da das Paradigma das Herunterbrechen von Problemen in Teilschritte von Grund auf vorsieht. Die Probleme werden in Teilfunktionen behandelt, und erst zum Schluss ergibt sich der Zusammenhang zu einem Programm. Ein globales Verständnis der Lösung ist nicht von Anfang an erforderlich.

	Objektorientierung	Funktionale Programmierung
Visuell/Verbal	Visuell/Verbal	Visuell/Verbal
Reflexiv/Aktiv	Reflexiv	Reflexiv
Sensorisch/Intuitiv	Sensorisch	Intuitiv
Sequentiell/Global	Global	Sequentiell

Abbildung 5: Vorteile der Lerntypen in den betrachteten Paradigmen Zusammenfassung

3.3 Ausprägung der Computational Thinking Aspekte in den Programmierparadigmen

Um die Zusammenhänge zwischen den Vor- und Nachteilen herzustellen, muss betrachtet werden, wie genau die einzelnen CT Aspekte in den Paradigmen ausgeprägt sind. Kein Aspekt kann in einem Paradigma vollständig ausgelassen werden, da CT generell für das Konzept des Programmierens benötigt wird. Allerdings beeinflussen die unterschiedlichen Schwerpunkte und Philosophien der Paradigmen, wie die Aspekte gewertet und gewichtet werden können. Aufgrund dieser Analyse können dann zusätzliche Schlüsse und Wichtungen der Vor- und Nachteile der Lerntypen gezogen werden.

3.3.1 Dekomposition

In der FP ist die Dekomposition sehr stark ausgeprägt. Jede Funktion hat nur eine einzige Verantwortung und gibt immer dasselbe Ergebnis wieder. Durch Vermeidung von Seiteneffekten ist die Funktionsweise jeder Methode eindeutig. Der

Programmablauf ist eher eine Abfolge von Befehlen als in der OO. In der OO erfolgt die Dekomposition nicht gezwungenermaßen auf Funktionsebene, sondern eher durch die einzelnen Klassen und deren Verantwortung.

Da in der FP die Dekomposition einen starken Fokus hat, könnten hierbei visuelle Lerntypen eventuell einen zusätzlichen Vorteil hinsichtlich des funktionalen Paradigmas mit sich bringen, je nachdem wie die Dekomposition in der Vorlesung behandelt wird.

3.3.2 Abstraktion

Die Abstraktion in der OO ist generell einfach zu visualisieren. Meistens werden die Probleme auf Klassen mit jeweils eigenen Funktionen heruntergebrochen. Diese Klassen wiederum können in Diagrammen oder Bildern mit ihren Funktionen dargestellt werden. In der FP ist die Abstraktion stark mit der Dekomposition verknüpft. Um die einzelnen Teilfunktionen zu entwerfen, muss das Problem zunächst in die kleineren Einheiten zerlegt werden. Es gibt eine sehr starke Abstraktion bis hin zu mathematischen Funktionen.

Da intuitive Typen tendenziell einen Vorteil in Abstraktion haben, könnten ihnen das Erlernen von FP insgesamt leichter fallen.

3.3.3 Algorithmen

Objektorientierte Sprachen bieten häufig ein breites Toolset für die Algorithmenentwicklung. Beispielsweise in Java sind Integer eine eigene Klasse, mit jeweils eigenen Operationen, die bei der Entwicklung verwendet werden können. In der FP hingegen ist das Toolset deutlich begrenzter, auch durch die Abwesenheit von Zuständen.

Da FP stark auf innovative Lösungsansätze setzt, können intuitive Typen hier besonders profitieren. Außerdem könnten sequenzielle Lerntypen einen zusätzlichen Vorteil in der FP haben. Die Umsetzung der Teilprobleme erfolgt isoliert voneinander und wird erst zum Schluss wieder in der Funktionskomposition zusammengesetzt. Ein globales Verständnis des Problems ist daher weniger essenziell als in der objektorientierten Programmierung. Dieser Teil kann ebenfalls mit der Dekomposition in Verbindung gebracht werden, die bei der Aufteilung in Teilprobleme behilflich ist. Da FP keine veränderlichen Werte und Zustände hat, gibt es zudem keine herkömmlichen Schleifen. Stattdessen erfolgt das wiederholte Ausführen einer Funktion über Rekursion, was ein Umdenken in der Problemlösung erfordert.

3.3.4 Debugging

In der OO gibt es häufiger ein „klassisches“ Debuggen. Klassen können diesen Prozess möglicherweise unübersichtlicher machen als in der FP. FP hingegen setzt stark auf Fehlervermeidung, indem Funktionen keine Seiteneffekte haben und deterministisch sind. Durch die eindeutige Verantwortung jeder Funktion treten weniger Flüchtigkeitsfehler oder unbeabsichtigte Seiteneffekte auf. Während FP tendenziell weniger Bugs durch Seiteneffekte produziert, erfordert das Debuggen trotzdem mehr Reflexion, weil die Fehlerquellen oft in der Funktionskomposition und Typensystemen liegen. Es zeigt sich daher, dass funktionale Programmiersprachen wie Clojure, Scala und Haskell tendenziell weniger nachträgliche Bugfixes in Relation zu anderen Commit Typen haben, als vergleichbare objektorientierte Sprachen wie Java oder C++.

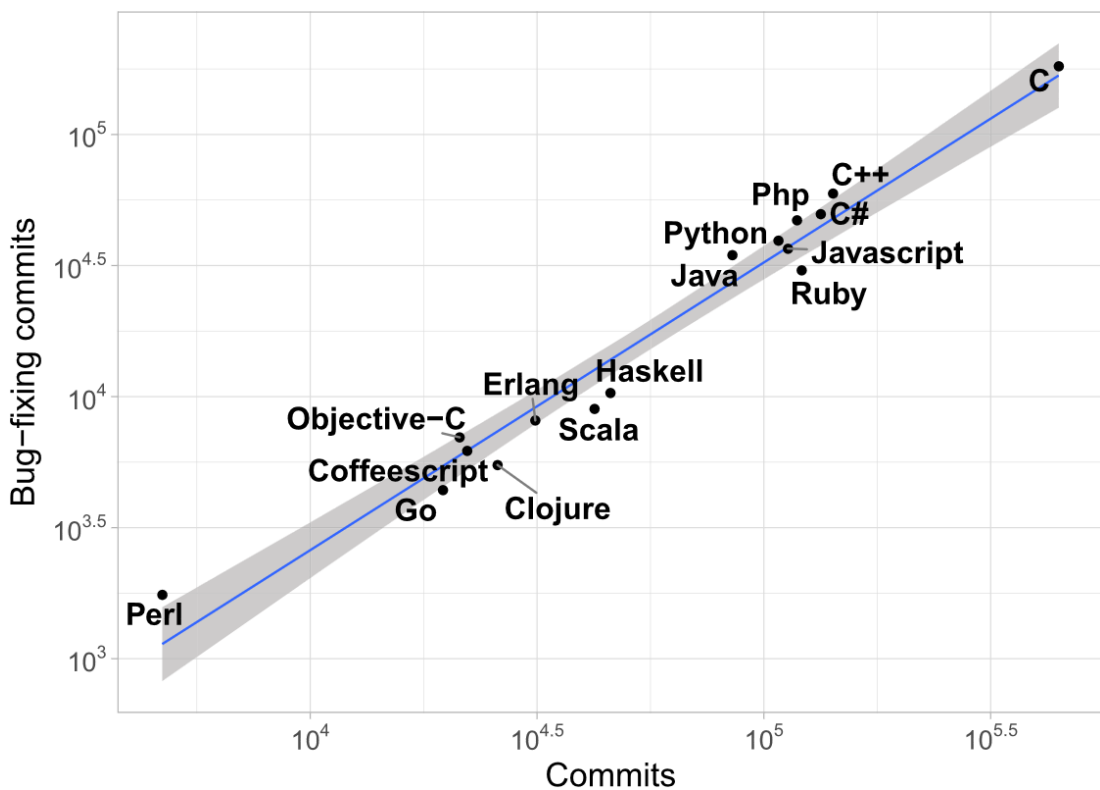


Abbildung 6: Commits und Bugfixing Commits in verschiedenen Programmiersprachen (Berger et al., 2019)

Aufgrund der besonders ausgeprägten Anforderung an die Reflexion beim Schreiben des Codes haben reflexive Typen möglicherweise einen zusätzlichen Vorteil in der FP. Debugging in der FP erfordert ein abstraktes Verständnis der Vorgänge und eine ständige Reflexion der Arbeitsschritte.

	Dekomposition	Abstraktion	Algorithmen	Debugging
Objektorientiert	Klassenebene	Objekt auf Klassen	Großes Toolset der einzelnen Klassentypen	Potential der Unübersichtlichkeit (Globale Übersicht nötig)
Funktional	Funktionsebene	Vorgänge auf Mathematische Funktionen	Begrenztes Toolset, Fokus auf Rekursion	Großer Fokus auf Maintainability

Abbildung 7: Ausprägungen der CT Aspekte in den jeweiligen Programmierparadigmen Zusammenfassung

3.4 Zusammenfassung

Zusammengefasst lassen sich also definitiv Hänge der Lerntypen zu bestimmten Paradigmen erkennen. In diesem Abschnitt werden die Vor- und Nachteile besonders im Hinblick auf die FP noch einmal übersichtlich dargestellt.

Für die CT Aspekte lässt sich zusammenfassend sagen, dass besonders Personen mit visuell, reflexiv, intuitiv und sequenziell ausgeprägten Veranlagungen einen Vorteil beim Erlernen der einzelnen Aspekte haben.

Für die Lerntypen lässt sich zusammenfassend sagen, dass besonders Personen mit reflexiv, intuitiv und sequenziell ausgeprägten Veranlagungen Vorteile beim Lernen mit funktionaler Programmierung haben. Personen mit reflexiv, sensorischen und global ausgeprägten Eigenschaften hingegen werden besser mit objektorientierter Programmierung lernen. Je nach Vorlesungsinhalt könnte ein visuell ausgeprägter Lerntyp noch einen zusätzlichen Vorteil in beiden Paradigmen haben.

Zu den einzelnen Ausprägungen der CT Aspekte in den Paradigmen lassen sich zusätzliche Vorteile der Lerntypen ziehen. Hat ein Lerntyp beispielsweise sowohl einen Vorteil beim Erlernen von algorithmischem Denken, als auch eine Neigung zu innovativen Lösungsansätzen, etwa durch eine besondere Ausprägung der intuitiven Lerntypendimension, wird dieser sich auch mit dem Toolset der FP leichter tun.

3.4.1 Vorteile der Lerntypen in funktionaler Programmierung

Durch die Analyse sowohl der Programmierparadigmen als auch der Computational Thinking Aspekte hinsichtlich der Lerntypen haben sich klare Vorteile einiger Ausprägungen ergeben. Insbesondere folgende Lerntypen werden demnach Vorteile haben, wenn sie ihren ProgrammierEinstieg mit funktionaler Programmierung machen.

Visuell Da durch einen Vorteil in der Abstraktion ein direkter Bezug zu FP im Zusammenhang mit mathematischen Formeln gemacht werden kann.

Ein weiterer Vorteil in der Dekomposition kann sowohl visuellen als auch verbalen Lerntypen zugeordnet werden, abhängig von den in der Vorlesung verwendeten Darstellungsformen.

Reflexiv Da Programmierung tendenziell eher in Einzelarbeit erfolgt, und der Lerntyp in herkömmlichen Vorlesungsstilen besser unterstützt wird. Der reflexive Typ profitiert stark durch die Fehlervermeidungsstrategie von FP. Durch ständige Reflexion der Teillösungen können potenzielle Fehler früh erkannt werden. Der Vorteil im Debugging bietet reflexiven Lerntypen einen entscheidenden Vorteil bei der Anwendung von FP. Zudem hat der reflexive Typ einen Vorteil in der effektiven Findung von Lösungen durch algorithmisches Denken.

Intuitiv Da dieser Typ in der FP einen besonderen Vorteil in der Abstraktion hat, die in der FP essenziell ist, da die meisten funktionalen Programmiersprachen auf dem Lambda-Kalkül basieren. Da Innovation bei neuen Algorithmen wichtig in der FP ist, hat dieser Typ auch hier einen Vorteil. Zudem kann eine intuitive Lernweise das natürliche Anwenden von Rekursion in der Lösungsstrategie begünstigen, da eine Ausprägung der intuitiven Lerndimension das Anwenden innovativer Algorithmen unterstützen kann.

Sequenziell Da FP einen eher sequenziell veranlagten Entwicklungsstil hat, in dem ein Teilproblem nach dem anderen angegangen wird. Zudem kann eine sequenzielle Veranlagung zusätzlich bei der Anwendung von Dekomposition helfen, um die einzelnen Teile der Problematik nacheinander zu betrachten.

Diese Ausprägungen decken sich zudem mit den ausgearbeiteten Vorteilen zum Erlernen von CT Aspekten, was einen zusätzlichen Vorteil bieten kann.

3.4.2 Nachteile der Lerntypen in funktionaler Programmierung

Andererseits haben im Umkehrschluss zum vorherigen Abschnitt wiederum folgende Lerntypen eher Nachteile bei FP.

Verbal Da verbal schwieriger ein Bezug zu den mathematischen Grundlagen von FP gemacht werden kann. Ob ein verbaler oder ein visueller Typ in der Dekomposition eher im Nachteil ist, ist jedoch wie bereits erwähnt größtenteils abhängig davon, welche Darstellungsformen in der Vorlesung verwendet werden.

Aktiv Da die Programmierung allgemein reflexive Typen bevorzugt. Ein aktiver Stil widerspricht der Fehlervermeidung in FP, da diese eine ausführliche

Reflexion der Arbeitsschritte erfordert. Zudem hat ein aktiver Typ Nachteile in der Findung von Lösungen durch algorithmisches Denken. Informationen können durch aktives Arbeiten und Diskussion besser verarbeitet werden und brauchen so mehr Zeit, um richtig entwickelt zu werden.

Sensorisch Da nicht so ein guter Bezug zu echten Problemen hergestellt werden kann. Zudem hat ein sensorischer Typ eher Nachteile beim Erlernen von Abstraktion und beim innovativen Ausarbeiten von neuen Lösungen für Probleme. Das Anwenden etablierter Methoden ist besonders beim Erlernen von FP schwieriger, da sich das Paradigma von anderen Programmierstilen stark unterscheidet.

Global Da eher ein sequenzielles Denken gefordert ist. In der OO ist ein globales Denken von Vorteil, da der Zusammenhang zwischen den Domänen eine Rolle spielt und ein Verständnis der gesamten Architektur nötig ist. In der FP allerdings werden die Probleme eher schrittweise angegangen, was für diesen Lerntypen ein Nachteil sein könnte.

4 Praktischer Selbstversuch

Um die Schlussfolgerungen aus dem Forschungsteil weiter zu untersuchen, wurde entschieden, die Vor- und Nachteile der Lerntypen in einem praktischen Selbstversuch zu betrachten. Es soll hierbei speziell untersucht werden, welche Schwierigkeiten erwartet werden, und welche tatsächlich bei der Bearbeitung auftreten. Hierzu werden die einzelnen Lerndimensionen der Autorin untersucht. Anschließend wird ein einfacher Algorithmus in einer funktionalen Programmiersprache umgesetzt. Es wurde sich für Haskell entschieden, da es die meistverwendete funktionale Programmiersprache in Einsteigerkursen im Bereich Informatik ist (siehe Untersuchung Curricula). Zur Vorbereitung wurde der Kurs CIS 194 (Yorgey, 2013) der University of Pennsylvania genutzt, der in der offiziellen Haskell-Dokumentation empfohlen wird (*First steps (How to learn Haskell proper)*, o. J.). Als Problem wurde „Türme von Hanoi“ gewählt. Weitere Erläuterungen zu der Art des Problems folgen in Abschnitt Problembeschreibung.

4.1 Lerntypenanalyse

Um den Selbstversuch mit den Vor- und Nachteilen der Lerntypen zu verbinden, musste zunächst eine eigene Lerntypenanalyse durchgeführt werden. Hierzu wurde der „Index of Learning Styles Questionnaire“ verwendet (Felder & Soloman, o. J.-b), ein Online-Tool, welches insgesamt 44 Fragen stellt und dann auf Basis der Antworten die Lerntypen eines Individuums einschätzen kann. Die Lerntypen werden als Gegensatzpaare auf einer Skala gezeigt, die darstellen soll, wie ausgeprägt der Hang zu einem Aspekt ist. Ein Wert von 1 bis 3 entspricht einer ausgewogenen Balance beider Aspekte der Dimension mit leichten Präferenzen, 5 bis 7 entspricht einem mäßigen Hang zu einem Aspekt und 9 bis 11 entspricht einer starken Präferenz für einen der Lernstile. Die Verlässlichkeit dieses Fragebogens wurde sowohl von den ursprünglichen Verfassenden des Lernmodells (Felder & Spurlin, 2005) als auch von anderen Quellen (Zywno & Zywno, 2003) geprüft und bestätigt.

Die Fragen des ILS Fragebogens wurden in einer zufälligen Reihenfolge beantwortet, um zu vermeiden, dass bereits vorab klar ist, welche Frage welcher Typendimension zugeordnet ist. Dies ist möglich, da der Test so aufgebaut ist, dass jeweils ein Frageblock von 4 Fragen jede Dimension behandelt. Die erste Frage beispielsweise fließt in die Bewertung der aktiv/reflexiven Dimension ein, ebenso wie die fünfte und neunte Frage.

4.1.1 Ergebnisse der persönlichen Evaluierung

Der Fragebogen wurde Online auf der offiziellen Seite der University of Pennsylvania bearbeitet. Die Ergebnisse des Fragebogens lauten wie folgt.

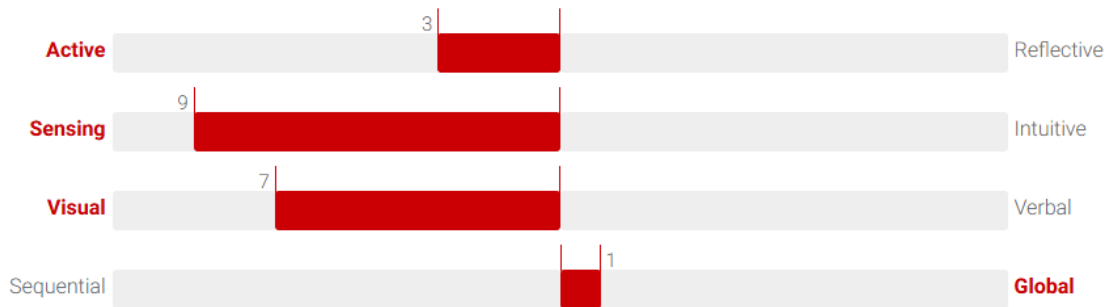


Abbildung 8: Ergebnisse des ILS Fragebogens

Die Autorin zeigt demnach folgende Ausprägungen in den Dimensionen.

- Ein ausgewogenes Verhältnis der aktiven/reflexiven Dimension, möglicherweise eine leichte Präferenz für den aktiven Typ
- Eine starke Präferenz für den sensorischen Typ
- Eine mäßige Präferenz für den visuellen Typ
- Ein ausgewogenes Verhältnis der sequenziellen/globalen Dimension, möglicherweise eine leichte Präferenz für den globalen Typ

Die individuellen Ausprägungen der Lerntypen werden im praktischen Versuch hinsichtlich des Erlernens und Anwendens der CT Aspekte und der verschiedenen Paradigmen untersucht. Die Ausprägungen sollten sich demnach wie folgt auswirken.

- Ein leichter Nachteil beim Debugging in funktionaler Programmierung (FP), und beim Erlernen von Programmieren in Einzelarbeit, allerdings auch ein schnelleres Ausprobieren von Ansätzen. Zudem ein leichter Nachteil in der Anwendung von algorithmischem Denken und der effektiven Findung von Lösungsansätzen.
- Ein Nachteil beim Anwenden von algorithmischem Denken und der Erkennung der Zusammenhänge zwischen FP und mathematischen Funktionen, sowie beim innovativen Arbeiten mit einem limitierten Toolset. Demnach ebenfalls ein Nachteil bei der Findung von rekursiven Lösungsansätzen
- Ein möglicher mäßiger Vorteil bei der Dekomposition aus der Perspektive eines visuellen Lerntypen, allerdings ein Nachteil bei der Abstraktion aufgrund einer eher sensorischen Ausprägung

- Kein besonderer zusätzlicher Vorteil bei der Dekomposition aus der Perspektive eines sequenziellen Lerntyps, sowie der generellen Anwendung von FP in einer sequenziellen Lösungsstrategie

4.2 Problembeschreibung

Um die aufgestellten Thesen zu prüfen, wurde entschieden, eine Programmieraufgabe in einem praktischen Teil zu lösen und alle Erfahrungen, Schwierigkeiten und Probleme in einer Art „Development Diary“ zu dokumentieren. Als Problem für den praktischen Versuch wurden die „Türme von Hanoi“ gewählt. Diese Entscheidung wurde getroffen, da die Aufgabe in einem Anfängerkurs mit eigener Aufgabenstellung vertreten war (Yorgey, 2013), sowie alle Aspekte des CT abdeckt (siehe Auswahl des Problems für den Selbstversuch). Das Problem erfordert zudem die Anwendung von Rekursion, was in der FP einen besonderen Stellenwert hat. Die Aufgabenstellung des Kurses CIS 194 bietet einen Ansatz in Form einer vorgegebenen Funktionsdefinition, sowie zwei vordefinierten Datentypen. Aufgrund dessen, dass auch der Aspekt der Abstraktion und Dekomposition betrachtet werden sollte, wurde die Aufgabenstellung des Kurses CIS 194 nicht verwendet. Stattdessen wurde von Grund auf selbst eine Lösung gefunden.

In dem Problem geht es um drei Holzstäbe, auf denen unterschiedlich große, runde Holzplatten gestapelt sind. Das Ziel der Lösung ist es, die aufsteigend gestapelten Platten vom Turm ganz links hin zum Turm ganz rechts zu transportieren. Hierbei gelten drei Regeln. Zum einen darf nie mehr als eine Platte gleichzeitig bewegt werden. Eine Platte darf nur bewegt werden, wenn diese die oberste im Stapel ist. Außerdem darf sich eine größere Platte niemals auf einer kleineren befinden.

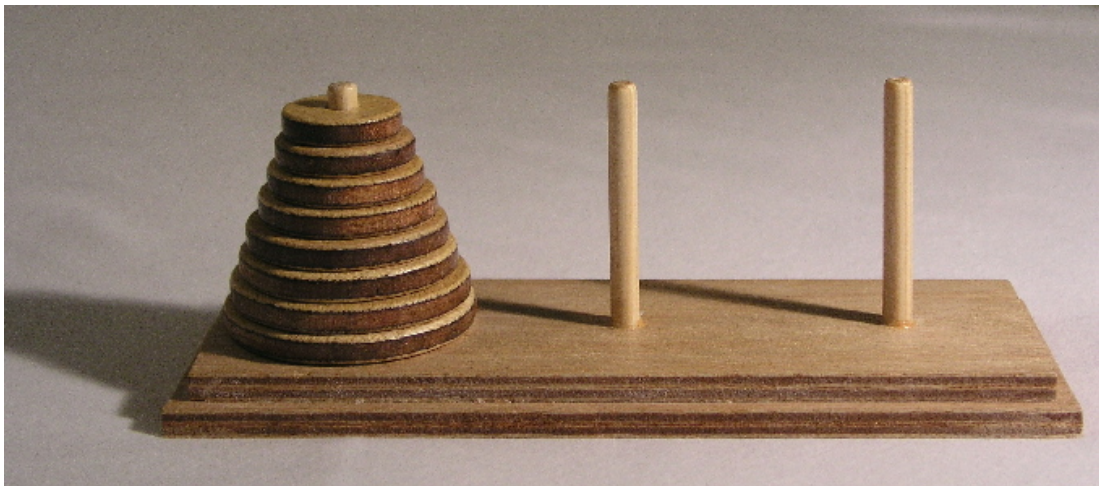


Abbildung 9: Ein Modellset der Türme von Hanoi mit 8 Holzplatten (Commons, 2024)

4.2.1 Theoretische Überlegungen

Um das Problem zu lösen, wurde zunächst unabhängig vom Code über das Problem nachgedacht. Es wurde entschieden, das Problem physisch nachzubauen. Diese Methode half besonders dabei, das Problem zu verstehen und erste Überlegungen zur Abstraktion des Problems durchzuführen. Zunächst wurde versucht, durch zufälliges Ausprobieren eine Lösung zu finden. Dies funktionierte, allerdings wurden keine besonderen Muster gefunden, die bei einer allgemeinen Lösung des Problems helfen könnten.

Die ersten tatsächlichen Lösungsansätze waren sehr kompliziert, da noch kein globales Verständnis für das Problem vorhanden war. Zunächst wurde überlegt, wie das Problem abstrahiert werden kann. Ein erster Versuch, das Problem in einer algorithmischen Denkweise zu betrachten, erfolgte mit dem Aufstellen von mehreren Regeln. Wenn eine Platte auf mehrere Pole bewegt werden kann, wird diese auf den ersten verfügbaren Pol von links gesetzt. Wir beginnen am linken Pol und bewegen die Platten so lange auf einen anderen Pol, bis keine legalen Züge mehr möglich sind. Danach wird der nächste Pol von links betrachtet, bis keine legalen Züge mehr ausgeführt werden können, und so weiter. Es wurde allerdings schnell beim Durchspielen dieses ersten Ansatzes am physischen Modell bemerkt, dass sich so schnell Endlosschleifen entwickeln.

In einem neuen Ansatz wurde das Problem in der reduziertesten Form betrachtet, mit insgesamt drei Holzplatten. Es zeigte sich, dass die größte Platte auf den rechten Pol zu setzen das erste „Teilziel“ des Problems darstellt. Die anderen Platten werden durch Hilsschritte geordnet und in der Mitte als Hilfspol zwischengelagert. Der Ansatz, auf Visualisierungen des Problems zurückzugreifen, half stark dabei, Muster in der Lösung zu erkennen. Bei der Überlegung, wie man die Lösung nach dem funktionalen Paradigma umsetzen kann, fiel auf, dass Rekursion zwingend nötig sein wird. Falls mehr als 3 Holzplatten vorhanden sind, kann das Problem so lange weiter reduziert werden, bis nur noch drei Platten betrachtet werden. Von dort aus lässt sich das Problem rekursiv mit $n-1$ Platten lösen.

4.2.2 Umsetzung

Im nächsten Schritt wurde überlegt, wie das Problem auf eine virtuelle Welt abstrahiert werden kann. Die Holzplatten wurden hierbei als Integer dargestellt, um die Größen der Platten vergleichen zu können. Die Pole selbst wurden als Listen dargestellt, in denen jeweils das erste Element die oberste Holzplatte darstellt. Ob ein Zug dem legalen Bewegungsset entspricht, wird anhand dessen bestimmt, ob das zu bewegend Element kleiner ist als das Element, auf dem dieses platziert

wird. Elemente werden dann so lange bewegt, bis die „Win Condition“ erreicht ist, in dem Fall, dass sich in den ersten beiden Listen keine Elemente mehr befinden. Demnach gibt es eine Aufteilung der Verantwortungen, in denen zwei Funktionen die Gültigkeit und Zielbedingungen des aktuellen Zustandes prüfen, und eine Funktion, welche die tatsächlichen Züge durchführt und von den anderen beiden Hilfsfunktionen Gebrauch macht.

Da diese Funktion rekursiv laufen soll, muss sie alle nötigen Parameter entgegennehmen. Hierzu gehören der Startpol, Zielpol und ein Hilfspol, auf dem Platten zwischen platziert werden können. Zudem wird die Anzahl von Holzplatten benötigt, die sich in jedem rekursiven Schritt reduzieren soll. Um das Endprodukt zu dokumentieren, wird zudem eine Liste von Lösungsschritten übergeben, die jeweils in einem Tupel den Startpol sowie den Endpol speichert. Hierbei muss aufgrund der Beschränkungen in der FP besonders darauf geachtet werden, wie die Lösungsschritte dokumentiert werden können. Diese können nicht etwa wie in einer prozeduralen Lösung nach jedem Schritt in der Konsole gedruckt werden, da FP keine Seiteneffekte zulässt.

Die Umsetzung des Problems im funktionalen Paradigma erfolgte in Haskell. Zunächst wurden die Funktionsdefinitionen und Datenstrukturen geschrieben, und anschließend der Basisschritt und der rekursive Schritt. Dabei fiel allerdings auf, dass die Umsetzung mit den einzelnen Polen als Listen von Integern nicht realistisch ist. Hierbei wurde in der Theorie zu sehr in herkömmliche prozedurale Denkstrukturen zurückgegriffen. Die Listen mit den Integern müssten demnach in jedem Schritt weiter verändert werden und haben somit eine Art Zustand, der nicht dem funktionalen Paradigma entspricht. Es wurde sich daher im Prozess der Implementierung entschieden, die Datenstrukturen wieder zu verwerfen und ausschließlich die Liste mit Lösungsschritten als Rückgabetypp zu geben. Da die Schrittfolge im Falle $n=3$ immer gleich ist und nur rekursiv wiederholt werden muss, müssen die Schritte zudem nicht gezwungenermaßen überprüft werden. Das Problem konnte letztendlich auf eine einzige Funktion mit nur einer Verantwortung reduziert werden. In einer späteren Reflexion der Lösung wurde entschlossen, dass dies die beste Lösung für eine funktionale Umsetzung ist.

4.3 Zusammenhang mit Lerntypen

Bei der Bearbeitung der Aufgabe konnten mehrere Beobachtungen gemacht werden, die sich direkt in Bezug auf die Erkenntnisse des Forschungsteils setzen lassen. Basierend auf den Ergebnissen des Fragebogens wurde erwartet, dass die mäßige Ausprägung der visuellen Dimension Vorteile für die Abstraktion und möglicherweise auch die Dekomposition mit sich bringt. Dabei ist letzteres abhän-

gig von den gewählten Darstellungsmethoden während der Bearbeitung. Tatsächlich erwies sich die Visualisierung des Problems durch einen physischen Nachbau als eine der hilfreichsten Strategien in der Lösungsfindung. Auch das Aufteilen des Problems in Teilverantwortungen fiel leicht, jedoch ist dies unabhängig von den Vor- und Nachteilen des visuellen Lerntypen zu betrachten, da hierbei keine spezifischen visuellen Methoden eingesetzt wurden.

Hinsichtlich der aktiven/reflektiven Dimension wurden aufgrund eines leichten Hanges zum aktiven Typen mit Schwierigkeiten des reflektierten Arbeitens beim Debugging und bei der Anwendung von algorithmischem Denken gerechnet. Tatsächlich ließ sich beobachten, dass eine sehr aktive Arbeitsweise verwendet wurde, bei der das schnelle Ausprobieren und Fehlschlagen im Vordergrund standen. Hierbei waren einige unterstützende Tools behilflich (siehe Vorgehen im Selbstversuch). Im Nachhinein lässt sich sagen, dass diese Vorgehensweise im Kontext von FP nicht optimal war. Sie führte dazu, dass mehr Zeitaufwand für eine rekursive Lösung nötig war, als mit einer reflektierteren Arbeitsweise nötig gewesen wäre.

In der intuitiv/sensorischen Dimension wurden ebenfalls einige Hürden erwartet. Sensorische Typen haben eher Schwierigkeiten bei der Arbeit mit Abstraktion und Algorithmen als ihr Gegenpaar. Besonders in der FP lassen sich etablierte Methoden und Denkweisen nur schlecht anwenden, da sich das Paradigma stark von anderen Programmierstilen unterscheidet. Im Kontext von FP bedeutet dies Schwierigkeiten bei der innovativen Entwicklung neuer Algorithmen, bei der Arbeit mit einem begrenzten Toolset, sowie Probleme beim rekursiven Denken. Einer der am deutlichsten erkennbaren Schwierigkeiten im praktischen Teil war es, etablierte Methoden schlechter anwenden zu können, besonders um den Algorithmus allgemeiner und rekursiv zu gestalten. Es brauchte einen größeren Zeitaufwand, um sich auf die neue Denkweise umzustimmen.

In der sequenziell/globalen Dimension wurde ein recht ausgewogenes Verhältnis festgestellt, mit einer leichten Präferenz zum globalen Typ. Das Problem wurde in der theoretischen Lösungsfindung zunächst stark sequenziell betrachtet. Sobald ein ausreichendes globales Verständnis des Problems vorhanden war, fiel es einfach, die Funktionen letztendlich zu implementieren.

5 Fazit

Abschließend werden noch einmal alle Ergebnisse der Bachelorarbeit insgesamt betrachtet und in Beziehung zueinander gesetzt. Zudem erfolgt eine kurze Reflexion zum Prozess der Arbeit, sowie mögliche Erweiterungen der Forschungsfrage in zukünftigen Arbeiten.

5.1 Schlussfolgerungen aus Theorie und Praxis

Abschließend lässt sich also sagen, dass die Ergebnisse der Arbeit darauf hindeuten, dass ein Zusammenhang zwischen den verschiedenen Lerntypen und ihrer Eignung für funktionale Programmierung (FP) besteht. Im Verlaufe der Arbeit wurden sowohl Vor- als auch Nachteile für die einzelnen Lerntypendimensionen identifiziert und mithilfe eines Selbstversuches analysiert. Die Erkenntnisse legen nahe, dass Probleme von Programmieranfängenden möglicherweise reduziert werden können, wenn die Stärken und Schwächen dieser berücksichtigt werden. Dies könnte zu einer geringeren Abbruchrate beitragen und den Einstieg in das Studium für neue Studierende generell einfacher gestalten. Dies könnte beispielsweise so umgesetzt werden, dass zu Beginn des Studiums eine Selbsteinschätzung der Studierenden stattfindet, hinsichtlich ihrer eigenen Lerntypdimensionen. Somit können individuelle Stärken unterstützt werden. Je nachdem, mit welchem Ansatz sich eine Person mehr identifiziert, könnten anschließend abgestimmte Aufgaben für die jeweiligen Gruppen verteilt werden. Wenn Studierende zu Beginn des Studiums unterstützt werden können, liegt es nahe anzunehmen, dass sie mithilfe der Grundlagen auch im weiteren Verlaufe ihres Studiums weniger Schwierigkeiten haben werden.

Auch könnten Studierende selbst ohne zusätzliches Lehrmaterial besser verstehen, wie sie am besten die Inhalte der Vorlesungen nachvollziehen können. Gegebenenfalls erkennen sie durch ein Arbeiten mit einem speziellen Programmierparadigma wie FP, dass noch Schwierigkeiten in einigen Computational Thinking (CT) Aspekten bestehen, auf die sie sich außerhalb der Vorlesung konzentrieren können.

Im Hinblick auf die zu Beginn formulierten Forschungsfragen lassen sich folgende Schlussfolgerungen ziehen.

1. **Fällt es jedem gleich leicht, Programmieren zu lernen, oder gibt es Hänge zu einem bestimmten Paradigma?** Abschließend lässt sich auf Basis der in der Arbeit erlangten Erkenntnisse sagen, dass Hänge zu Paradigmen erkannt wurden. Welche Paradigmen von wem bevorzugt werden, hängt von den individuellen Lerntyp-Präferenzen ab, könnte allerdings

auch von der Gestaltung der Vorlesungen beeinflusst werden. Eine genauere empirische Überprüfung der Thesen steht noch aus.

2. **Welche Vor- und Nachteile haben gängige Paradigmen für die jeweiligen Lerntypen?** Es wurden im Laufe der Arbeit eindeutige Vor- und Nachteile für FP und Objektorientierung (OO) gefunden. Andere Paradigmen wurden zunächst nicht betrachtet, könnten allerdings in der Zukunft auch in den Kreis der zu betrachtenden Paradigmen aufgenommen werden.
3. **Welchen Lerntypen würde funktionale Programmierung beim Lernen von Entwicklung helfen, und welchen Typen würde der Ansatz eher schaden?** Im Forschungsteil der Arbeit konnte identifiziert werden, welche Kombination von Ausprägungen der Lerntypdimensionen ein Lernen von FP besonders begünstigt. Die Erkenntnisse stützen sich hierbei auf den in der Arbeit gezogenen Schlussfolgerungen.

5.2 Reflexion

Dieser Abschnitt betrachtet die Umsetzung der Bachelorarbeit in Hinsicht auf die ursprünglich geplante Durchführung und wägt ab, wieso Schwierigkeiten auftraten, und welche Verbesserungen sinnvoll wären.

5.2.1 Erfolge

Einige Aspekte der Bachelorarbeit fielen leichter als zunächst angenommen. Besonders hat hierbei das Felder-Silverman Lerntyp Modell (FSLSM) geholfen. Dies konnte man sehr gut in den Kontext der Informatik setzen, unter anderem weil das Modell auch in der Vergangenheit in anderen Studien des Feldes als Richtlinie verwendet wurde.

Die Verfassenden des Modells legen zudem einen besonderen Wert darauf, dass das Modell und die Theorie weiterhin frei zugänglich bleiben, und stellen auf ihrer Webseite sehr viele Links und Ressourcen zur Verfügung. Die Ressourcen beinhalten unter anderem auch Belege zur Verlässlichkeit des Lerntypenmodell-Fragebogens, was die Entscheidung zur Verwendung des Modells als Mittel sehr einfach machte.

Der Fragebogen der offiziellen Seite war zunächst nicht funktional, aber auf Anfrage stellte sich heraus, dass dies daran lag, dass international versucht wurde, auf eine Webseite der University of Pennsylvania zuzugreifen. Mithilfe eines Virtual Private Networks (VPN) konnten diese Probleme behoben werden, und die Seite konnte normal verwendet werden.

Weiterhin war das Lernen mithilfe des Kurses der University of Pennsylvania sehr hilfreich. Der Kursinhalt war leicht verständlich und gut aufgebaut. Als Vorbereitung auf den praktischen Selbstversuch wurden einige Übungsaufgaben des Kurses bearbeitet. Dies half erheblich mit dem Erlernen der Haskell Syntax und der generellen Denkweise von FP.

5.2.2 Schwierigkeiten

Eines der frühesten Probleme der Bachelorarbeit fand sich im Rechercheteil. Zum Begriff CT ließ sich keine einheitliche Definition finden, und es war sehr schwer abzuwägen, welche Studien am vertrauenswürdigsten sind. Um diesem entgegenzuwirken, wurden mehrere Studien und Literatur zu dem Thema betrachtet, die es sich vornehmen, die bisherigen Forschungsergebnisse aller Studien zusammenzufassen und auf einen einheitlichen Stand zu bringen. Um eine möglichst objektive Auswahl zu treffen, wurden vier Aspekte gewählt, die in der Forschung am häufigsten diskutiert und untersucht wurden. Hierzu wurden mögliche Unteraspekte unter größeren CT Aspekten gruppiert.

Zudem war eines der größten Probleme der Bachelorarbeit, einen Zusammenhang der drei Themenfelder, CT, FP und das FSLSM, zu finden. Es wurde ein klarer Zusammenhang zwischen CT und den Lerntypen, sowie FP und den Lerntypen gefunden. Diese Vor- und Nachteile dann allerdings wiederum in der dritten Ebene zu kombinieren erwies sich als besonders zeitaufwendig. Es war sehr schwierig zu bestimmen, wie CT und FP zusammenhängen, und welche Aspekte eine größere oder kleinere Rolle in den einzelnen Paradigmen spielen. Jeder CT ist in seiner eigenen Weise für das Programmieren allgemein wichtig. Da es keine vorhandene Forschung zum Zusammenhang von Programmierparadigmen und CT gab, basieren die Ergebnisse dieses Abschnitts größtenteils auf eigenen Schlussfolgerungen und Annahmen. Es wurden verschiedene Überschriften für die Themen entworfen, die die Gedankenprozesse ein wenig eingrenzten. Letztlich wurden die Abschnitte zunächst wieder gelöscht und durch eine stichpunktartige Herangehensweise neu formuliert, um die Gedanken klarer zu formulieren. Zudem half es, eine visuelle Zusammenfassung der Schlussfolgerungen der einzelnen Abschnitte in Form der Tabellenabbildungen zu haben. Auf die Visualisierungen wurde auch immer wieder zurückgegriffen. Letztendlich orientierte sich die Struktur stärker an den Forschungsfragen als an der ursprünglich geplanten Gliederung der Abschnitte.

Zusätzlich erwies sich als herausfordernd, die Wichtigkeit der einzelnen CT und FP Aspekte angemessen herauszustellen. Beispielsweise ist Rekursion ein zentrales Konzept in der FP, ließ sich im Forschungsteil aber nur mit algorithmischem Denken in Verbindung setzen. Es war schwierig angemessen zu artikulieren,

welche Aspekte der FP besonders relevant sind, insbesondere durch den Mangel an nötiger praktischer Erfahrung im Thema. Dadurch ist es möglich, dass einige wichtige Schwerpunkte der FP nicht ausreichend im Forschungsteil berücksichtigt wurden.

Eine weitere Schwierigkeit war insgesamt, die Forschungsfragen mit dem Selbstversuch zu belegen. Der Zweck des Selbstversuches war es, einen Programmieranfänger zu simulieren, der noch keine Erfahrungen in funktionaler Programmierung gemacht hat. Die Autorin hatte zwar keine funktionalen Programmierkenntnisse, allerdings bereits generelles Programmierwissen. Es bleibt daher unklar, inwiefern die bereits angeeigneten CT Aspekte in der Lösungsfindung geholfen haben. Insbesondere die Aspekte, in denen ein Vorteil gegeben war, also die Abstraktion und Dekomposition, sind Aspekte der Informatik, die im Verlaufe des Studiums immer wieder weitergebildet und trainiert werden. Besonders bei der Implementierung fiel auf, dass eher auf herkömmliche prozedurale Denkstrukturen zurückgegriffen wurden, die anschließend in funktionale Funktionen umgewandelt wurden. Stattdessen sollten die Programmierfähigkeiten von Grund auf mit funktionaler Programmierung betrachtet werden. Es ist zudem unklar, ob die Schwierigkeiten, die im Laufe des praktischen Teiles auftraten, tatsächlich in Verbindung mit den Lerntypen gebracht werden können, oder ob diese nicht möglicherweise eher mit FP grundsätzlich zusammenhängen. Eine Diskussion dazu, wie diese Probleme umgangen werden könnten, findet sich im Abschnitt Untersuchung der Vor- und Nachteile in funktionaler Programmierung.

Zudem, als ein eher allgemeines Problem des Selbstversuches, war es sehr schwer und zeitaufwendig, sich mit der neuen Syntax von Haskell auseinanderzusetzen. Die Sprache ist fern genug von allen im Studium erlernten Programmierkenntnissen, um in der Bearbeitung eine Schwierigkeit darzustellen. Das lokale Einrichten der Sprache, sowie die Verwendung der REPL war allerdings sehr einfach, und gab schnelles Feedback beim Durchführen der Aufgaben.

5.3 Ausblick und Weiterführende Forschung

Abschließend werden mögliche Weiterentwicklungen und offene Fragestellungen in Hinblick auf die Forschungsfragen betrachtet, die sich aus den Schlüssen der Arbeit ergeben.

5.3.1 Unterstützung von benachteiligten Lerntypen

Ein Aspekt, der in der Arbeit vernachlässigt wurde, ist die Nachteilsausgleichung. Laut des FSLSM sollen die Lerntypen idealerweise nicht als Richtlinie für Lernmethoden verwendet werden. Allerdings ist es möglich, Personen mit schwächer

ausgebildeten Dimensionen der Lerntypen zu unterstützen. Beispielsweise sind herkömmliche Vorlesungen in Informatikstudiengängen eher in einer Art und Weise strukturiert, die verbalen Lerntypen einen Vorteil bieten würden. Personen mit schwächer ausgebildeten verbalen Kompetenzen könnten dabei unterstützt werden, diese Kompetenzen zu entwickeln. Parallel könnte diesen Personen allerdings auch gefördert werden, erste CT Kompetenzen zu erwerben, damit diese über einen längeren Zeitraum nicht den Anschluss an die Lerninhalte verlieren. Im Feld der Informatik wäre dies zum Beispiel durch die Verwendung der blockbasierten, visuellen Programmiersprache Scratch möglich. Letztendlich geht es nicht darum, allen Lerntypen gerecht zu werden, sondern gemeinsame Lösungen zu finden, die sicherstellen, dass alle im gleichen Tempo lernen können.

Die Frage, wie dieser gemeinsame Grund für alle Programmierneinsteiger in Studiengängen der Informatik mithilfe der Lerntypen gefunden werden kann, könnte eine eigene Forschungsfrage für eine zukünftige Arbeit sein. Hierbei könnten verschiedene Ansätze zur Umstrukturierung einer „klassischen“ Vorlesung entwickelt werden und gegebenenfalls unter Studienbeginnenden der Informatik vorgestellt werden.

5.3.2 Erweiterung der Curricula-Analyse

Aufgrund des Zeitmangels und weil die Curricula-Analyse nicht der Schwerpunkt der Bachelorarbeit war, wurden die Daten ausschließlich auf verwendete Programmiersprachen sowie Paradigmen untersucht. Was allerdings noch eine interessante Erweiterung wäre, könnte sein, die Kursinhalte selbst noch näher zu untersuchen. Dazu könnten entweder die Modulhandbücher herangezogen werden, die allerdings begrenzte Informationen über die Vorlesungen selbst zur Verfügung stellen. Andererseits könnten ausgewählte Universitäten und Hochschulen kontaktiert werden, um Zugang zu den detaillierten Lehrinhalten zu erlangen. Beispielsweise könnten die wenigen Kurse, die sich mit funktionaler Programmierung beschäftigen, darauf untersucht werden, wie tief die Lehrinhalte hier wirklich gehen. Wird beispielsweise FP in der Praxis angewandt? Oder wird diese nur in der Theorie gelehrt und ist möglicherweise nur Teil einer kurzen Übersicht über alle verbreiteten Programmierparadigmen?

5.3.3 Untersuchung anderer Paradigmen in Hinsicht auf CT Aspekte und Lerntypen

Im Rahmen der Bachelorarbeit wurde speziell OO und FP im Kontext CT und Lerntypen betrachtet. Bei der Curricula Analyse ist allerdings aufgefallen, dass oft OO und Prozedurale Programmierung zusammen gelehrt werden. Hierbei stellt

sich die Frage, welche Kompetenzen die jeweiligen Ansätze besonders gut vermitteln können. Möglicherweise gibt es noch ausgeprägtere Hänge der bestimmten Lerntypdimensionen zu einem anderen Paradigma, welches im Rahmen der Forschungsarbeit nicht betrachtet wurde.

5.3.4 Untersuchung weiterer CT Aspekte

Aufgrund dessen, dass es keine einheitliche Definition von CT gibt, wurden in der Arbeit die vier häufigsten Aspekte betrachtet. Es wurden allerdings zudem auch weitere Aspekte herausgestellt, die ebenso relevant für die Anwendung von CT sind. Diese Aspekte wurden im Rahmen der Arbeit nicht betrachtet, könnten allerdings in Zukunft ein weiterer Fokuspunkt einer weiterführenden Forschungsfrage sein. Hierbei könnten die zusätzlichen CT Aspekte weiter kategorisiert und hierarchisch geordnet werden.

5.3.5 Untersuchung der Vor- und Nachteile in funktionaler Programmierung

Aufgrund der zeitlichen Begrenzung wurde der praktische Versuch zur Untersuchung der Vor- und Nachteile der Lerntypen in funktionaler Programmierung mit der Autorin selbst durchgeführt. Da allerdings bereits vorher bekannt war, welche Lerndimensionen eher ausgeprägt sind, gab es einen Bias in der Untersuchung der Schwierigkeiten und Vorteile. Um eine verlässlichere Untersuchung durchzuführen, wäre es sinnvoll, den Versuch erneut anhand fremder Versuchspersonen durchzuführen, die sich im ersten Semester eines Informatikstudiums befinden. Hierzu könnte ein sehr kurzer Einleitungskurs zu FP entworfen werden, anhand dessen die Probanden eine Aufgabe durchführen. Im Anschluss an den Programmierteil wird dann der Lerntyp der einzelnen Personen mittels des Felder Silverman Fragebogens festgestellt.

Außerdem ist es möglich, dass das betrachtete Problem im praktischen Versuch letztendlich doch etwas kurz war, um alle CT Aspekte zu betrachten. Beispielsweise war es schwierig, das Problem in Teilverantwortungen im herkömmlichen Sinne zu unterteilen. Auch die sequenziell/globale Dimension konnte aufgrund dieses Aspektes nicht viele Erkenntnisse gewinnen. Eine Aufgabe wie der Sudoku Solver hätte letztendlich den zeitlichen Rahmen der Arbeit gesprengt, wäre allerdings zur genaueren Untersuchung der Vor- und Nachteile geeigneter gewesen. Auch in der Diskussion war kurzzeitig, statt einer kleineren Aufgabe vier Aufgaben zu bearbeiten, die sich jeweils auf einen CT Aspekt fokussieren. Dies wäre allerdings ebenfalls nicht mit dem gegebenen Zeitrahmen vereinbar gewesen, könnte allerdings eine gute Methodik darstellen, um die Forschungsfrage

in Zukunft weiter zu untersuchen.

5.3.6 Weitere Forschung zur Ausprägung der CT Aspekte in den Paradigmen

Aufgrund dessen, dass im Rahmen der Bachelorarbeit wenige Quellen zum Thema CT im Zusammenhang mit Programmierparadigmen gefunden wurden, basieren die meisten Schlussfolgerungen des Forschungsteils auf eigenen Annahmen. Allerdings könnten die Ausprägungen der einzelnen CT Aspekte ebenfalls in der Praxis untersucht werden. Durch die Stärken und Schwächen der einzelnen Lerntypen würden sich in einem praktischen Versuch möglicherweise Rückschlüsse darauf ziehen lassen, wie sich die Paradigmen auf einzelne CT Aspekte fokussieren. Diese Art von Forschungsfrage könnte sich ebenfalls mit der Frage auseinandersetzen, wie man Programmieranfängende am besten in ihrem Lernprozess unterstützen kann.

6 Anhang

Im Anhang wird das Vorgehen in den verschiedenen Abschnitten näher erläutert. Es werden alle Hintergrundinformationen, die nicht direkt für die Arbeit selbst relevant sind, gesammelt und dargestellt.

6.1 Vorgehen bei der Untersuchung der Curricula

Um auszuarbeiten, wie die aktuelle Situation an deutschen Hochschulen und Universitäten ist, wurde der Hochschulkompass der Hochschulrektorenkonferenz verwendet (<https://www.hochschulkompass.de/home.html>, o. J.). Auf der Webseite wurden Studiengänge der Informatik gesucht und nach folgenden Kriterien mithilfe der Filterfunktion der Webseite sortiert:

- Abschluss Bachelor/Bakkalaureus (Da Programmieranfänger betrachtet werden sollen, ergibt es im Kontext keinen Sinn, weiterführende Studiengänge oder Masterprogramme zu berücksichtigen)
- Studententyp Grundständig (Siehe Abschluss Bachelor/Bakkalaureus)
- Fachsuche Informatik (Speziellere Studiengänge wie Bioinformatik und Wirtschaftsinformatik wurden ausgeschlossen, um Überschneidungen zwischen den Hochschulen zu meiden. Es wurde immer ein Studiengang pro Institution untersucht, der sich möglichst nah an der Allgemeinen Informatik kategorisieren lässt)
- Studienfeld Angewandte Informatik oder Informatik (Siehe Fachsuche)
- Studienformen Vollzeitstudium (Ein weiteres Kriterium, um Überschneidungen zu meiden und die Studiengänge weiter auszusortieren)
- ohne Lehramt (Wurde in den Filtern aussortiert, da die Lerninhalte sowohl Informatik als auch Erziehungswissenschaften umfassen, und somit nicht im Fokus der Arbeit liegen)

Nach der Anwendung der Filter wurden noch insgesamt 425 Treffer angezeigt, die erst im späteren Verlauf der Arbeit weiter reduziert wurden (siehe Untersuchung der Daten).

6.1.1 Zeit Verwaltung

Da die Curricula-Analyse nicht der Schwerpunkt der Bachelorarbeit sein sollte, musste entschieden werden, wie viel Zeit in das Thema investiert werden soll.

Hierbei wurden mehrere Risiken gesehen. Zum einen ist es möglich, dass man durch Internetrecherche alleine nicht erschließen kann, welche Inhalte ein Modul hat. Zum anderen ist das größere Risiko wahrscheinlich der Zeitfaktor. Es ist ungewiss, wie lange es dauert, jedes Modul zu untersuchen, da jede Institution ihre Informationen anders sortiert, bereitstellt und handhabt.

Um die Zeit in einem realistischen Rahmen zu halten, wurde in Erwägung gezogen, einen Zeitraum festzulegen, in dem so viele Studiengänge wie möglich betrachtet werden, und diese anschließend die Forschungsmenge darstellen. Dieser Zeitraum könnte etwa auf 2 Arbeitstage fallen. Auch wurde abgewägt, die Studiengänge nach Anzahl der Studierenden zu sortieren und die 50 am besten besuchten Institutionen zu betrachten.

Es wurde allerdings ebenfalls notiert, dass das unsicherste Kriterium hier die Zeit war, die benötigt wird, um einen Studiengang zu untersuchen. Möglicherweise müssen die Methoden zur Reduktion der Zeit gar nicht angewandt werden, wenn sich die Risiken nicht erfüllen. Zunächst wurde daher versucht, abzuwägen, wie lange Analysezeit grob einzuschätzen war. In einer isolierten Probe wurden fünf zufällige Universitäten betrachtet (In diesem Fall die ersten fünf Suchergebnisse des Hochschulkompasses, die HS Furtwangen, die Ruhruniversität Bochum, die Hochschule Fulda, die Friedrich-Schiller-Universität Jena, und die Hochschule Konstanz). Es dauerte etwa 10 Minuten, um entsprechende Informationen über alle 5 Studiengänge zu erlangen. Bei den 121 verbleibenden Informatik-Studiengängen wurde die Zeitdauer also grob auf 4 Stunden eingeschätzt, ein realistischer Zeitraum zur Sammlung der Daten. Mit diesen neuen Informationen wurden die Methoden zur Zeitreduktion wieder verworfen.

Letztendlich wurde für die Analyse doch ein ganzer Arbeitstag benötigt, aber die reduzierte Menge der Studiengänge durch die Filter war bereits genügend, um die Daten in einem angemessenen Zeitraum zu sammeln.

6.1.2 Untersuchung der Daten

Bei der Analyse der Curricula wurde systematisch vorgegangen. Mithilfe eines Webscrapers wurden alle nötigen Informationen als JSON extrahiert und in einer Excel-Tabelle sortiert. Es wurde festgestellt, dass die Filterfunktion des Hochschulkompasses nicht ausreichend war, um die Datenmenge zu reduzieren. Die Studiengänge wurden daher noch einmal manuell aussortiert, nach weiteren Kriterien. Ein Studiengang wurde demnach aussortiert, wenn eines der folgenden Kriterien zutraf.

- Kein allgemeiner Informatikstudiengang (Zum Beispiel Bioinformatik oder Wirtschaftsinformatik)

- Zweitfach oder Nebenfach Informatik
- Teilzeitstudium
- Mehrere Studienorte für einen Studiengang (Hierbei sind die angebotenen Module teils nicht eindeutig zuordenbar)
- Doppelte Studiengänge für eine Institution (Etwa wenn sowohl Angewandte als auch Allgemeine Informatik angeboten wird. Es wurde immer der allgemeinere Studiengang gewählt. Zwischen internationalen und deutschen Studiengängen wurde immer der deutsche Studiengang gewählt)

Für die meisten Studiengänge ließen sich die nötigen Informationen in sehr kurzen Zeiträumen mit einer Suche nach dem Studienverlaufsplan, sowie dem Modulhandbuch für die Informatik finden. Hierbei wurde der Studienverlaufsplan genutzt, um herauszufinden, welcher Kurs als Einführung in die Programmierung im ersten Semester dient, und das Modulhandbuch, um zu extrahieren, welche Programmierparadigmen und Sprachen im Kurs verwendet werden.

Unterschied der Datensätze Unspezifisch und Keine Infos Nicht alle Module listeten die benötigten Informationen. Es wird in der Analyse grundsätzlich unterschieden zwischen zwei Fällen. Zum einen, Studiengänge, die zwar ein Modulhandbuch zur Verfügung stellten, dieses aber nicht explizit spezifiziert, welche Paradigmen und Programmiersprachen verwendet werden (Markiert als „Unspezifisch“). Zum anderen gibt es noch den Fall, dass kein Modulhandbuch öffentlich zur Verfügung steht. Dies kann etwa der Fall sein, wenn die Institution Informationen nur auf Anfrage herausgibt, oder das Modulhandbuch einfach nicht aufzufinden war. Es wurde davon abgesehen, die Institutionen zu kontaktieren, um einen neuen ungewissen Faktor in der Arbeit zu vermeiden. Die Studiengänge ohne Modulhandbuch wurden leer gelassen und sind in der Excel-Tabelle rot markiert. In der Grafik zur Darstellung der Ergebnisse sind diese Datensätze in „Keine Infos“ kategorisiert.

Die Excel-Tabelle mit den extrahierten Daten lässt sich im Repository der Bachelorarbeit finden (*Curricula Analyse Ergebnisse*, o. J.).

6.2 Ähnliche Forschungsergebnisse zum Forschungsteil

Da zum Zusammenhang von CT, Lerntypen und Programmierparadigmen wenig vorhandene Quellen gefunden wurden, basieren viele Annahmen des Forschungsteils auf Spekulationen, die auf Basis des Rechercheteils gemacht wurden. Um genauere Forschungsergebnisse zu erzielen, würde es sich anbieten, eine eigene

Studie im Rahmen der Bachelorarbeit durchzuführen. Dies konnte in dieser Arbeit aufgrund von zeitlichen Einschränkungen nicht umgesetzt werden, wäre allerdings eine mögliche Erweiterung der Forschungsfragen.

Die Quelle zum Thema CT und Lerntypen (Chen et al., 2023) bestätigte vorhandene Forschungsergebnisse nachträglich und erweiterte diese leicht. Es gibt auch andere Quellen, die allerdings nicht komplett mit den betrachteten Themen übereinstimmen. Hierzu wurden mehrere Untersuchungen von CT gefunden, allerdings nicht speziell im Kontext von Programmierparadigmen und Lerntypen.

Beispielsweise wurde hierbei vor der Verfassung des Forschungsteils eine Arbeit speziell zum Thema Entwicklung von CT mit Lernstilen betrachtet (Liu, Chen, Yu & Shih, 2022), allerdings verwendete die Arbeit das Lerntypenmodell nach David Kolb. Die Arbeit analysierte zudem eher die Methodiken, die verwendet werden können, um CT Kompetenzen möglichst effizient zu lehren, und bietet weniger Inhalt hinsichtlich des Zusammenhanges mit einzelnen Lerntypen.

Eine weitere Quelle (Michaelson, 2018) zum Thema Programmierparadigmen und CT half, die Wahl der häufigsten Paradigmen im Rechercheil zu bestätigen. Allerdings stellt diese Arbeit eher fest, dass CT eigentlich ein eigenes Paradigma der Programmierung ist und als solches angesehen werden sollte. Weniger wird CT in den Kontext von anderen Programmierparadigmen wie OO Programmierung, Prozedurale Programmierung und weitere gesetzt. Von daher konnten die Schlussfolgerungen der Arbeit eher schwierig für den Forschungsteil verwendet werden.

6.3 Auswahl des Problems für den Selbstversuch

Die Auswahl des Programmierproblems in Abschnitt 4 der Arbeit erfolgte an mehreren Kriterien. Es sollte ein relativ einfaches Problem gewählt werden, das eher einen Zweck demonstriert als kreative Lösungen erfordert. Zunächst wurde hierbei entschieden, einen Algorithmus zur automatischen Lösung und Erstellung eines Sudoku zu schreiben. Bei einem Sudoku handelt es sich um ein klassisches Rätsel, bei dem auf einem 9x9 großen Feld in jeder Zeile, Spalte, und in jeder 3x3 großen Box jede Zahl von 1 bis 9 nur ein mal vorkommen darf.

Das Problem eignete sich zum Untersuchen und Lernen von funktionaler Programmierung (FP), da alle wichtigsten Aspekte von FP benötigt wurden, um das Problem im Paradigma erfolgreich umzusetzen (Rekursion, Funktionen höherer Ordnung, Komposition, Unveränderlichkeit). Es wurden ebenfalls alle CT Aspekte ausreichend abgedeckt.

Dekomposition Herunterbrechen in Teilprobleme. Beispielsweise Validierung des Sudoku, dann Untersuchen der Spalten, Zeilen und Boxen separat.

Abstraktion Rätsel in Datenstrukturen umwandeln. Beispielsweise Überlegungen, wie das Brett und die Felder am besten im Code abgebildet werden können.

Algorithmen Zur Implementierung des Solvers.

Debugging Wie kann das Sudoku effizient gelöst werden? Evaluierung der Lösung und Erkennen von Fehlerpotenzial in den einzelnen Unterfunktionen.

Letztendlich wurde allerdings doch ein anderes Problem gewählt, da Sorgen hinsichtlich der Umsetzbarkeit im gegebenen Zeitrahmen bestanden. Es wurde letztendlich die Türme von Hanoi als Aufgabe für den Selbstversuch entschieden. Dies geschah hauptsächlich aus zwei Gründen. Zum einen gab es eine persönliche Empfehlung des betreuenden Prüfers. Zum anderen wurde das Problem im ersten Aufgabenblatt des verwendeten Kurses betrachtet, um Haskell zu lernen. Das Vorhandensein der Türme von Hanoi in einem Anfängerkurs für FP vermittelte den Eindruck, dass die Aufgabe besonders geeignet sein könnte, um den Einstieg in die FP zu simulieren. Das Problem vertritt ebenfalls wichtige Aspekte von FP, vor allem das Konzept des Backtrackings. Auch die CT Aspekte waren wiederum erkennbar vertreten, was letztendlich zur Entscheidung führte.

Dekomposition Überlegung, welche Teilprobleme es gibt. Ein bisschen weniger offensichtlich als beim Sudoku Solver, aber beispielsweise die Validierung der Züge und die tatsächliche Durchführung sind zwei verschiedene Verantwortungen.

Abstraktion Überlegung, wie die Pins und Holzplatten abgebildet werden können. Wie wird deutlich, welche Platte zu welchem Pin gehört? Wie wird die Größe der Platten deutlich? Hierbei können beispielsweise Integer verwendet werden.

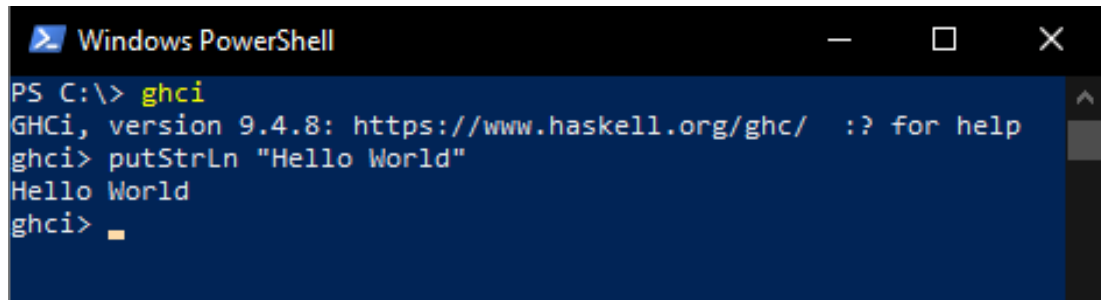
Algorithmen Zur tatsächlichen Implementierung des Problems.

Debugging Ebenso wie der Sudoku Solver gibt es hier mehrere Lösungen, die abgewägt werden können.

6.4 Vorgehen im Selbstversuch

Der Autor hatte vor der Bachelorarbeit keinerlei Vorkenntnisse in FP und Haskell. Um die Sprache zu lernen, wurde den offiziellen Empfehlungen der Haskell-Dokumentation gefolgt. Hierbei war das sogenannte read-eval-print-loop (REPL) besonders hilfreich. Die interaktive Programmierumgebung nimmt einzelne Nutzerinputs, evaluiert diese und gibt den Output an den Nutzer zurück. Im Kontext

der CIS 194 Vorlesung wurde das REPL ebenfalls zur Einführung in Haskell-Variablen verwendet. Personen ohne Vorkenntnisse können so schnell und barrierefrei mit der Syntax und den Ausdrücken in Haskell experimentieren und so praktische, erste Programmiererfahrungen machen. Die Arbeit mit der REPL begünstigt zudem durch die extrem schnelle Feedbackschleife eine aktivere Arbeitsweise, die ein schnelles Testen und Fehlschlagen ermöglicht.

A screenshot of a Windows PowerShell terminal window. The title bar reads 'Windows PowerShell'. The prompt is 'PS C:\>'. The user has entered 'ghci' in green text. The response is 'GHCi, version 9.4.8: https://www.haskell.org/ghc/ :? for help'. The user then enters 'ghci> putStrLn "Hello World"'. The output is 'Hello World'. The prompt returns to 'ghci>' with a cursor. A vertical scrollbar is visible on the right side of the terminal window.

```
PS C:\> ghci
GHCi, version 9.4.8: https://www.haskell.org/ghc/ :? for help
ghci> putStrLn "Hello World"
Hello World
ghci>
```

Abbildung 10: Verwendung von GHCi in der Windows PowerShell

Um Haskell Syntax und generelle Prinzipien zu lernen, wurden zunächst die ersten Aufgaben der CIS 194 Hausaufgaben bearbeitet.

Literaturverzeichnis

Berger, E. D., Hollenbeck, C., Maj, P., Vitek, O. & Vitek, J. (2019). On the impact of programming languages on code quality. *ACM Transactions on Programming Languages and Systems*, 41 (4), 21:1–21:24. Zugriff auf <https://doi.org/10.1145/3340571> doi: 10.1145/3340571

Chen, Y.-Y., Su, S.-W., Chen, L.-X., Liao, C.-H. & Yuan, S.-M. (2023). Effect of learning style on non-programmed computational thinking activities. *2023 IEEE 5th Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability*. doi: 10.3390/engproc2023055040

Commons, W. (2024). *File:tower of hanoi.jpeg — wikimedia commons, the free media repository*. Zugriff am 2025-03-14 auf https://commons.wikimedia.org/w/index.php?title=File:Tower_of_Hanoi.jpeg&oldid=969783102 (Online; accessed 23-March-2025)

Curricula analyse ergebnisse. (o.J.). Zugriff auf https://github.com/AnoukMartinez/BA_WS25/blob/main/Data/Hochschulkompass_Data.xlsx

Curzon, P. & Curzon, P. (2018). *Computational thinking: Die welt des algorithmischen denkens – in spielen, zaubertricks und rätseln*. doi: 10.1007/978-3-662-56774-6

(Destatis), S. B. (2024). *Studierende in mathematik, informatik, naturwissenschaft (mint) und technik-fächern*. Zugriff am 2025-03-22 auf <https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Bildung-Forschung-Kultur/Hochschulen/Tabellen/studierende-mint-faechern.html>

Felder, R. M. (1988, 01). Learning and teaching styles in engineering education. *Journal of Engineering Education Washington*, 78, 674-681.

Felder, R. M. & Soloman, B. (o. J.-a).

Felder, R. M. & Soloman, B. A. (o. J.-b). *Index of learning styles questionnaire*. Zugriff am 2025-03-08 auf <https://learningstyles.webtools.ncsu.edu/>

Felder, R. M. & Spurlin, J. E. (2005). Applications, reliability and validity of the index of learning styles.

First steps (how to learn haskell proper). (o.J.). Zugriff am 2025-03-11 auf <https://www.haskell.org/ghcup/steps/>

Heublein, U., Hutzsch, C. & Schmelzer, R. (2022). *Die entwicklung der studienabbruchquoten in deutschland* (Bericht). Deutsches Zentrum für Hochschul- und Wissenschaftsforschung (DZHW). Zugriff am 2025-02-21 auf https://www.dzhw.eu/pdf/pub_brief/dzhw_brief_05_2022.pdf

(o.J.). Zugriff auf <https://www.hochschulkompass.de/home.html>

Jung, C. (1921). *Psychologische typen*. Rascher.

Khine, M. (2018). *Computational thinking in the stem disciplines foundations and research highlights: Foundations and research highlights*. doi: 10.1007/978-3-319-93566-9

Kolb, D. (1984). *Experiential learning: experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall. Zugriff auf <http://www.learningfromexperience.com/images/uploads/process-of-experiential-learning.pdf>! (dateofdownload:31.05.2006)

Kumar, A. N. (2017). Learning styles of computer science i students. , 1-6. doi: 10.1109/FIE.2017.8190464

Liu, H.-C., Chen, H.-R., Yu, S.-C. & Shih, Y.-T. (2022). *Effect of learning computational thinking using board games in different learning styles on programming learning*. doi: 10.1007/978-3-031-15273-3_56

McDonald, C. (2018). Higher education computer science: A manual of practical approaches. *Cambridge International Law Journal*, 75-93. doi: 10.1007/978-3-319-98590-9

Michaelson, G. J. (2018). Programming paradigms and computational thinking.. Zugriff auf <https://api.semanticscholar.org/CorpusID:231747966>

Myers, I. B., McCaulley, M. H., Quenk, N. L. & Hammer, A. L. (1998). *Mbti manual: A guide to the development and use of the myers-briggs type indicator* (3rd Aufl.). Mountain View, CA: Consulting Psychologists Press.

Normak, K. (2017). *Overview of the four main programming paradigms*. Zugriff am 2025-02-21 auf https://homes.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. USA: Basic Books, Inc. doi: 10.1007/978-3-0348-5357-6

- Shute, V. J., Chen, S. & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*. doi: 10.1016/j.edurev.2017.09.003
- Stackoverflow. (2024). *Stack overflow developer survey 2024*. Zugriff auf <https://survey.stackoverflow.co/2024/technology>
- Wing, J. (2006, 03). Computational thinking. *Communications of the ACM*, 49, 33-35. doi: 10.1145/1118178.1118215
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*. doi: 10.1098/rsta.2008.0118
- Yorgey, B. (2013). *Cis 194: Introduction to haskell (spring 2013)*. Zugriff am 2025-03-11 auf <https://www.cis.upenn.edu/~cis1940/spring13/>
- Zywno, M. & Zywno, M. S. (2003). A contribution to validation of score meaning for felder- soloman's index of learning styles.
doi: 10.18260/1-2--12424