Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie

Kognitive Strategien von Novizen im Programmierunterricht

MASTERARBEIT

2018

Autorin
Amanda Folie

Begleitung
Prof. Dr. Carmen Zahn

Praxispartner
Hochschule für Angewandte Psychologie, Olten

Zusammenfassung

Der Bedarf an Programmierenden steigt vorzu. Auch die Nachfrage an Programmierkursen wird immer grösser. Programmieren ist eine komplexe Aufgabe, welche eine hohe kognitive Leistung fordert. Es dauert bis zu zehn Jahren, sich zum Experten in Programmierung zu entwickeln. Experten wenden bei der Programmierung kognitive Strategien an, um eine Aufgabe mit Erfolg zu lösen. Um den Programmierunterricht für Neulinge (Novizen) besser gestalten zu können, ist ein besseres Verständnis über kognitive Strategien notwendig. Zu diesem Zweck, wurden kognitive Strategien von Novizen in Rahmen einer sequentiellen Mixed-Methods-Studie exploriert und analysiert. Es wurde untersucht wie kognitive Strategien entstehen und in welcher Verbindung diese mit mentalen Modellen, der schulischen Leistung und Vorkenntnisse stehen. Interviews (n=9) und quantitative Resultate von 106 Novizen deuten an, dass vollständigere mentale Modelle die Anwendung konzeptueller Strategien erhöhen. Weiter weisen die Resultate auf einen Mangel an Strategien zum Programmverständnis hin. Weder die schulische Leistung, noch Vorkenntnisse zeigten eine Verbindung mit kognitiven Strategien. Gestaltungsvorschläge wurden abgeleitet.

Schlüsselwörter

Kognitive Strategien, Programmierung, Novizen, mentale Modelle, Vorkenntnisse, schulische Leistung

Abstract

Today, the need of programmers rises continually. As a result, the demand for programming courses is increasing. Programming is a very complex task and involves high cognitive performance. It takes up to ten years to become an expert. Prior studies have revealed that programming experts when writing a program use cognitive strategies to successfully solve the task. Therefore, to improve how programming is taught to novices, it is important to know more about cognitive strategies. For this reason, the results of a sequential mixed-method study are presented to explore and analyze the cognitive strategies used by programming novices. How cognitive strategies are learned and how they relate to mental models, academic performance and prior knowledge were examined. Interviews (n=9) and quantitative results from 106 programming novices imply that more complete mental models increase the use of conceptual cognitive strategies. The results further indicate a lack of program comprehension strategies in early stages of learning to program. Both academic performance and prior knowledge have not shown a relation to cognitive strategies. Proposals to optimize programming courses were made.

Keywords

cognitive strategies, programming, novice programmers, mental models, prior knowledge, academic performance

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit, ist das Erlernen von Programmiersprachen sehr gefragt. Robins, Rountree und Rountree (2003) schreiben, dass die Nachfrage an Programmierenden sowie auch das Interesse an der Programmierung in den letzten Jahren stark zugenommen hat. So zeigt auch die Erhebung vom Bundesamt für Statistik (2017, S. 19), dass bei höheren Fachschulen die Ausbildung in Software- und Applikationsentwicklung und -analyse an vierter Stelle der gewählten Ausbildungsfelder steht. Doch das Erlernen von Programmiersprachen ist ein sehr komplexer Prozess und stellt für Lernende eine grosse Herausforderung dar (Bringula, Tolentino, Manabat & Torres, 2012). Dies wird auch durch die überaus hohe Ausstiegsquote in den Programmierkursen ersichtlich (Robins et al., 2003). In der Literatur wird zwischen Experten und Novizen, im Englischen bekannt als *novice programmers,* unterschieden (Robins et al., 2003). Es benötigt, nach Winslow (1996, S. 18) bis hin zu zehn Jahren, um sich zum Experten in der Programmierung zu entwickeln.

1.1 Programmierung als Aufgabe

Die Programmierung wird in der Literatur als komplexe, problemlösende Aufgabe beschrieben (Rogalski & Samurçay, 1993, van Merriënboer & Kirschner, 2018). Dass die Aufgabe des Programmierens eine hohe kognitive Leistung erfordert, geht auch aus der Aufgabenbeschreibung von Rogalski und Samurçay (1993) klar hervor. In der untenstehenden Abbildung 1 ist anhand einer Schleife der Prozess dargestellt, wie dieser beim Lösen einer Programmieraufgabe von professionellen Programmierenden durchlaufen wird.

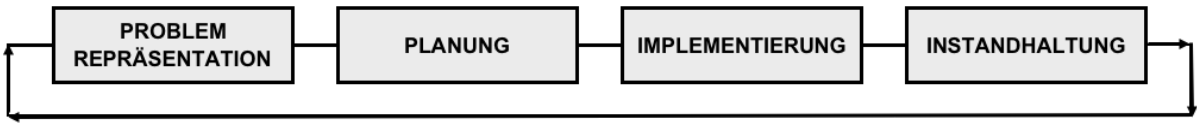


Abbildung 1. Ablauf einer Programmieraufgabe von professionellen Programmierenden (Rogalski & Samurçay, 1993, S. 8)

Diese Schleife gilt für sämtliche Programmieraufgaben, unabhängig deren Komplexität. Die vier beinhalteten Phasen können jedoch mehr oder weniger stark ausgeprägt sein, je nach Kontext. So weisen Rogalski und Samurçay (1993) darauf hin, dass zu Beispiel bei schulischen Übungsaufgaben die Phasen nicht im selben Umfang auftreten, wie bei üblichen Programmieraufgaben.

Zu Beginn einer Programmieraufgabe, muss die Problemstellung mental repräsentiert und verstanden werden (Rogalski & Samurçay, 1993). Verfügen Programmierende über mentale Modelle, so können diese die Problemstellung erfolgreich repräsentieren, denn Ormerod

(1990, S. 76f) besagt, „a mental model is constructed out of propositions to form an analogue of the real world representation of a problem“. Anschliessend wird in der Planungs-, oder auch Designphase, auf Basis der mentalen Repräsentation, die Lösung der Aufgabe ausgearbeitet, um diese in der nächsten Phase implementieren zu können (Rogalski & Samurçay, 1993). Wie komplex diese Phase ist, hängt gemäss Rogalski und Samurçay (1993) davon ab, wie komplex der Algorithmus und die Aufgabenstellung an sich ist. Die Implementierung beinhaltet das Verschriftlichen in einer gegebenen Programmiersprache und der gegebenen Technik (Rogalski & Samurçay, 1993). Nach der Implementierung folgt die Aufrechterhaltung des Programmes, welche das Testen und Fehlerbeheben beinhaltet und unter Umständen die ganze Lebensdauer eines Programms betreffen kann (Rogalski & Samurçay, 1993).

Dass die einzelnen Abläufe, dargestellt in Abbildung 1, nicht komplett getrennt voneinander betrachtet werden können und es nicht möglich ist, diese allein durch Wissen erfolgreich zu bewältigen, zeigen Rogalski und Samurçay (1993) auf:

> These programming sub-tasks have multiple interconnections and each requires various types of knowledge such as domain knowledge, design strategies, programming algorithms and methods, debugging, testing […]. It is clear that cognitive requirements are not uniform for all these tasks, and that training cannot be seen as a linear knowledge accumulation. (S. 8)

Vielmehr werden zum Programmieren, neben dem Wissen auch mentale Modelle sowie kognitive Strategien benötigt (Davies, 1993, Robins et al., 2003; Rogalski & Samurçay, 1993; Winslow, 1996). Arzarello, Chiappini, Lemut, Malara und Pellerey (1993) schreiben, dass beim Programmieren, das Lesen und das Schreiben unterschiedliche Bestandteile des Programmierens, mit unterschiedlichen Aktivitäten, darstellen. Während das Lesen, das Verstehen, Analysieren und Evaluieren eines bestehenden Programmes beinhaltet, werden beim Schreiben, Programme oder einzelne Programmteile entworfen, implementiert, getestet und Fehler behoben (Arzarello et al., 1993). Auch hier weisen Arzarello et al. (1993, S. 285) darauf hin, dass bereits beim Lesen eines Programmes, semantisches Wissen allein nicht ausreichend ist, sondern die Semantik auch beherrscht werden muss. Davies (1993) reklamiert, dass das Programmierwissen zwar als wichtig eingestuft wird, doch in der überzähligen Literatur nicht darauf eingegangen wird, welches Wissen notwendig ist und wie dieses letztendlich eingesetzt wird. Diverse Studien haben erwiesen, dass gerade bei Novizen oftmals vorhandenes Wissen nicht zielführend angewendet werden kann (Robins et al., 2003, Winslow, 1993). „Novice programrners know the syntax and semantics of individual

statements but they do not know how to combine these features into valid programs" beschreibt Winslow (1993, S. 17) und macht einmal mehr deutlich, dass die Anwendung von Strategien ein wichtiger Aspekt der Programmierung darstellt. Denn Winslow (1993) besagt, dass Experten unter Anderem, (a) über bessere taktische und strategische Fähigkeiten verfügen, (b) mentale Modelle zu ihrem Vorteil nutzen können, (c) lieber Algorithmen als spezifische Syntax einsetzen, (d) ihr gesamtes Wissen auch anwenden können. Novizen wiederum unterscheiden sich in diesen Punkten gegenüber den Experten, da diese oftmals weniger strukturiert vorgehen und Strategien vernachlässigen (Winslow, 1993).

1.2 Ausgangslage und Problemstellung

Für die Gestaltung des Programmierunterrichts sowie den dabei eingesetzten Lehrmitteln, ist es wichtig, die Inhalte geeignet für die Lernenden zu vermitteln. Im Programmierunterricht, werden Inhalte grösstenteils wissensbasiert vermittelt und die Vermittlung von Strategien werden vernachlässigt, stellen Robins et al. (2003) fest. Die Strategische Initiative Education Naturwissenschaft und Technik (EduNaT) der Fachhochschule Nordwestschweiz (FHNW) befasst sich in ihren Projekten mit dem Thema Bildung in den Bereichen Naturwissenschaft und Technik (FHNW, 2018a). Der Bildung in naturwissenschaftlichen und technischen Berufen ist eine hohe Bedeutung beizumessen, denn es hängt „der wirtschaftliche Erfolg und damit der hohe Lebensstandard in der Schweiz zu einem erheblichen Teil von naturwissenschaftlich-technischen Innovationen ab" (FHNW, 2018a). Im Rahmen des Forschungsprojektes der EduNat mit dem Titel *Software Teaching Environment for the Classroom* (STEC), beschäftigten sich in einer Zusammenarbeit, die Hochschulen für Life Sciences, Technik, Wirtschaft und Angewandte Psychologie mit dem Thema Programmierunterricht. Gemeinsam haben die unterschiedlichen Hochschulen eine Software entwickelt, welche zur didaktischen Unterstützung des Programmierunterrichts eingesetzt werden kann (FHNW, 2018a). Denn mit der entwickelten Software, können Studierende eine Aufzeichnung von der Lehrperson in Echtzeit vorprogrammierten Änderungen im Code nochmals abspielen und nachverfolgen. Da die Software keine Bildfolgen, sondern den geschriebenen Inhalt aufzeichnet, können die Lernenden zudem direkt den Code an einer beliebigen Stelle weiterentwickeln.

Im Zentrum des STEC-Projektes der EduNaT stand die Entwicklung einer Software zur didaktischen Unterstützung im Programmierunterricht. Im Anschluss an dieses Projekt soll nun im Rahmen der Master Thesis an der Hochschule für Angewandte Psychologie, der Fokus auf Programmierlernende gelegt werden. Denn um Programmierlernende bestmöglich ausbilden zu können ist wichtig, auch über Kenntnisse zu den lernspezifischen Fakto-

ren von Programmierlernenden zu verfügen. Es ist bekannt, dass während einige Novizen erfolgreich im Erlernen der Programmierung sind, andere weitaus weniger Erfolg aufweisen (Robins et al., 2003). Dieser Unterschied kann nicht einzig auf das Wissen hinsichtlich der Programmierung zurückzuführen sein, denn Robins et al. (2003, S. 165) gehen davon aus, dass "the most significant differences between effective and ineffective novices relate to strategies rather than knowledge". Robins et al. (2003) proklamieren, dass beim Unterrichten der Fokus nicht einzig auf das komplexe Endprodukt von Programmierwissen gelegt werden sollte, sondern auch erfolgreiche Handlungsweisen von Novizen fokussieren sollten. Dazu ist es notwendig, ein tieferes Verständnis von erfolgreichen und nicht erfolgreichen Novizen zu erlangen. Welche Strategien von erfolgreichen Novizen angewendet werden und in welchem Zusammenhang diese mit dem Vorwissen und mentalen Modellen stehen, wurde bisher noch nicht empirisch untersucht. (Robins et al., 2003). Mit der vorliegenden Master Thesis soll diese Lücke geschlossen werden. Im Zentrum dieser Arbeit stehen kognitive Strategien von Novizen. Es wird beleuchtet, welche kognitiven Strategien angewendet werden und zudem deren Zusammenhang mit der schulischen Leistung, Vorwissen und mentalen Modellen untersucht. Anschliessend werden aus den Resultaten Handlungsempfehlungen für den Programmierunterricht abgeleitet.

1.3 Aufbau des Berichts

Der Bericht ist insgesamt in fünf weitere Kapitel gegliedert und befasst sich zunächst, im zweiten Kapitel mit dem theoretischen Hintergrund. Dabei werden psychologische Konzepte und Begriffe im Bereich der Programmierung und des Lernens beschrieben, welche zum Verständnis der Arbeit grundlegend sind. Im dritten Kapitel wird die Leitfrage mit deren Unterfragestellungen erläutert und zu den nicht explorativen Unterfragestellungen jeweils Hypothesen abgeleitet. Im nachfolgenden vierten Kapitel wird die Methodik aufgezeigt, welche zur Beantwortung der Fragestellungen angewendet wurde. Nach dem Beschrieb der Methode werden im fünften Kapitel die gewonnenen Resultate dargestellt. Die Interpretation und Diskussion der Resultate und die Beantwortung der Fragestellung erfolgt nachfolgend im sechsten Kapitel. Ebenfalls im sechsten Kapitel werden abschliessend Handlungsempfehlungen für den Programmierunterricht aus den gewonnenen Erkenntnissen abgeleitet und einen Ausblick auf mögliche zukünftige Forschungsfragen wird gegeben.

2 Theoretischer Hintergrund

In den nachfolgenden Unterkapiteln werden Begriffe und psychologische Konzepte erläutert, welche für das weitere Verständnis von Bedeutung sind. Die in diesem Kapitel dargestellten Erkenntnisse stammen aus psychologischen Lehrbüchern sowie wissenschaftlichen Artikeln aus den Online-Datenbanken ScienceDirect und PsyARTICLES. Es wird zunächst auf die kognitiven Aspekte der Programmierung eingegangen. Darauffolgend werden kognitive Strategien und mentale Modelle beschrieben, welche bei der Programmierung von Bedeutung sind. Vor dem zusammenfassenden Fazit wird das Erlernen der Programmierung genauer beleuchtet und mit Modellen zum Lernen komplexer Inhalte in Bezug gebracht.

2.1 Kognition und Programmierung

Dass bei der Programmierung kognitive Prozesse eine wichtige Rolle innehaben, lässt sich bereits aus der in Kapitel 1.1 vorgestellten Aufgabenbeschreibung von Rogalski und Samurçay (1993) erkennen. Viele weitere Reviews und Studien zeigen auf, dass die Programmierung mit kognitiven Prozessen in Zusammenhang steht (vgl. Linn & Dalbey, 1989, Robins et al., 2003). Robins et al. (2003) beschreiben fünf ineinandergreifende Aspekte, welche bei der Programmierung involviert sind und gerade bei Novizen zu Schwierigkeiten führen können:

> These are: (1) general *orientation*, what programs are for and what can be done with them; (2) the *national machine*, a model of the computer as it relates to executing programs; (3) *notation*, the syntax and semantics of a particular programming language; (4) *structures*, that is, schemas/plans […]; (5) *pragmatics*, that is, the skills of planning, developing, testing, debugging, and so on. (S. 184)

Weiter differenzieren Linn und Dalbey (1989) drei übergeordneten Kategorien bezüglich kognitiven Voraussetzungen, nämlich (a) das Wissen einer Programmiersprache betreffend deren Eigenschaften, (b) die Fähigkeiten zur Planung und (c) die allgemeinen Fähigkeiten des Problemlösens.

2.1.1 Programmierwissen

Um eine Programmieraufgabe erfolgreich lösen zu können, ist es auch wichtig, die Eigenschaften und Elemente von Programmiersprachen zu kennen, so Linn & Dalbey (1989). Programmierwissen beinhaltet beispielsweise das Wissen zu Schleifen (IF, DO-WHILE), Variablen, Datentypen, Funktionen, Anweisungen (SWITCH, USE-CASE) und die Syntax.

Im Unterricht wird dieses Wissen oftmals so erlernt, dass Lernende vorhersagen müssen, was einzelne Elemente in einem Programm bewirken und bestehende Codes leicht verändern müssen (Linn & Dalbey, 1989). So müssen Novizen im Unterricht beispielsweise den Inhalt einer Schleife leicht verändern, um deren Verhalten zu beeinflussen. Robins et al. (2003, S. 140) machen zudem auf Weiteres notwendiges Wissen in Bezug auf die Programmierung aufmerksam, denn sie beschreiben, "programming ability must rest on a foundation of knowledge about computers, a programming language or languages, programming tools and resources, and ideally theory and formal methods". Aber das Programmierwissen allein ist nicht ausreichend, denn somit können lediglich einzelne Zeilen manipuliert, nicht aber ein funktionierendes Programm erstellt werden (Linn & Dalbey, 1989). Die Aufgabe der Programmierung beinhaltet, gemäss Linn und Dalbey (1989), zudem kognitive Prozesse und Strategien, welche beim Lösen von Problemen bedeutend sind.

2.1.2 Programmierung als problemlösende Aufgabenstellung

Brooks (1990) hält fest, dass es bei der Programmierung nicht darum geht einen Code zu produzieren der eine gewünschte Aktion zur Folge hat, denn viel mehr ist es das Ziel ein programmspezifisches Problem zu lösen. Ein Problem umfasst die vier übergeordneten Eigenschaften (a) die Ausgangslage, (b) das angestrebte Ziel, (c) die möglichen Handlungen, welche von der Ausgangslage zu angestrebten Ziel führen und (d) die Restriktionen, welche mit den Handlungen verbunden sind (Ormerod, 1990). Ormerod (1990) besagt, dass die Programmierung als eine problemlösende Aktivität angesehen werden kann und führt die Aufgabe der Programmierung auf die übergeordneten Eigenschaften eines Problems wie folgt zurück:

> Programming may be seen as a problem-solving activity, where the initial state is the problem for which a program is required, and the goal state is both the solution which the program can calculate, and also the program itself. The operators consist of the syntactic and semantic features of the language and the cognitive skills of the programmer, and the operator restrictions are imposed by limitations of the language and the problem description. (S. 71f)

Viele Studien zu Unterschieden von Novizen und Experten haben aufgezeigt, dass gerade Unterschiede bestehen betreffend den Aspekten beim Lösen von Problemen, der Art wie Probleme repräsentiert werden und den angewendeten Strategien zur Problemlösung (Ormerod, 1990). „Programming novices work forwards, writing a program line by line, whereas

experts work backwards, breaking the program goal into modular units", nennt Ormerod (1990, S. 72) als ein Beispiel. Studien zu den kognitiven Strategien werden des Weiteren im Kapitel 2.3 beschrieben.

2.1.3 Programmverständnis und Programmgenerierung

Es sind weitgehend zwei unterschiedliche Arten von Studien zu finden, wobei sich die einen Studien auf das Programmverständnis fokussieren, während für andere Studien die Programmgenerierung zentral ist (Robins et al., 2003). Robins et al. (2003) beschreiben, dass bei den Studien zum Programmverständnis, die Teilnehmenden einen Text erhalten und das Verständnis zu diesem demonstrieren sollen. Anders wiederum ist es bei Studien mit Fokus Programmgenerierung, bei welchen Teilnehmende aufgefordert werden, ein Teil eines Programmes selbstständig zu programmieren, um eine Aufgabestellung oder eine Problemstellung zu lösen (Robins et al., 2003). Es liegen vermehrt Studien und Ergebnisse zur Programmverständnis vor. Robins et al. (2003) vermutet den Grund darin, dass Aufgaben zum Programmverständnis eingeschränkter sind und das Verhalten der Teilnehmenden somit besser interpretiert und beschrieben werden kann. Es muss jedoch beachtet werden, dass diese Aktivitäten nicht strikt getrennt angesehen werden können, da die Fähigkeit ein Programm zu schreiben mit dem Verständnis eines Programms zusammenhängt (Winslow, 1996).

2.1.4 Weitere Aspekte

In den veröffentlichten psychologischen Studien im Gebiet der Programmierung, wurde der Fokus insbesondere auf die Repräsentation des Wissens und problemlösende Strategien gelegt (Robins et al., 2003). Somit erhalten gerade mentale Modelle und kognitive Strategien eine bedeutende Rolle für psychologische Studien im Bereich der Programmierung. Definitionen und empirische Erkenntnisse zu den mentalen Modellen und kognitiven Strategien bei der Programmierung werden in den nachfolgenden Kapiteln 2.2 und 2.3 im Detail beschrieben.

2.2 Mentale Modelle

Das Programmieren ist keine natürliche Verhaltensweise eines Menschen und weitaus komplexer zu erlernen als eine Landessprache (van der Veer, 1993). Denn laut van der Veer (1993), müssen bei der Programmierung Analysen einer gegebenen Problemstellung erarbeitet sowie Synthesen einer Lösung entwickelt werden. Hierbei sind mentale Modelle von Bedeutung, welche eine Art mentale Repräsentation darstellen, welche von einer Per-

son individuell entwickelt werden, um die Komplexität des Materials bewältigen zu können (van der Veer, 1993).

In seinem Buch über mentale Modelle beschäftigt sich Johnson-Laird (1983) zu Beginn mit dem Wesen der Erklärung. Dabei besagt Johnson-Laird (1983), dass eine Erklärung immer auf dem Verständnis der jeweiligen Sache basiert. Denn wenn man etwas nicht versteht, so kann man es auch nicht erklären. Weiter schreibt Johnson-Laird (1983), dass das Verständnis wiederum von dem vorhandenen Wissen und dem Glauben stammt:

> If you know what causes a phenomenon, what results from it, how to influence, control, initiate, or prevent it, how it relates to other states of affairs or how it resembles them, how to predict its onset and course, what its internal or underlying 'structure' is, then to some extent you understand it. (S. 2)

Liegt dieses Verständnis über ein Phänomen vor, verfügt man über eine mentale Repräsentation, welche als Modell dient, bei welchem Johnson-Laird (1983) von einem funktionierenden Modell des Phänomens spricht. Um ein besseres Verständnis über mentale Modelle zu erlangen ist es wichtig, vorab über ein grundlegendes Verständnis zu Modellen im Allgemeinen zu verfügen, so Dutke (1994). Eine umfassende Definition eines Modells mit Berücksichtigung aller bedeutenden Aspekte, liefert dazu Oberquelle (1984):

> A model is a communicable description
> - of a certain aspect (the view)
> - of a section of reality (the system)
> - at some level (of abstraction or detail)
> - as perceived by a human being (model builder)
> - which is to serve the purposes of its users. (S. 27)

Die Bezeichnung mentales Modell kommt daher, dass das Model lediglich gedanklich und nicht gegenständlich vorliegt (Dutke, 1994). Ormerod (1990) definiert, dass ein mentales Model in seinem eigentlichen Sinne, eine mentale Repräsentation eines Problemfeldes darstellt. Auch Dutke (1994) beschreibt mentale Modelle als Ausdruck des Verstehens eines Ausschnitts der realen Welt. Mentale Modelle sind individuell, so wie sie der jeweiligen Person dienlich sind gestaltet und können ihre eigenen Schwerpunkte aufweisen (Dutke, 1994). Manche mentalen Modelle sind stärker verstehensorientiert, andere wiederum eher handlungsorientiert, erklärt Dutke (1994). Wichtig zu beachten ist, dass mentale Modelle weder zwingend korrekt noch vollständig sind (vgl. Dutke, 1994, Johnson-Laird, 1983). Van Merriënboer und Kirschner (2018) besagen, dass mentale Modelle Aufgabenlösenden da-

bei helfen die Domäne der jeweiligen Aufgabe zu verstehen, Schlüsse über die Domäne ziehen zu können, Erklärungen zu finden und Vorhersagen zu treffen. Zudem schreiben van Merriënboer und Kirschner (2018) mentalen Modellen eine wichtige Rolle beim Lösen von Problemen und der Entscheidungsfindung zu. Mentale Modelle sind gerade auch im Lernkontext der Programmierung von Bedeutung, denn Ormerod (1990) besagt, dass Lernende beim Lösen von Problemen die Lösung in Form eines Schemas in eine neue, ähnliche Problemstellung übertragen und wo sinnvoll, anwenden können.

2.2.1 Mentale Modelle zur Aufgabendomäne

Mentale Modelle repräsentieren, gemäss van Merriënboer und Kirschner (2018), wie eine Aufgabendomäne organisiert ist. Es können drei Arten von Submodellen unterschieden werden, welche das mentale Modell bilden. Van Merriënboer und Kirschner (2018) differenzieren dabei konzeptuelle, strukturelle und kausale Modelle.

2.2.1.1 Konzeptuelle Modelle

Konzeptuelle Modelle befassen sich mit der Antwort auf die Frage was etwas genau ist, in Form von Objekten, Ereignissen und weiteren Einheiten anhand deren Charakteristiken (van Merriënboer & Kirschner, 2018). Solche Konzepte ermöglichen konkrete Objekte einer spezifischen Klasse zuzuordnen. Dies veranschaulichen van Merriënboer und Kirschner (2018) an dem Beispiel eines Computers, welcher beispielsweise nach dessen Art (zum Beispiel Laptop), dessen Farbe (schwarz), dessen Prozessor (Intel Core i7), oder dessen Betriebssystem (Windows 10) und weiteren Aspekten klassifizieren. Betreffend einer spezifischen Aufgabe sind gemäss, van Merriënboer und Kirschner (2018), manche Konzepte jeweils nützlicher als andere. Für einen Programmierer ist es beispielsweise notwendig ein Konzept dazu zu haben, ob das Programm auf einem Tablet oder Laptop wiedergegeben wird. Die Farbe des Geräts jedoch, ist für das Lösen einer Programmieraufgabe unerheblich. Auch zu den konzeptionellen Modellen gehören Konzepte bezüglich Beziehungen zwischen den einzelnen Objekten. Van Merriënboer und Kirschner (2018) unterscheiden dabei die Beziehungen, bei welchen das Objekt als Teil einer Objektgruppe, auch Kategorie, zugeordnet oder bei welcher ein Objekt als Bestandteil eines anderen Objekts untergeordnet wird.

2.2.1.2 Strukturelle Modelle

Bei strukturellen Modellen ist die zeitliche und örtliche Verankerung zentral, in welcher die Konzepte eingeordnet werden können, beschreiben van Merriënboer und Kirschner (2018). Daraus ergibt sich eine Art Plan, welcher es ermöglicht Zusammenhänge und Abläufe zu

verstehen und Vorhersagen für zukünftige Ereignisse treffen zu können (van Merriënboer & Kirschner, 2018). In der Programmierung können sich solche Pläne, laut van Merriënboer und Kirschner (2018), auf das Grundgerüst eines Programmes, wie beispielsweise deren Kopfzeile, Deklarationen, Prozeduren und Hauptprogramm beziehen:

> These plans are related to less abstract plans providing a general representation of basic programming structures such as procedures, looping structures, and decision structures. These, in turn, are related to concrete plans providing a representation of structures that are close to the actual programming codes, such as specific templates for looping structures (e.g., WHILE-loops, FOR-loops, REPEAT-UNTIL-loops), conditions (e.g., IF-THEN, CASE), and so on. (S. 190)

Die Antwort darauf wie etwas aufgebaut und organisiert ist, liefern somit strukturelle Modelle (van Merriënboer & Kirschner, 2018).

2.2.1.3 Kausale Modelle

Mit kausalen Modellen stehen Auswirkungen und natürliche Beziehungsprozesse zwischen Konzepten im Mittelpunkt, schreiben van Merriënboer und Kirschner (2018). Daraus werden Prinzipien gebildet, welche Antwort darauf geben, wie etwas funktioniert oder weshalb etwas nicht funktioniert (van Merriënboer & Kirschner, 2018). Solche Auswirkungen und natürliche Beziehungsprozesse zeichnen sich dadurch aus, dass eine Veränderung immer auch eine weitere Veränderung mit sich bringt, erklären van Merriënboer und Kirschner (2018). Wenn man bei einer FOR-Schleife beispielsweise den Inhalt der Variable ändert, welche angibt unter welcher Bedingung die Schleife durchlaufen wird, so hat dies zur Folge, dass die Schleife entweder nicht, weniger, mehr, oder endlos durchlaufen wird. Durch kausale Modelle können somit Vorhersagen für Verhaltensänderungen, Rückschlüsse und Interpretationen gemacht werden, was auch dazu beiträgt bei unerwünschtem Verhalten oder auch bei Fehlermeldungen die Ursache aufzudecken (van Merriënboer & Kirschner, 2018). Wird somit eine programmierte Schleife endlos ausgeführt, aufgrund dessen ein Programm nicht lauffähig ist, kann der Programmierer dies auf die ursächliche Variable zurückführen.

2.3 Kognitive Strategien

Beim Lösen einer Programmieraufgabe stösst man immer wieder auf neue Problemstellungen, welche einer früheren Problemstellung ähnlich sind, aber Aspekte enthält, welche für die programmierende Person neu sind und übliche Routinen nicht mehr zur Lösung beitragen. An dieser Stelle kommt es zur Anwendung kognitiver Strategien, proklamieren van Merriënboer und Kirschner (2018). Als erstes soll der Unterschied zwischen kognitiven Stra-

tegien und Programmierwissen (siehe Kapitel 2.1.1) abgegrenzt werden. Sprechen wir von Programmierwissen, ist das rein deklarative Wissen hinsichtlich der Programmierung oder einer Programmiersprache gemeint (Davies, 1993). Dieses Programmierwissen kann beispielsweise bedeuten, dass die Person fähig ist zu beschreiben, wie eine IF-Schleife funktioniert. Hingegen handelt es sich bei kognitiven Strategien, gemäss Davies (1993), auf die Art und Weise, wie Wissen genutzt und angewandt wird. Dies wäre beispielsweise, die IF-Schleife in einem Programm angemessen einzusetzen. Mittels kognitiver Strategien kann Wissen somit angewendet werden und Aufgaben möglichst effizient angegangen werden, denn „strategies involve actions, goals, and some notion of optimality: Intuitively, a strategy is the idea of an agent about the best way to act in order to reach a goal" definieren van Dijk und Kintsch (1983, S. 64f). Bei dem Verständnis von van Dijk und Kintsch (1983) bauen kognitive Strategien auf mentalen Modellen auf und beinhalten hohe Ansprüche der Informationsverarbeitung. Dies verdeutlicht sich, wenn man beachtet, welche kognitiven Aktivitäten, dargestellt in Abbildung 2, allein schon in der Planungsphase einer Programmieraufgabe involviert sind.

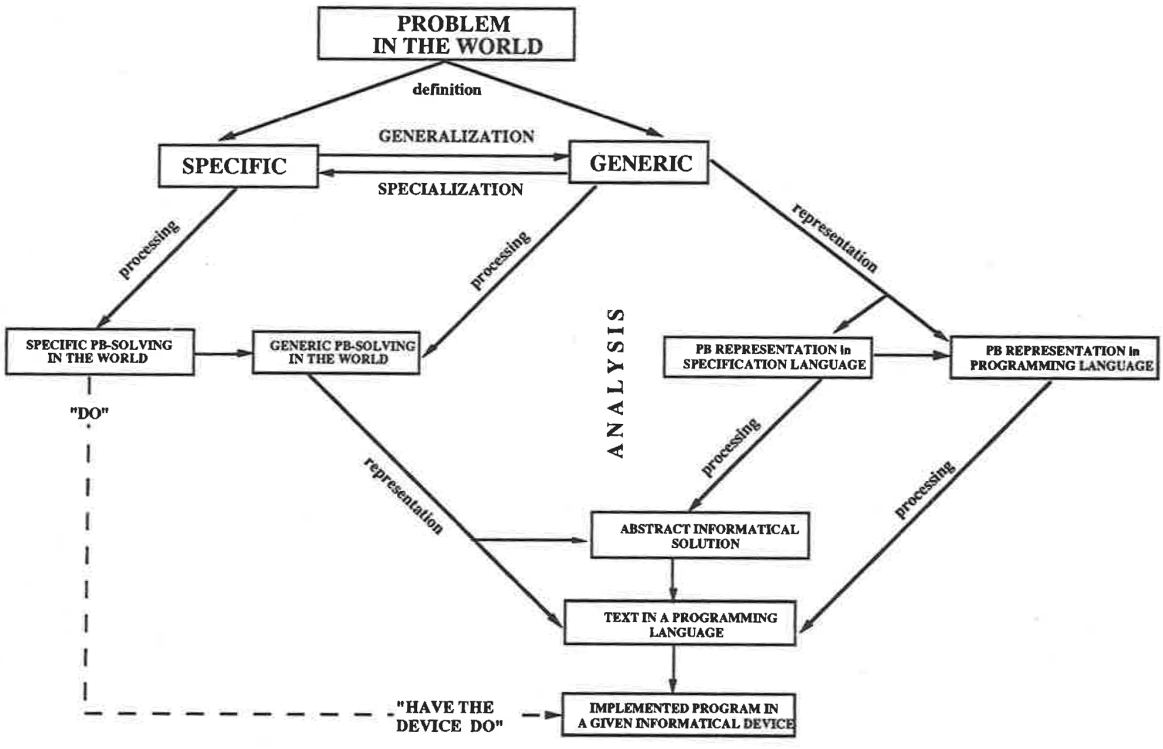


Abbildung 2. Darstellung kognitiver Aktivitäten während der Planungsphase einer Programmieraufgabe von Rogalski & Samurçay, 1993. In E. Lemut, B. du Boulay & G. Dettori (Eds.), Cognitive Models and Intelligent Environments for Learning Programming. (NATO ASI Series, Series F: Computer and Systems Sciences, Vol 111, pp. 6-19). Springer: Berlin, Heidelberg.

Es wird dabei aufgezeigt, dass bei der Phase Design die zwei verschiedenen Dimensionen Repräsentation und Verarbeitung unterschieden werden können (Rogalski & Samurçay,

1993). Weiter beschreiben Rogalski und Samurçay (1993), dass es zwei Wege gibt, das in der vorhergehenden Phase repräsentiere Problem der realen Welt, in einer Programmiersprache zu implementieren. So kann das Problem entweder zuerst in der entsprechenden Domäne gelöst und in Programmcode übersetzt werden, oder die Aufgabestellung kann so angegangen werden, dass das Problem direkt in einer Programmiersprache oder einem Pseudocode repräsentiert und anschliessend direkt im Kontext der Programmierung gelöst wird (Rogalski & Samurçay, 1993). Ein Pseudocode ist die Verschriftlichung des Programmablaufs in einer natürlichen Sprache. Dabei wird noch keinen Code in einer Programmiersprache geschrieben. Rogalski und Samurçay (1993) weisen darauf hin, dass bei realen Programmieraufgaben die Strategien nicht in zuerst Repräsentieren oder zuerst Verarbeiten eingeteilt werden können, da beides angewendet wird. Während dem Lösen einer Programmieraufgabe können verschiedene Strategien angewendet werden.

2.3.1 Konzeptuelle versus sequentielle Strategien

Die Unterscheidung von kognitiven Strategien sieht Schwank (1993) zwischen konzeptuellen und sequentiellen Strategien. In Abbildung 3 werden die beiden unterschiedlichen Arten kognitiver Strategien in einer grafischen Darstellung einander zusammengefasst gegenübergestellt.

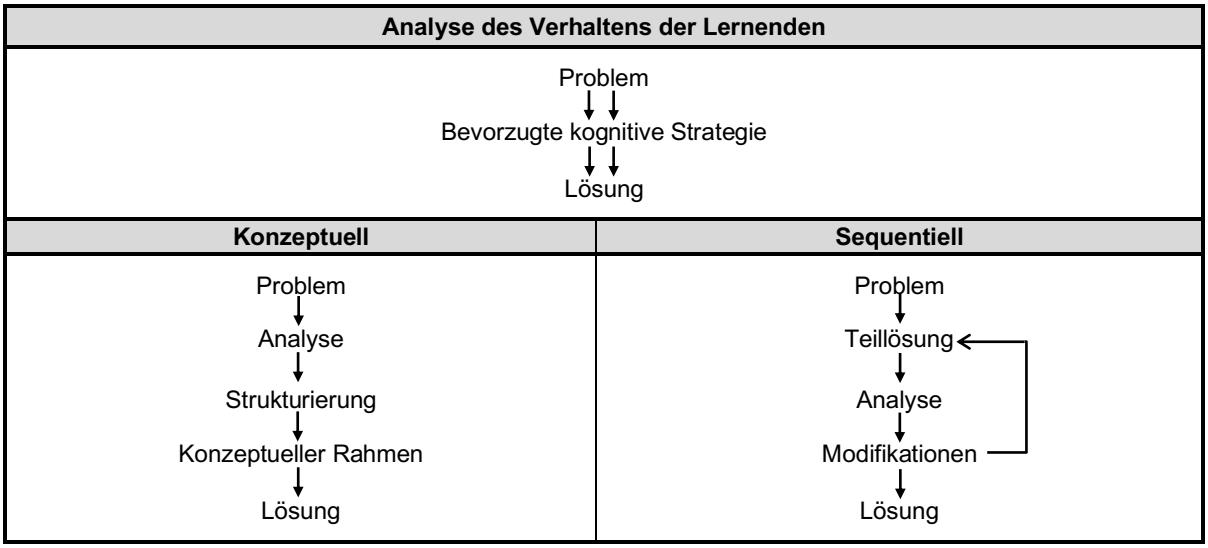


Abbildung 3. *Grafische Gegenüberstellung konzeptueller und sequentieller Strategien nach Schwank (1993, S. 254)*

Verwenden Programmierlernende konzeptuelle Strategien, so beginnen diese mit der Analyse einer Problemstellung, strukturieren diese und versuchen, mit Hilfe der Vorkenntnisse, einen konzeptuellen Rahmen zu entwickeln, beschreibt Schwank (1993). Programmierlernende, welche sequentielle Strategien anwenden, gehen hingegen anders vor. Diese be-

ginnen mit einer ersten Lösung, bevor sie ihre Gedanken komplett strukturiert und im Detail ausgearbeitet haben (Schwank, 1993). Dabei gehen die Programmierlernenden gemäss Schwank (1993) so vor, dass die Ideen jeweils im Dialog mit der Materie entwickelt werden und die finale Lösung entsteht durch das Analysieren und Modifizieren von Teillösungen.

2.3.2 Strategien zum Programmverständnis nach Pennington (1987)

Pennington (1987) hat an 40 professionellen Programmierenden untersucht, welche kognitive Strategien zum Programmverständnis verwendet werden. Dazu wurde den Teilnehmenden ein bestehender Programmcode gegeben, welcher sie lesen und anpassen sollten. Dabei wurden die Teilnehmenden angehalten ihre Gedankengänge aufzusagen, sodass die Strategien entnommen werden konnten. Anhand der aufgetretenen Fehler bei der Programmierung wurde anschliessend deklariert, ob das Vorgehen der jeweiligen Person effizient war oder nicht. Daraus konnte Pennington (1987) die folgenden drei Strategien herausfiltern, welche auch in weiteren Studien immer wieder Beachtung finden (vgl. Corritore & Wiedenbeck, 2001, Robins et al., 2003).

2.3.2.1 Programmebene-Verständnis-Strategie

Die Programmebene-Verständnis-Strategie zeichnet sich dadurch aus, dass beinahe gar keine Verbindungen, also Querverweise zu anderen Objekten und Vorgängen während der Programmierung gemacht werden (Pennington, 1987). Pennington (1987) hält fest, dass bei dieser Strategie nach Plan vorgegangen wird und sich diese fast ausschliesslich auf die Programmebene fokussiert. In der Studie von Pennington (1987) konnte diese Strategie mit einer schlechteren Programmierleistung in Verbindung gebracht werden. Pennington (1987) führt dies darauf zurück, dass keine Verbindung zur Domäne hergestellt wird, was die Prozesse erleichtern würde und zum Verständnis von Kausalitäten in Abläufen beiträgt.

2.3.2.2 Aufgabendomäne-Verständnis-Strategie

Im Gegensatz dazu, ist die Aufgabendomäne-Verständnis-Strategie überhäuft mit Verbindungen, sodass laut Pennington (1987), allein das Lesen einer Variable dazu verleitet, Hypothesen und Fantasien zu Konstrukten aus der realen Welt aufzustellen. Auch diese Strategie zeigte keine besonders guten Resultate beim Programmverständnis, denn die Hypothesen häuften sich lediglich vorzu und haben sich nur selten bewahrheitet (Pennington, 1987).

2.3.2.3 Querverweis-Verständnis-Strategie

Die Strategie mit den besten Ergebnissen war die Querverweis-Verständnis-Strategie, bei welcher immer wieder Verbindungen zwischen der Programmebene und der Aufgabendo-

mäne hergestellt wurden, die Verbindung jedoch vorgängig jeweils von einem Stimulus initiiert wurde (Pennington, 1987). Dabei wurde jeweils der ursprüngliche Plan beachtet und es wurde nach Erklärungen gesucht, weshalb ein Ereignis eintrifft, schreibt Pennington (1987).

2.3.3 Top-Down-Strategien versus Bottom-Up-Strategien

„The direction of comprehension concerns the programmer's strategic approach to program comprehension, which may be top-down, bottom-up, or a combination of the two" schreiben Corritore und Wiedenbeck (2001). Bei der Bottom-Up-Strategie werden einzelne Programmzeilen enkodiert und im Kurzzeitgedächtnis gespeichert, um die einzelnen Zeilen anschliessend zusammenfassend repräsentieren zu können (Corritore & Wiedenbeck, 2001). Anschliessend, erklären Corritore und Wiedenbeck (2001), werden die zusammengefassten Einheiten im Langzeitgedächtnis auf Basis des vorhandenen Programmierwissens verstanden. Bei der Top-Down-Strategie stellen die Programmierenden, basierend auf der Programmfunktion und weiteren Informationen ausserhalb des Programms, Hypothesen auf und suchen im Programmcode nach Indikatoren um die Hypothese zu bestätigen oder zu verwerfen (Corritore & Wiedenbeck, 2001).

In der Literatur beschriebene Studien unterscheiden zum Teil auch das Vorgehen der prozeduralen und der objekt-orientierten Programmierung (Robins et al., 2003). Corritore und Wiedenbeck (2001) untersuchten an 30 professionellen Programmierenden, ob es betreffend angewendeten Strategien zum Programmverständnis Unterschiede zwischen Experten der prozeduralen und der objektorientierten Programmierung gibt. Dabei konnte festgestellt werden, dass Experten der objektorientierten Programmierung zu einer starken Top-Down-Strategie tendierten, während Experten der prozeduralen Programmierung vermehrt mit einer Bottom-Up-Strategie gearbeitet haben (Corritore & Wiedenbeck, 2001). Corritore und Wiedenbeck (2001) halten jedoch fest, dass diese Ergebnisse einzig auf das Programmverständnis bezogen werden kann.

2.3.4 Trial-and-Error-Strategie

Die Trial-and-Error-Strategie beschreibt ein Vorgehen, dem kein spezifischer Plan vorausgeht, beschreiben Carbone, Hurst, Mitchell und Gunstone (2009). Azarello et al. (1993) proklamieren, dass Novizen beim Lösen einer Aufgabestellung fähig sein müssen, die Aspekte der Planung und der Implementierung zu separieren. Bei Novizen, welche diese Trennung nicht vornehmen, beobachteten Azarello et. al, (1993), dass verschiedene Möglichkeiten ausprobiert werden. Azarello et al. (1993) sehen die Trial-and-Error-Strategie als eine Vor-

hergehensweise, welche dann angewendet wird, wenn kein Plan zur Problemlösung entwickelt werden konnte:

> Not always novices are able to make this transition. At opposite extremes we find those who keep using a performing approach based on trial-and-error methods, and those who' at a higher cognitive level, make a global plan and/or revision, looking at the program as a whole from outside. (S. 290)

Wird die Problemstellung nicht verstanden, so beginnen Programmierlernende direkt mit der Programmierung ohne systematische Strategie die zur Lösung beiträgt, programmieren planlos und stützen sich auf die Rückmeldungen des Systems (Carbone et al., 2009). Carbone et al. (2009) führen diese Vorhergehensweise auf den Mangel an Fähigkeiten hinsichtlich des Problemlösens zurück. Dass im Lernprozess der Programmierung die Verwendung von Trial-and-Error-Strategien auch förderlich sein können, zeigen Salleh, Shukur und Judi (2013) auf. Denn um Programme schreiben zu können, ist die Verwendung einer Entwicklungssoftware notwendig. Auch die Bedienung einer solchen Software benötigt Fähigkeiten und muss erlernt werden. Hierbei wird die Trial-and-Error-Strategie positiv konnotiert, denn “skills in using programming tools could be gained only through informal learning, trial and error process, using either Internet sources, help functions, or from insertion notes supplied by the software” schreiben Salleh et al. (2013).

2.3.5 Kognitive Strategien und Novizen

Zu kognitiven Strategien findet sich viel Literatur, gerade aus den Jahren 1980 bis 2005, welche unterschiedliche Strategien benennen. Viele der beschriebenen Strategien sind jedoch ähnlich, oder bauen auf weiteren Erkenntnissen auf (vgl. Corritore und Wiedenbeck, 2001, Pennington, 1987). Es lässt sich jedoch festhalten, dass die kognitiven Strategien in den meisten Fällen an Experten erforscht wurden (Robins et al., 2003). In einem Review zeigt Winslow (1996) jedoch auf, dass zwischen Novizen und Experten grosse Unterschiede bestehen. So deutet Winslow (1996) insbesondere darauf hin, dass Novizen den Code von Zeile zu Zeile abarbeiten, ihr Wissen nicht adäquat einsetzen können und Strategien bei der Programmierung vernachlässigen.

2.4 Erlernen der Programmierung

Aus den vorhergehenden Kapiteln geht hervor, dass das Erlernen der Programmierung ein komplexer Prozess darstellt. Luxton-Reilly (2016) macht darauf aufmerksam, dass die hohen Erwartungen an Programmierlernende das Erlernen der Programmierung und das Erreichen einer guten schulischen Leistung erschweren. Damit begründet Luxton-Reilly (2016) auch die hohen Ausstiegsquoten von Programmierkursen. Dass die Programmierung komplexe, kognitive Fähigkeiten erfordert, schreibt van Merriënboer (1997), welcher sich in seinen Büchern und Artikeln mit der Thematik des Erlernens kognitiver Fähigkeiten befasst.

2.4.1 Cognitive Load Theory

Ein Aspekt, welcher das Erlernen komplexer, kognitiver Fähigkeiten erschwert, ist die hohe kognitive Belastung. Sowohl das Erlernen, als auch die Leistung komplexer Fähigkeiten, sind stark gebunden an die limitierte Prozesskapazität des menschlichen Gehirns (van Merriënboer, 1997). Sweller, van Merriënboer und Paas (1998) beschreiben, dass bei Instruktionen das primäre Ziel für das Abrufen des nützlichen mentalen Modelles zur Lösung der Problemstellung dient. Obwohl mentale Modelle im Langzeitgedächtnis gespeichert sind, müssen Informationen aus dem Arbeitsgedächtnis verarbeitet werden, um das mentale Modell zu konstruieren (Sweller et al., 1998). Dies führt laut Sweller et al. (1998) zu einer kognitiven Belastung. In der kognitiven Belastung beim Programmieren, sehen Matthews, Hin und Choo (2015) den Grund darin, dass das Erlernen der Programmierung für Novizen so schwierig ist:

> One of the reasons for this is the cognitive load in programming learning materials. Learners are bombarded with various textual and visual representations to learn the concepts of programming. These include the flow of control structure (e.g., for loop, while loop, and IF statements) and programming syntax and codes; as well as the key points and explanation, flowcharts, algorithms and/or pseudo code used to explain the concepts and programming codes. (S. 238)

Carbone et al. (2009) zeigen auf, dass die Kapazität des Gehirns und die Motivation personenbezogene Faktoren darstellen, welche das Erlernen der Programmierung beeinflussen.

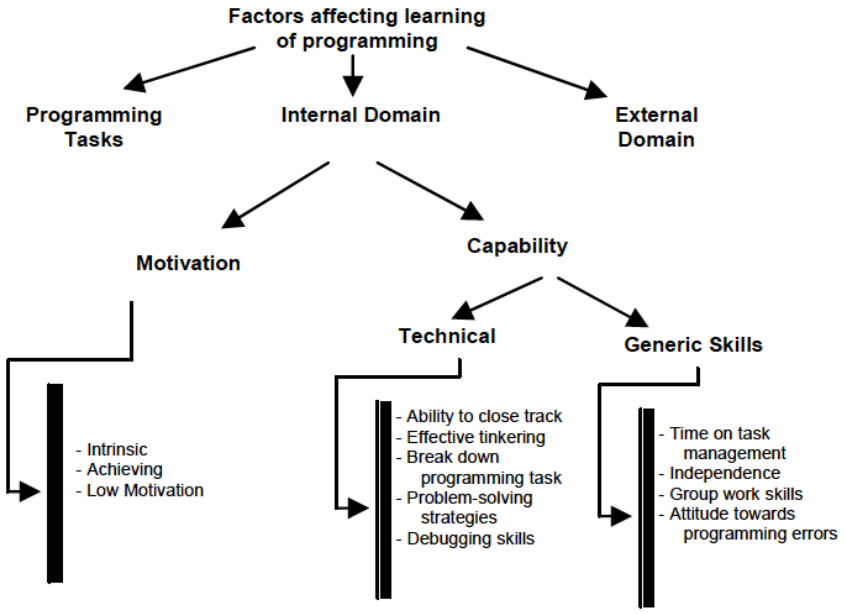


Abbildung 4. Darstellung personenbezogener Aspekte beim Erlernen der Programmierung von Carbone, A., Hurst, J.R., Mitchell, I.J., & Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. ACE '09, 1-10.

Wie Abbildung 4 aufzeigt, hat die Kapazität des Gehirns beim Erlernen der Programmierung einen Einfluss auf allgemeine sowie technische Fähigkeiten (Carbone et al., 2009). Diese beeinflussen somit auch die Fähigkeiten, welche mit den Strategien zum Problemlösen verwendet werden (vgl. Carbone et al., 2009). Damit es zu keiner kognitiven Überbelastung kommt, empfiehlt van Merriënboer (1997), nicht alle notwendigen Fähigkeiten zusammen zu vermitteln und kognitive Belastungen, welche nicht relevant sind, zu vermindern.

2.4.2 Liveprogramming als Methode im Programmierunterricht

Eine häufig angewendete Unterrichtsmethode im Programmierunterricht ist das *Liveprogramming*. Beim Liveprogramming werden die Prozesse der Planung und Implementierung eines Projekts vor der gesamten Klasse, während der Unterrichtsstunde in Echtzeit vorprogrammiert (Paxton, 2002). Die Lernenden haben so die Möglichkeit, gleichzeitig den Programmcode mitzuschreiben. In einer Experimentalstudie mit 146 Teilnehmenden, erforschte Rubin (2013) die Effizienz dieser Methode. Die Resultate zeigten, dass Liveprogramming eine sehr effiziente Methode darstellt, welche Lernende besser auf grössere Aufgabenstellungen vorbereitet. Auch Paxton (2002) konnte anhand eines Fragebogens bei 30 Programmierenden erheben, dass als positiv wahrgenommen wurde, wie beispielsweise, dass (a) Lernende aktiv in die Lektionen miteingebunden werden, (b) der Designprozess inklusive Überführung in funktionierenden Programmcode illustriert wird und (c) Experten beim Lösen von Problemen beobachtet werden können. Aber das Liveprogramming stellt auch

Schwierigkeiten dar. So empfinden manche Lernende das gemeinsame Programmieren in der Klasse als zu langsam, während andere Lernende das Tempo wiederum als zu schnell wahrnehmen (Paxton, 2002). Auch Fehler im Programmcode des Dozierenden, repetitive Programmteile, oder das Nachverfolgen von Änderungen gegenüber vorherigen Versionen stellen laut Paxton (2002) eine Hürde dar. Als Empfehlung für die Methode Liveprogramming schlägt Paxton (2002) vor, grössere Projekte mit der Klasse zu realisieren und zusätzlich online Materialien in den Unterricht zu integrieren.

2.4.3 Die Rolle der Vorkenntnisse

Um die Programmierung erfolgreich erlernen zu können, sind verschiedene Vorkenntnisse notwendig (vgl. Linn & Dalbey, 1989). Brooks (1990) deutet darauf hin, dass einerseits Wissen über die jeweilige Domäne ein effizientes Programmieren begünstigt, aber auch das Vorwissen betreffend Softwarearchitekturen wichtig ist. Auch Schwank (1993) bringt die Vorkenntnisse in Verbindung mit kognitiven Strategien, denn sie besagt, dass die Entwicklung eines konzeptuellen Rahmens, durch Anwendung der Vorkenntnisse ermöglicht wird.

2.5 Fazit

Das Erlernen der Programmierung ist eine sehr komplexe Aufgabe, denn die Programmierung beinhaltet viele kognitive Prozesse, wobei Wissen, mentale Modelle und kognitive Strategien von Bedeutung sind (Robins et al., 2003). Mentale Modelle sieht Dutke (1994, S. 2) als „Grundlage zur Planung und Steuerung von Handlungen“. Dass mentale Modelle mit kognitiven Strategien in Zusammenhang stehen zeigen auch klar die Ergebnisse von Pennington (1987), bei welchen Strategien zum Programmverständnis auf das Modell der Aufgabendomäne zurückgreifen. Während mentale Modelle beschreiben, wie die Welt organisiert ist, beschreiben kognitive Strategien wie die Handlungen der Programmierenden organisiert sind (van Merriënboer & Kirschner, 2018). Kognitive Strategien beinhalten laut, van Merriënboer und Kirschner (2018), Wissen über Wirkungsbeziehungen für die Anwendung von Daumenregeln, während mentale Modelle dieselben Beziehungen in dem Kontext der Aufgabendomäne verorten. Vergleicht man Novizen mit Experten, so lässt sich erkennen, dass Novizen über weniger adäquate mentale Modelle verfügen, Strategien oftmals vernachlässigen und ihr Wissen nicht in effiziente kognitive Strategien überführen können. Obwohl die Programmierung in der Forschung grosse Beachtung findet, machen Robins et al. (2003) in ihrem Review darauf aufmerksam, dass zurzeit noch wenig Literatur zur Programmgenerierung vorliegt und Studien zu mentalen Modellen und kognitiven Strategien bei Novizen noch kaum vorliegen.

3 Fragestellung

In diesem Kapitel wird zuerst die Fragestellung mit deren Unterfragestellungen abgeleitet. Auf Basis der abgeleiteten Fragestellungen, werden im Anschluss zu prüfende Hypothesen aufgestellt. Am Ende des Kapitels wird eine Abgrenzung zu benachbarten Themengebieten festgehalten.

3.1 Ableitung der Fragestellung

Da im Rahmen des vorhergehenden STEC-Forschungsprojektes der FHNW die Unterrichtenden und technische Aspekte untersucht worden sind, hat die vorliegende Master Thesis zum Ziel, ergänzend bedeutende lernspezifische Faktoren von Programmierlernenden zu untersuchen.

Robins et al. (2003) weisen darauf hin, dass Novizen der Programmierung bisher noch wenig erforscht worden sind. Gerade bei der Anwendung von kognitiven Strategien während der Programmierung unterscheiden sich Novizen von Experten (Winslow, 1996). Bereits beim Lernen der Programmierung wird der Anwendung von kognitiven Strategien eine grosse Bedeutung beigemessen (vgl. Robins et al., 2003, van Merriënboer & Kirschner, 2018, Winslow, 1996). Dennoch wurden kognitive Strategien bei Novizen, in Bezug deren Hintergründe und im Kontext Programmierunterricht noch kaum untersucht. Diese Lücke soll mit der vorliegenden Arbeit geschlossen werden. Anhand einer empirischen Untersuchung sollen kognitive Strategien von Lernenden erforscht werden. Die Master Thesis beschäftigt sich mit der folgenden Leitfrage:

> **Welche Rolle nehmen kognitive Strategien von Novizen im Programmierunterricht ein?**

Aus der Leitfrage lassen sich nachfolgende Unterfragestellungen ableiten, welche für die Nachvollziehbarkeit der Leitfrage zentral sind.

1. *In welcher Verbindung stehen Vorkenntnisse und kognitive Strategien?*
2. *In welcher Verbindung stehen mentale Modelle und kognitive Strategien?*
3. *Wie entstehen kognitive Strategien?*
4. *Welche kognitiven Strategien führen zu einer höheren schulischen Leistung?*
5. *Wie können kognitive Strategien beim Liveprogramming gefördert werden?*

Zur Untersuchung der oben genannten Fragestellungen wurde ein sequenzielles Mixed-Methods-Untersuchungsdesign geplant. Dabei werden zu Beginn auf explorative Weise,

qualitativ Erkenntnisse zu kognitiven Strategien gesammelt, um diese darauffolgend in der quantitativen Studie auf die Allgemeinheit zu prüfen.

3.2 Hypothesen

Zu den Fragen, in welchem Zusammenhang kognitive Strategien mit mentalen Modellen und der schulischen Leistung stehen, wurden im Vorfeld Hypothesen formuliert, welche in den nachfolgenden Unterkapiteln beschrieben werden. Weitere Hypothesen zum Zusammenhang kognitiver Strategien mit der schulischen Leistung sowie Hypothesen zum Zusammenhang kognitiver Strategien und Vorkenntnissen werden später, bei den Resultaten der qualitativen Studie abgeleitet.

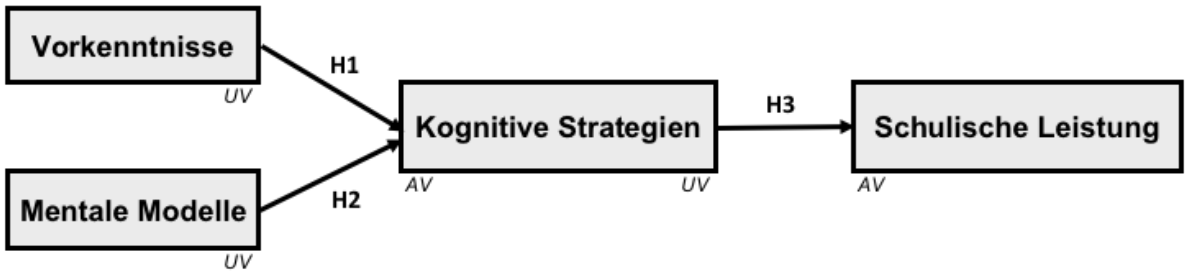


Abbildung 5. Zusammenhang der verschiedenen Hypothesen

In Abbildung 5 wird der Zusammenhang der einzelnen Hypothesen grafisch dargestellt. Zur Beantwortung der Frage, wie kognitive Strategien entstehen, wurde keine vorhergehende Hypothesen entwickelt. Auch wie kognitive Strategien beim Liveprogramming gefördert werden können, wird ohne Hypothese erforscht. Diese Frage wird in Form von Handlungsempfehlungen aufgrund der Ergebnisse und der aktuellen Literatur beantwortet.

3.2.1 Hypothese 1: Kognitive Strategien und Vorkenntnisse

Um sich eine Fähigkeit anzueignen sind bestimmte Vorkenntnisse notwendig, welche ein effizientes Lernen überhaupt ermöglichen (van Merriënboer, 1997). Auch Schwank (1993) sieht eine Verbindung zwischen den Vorkenntnissen von Novizen in der Programmierung und deren Verwendung kognitiver Strategien. Die erste Unterfrage dieser Arbeit beschäftigt sich somit mit der Frage, in welcher Verbindung Vorkenntnisse und kognitive Strategien stehen. Es soll der Zusammenhang erhobener Vorkenntnisse mit einzelner, in der qualitativen Studie explorierter kognitiver Strategien untersucht werden. Dazu werden aus den Ergebnissen der qualitativen Studie später die Hypothesen abgeleitet.

3.2.2 Hypothese 2: Kognitive Strategien und mentale Modelle

Neben kognitiven Strategien stellen auch mentale Modelle einen wichtigen Aspekt beim Erlernen von Programmiersprachen dar (vgl. Robins et al., 2003, van Merriënboer & Kirschner, 2018). Van Merriënboer und Kirschner (2018) differenzieren die drei Arten konzeptuelle, strukturelle und kausale Modelle, welche das mentale Modell bilden. Laut Schwank (1993) gehen sequentielle Strategien mit der Reaktion ständiger Rückmeldungen des Systems einher. Es kann vermutet werden, dass sequentielle Modelle angewendet werden, wenn das mentale Modell nicht ausreichend ist und somit weitere Informationen benötigt werden. Daher wurden folgende Hypothesen aufgestellt:

Hypothese 2a: *Novizen, welche über vollständige mentale Modelle verfügen, unterscheiden sich in der Ausprägung der Anwendung konzeptueller kognitiver Strategien, gegenüber Novizen, welche nicht über vollständige mentale Modelle verfügen.*

Hypothese 2b: *Novizen, welche über vollständige mentale Modelle verfügen, unterscheiden sich in der Ausprägung der Anwendung sequentieller kognitiver Strategien, gegenüber Novizen, welche nicht über vollständige mentale Modelle verfügen.*

3.2.3 Hypothese 3: Kognitive Strategien und die schulische Leistung

Die von Novizen verwendeten kognitiven Strategien sind erst nach der Erhebung und Auswertung der qualitativen Studie bekannt. Da Schwank (1993) konzeptuelle Strategien mit einer strukturierten Vorgehensweise assoziiert, scheint es wahrscheinlich, dass konzeptuelle Strategien mit einer höheren schulischen Leistung einhergehen. Folgende Hypothese wurde somit abgeleitet:

Hypothese 3a: *Novizen, welche konzeptuelle Strategien oft oder immer verwenden, erreichen eine höhere schulische Leistung als Novizen, welche konzeptuelle Strategien nie oder selten verwenden.*

Zudem soll der Zusammenhang einzelner, in der qualitativen Studie explorierter kognitiver Strategien mit der schulischen Leistung untersucht werden. Dazu wird aus den Ergebnissen der qualitativen Studie später eine weitere Hypothese abgeleitet.

3.3 Abgrenzung

Die vorliegende Arbeit untersucht kognitive Strategien im Zusammenhang von Lernerfolg im Programmierunterricht. Im Zentrum stehen dabei die Lernenden. Faktoren seitens der Unterrichtenden und technischen Bedingungen werden daher nicht untersucht. Des Weiteren liegt der Fokus auf den kognitiven Aspekten während des Lernprozesses beim Erlernen der Programmierung. Weitere mögliche Einflussfaktoren wie beispielsweise die Motivation der Lernenden sind nicht Bestandteil dieser Arbeit und werden nicht im Detail erläutert.

4 Methode

In diesem Kapitel wird das methodologische Vorgehen beschrieben und begründet.

4.1 Untersuchungsdesign

Für die Beantwortung der Fragestellung und deren Unterfragestellungen wurde ein sequentielles Mixed-Methods-Untersuchungsdesign durchgeführt. Dabei hatte die vorgehende qualitative Studie ein exploratives Motiv, bei welchem das Ziel bestand, kognitive Strategien von Novizen der Programmierung und deren Herkunft aufzudecken. Die Ergebnisse wurden anschliessend qualitativ ausgewertet und operationalisiert, sodass diese in den standardisierten Fragebogen übertragen werden konnten. Die nachfolgende quantitative Datenerhebung mittels Fragebogen, hatte zum Ziel, „genauere Zahlenangaben für die entdeckten Tatbestände und Zusammenhänge zu erhalten und die Ergebnisse zu generalisieren“ (Kuckartz, 2014, S. 67).

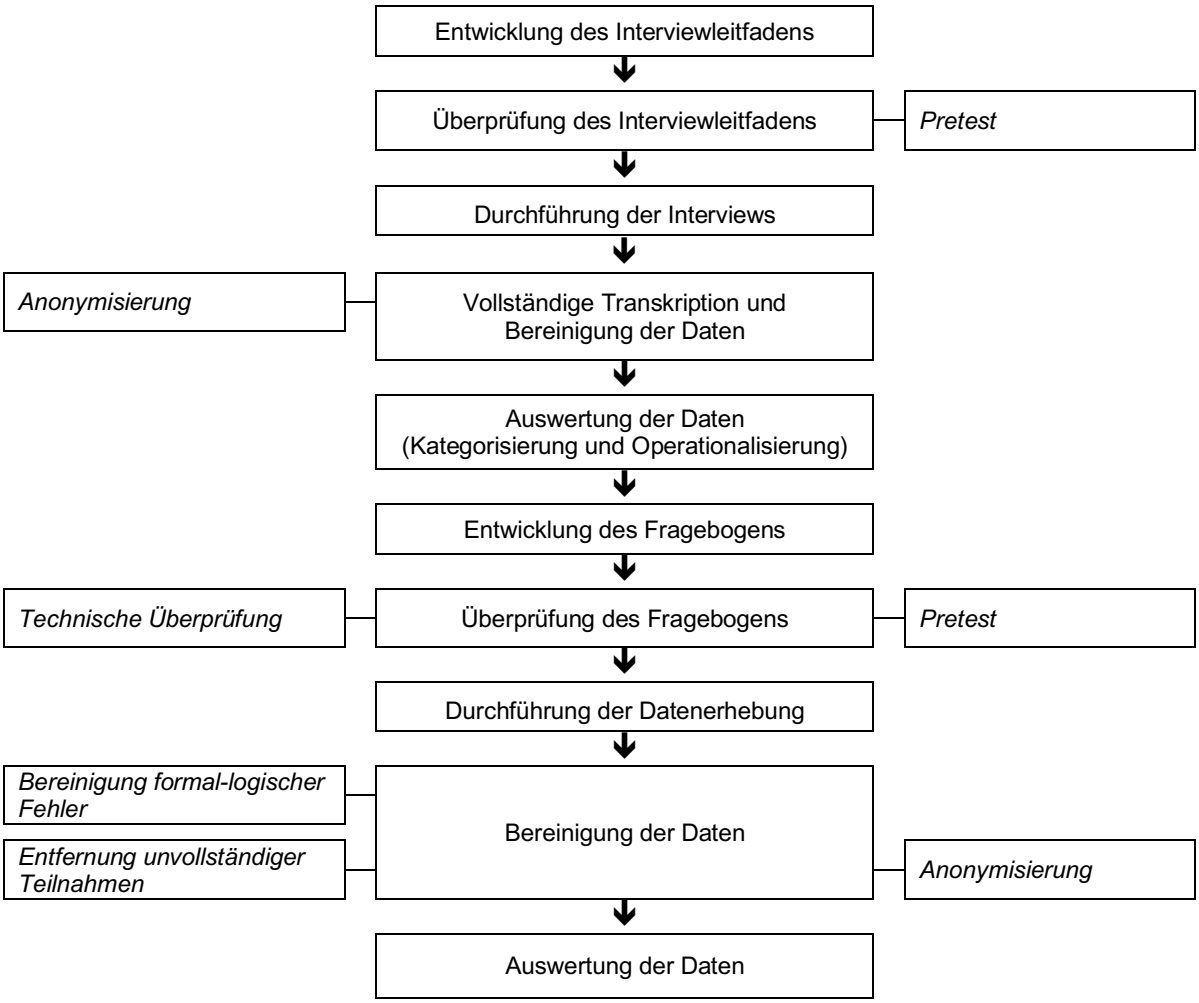


Abbildung 6. Darstellung des Ablaufes der angewendeten Untersuchungsdesigns (eigene Darstellung)

In Abbildung 6 ist der Ablauf des angewendeten Untersuchungsdesigns dargestellt. Das Untersuchungsdesign ist in zehn übergeordnete Teilschritte gegliedert, wobei die ersten fünf Teilschritte der qualitativen Studie (s. Kapitel 4.2) und weitere fünf Teilschritte der quantitativen Studie (s. Kapitel 4.3) zuzuordnen sind.

4.2 Studie 1

Als erstes wurde die qualitative Studie durchgeführt, bei welcher Novizen zu deren kognitiven Strategien und deren Herkunft sowie den Voraussetzungen zum Erlernen der Programmierung befragt wurden. Alternativ wäre möglich gewesen, bereits erhobene kognitive Strategien aus der Literatur für die quantitative Studie zu verwenden. Da jedoch bisher grösstenteils Experten befragt wurden und wenig aktuelle Ergebnisse vorliegen, war eine aktuelle, repräsentativere Erhebung der Daten notwendig. Die Stichprobe der qualitativen Studie sowie das Vorgehen während den einzelnen Teilschritte wird in den folgenden Unterkapiteln beschrieben.

4.2.1 Stichprobenbeschreibung

Die Interviews wurden mit insgesamt N = 9 Studierenden mit Fachrichtung Life Science der FHNW durchgeführt. Von den Teilnehmenden waren N = 3 Frauen und N = 6 Männer. Sämtliche Teilnehmende verfügten über 0.5 bis 1.5 Jahre Programmiererfahrung. Eine der teilnehmenden Personen gab zudem an, im Rahmen einer vorhergehenden Lehre, bereits praktische Erfahrungen in der Programmierung gemacht zu haben.

4.2.2 Rekrutierung der Stichprobe

Da die Studie in Verbindung mit der vorgehenden Studie der EduNat steht, konnte über einen Kontakt der FHNW eine Ausschreibung per E-Mail an Studierende der FHNW in Muttenz weitergeleitet werden. Um möglichst viele Interviewpartner gewinnen zu können, wurde eine Teilnahme mit CHF 40.- verdankt.

Da die Studie einen explorativen Charakter aufwies, wurde vorab keine feste Stichprobengrösse bestimmt. Es wurden stattdessen solange Interviews durchgeführt, bis eine Sättigung eingetroffen ist. Die war der Fall als durch neue Interviews keine neuen Informationen mehr gesammelt werden konnten.

4.2.2.1.1 Limitationen bei der Rekrutierung der Stichprobe

Bei der Rekrutierung der Teilnehmenden konnten lediglich Novizen mit bis zu 1.5 Jahre Programmiererfahrung für eine Teilnahme gewonnen werden. Der anschliessende Fragebogen wurde mit Teilnehmenden mit bis zu fünf Jahre Programmiererfahrung durchgeführt. Es wäre daher möglich, dass aufgrund der homogeneren Programmiererfahrung der Teilnehmenden der qualitativen Studie nicht sämtliche kognitive Strategien der befragten Teilnehmender der quantitativen Studie erhoben werden konnten.

4.2.3 Entwicklung des Interviewleitfadens

Zeitgleich mit der Rekrutierung der Teilnehmenden wurde der Leitfaden für die Interviews entwickelt. Um die kognitiven Strategien zu eruieren, welche von den Novizen eingesetzt werden, wurde der strukturierte Interviewleitfaden entwickelt, mit Orientierung an der Methode der kritischen Ereignisse (CIT), erarbeitet von Flanagan (1954). Bei dieser Methode wird zu Beginn eine Beschreibung einer Situation erfragt, um danach genauer auf diese einzugehen und das Vorgehen zum Lösen der Situation und deren Ergebnis zu eruieren (Flanagan, 1954). Diese Methode schien als besonders geeignet, da zuerst das Thema, bei dieser Studie die Programmierung, mit einer Frage nach den Hauptzielen eröffnet wird, um anschliessend zu erfragen, wie bei einer konkreten Aufgabestellung vorgegangen wurde. In dieser Untersuchung wurde eine jeweils von den Teilnehmenden erinnerte, selbst gewählte Programmieraufgabe aus der Vergangenheit erfragt. Anschliessend erzählten die Teilnehmenden, wie sie bei der Lösung der Aufgabestellung vorgegangen sind und ob die Aufgabe erfolgreich gelöst werden konnte. Es wurde somit ein möglichst typisches Vorgehen, einer komplexen Aufgabenstellung erfragt. Die Methode CIT findet Verwendung in der Eignungsdiagnostik und hat sich sehr etabliert.

Die Methode CIT wurde angewendet, um das Thema zu eröffnen, kognitive Strategien aufzudecken und zu ermitteln, welche kognitiven Strategien zu einer erfolgreichen Lösung einer Aufgabenstellung geführt haben. Neben diesen Fragen, sollten bei den Interviews zusätzlich erfragt werden, woher kognitive Strategien stammen, welche Voraussetzungen beim Erlernen der Programmierung notwendig sind und wie Programmierunterricht für die Lernenden optimal gestaltet werden kann. Da diese Bestandteile nicht in der CIT enthalten sind, wurden weitere Fragen für den Leitfaden entwickelt. Für die Auswahl dieser Zusatzfragen, wurde die SPSS-Methode nach Helfferich (2011, S.182ff) angewendet. Nach Helfferich (2011) ist SPSS ein Akronym für Sammeln, Prüfen, Sortieren und Subsumieren. Dabei werden zu Beginn alle Fragen gesammelt, welche von Interesse sein könnten. Anschliessend werden die Fragen nochmals auf deren Eignung geprüft und ungeeignete Fra-

gen werden, gemäss Helfferich (2011) eliminiert. Die verbleibenden Fragen werden sortiert und am Schluss soweit als möglich zusammengefasst. Die SPSS-Methode nach Helfferich (2011) wurde gewählt, weil dabei die einzelnen Fragen nochmals nach deren Eignung und Offenheit überprüft werden und durch das Subsumieren der Interviewleitfaden wo nötig gekürzt und in eine gute Struktur gebracht werden konnte. Der Interviewleitfaden ist als Anhang A beigelegt.

4.2.3.1 Pretest

Nach der Entwicklung des Entwurfs für den Interviewleitfaden wurde zur Prüfung der Logik der Fragenreihenfolge sowie zur Prüfung der Verständlichkeit der einzelnen Fragen ein Pretest durchgeführt. Der Pretest wurde mit zwei Personen durchgeführt, welche über wenig Erfahrung in der Programmierung verfügen. Es zeigte sich, dass der Fachbegriff der kognitiven Strategien nicht gut verstanden werden konnte. Es wurde daher der Interviewleitfaden insofern abgeändert, dass nicht mehr von kognitiven Strategien gesprochen wurde, sondern es wurde stellvertretend das Wort Vorgehensweisen verwendet. Ansonsten waren keine weiteren Modifizierungen am Interviewleitfaden nötig.

4.2.4 Durchführung der Interviews

Die Interviews wurden im Zeitraum vom 22. November 2017 bis zum 04. Dezember 2017 in den Räumlichkeiten der FHNW in Olten durchgeführt. Da die Teilnehmenden aus der weiteren Umgebung kamen, bestand die Möglichkeit per Skype, in Form einer Videobesprechung, befragt zu werden. Von den 9 Teilnehmenden wurden vier Teilnehmende in Olten und fünf Teilnehmende per Skype befragt. Alle Interviews wurden nach dem Einverständnis der befragten Person mittels Tonaufzeichnung mitgeschnitten. Für die Interviews in Olten wurden die Aufzeichnungen mit der Diktierfunktion eines Smartphones aufgenommen. Bei den Interviews per Skype wurde das Programm *Quicktime Player* für die Audioaufzeichnung verwendet. Die Durchführung der Interviews dauerte jeweils zwischen 8 und 30 Minuten.

4.2.5 Transkription und Bereinigung der Daten

Nachdem alle Interviews durchgeführt worden sind, wurden alle Interviews vollständig transkribiert. Für die Transkription der Interviews wurden im Vorfeld genaue Regeln festgehalten, wie die Transkripte erstellt werden müssen. So sollte beispielsweise auf Abkürzungen verzichtet werden und nicht verständliche Textstellen mit drei Punkten in Eckklammern festgehalten werden. Von den Befragten lediglich begonnene Sätze wurden bei der Tran-

skription mit drei Punkten geschlossen. Da Emotionen bei der Befragung nicht zentral waren, wurden keine Mimiken und Gestiken in den Transkripten beschrieben.

Wurden die Transkripte fertiggestellt, wurden diese anschliessend noch bereinigt. Die Bereinigung beinhaltete die Satzstruktur, die Rechtschreibung sowie das Anonymisieren der Daten. Da die meisten Befragungen in schweizerdeutsch durchgeführt wurden und in Schriftsprache transkribiert wurden, sind die Satzstellungen in der Schriftsprache nicht immer syntaktisch korrekt. Hätte aufgrund dessen eine Aussage anders interpretiert werden können, wurde die Satzstellung wo nötig korrigiert. Ansonsten wurde die Satzstellung beibehalten, um keine Aussage zu verfälschen. Anschliessend wurde die Rechtschreibung der Transkripte nochmals überprüft. Zum Schluss wurden die Transkripte noch anonymisiert. Dazu wurden sämtliche Angaben wie Namen, Orte und Schulklassenzugehörigkeit durch das Wort *anonymisiert* in Eckklammern ersetzt.

Nach Abschluss der Transkription und Bereinigung der neun Interviews wurden sämtliche Tonaufzeichnungen unwiderruflich gelöscht, um den Datenschutz zu gewährleisten.

4.2.6 Auswertung

Um anschliessend die transkribierten Interviews auszuwerten, wurde zuerst eine fallorientierte Erkundung der Daten vorgenommen. Danach wurden die Interviews qualitativ ausgewertet.

4.2.6.1 Fallorientierte Erkundung der Daten

Um ein erstes Gefühl hinsichtlich der erhobenen Daten zu gewinnen, wurde vor der qualitativen Auswertung eine fallorientierte Erkundung der Daten, in Form von *Case Summarys* (s. Anhang B), durchgeführt. Zu diesem Zweck wurde zu jedem Interview, für die Studie interessante Angaben und Aussagen, zusammengetragen. Die offenen Aussagen wurden dabei paraphrasierten Aussagen. „Sinn und Zweck dieser Form der Exploration der Daten einzelner Personen ist die gemeinsame Betrachtung von Antworttexten [...] und relevanten Hintergrundvariablen“, proklamieren Kuckartz , Ebert, Rädiker und Stefer (2009, S. 68).

4.2.6.2 Qualitative Auswertung

Die transkribierten Interviews wurden zur qualitativen Auswertung in das Programm MAXQDA importiert. Diese wurden anschliessend kategorienbasiert verortet und ausgewertet. Dazu wurde anhand einer deduktiven und induktiven Vorgehensweise ein Kategoriensystem erstellt. Dazu wurde in einem ersten Schritt zu jeder Frage eine übergeordnete Kategorie erstellt. In einem nächsten Schritt wurde fallweise jede Antwort einer der Kategorien

zugewiesen und passend zum genannten Schwerpunkt der Antwort eine Subkategorie erstellt. Hat beispielsweise jemand bei der Kategorie "Kognitive Strategien der Programmgeneration" bemerkt, dass die beschriebene Aufgabe durch das freie Herumtippen angegangen wurde, so wurde bei der Kategorie "Kognitive Strategien der Programmgeneration" die Subkategorie "Trial-and-Error" hinzugefügt und die Aussage dieser Kategorie zugeordnet. Dabei wurde möglichst differenziert vorgegangen, sodass neue Aspekte jeweils einer neuen Subkategorie untergeordnet wurden. Jeder Kategorie wurde zudem ein Ankerbeispiel hinzugefügt, sodass auch zu einem späteren Zeitpunkt möglich ist, den Inhalt einer Kategorie nachvollziehen zu können. Zur Qualitätssicherung und um zu vermeiden, dass die Kategorienzuweisung nicht einheitlich vorgenommen wurde, wurden vorab konkrete Codierregeln (s. Anhang C) festgelegt. Dabei wurde beispielsweise festgehalten, in welchem Umfang Textstellen kategorisiert werden sollten.

Tabelle 1. Exemplarischer Auszug aus dem Kategoriensystem am Beispiel der Hauptkategorie „Kognitive Strategien der Programmgeneration“

Hauptkategorien	Subkategorien	Beschreibungen	Ankerbeispiele
Kognitive Strategien der Programmgeneration		Sämtliche Kommentare zu den kognitiven Strategien beim Erstellen von Programmen werden in den Subkategorien dieser Hauptkategorie zugeordnet.	
	„Abarbeiten des Codes von oben nach unten“-Strategie	In dieser Kategorie werden alle genannten Strategien gespeichert, bei denen der Code von oben nach unten bearbeitet wird.	*„Mein allgemeiner Vorgang ist, ich versuche mal etwas, so eine Grundidee und dann schaue ich was nicht funktioniert. Und man lernt es ja so dass, man sollte sich alles auf Papier aufschreiben so ein Ablaufdiagramm oder so. Ich bin immer so, ich fange dann halt mit dem Grundgerüst an, was brauche ich überhaupt und dann schaue ich welche von denen ich ganz einfach schreiben kann... Also zum Beispiel kann ich oft... Die Variabledeklaration ist dann einfach gemacht. Und dann habe ich so verschiedene Methoden zum Beispiel und dann geh ich also eigentlich Stück... Ja, dann arbeite ich diese einfach ab und schaue wo es ein Problem gibt und dort wo es Probleme gibt... ja, das Problem ist, oft findet man einfach keine Lösung. Dann hilft es oft eine Pause zu machen und dann nochmals quasi neu darauf zu schauen, ja.“ (TN 2)*
	Trial-and-Error	In dieser Kategorie werden alle genannten Strategien gespeichert, bei denen ohne Konzept vorgegangen wird.	*„Und dann schreibe ich Pseudocodes und dann gebe ich am Schluss den richtigen Code ein und teste dann aus. Und dann danach fange ich eigentlich an herum zu töggeln. Das ist dann halt nicht mehr so schön, wie man es machen sollte.“ (TN 8)*
	Planung auf Papier	In diese Kategorie kommen alle Kommentare, bei denen das Ablaufdiagramm, oder der Schreibtischtest bei der Erstellung genannt werden.	*„Meistens wenn ich sehe, dass es eine komplexe Aufgabe ist, mache ich eine Skizze, zeichne etwas auf das die Aufgabenstellung zeigt und dann schreibe mal von Hand auf, was heraus kommen sollte am Schluss. Und dann schreibe ich Pseudocodes und dann gebe ich am Schluss den richtigen Code ein und teste dann aus.“ (TN 8)*

Die Tabelle 1 stellt ein Auszug aus dem Kategoriensystem dar. Es ist dabei ersichtlich, wie die in den Interviews genannten kognitiven Strategien kategorisiert wurden. Die gebildeten Kategorien wurden anschliessend für den Fragebogen operationalisiert und als Items formuliert. Die Beschreibung dazu folgt in Kapitel 4.3.2.

4.2.7 Alternative Möglichkeit der Datengewinnung

Bei der Planung der qualitativen Studie wurde ebenfalls in Erwägung gezogen, die kognitiven Strategien mittels der Methode des lauten Denkens anzuwenden, welche auch in anderen Studien angewendet wurde (vgl. Pennington, 1987). Dabei sollten die Teilnehmenden eine Aufgabenstellung erhalten und diese in Anwesenheit der befragenden Person lösen. Während dem Lösen der Aufgabe sollten die Teilnehmende jeweils sämtliche Gedankenschritte laut aussprechen. Für diese Methode hätte gesprochen, dass die kognitiven Strategien nicht erinnert werden mussten, sondern direkt beschrieben wurden. Es wäre somit möglich gewesen auch implizite Strategien zu erheben. Diese alternative Möglichkeit der Datengewinnung wurde jedoch aus mehreren Gründe verworfen. Der erste Argumentationspunkt gegen diese Alternative war die unterschiedliche Erfahrung der Teilnehmenden. Wäre eine komplexe Aufgabenstellung vorgegeben worden, wäre es notwendig gewesen, dass die Teilnehmenden die gleichen Erfahrungen bezüglich deren Kenntnisse in Programmierung hatten sowie auch Erfahrung mit derselben Programmiersprache. Dies wäre jedoch nicht repräsentativ für die Teilnehmenden der quantitativen, nachfolgenden Studie gewesen, bei welcher Programmierlernende mit bis zu fünf Jahre Erfahrung in Programmierung und mit unterschiedlichen Programmiersprachen, befragt wurden. Das zweite Argument gegen diese Erhebungsalternative war, dass dabei lediglich die kognitiven Strategien selbst, nicht aber deren Herkunft sowie die weiteren in dieser Studie erfragten Aspekte hätten erhoben werden können. Eine vollständige Erhebung wäre gewesen, wenn beide Möglichkeiten der Erhebung durchgeführt worden wären. Aus zeitlichen Gründen war dies jedoch in diesem Rahmen nicht möglich.

4.3 Studie 2

Im Anschluss an die qualitative Studie wurde die quantitative Studie durchgeführt, welche einen verallgemeinernden Charakter aufwies. Es bestand dabei das Ziel, die Ergebnisse der vorgehenden, qualitativen Studie sowie die formulierten Hypothesen an einer grösseren Stichprobe zu überprüfen. Die Stichprobe der quantitativen Studie und deren Rekrutierung sowie das Vorgehen während den einzelnen Teilschritte der angewendeten Methode wird in den folgenden Unterkapiteln beschrieben.

4.3.1 Stichprobenbeschreibung

Der Fragebogen wurde von N = 144 Personen beendet. Nach Ausschluss der Teilnehmenden, welche keine oder mehr als fünf Jahre Programmiererfahrung aufweisen, konnten insgesamt N = 106 Teilnahmen für die Auswertungen verwendet werden (s. Tabelle 2).

Tabelle 2. Übersicht zur Entstehung der Stichprobe

	N	Prozent (%)
Bruttobeteiligung	261	100
Abbrüche	117	44.8
Nettobeteiligung	144	100
Stichprobenrelevante Ausfälle	38	26.4
Verwendbare Datensätze	106	73.6

Anmerkungen. Bei der Bruttobeteiligung handelt es sich um die Anzahl Personen, welche den Fragebogen aufgerufen haben. Die Bruttobeteiligung sind die Anzahl Personen, welche den Fragebogen beendet haben. Stichprobenrelevante Ausfälle sind die Teilnehmenden mit keiner oder mehr als fünf Jahre Programmiererfahrung.

Teilgenommen haben 11 weibliche (10.4%) und 79 männliche (74.5%) Personen. Die weiteren 16 Teilnehmenden (15.1%) haben keine Angabe zu ihrem Geschlecht gemacht. Der Mittelwert des Alters aller Teilnehmenden entspricht 23.9 Jahren (SD = 4.285, Wertebereich: 16-40 Jahre, N = 90, fehlende Werte = 16). Es wurde versucht eine möglichst heterogene Stichprobe hinsichtlich der Art der Aus- oder Weiterbildung zu gewinnen. Von den 90 Teilnehmenden, welche eine Aussage betreffend Aus- und Weiterbildung gemacht haben, gaben bei der Frage nach dem höchsten Schulabschluss 35.6% die Matura, 25.6% die Grundschule, 22.2% eine Berufslehre, 10% ein Bachelor-Studium und 6.7% eine höhere Fachschule (HF) an. Auch bezüglich der aktuellen Aus- oder Weiterbildung in Informatik zeigte sich eine Heterogenität bei den 90 Teilnehmenden. Denn 26.7% der Teilnehmenden gaben an, aktuell die Berufsschule zu absolvieren. Weitere Teilnehmende gaben an im

Moment zu studieren auf dem Niveau Bachelor (43.3%), Master (3.3%) und HF (22.2%). Lediglich 4.4% gaben an, derzeit keine Aus- oder Weiterbildung in Informatik zu absolvieren.

Weiter wurden alle Teilnehmenden nach deren Erfahrung in der Programmierung befragt. Während Winslow (1996) proklamiert, dass es zehn Jahre benötigt um Experte in Programmierung zu werden, stellt Dreyfus (2004) die aufbauenden Etappen (a) Novize, (b) fortgeschrittener Beginner, (c) Kompetenz, (d) geübt und (e) Experte vor. Wie viele Jahre die jeweiligen Etappen dauern, wird in der Literatur nicht definiert. Auch ist es kaum möglich, eine Etappe in Zeit zu messen, da beispielsweise in verschiedenen Lehrgängen und je nach Studienpensum die Anzahl an Programmiermodulen variieren kann und somit auch die in das Erlernen der Programmierung investierte Zeit. Für die vorliegende Studie wurde daher entschieden Programmierlernende mit bis zu 5 Jahren Programmiererfahrung zu befragen (s. Abbildung 7).

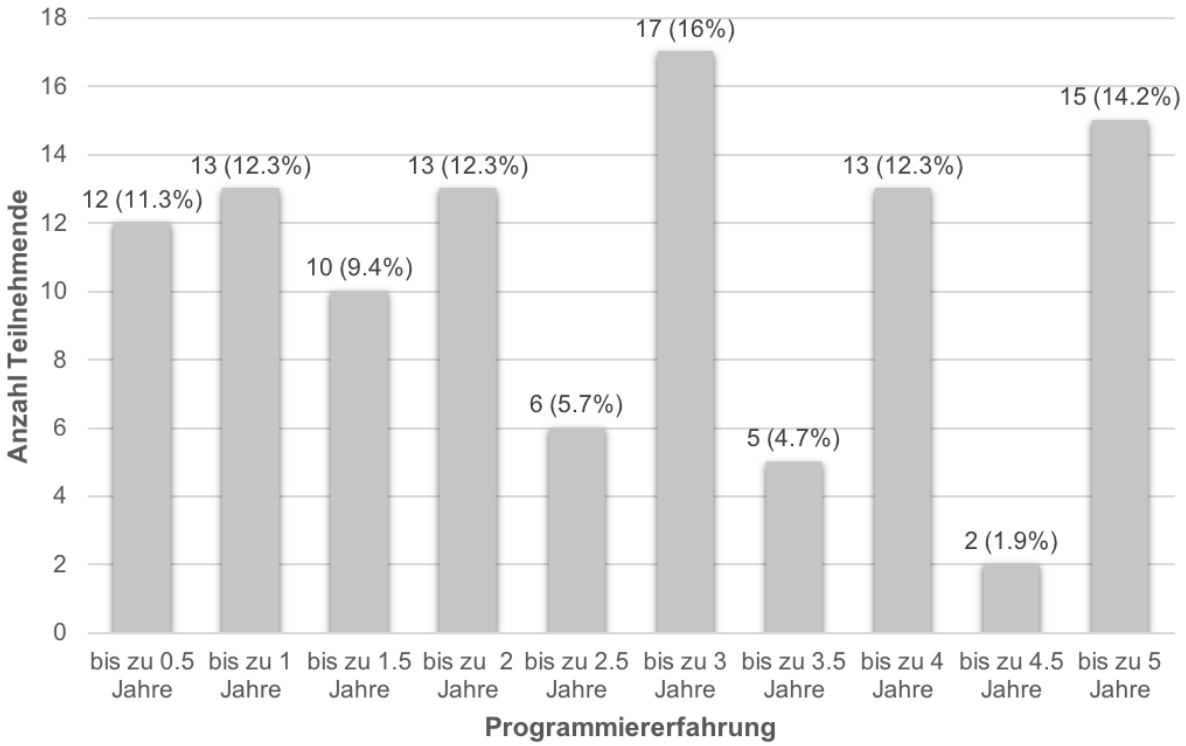


Abbildung 7. Aufstellung der Programmiererfahrung der Teilnehmenden (N=106) in Jahren

Wie Abbildung 8 zeigt, hatten die Teilnehmenden unterschiedlich hohe Erfahrungen in verschiedenen Programmiersprachen. Mit der Programmiersprache Java gaben die Teilnehmenden mit einem Mittelwert von 3.13 an, am meisten Erfahrung (SD = 1.147, Wertebereich: 1-5) zu verfügen.

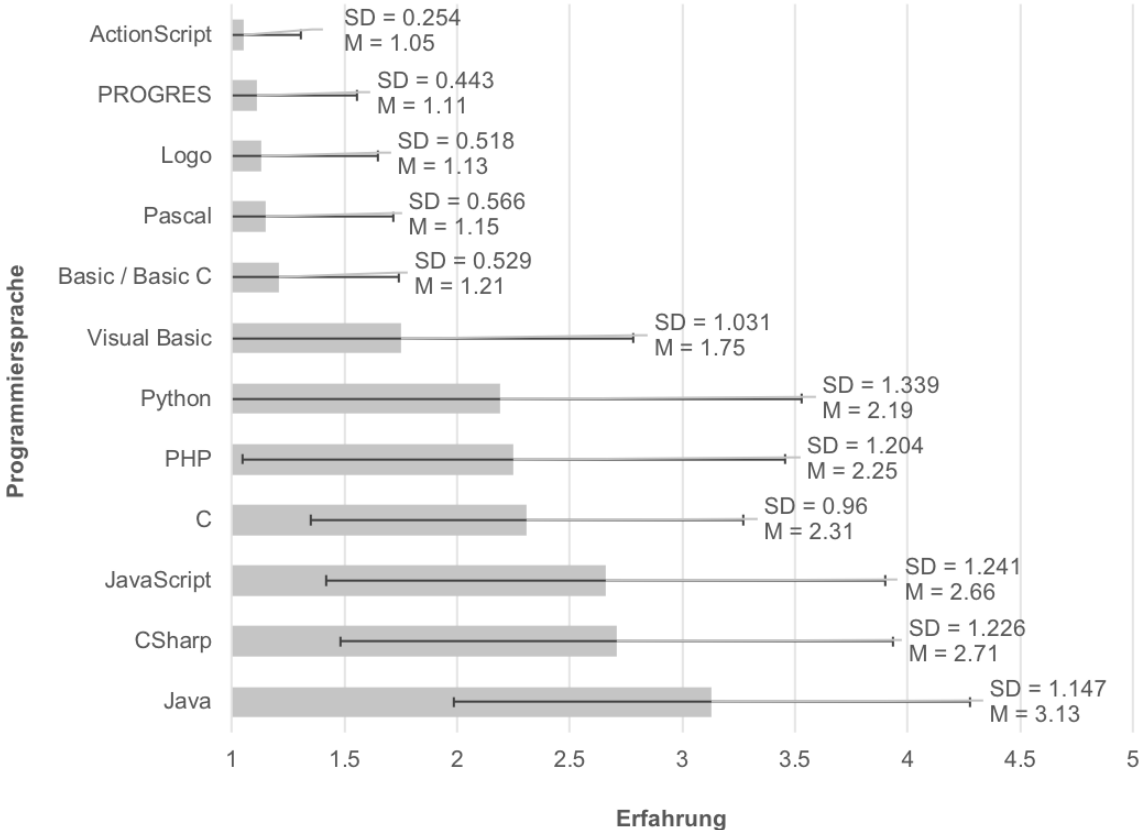


Abbildung 8. Mittelwerte und Standardabweichung zur Erfahrung der Teilnehmenden (N=106) hinsichtlich unterschiedlicher Programmiersprachen. Skalierung von 1 (keine Erfahrung) bis 5 (sehr viel Erfahrung)

Da bei unterschiedlichen Studien die kognitiven Strategien bei der prozeduralen gegenüber der objektorientierten Programmierung differenzierten, wurde auch die Erfahrung bezüglich den Programmierstilen erfragt (s. Abbildung 9). Mit einem Mittelwert von 3.02 bei der prozeduralen Programmierung (SD = 1.138, Wertebereich: 1-5) und einem Mittelwert von 33.58 bei der objektorientierten Programmierung (SD = 1.032, Wertebereich: 1-5) sind keine grossen Unterschiede betreffend Erfahrung bei den beiden Programmierstilen erkennbar.

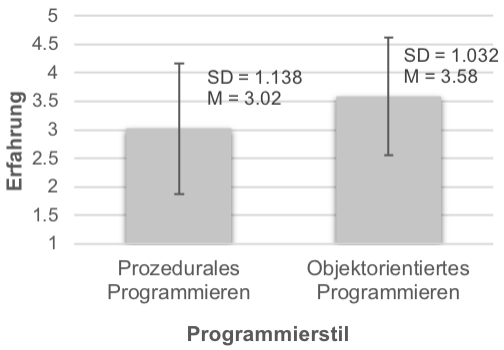


Abbildung 9. Mittelwerte und Standardabweichung zur Erfahrung der Teilnehmenden (N=106) hinsichtlich Programmierstilen. Skalierung von 1 (keine Erfahrung) bis 5 (sehr viel Erfahrung)

Der Mittelwert der schulischen Leistung der Teilnehmenden, anhand von Schulnoten, liegt bei 4.88 und ist somit eher hoch (SD = .671, Wertebereich: 3-6).

4.3.2 Entwicklung des Fragebogens

Als Grundlage für die Entwicklung des Fragebogens dienten die Ergebnisse der qualitativen Studie sowie die zu prüfenden Hypothesen. Dazu wurden in einem ersten Schritt die gebildeten Kategorien hinsichtlich kognitiver Strategien der qualitativen Studie mit deren Unterkategorien in ihre kleinste Einheit operationalisiert und für den Fragebogen als Items formuliert. Zudem wurde eine Frage zu mentalen Modellen aufgenommen sowie Items zur Selbsteinschätzung betreffend Vorkenntnissen. Auch war eine Teilnahme am Wettbewerb möglich. Der Fragebogen enthielt sowohl geschlossene, als auch offene Antwortformate. Als Antwortformat für die geschlossenen Fragen wurden 5-stufige Likert-Skalen verwendet. Der Differenzierungsgrad von fünf Stufen wurde als geeignet betrachtet, weil laut Buehner (2011) die Validität und Reliabilität bis hin zu sieben Antwortkategorien ansteigt, bei zu vielen Kategorien jedoch die Gefahr besteht, dass Teilnehmende mit dem Differenzierungsgrad überfordert sind.

Um eine möglichst grosse Akzeptanz der Teilnehmenden zu erhalten und um den Verlust von Aufmerksamkeit möglichst zu vermeiden, wurde bei der Entwicklung darauf geachtet, dass der Fragebogen innert 15 - 20 Minuten vollständig bearbeitbar war.

4.3.2.1 Aufbau des Fragebogens

Die Fragen wurden in die thematischen Blöcke (a) Erfahrung mit Programmierung, (b) schulische Leistung, (c) Vorkenntnisse, (d) kognitive Strategien, (e) Herkunft kognitiver Strategien, (f) mentale Modelle sowie (g) sozialstatistische Daten unterteilt. Der Fragebogen hat mit einer Einstiegsseite mit einem Einleitungstext zur Studie inklusive Kontaktangaben für Rückfragen und Angaben zu dem Wettbewerb begonnen. Ebenfalls wurde auf der Einstiegsseite die Anonymität bei der Teilnahme an der Studie zugesichert. Sozialstatistische Daten, wie Alter und Geschlecht, wurden ganz zum Schluss abgefragt, da sich diese auch bei gesunkener Aufmerksamkeit gut beantworten lassen (Kuckartz et al,, 2009). Wünschten die Teilnehmende am Wettbewerb mitzumachen, so wurde eine Zusatzseite zur Angabe der Adressdaten zum Schluss angezeigt.

4.3.2.1.1 Erfahrung mit Programmierung

Da lediglich Programmierlernende mit maximal bis zu fünf Jahren Programmiererfahrung befragt wurden, wurde als erstes die Programmiererfahrung abgefragt. Es konnte bei der Erfahrung von *keine* bis *mehr als 5 Jahre* ausgewählt werden. Dazwischen konnte in einem Halbjahrestakt angegeben werden, wie viele Jahre Programmiererfahrung vorhanden sind. Lernte beispielsweise jemand aktuell im dritten Semester Programmierung, wurde *bis zu 1.5 Jahre* ausgewählt.

Des Weiteren wurde zur Beschreibung der Stichprobe erhoben, wieviel Erfahrung die Teilnehmenden bei den verschiedenen Programmiersprachen und den unterschiedlichen Programmiermethoden prozedurales und objektorientiertes Programmieren haben. Zudem wurde gefragt, wieviel Erfahrung bereits in den unterschiedlichen Aufgaben der Programmierung, also mit Fokus auf die Programmgeneration und das Programmverständnis, gesammelt werden konnte. Als Antwortformat wurde dazu jeweils eine 5-stufige unipolare Skala gewählt, welche von keiner bis hin zu sehr viel Erfahrung misst.

Tabelle 3. Darstellung der 5-stufigen Skala des Fragebogens der Items zur Erfahrung in der Programmierung

1	2	3	4	5
Keine Erfahrung				Sehr viel Erfahrung

Es wurden, wie in Tabelle 3 ersichtlich, jeweils lediglich die beiden Endpole der Antwortskala beschriftet.

4.3.2.1.2 Schulische Leistung

Um die schulische Leistung der Teilnehmenden zu messen, wurden diese gefragt, wie gut sie ihr Können bei der Programmierung anhand von Schulnoten einschätzen. Sollten bereits Schulnoten in der Programmierung vorliegen, wurden die Teilnehmenden darum gebeten, anstelle einer Schätzung ihre zuletzt erhaltene Schulnote anzugeben. Als Antwortformat wurde ein Schieberegler verwendet, welcher im Abstand von jeweils einer halben Note, von 1 bis 6 beschriftet war. Schulnoten stellten ein geeignetes Mass dar, um die schulische Leistung zu messen, da Schulnoten ein in der Schweiz allgemein bekanntes und angewandtes Mass darstellt. Somit konnte sichergestellt werden, dass keine Verfälschungen aufgrund unterschiedlicher Interpretationen der Werte aufkamen.

4.3.2.1.3 Vorkenntnisse

Um den Zusammenhang der Vorkenntnisse mit den kognitiven Strategien eruieren zu können, wurde abgefragt, wie gut die Kenntnisse in Sprachkenntnissen, Mathematik und Informatik sind. Die einzelnen Items aus den drei Bereichen wurden auf Basis recherchierter Literatur sowie aus den Ergebnissen der qualitativen Studie entwickelt. Als Antwortformat wurde dazu jeweils eine 5-stufige Skala bipolare gewählt, welche Werte von sehr schlechten bis sehr guten Kenntnisse aufnimmt.

Tabelle 4. Darstellung der 5-stufigen Skala des Fragebogens der Items für die Vorkenntnisse

1	2	3	4	5
Sehr schlecht				Sehr gut

Für die Darstellung der Skala wurden, wie Tabelle 4 zeigt, jeweils nur die beiden Endpole der Antwortskala beschriftet.

4.3.2.1.4 Kognitive Strategien

Für die Abfrage der kognitiven Strategien wurde zwischen kognitiven Strategien für die Programmgeneration und kognitive Strategien für das Programmverständnis differenziert. Somit wurden zwei Fragen zu kognitiven Strategien beim Fragebogen erstellt. Es wurde auch beim Fragebogen jeweils der Begriff Vorgehensweisen verwendet, sodass die Inhalte zielgruppengerecht und allgemein verständlich waren.

Damit sich die Teilnehmenden in eine reale Situation einfühlen konnten, wurde ihnen zuerst eine reale Aufgabenstellung präsentiert. Es wurde alternativ in Erwägung gezogen, keine Aufgabestellung zu formulieren und lediglich zu erfragen, wie normalerweise vorgegangen wird. Für eine vorher beschriebene Aufgabenstellung hat gesprochen, dass dadurch die Komplexität der Aufgabe vorgegeben und für alle somit gleich war. Dadurch konnten komplexitätsbedingte Unterschiede der Antworten umgangen werden. Zudem wurde angenommen, dass durch eine festgelegte Aufgabenstellung das Nachempfinden einer realen Vorhergehensweise erleichtert wird. Die folgende Aufgabe von Soloway, Bohnar und Ehrlich (1989, S. 197), wurde ins Deutsch übersetzt wurde und diente als Aufgabenstellung:

> Schreiben Sie ein Plan für ein Programm, welches jeweils Ganzzahlen einliest und nach dem Einlesen, den exakten Durchschnitt der eingegebenen Zahlen ausgibt. Das Programm soll solange Ganzzahlen entgegennehmen, bis die Zahl 99999 eingegeben wird. Bitte beachten Sie: Die letzte Zahl 99999 sollte nicht in der Berechnung des Mittelwerts enthalten sein.

Die verwendete Aufgabenstellung ist eine Programmieraufgabe, die seit den Achtzigerjahren immer wieder Verwendung in diverser Studien zum Thema Programmierung findet (Fisler, 2014). Obwohl diese Aufgabenstellung weder besonders komplex oder schwierig ist, bemerken Soloway et al. (1989), dass Novizen oftmals Schwierigkeit haben diese Aufgabe zu lösen. Laut Fisler (2014) findet die Aufgabe gerade dann Verwendung, wenn es darum geht, die Planung einer Lösung zu erforschen.

Die Teilnehmenden erhielten im Anschluss an die Aufgabenstellung eine Liste von kognitiven Strategien formuliert als Items erhalten, bei welcher sie jeweils angaben, in welcher Häufigkeit sie die kognitive Strategie anwenden. Die Items zu den kognitiven Strategien für die Programmgeneration konnten auf Basis der Ergebnisse der qualitativen Studie entwi-

ckelt werden. Die Frage, welcher die kognitiven Strategien als Items zugeordnet waren, wurde nicht als eigentliche Frage, sondern Aussage in der ich-Form formuliert.

Frage mit Beispielitem:

Zum Lösen der Aufgabenstellung gehe ich folgendermassen vor...

... Ich arbeite den Code von oben nach unten ab

Dadurch, dass die Frage in der ich-Form formuliert wurde, sollte es für die Teilnehmenden einfacher gestaltet sein, sich in die eigene Verhaltensweise hineinzudenken. Für die Antwort wurde das Format einer 5-stufigen unipolaren Skala verwendet.

Tabelle 5. Darstellung der 5-stufigen Skala des Fragebogens der Items zu kognitiven Strategien bei der Programmgeneration

1	2	3	4	5
nie	selten	gelegentlich	oft	immer

Wie in Tabelle 5 ersichtlich, bot die Antwortskala eine Auswahl von *nie* bis *immer*, im Bezug auf die Anwendung einer kognitiven Strategie zur Programmgeneration. Es wurden alle Skalenpunkte beschriftet.

In einer weiteren Frage, wurden die kognitiven Strategien für das Programmverständnis erfragt. Diese wurden als offene Frage erhoben. Dabei wurde ein Szenario geschildert, bei welchem die zuvor verwendete Aufgabestellung von Soloway et al. (1989) von einer anderen lernenden Person begonnen wurde und diese sich nun um Hilfe an die teilnehmende Person wendet. Die Aufgabestellung von Soloway et al. (1989), wurde dazu wie folgt in das weitere Szenario übernommen:

> Das Programm liest nun vorzu Ganzzahlen ein. Das Programm soll solange Ganzzahlen entgegennehmen, bis die Zahl 99999 eingegeben wird. Nun werden aber bei der Eingabe von 99999 weiterhin Ganzzahlen entgegengenommen. Die Berechnung des Durchschnittes wurde bereits implementiert, konnte jedoch noch nicht getestet werden, da das Programm bei der Eingabe von 99999 nicht mit dem Einlesen endet. Sie erhalten nun den Code und werden gebeten, den Fehler zu suchen.

Die Teilnehmenden wurden daraufhin gebeten, einen möglichste detaillierten Plan zu beschreiben, wie sie beim Lösen der Aufgabe vorgehen würden.

4.3.2.1.5 Herkunft kognitiver Strategien

Des Weiteren wurde erhoben, was die Herkunft kognitiver Strategien sein kann. Die Items dazu wurden aus den Ergebnissen der qualitativen Studie abgeleitet. Auch bei der Frage nach der Herkunft kognitiver Strategien, wurde keine eigentliche Frage, sondern eine Aussage in der ich-Form verwendet.

> **Frage mit Beispielitem:**
> *Meine Strategien hinsichtlich der Programmierung habe ich...*
> *... aus dem Unterricht*

Für die Antwort wurde das Format einer 5-stufigen unipolaren Skala verwendet, welche Skalenpunkte von *trifft gar nicht zu* bis *trifft völlig zu* beinhaltete.

Tabelle 6. Darstellung der 5-stufigen Skala des Fragebogens der Items zur Herkunft kognitiven Strategien bei der Programmgeneration

1	2	3	4	5
trifft gar nicht zu	trifft wenig zu	trifft teils-teils zu	trifft wenig zu	trifft völlig zu

Wie in Tabelle 6 ersichtlich, wurden sämtliche Skalenpunkte beschriftet.

4.3.2.1.6 Mentale Modelle

Für die Erhebung von mentalen Modelle wurde die Teach-Back-Methode, in Anlehnung von van der Veer (1993) verwendet. So wurde in wenigen Worten eine Aufgabenstellung beschrieben und die Teilnehmenden haben die Aufgabe erhalten, eine Instruktion für eine andere lernende Person zu schreiben. Folgende Situation war gegeben:

> Es soll ein Programm geschrieben werden, bei welchen Noten (bspw. 4.6) eingegeben werden können und daraufhin der Notenschnitt berechnet und auf 0.5 gerundet ausgibt. Instruieren Sie Ihre Mitschülerin betreffend dieser Aufgabe.

Die Instruktion sollte das Programm mit dessen Kontext und wichtige Aspekte für die Programmierung enthalten. Die Teilnehmenden wurden gebeten nichts zu löschen, sodass alle Gedankenschritte erhoben werden konnten.

In der Studie zu kognitiven Strategien beim Programmverständnis von Experten, beschrieb Pennington (1987) sieben unterschiedliche Aspekte von Informationen, welchen einen Einfluss auf die angewendeten kognitiven Strategien aufweisen. Diese wurden als Items ebenfalls erfragt (s. Tabelle 7).

Tabelle 7. Operationalisierte Items nach Pennington (1987) zur Herkunft kognitiver Strategien

Zu Beginn beim Lösen der Aufgabe mache ich mir Gedanken zu...
... den Operationen und Aktionen, welche einzelne Funktionen beinhalten
... den benötigten Variablen, mit Datentypen und Inhalt
... den genauen Abläufen bei der Programmausführung
... den Veränderungen von Zusammenhängen und Inhalte von Objekten
... Zustände von Objekten während der Programmausführung
... den Hauptzielen des Programmes
... den Unterzielen des Programmes, welche zum Erreichen des Hauptziels nötig sind

Wie auch zuvor bei den kognitiven Strategien wurde hier eine 5-stufige, unipolare Antwortskala von *nie* bis *immer* verwendet.

4.3.2.1.7 Sozialstatistische Daten

Zuletzt wurden sozialstatistische Daten erhoben. Dabei wurde der Ausbildungsstand, die Geschlechterzugehörigkeit sowie das Alter der Teilnehmenden erfragt. Zudem konnten die Teilnehmenden angeben, ob sie am Wettbewerb teilnehmen, oder über die Resultate der Studie informiert werden mochten. Bei Interesse an den Resultaten der Studie musste die E-Mailadresse angegeben werden. Für eine Teilnahme am Wettbewerb war die Angabe der vollständigen Adresse notwendig.

4.3.2.2 Mögliche Störvariablen

Bei der Planung und Entwicklung der Studie wurden mögliche Störvariablen bedacht und es wurde versucht, diese womöglich zu umgehen, oder deren Auftretenswahrscheinlichkeit zu verringern.

4.3.2.2.1 Soziale Erwünschtheit

Dass Teilnehmende bei Erhebungen auch sozial erwünscht antworten ist eine bekannte Problemstellung (Mummendey und Bolten, 1981). In dieser Studie war dies ebenfalls bei der Planung ein wichtiger Aspekt, da die schulische Leistung sowie Kenntnisse per Selbsteinschätzung erhoben wurden. Um das Auftreten von sozialer Erwünschtheit soweit möglich zu vermeiden, wurde bei der Instruktion des Fragebogens nochmals auf die Anonymität hingewiesen. Denn Mummendey und Bolten (1981, S. 152) schreiben, eine Person wird „vermutlich weniger [von sozialer Erwünschtheit beeinflusste] Antworten geben, wenn sie davon überzeugt werden kann, daß die Antworten für sie selbst keine persönlichen Konsequenzen haben werden“, sondern davon ausgehen, dass diese lediglich der Wissenschaft dienen.

4.3.2.2.2 Antworttendenzen

Die Antworttendenzen, welche bei dieser Studie möglicherweise von Bedeutung sind, ist die Tendenz vermehrt in der Mitte der Ratingskalen zu antworten und die Tendenz, vermehrt äusserste Ausprägungen anzukreuzen. Um auffällige Datensätze ausschliessen zu können, wurde der Datensatz mit den Teilnahmen nach der Datenerhebung überprüft.

4.3.2.2.3 Motivation

Um die Motivation der Teilnehmenden aufrechtzuerhalten, wurde bei der Entwicklung des Fragebogens darauf geachtet, dass dieser keine unnötigen Items enthält und innert 20 Minuten beendet werden konnte.

4.3.2.3 Implementierung in Unipark

Der fertige Entwurf des Fragebogens wurde anschliessend in das online Tool Unipark von *questback* implementiert, welches eine Teilnahme zum Fragebogen über das Internet ermöglicht. Der grosse Vorteil dieses online Tools ist, dass der Fragebogen auf unterschiedlichen Geräten bearbeitet werden kann und ein direkter Export für das Statistikprogramm SPSS ermöglicht wird.

4.3.2.4 Pretest

Vor der Datenerhebung wurde der Fragebogen nach dessen technischen Aspekten geprüft. Für die inhaltliche Überprüfung wurden Pretests mit N = 5 Personen durchgeführt. Dabei sollten Fragen oder Abläufe aufgedeckt werden, welche nicht ausreichend verständlich sind. Dazu wurden die Personen gebeten ein strukturiertes Feedbackformular (s. Anhang E) auszufüllen. Notwendige Anpassungen, welche aus dem Pretest hervorgegangen sind wurden vorgenommen. Anschliessend konnte der Link zum Fragebogen an die Schulen zur Rekrutierung der Teilnehmenden versendet werden.

4.3.3 Rekrutierung der Stichprobe

Um zu bestimmen, welche Stichprobengrösse für die Prüfung der Hypothesen benötigt wird, wurde eine G-Power-Analyse durchgeführt. Aufgrund dessen wurde eine Mindestanzahl an Teilnahmen von N = 100 Personen festgelegt.

Zur Rekrutierung von Programmierlernenden für den Fragebogen wurden insgesamt 28 Schulen per E-Mail angeschrieben mit der Bitte, die Anfrage an Lernende und Studierende mit Programmiermodulen weiterzuleiten. Bei den angeschriebenen Schulen handelte es sich um Berufsschulen, höhere Fachschulen, Fachhochschulen sowie Universitäten und die ETH. Nach einer Woche wurde an alle Schulen, welche keine Rückmeldung gesendet ha-

ben, ein Erinnerungsschreiben versendet. Von den angeschriebenen Schulen haben neun bestätigt, die Anfrage per E-Mail weitergeleitet zu haben. Drei weitere Schulen haben die Anfrage als Aushang in den Schulgebäuden verteilt und eine weitere Schule hat die Anfrage über Facebook bei ihren Studierenden verbreitet. Von zwölf der angeschriebenen Schule wurde keine Rückmeldung erhalten und weitere drei gaben an, keine Studierende und Lernende in der Programmierung auszubilden. Einzig eine der Schulen meldete zurück, die Anfrage nicht weitergeleitet zu haben. Im Laufe der Datenerhebung wurde von einer teilnehmenden Person per E-Mail informiert, dass diese die Anfrage in einem online Platz für den Austausch unter Programmierlernenden, zusätzlich veröffentlicht hat.

Um möglichst viele Teilnehmende gewinnen zu können, bestand nach Ausfüllen des Fragebogens die Möglichkeit an einem Wettbewerb teilzunehmen. Die Gewinne hatten gesamthaft einen Wert von CHF 280.- und es handelte sich dabei um Gutscheine für die ersten drei Plätze und Schokolade für die Plätze 4 bis 10.

4.3.3.1.1 Limitationen bei der Rekrutierung der Stichprobe

Die Rekrutierung der Teilnehmenden fiel schwer, da die Personen über keinen direkten Weg angesprochen werden konnten. Die Schulen durften aus Gründen des Datenschutzes keine Kontaktangaben von Lernenden herausgeben und manche Schulen durften aufgrund von internen Vorschriften die Anfrage nicht weiterleiten. Als erstes mussten jedoch geeignete Schulen recherchiert werden. Da nicht alle Schulen die Lehrpläne veröffentlicht haben, war nicht immer klar, welche Schulen Programmiermodule anbieten. Zudem wurde explizit nach Informatik- und Technikschulen recherchiert, obwohl die Programmierung auch in anderen Bereichen zum Lehrplan gehört. Dies, da es schwer herauszufinden war, welche nicht-technischen Lehrgänge Programmiermodule beinhalten. Aufgrund der schweren Erreichbarkeit von Programmierlernenden konnte die Anzahl an Teilnehmenden lediglich mit viel Aufwand rekrutiert werden. Hätte es mehr Teilnahmen benötigt, wäre es schwer möglich gewesen genügend Teilnehmende zu gewinnen.

4.3.4 Durchführung der Datenerhebung

Zeitgleich mit der Rekrutierung der Teilnehmenden fand die eigentliche Datenerhebung statt. Die Datenerhebung lief vom 08.03.2018 bis zum 31.03.2018 und während dieser Zeit wurden täglich Datensicherungen der aktuellen Teilnahmen gemacht.

4.3.5 Aufbereitung und Bereinigung der erhobenen Daten

Nach der Erhebung der Daten, wurden diese bereinigt und für die Auswertungen vorbereitet. Für die Bereinigung der Daten wurden diese aus dem online Tool Unipark für SPSS exportiert. Anschliessend wurden die abgebrochenen Teilnahmen geprüft. Dabei wurde ein Abbruch gefunden, bei einer sonst vollständigen Teilnahme, bei welcher lediglich die letzte Seite mit den Angaben für den Wettbewerb nicht ausgefüllt wurde. Die entsprechende Teilnahme wurde somit zu den beendeten Teilnahmen gezählt. Zudem hat sich gezeigt, dass N = 15 Teilnehmende auf der vorletzten Seite abgebrochen haben. Bei diesen Teilnahmen fehlte die Beschreibung bezüglich mentaler Modelle sowie die Angaben zu den soziodemografischen Daten. Da jedoch die vorgehenden, vollständigen Angaben dieser Teilnehmenden, bis auf die Hypothese betreffend mentalen Modellen, für die Prüfung sämtlicher Hypothesen verwendet werden konnten, wurden diese ebenfalls als abgeschlossene Teilnahmen gewertet und die nicht vorhandenen Angaben als fehlende Werte betrachtet.

Anschliessend wurden sämtliche formallogischen Fehler gesucht. Als Daten mit formallogischen Fehlern werden Daten verstanden, bei welchen der Datensatz leer oder die Daten inkonsistent oder unplausibel sind, wie beispielsweise ein Alter von über 120 Jahren (Kuckartz et al., 2009).

In einem letzten Schritt, wurde eine vollständige Anonymisierung der Daten vorgenommen. Dazu wurden die offenen Fragen nach Nennungen von Namen und Orten geprüft. Zudem wurden sämtliche Kontaktdaten, welche zur Teilnahme des Wettbewerbs oder Interesse an den Studienergebnissen hinterlegt wurden, getrennt von den Teilnahmedaten zur Studie abgespeichert. So waren keinerlei Rückschlüsse auf Personen und Orte möglich.

4.3.6 Auswertung

Nachdem die erhobenen Daten aufbereitet und bereinigt wurden, wurden die statistischen Auswertungen durchgeführt. Zur Prüfung der Hypothesen wurden unterschiedliche statistische Verfahren genutzt. So wurden deskriptive Statistiken, Korrelationen nach Pearson, lineare Regressionen, multiple lineare Regressionen, Reliabilitätsanalysen, Frequenzanalysen und Tests auf signifikante Unterschiede berechnet.

4.3.6.1 Interpretation von Zusammenhängen

Die Stärke der Zusammenhänge wurden jeweils nach Holling und Gediga (2011), wie in Tabelle 8 dargestellt, interpretiert:

Tabelle 8. Interpretation der Zusammenhangsmasse (r)

r	Zusammenhang
.0 - .09	kein Zusammenhang
.1 - .29	kleiner Zusammenhang
.3 - .49	mittlerer Zusammenhang
>.5	grosser Zusammenhang

4.3.6.2 Skalenbildung

Um die Hypothesen bestmöglich prüfen zu können, wurden zusammengehörige Items bei den Vorkenntnissen und den kognitiven Strategien jeweils zu einer Skala zusammengefasst (s. Anhang G).

Tabelle 9. Übersicht der gebildeten Skalen mit deren Reliabilität

Kategorie	Skala/Items	Hypothese	Anzahl Items	Entfernte Items	N	$\bar{x}$	SD	Cronbachs α
Vorkenntnisse	Mathematik	1	2	-	106	7.41	2.01	.92
	Sprache	1	2	1	106	7.99	1.61	.77
	Informatik	1	3	-	106	11.85	1.97	.61
Kognitive Strategien	Konzeptuelle Strategien	2/3	3	1	106	7.22	2.34	.42
	Sequentielle Strategien	2/3	1	-	106	3.45	.92	-
	Abarbeiten des Codes von unten nach oben	2/3	1	-	106	3.33	1.12	-

In Tabelle 9 werden die gebildeten Skalen und deren statistischen Werte dargestellt. Um die interne Konsistenz zu bestimmen, wurde Cronbachs Alpha für die jeweiligen Skalen berechnet. Items, welche Cronbachs Alpha einer Skala auf weniger als 0.7 senkten, wurden weggelassen. Obwohl Schecker (2014) den Wert 0.7 als üblichen Schwellwert für Cronbachs Alpha identifiziert, hält er fest, dass auch Skalen mit durchaus tieferem Wert verwendet werden dürfen, wenn diese im gleichen Themengebiet denselben Inhalt abbilden. Die interne Konsistenz von Mathematik- und Sprachkenntnissen waren über dem üblichen

Schwellwert von 0.7, lediglich die Informatikkenntnisse leicht darunter. Bei den verwendeten Items zur Skala der Informatikkenntnisse, handelt es sich jedoch um explizite Themen der Informatik, weshalb die Skala verwendet wurde. Bei den Skalen zu den kognitiven Strategien, wurden die Beschreibungen von Schwank (1993) zur Differenzierung zwischen konzeptuellen und sequentiellen Strategien verwendet. Auch wenn bei den konzeptuellen Strategien ein Cronbachs Alpha von lediglich .42 vorliegt, konnten die beinhalteten Items der Skala zugewiesen werden. Dies, da bei allen enthaltenen Strategien eine vorhergehende Planung zentral ist. Wurde das Abarbeiten des Codes in die Skalen konzeptuelle und sequentielle Strategie mitaufgenommen, so führte dies zu einem negativen Wert des Cronbachs Alpha. Da Ormerod (1990) dies aber als eine typische Strategie von Novizen erachtet, wurde dieses Item separat verwendet.

4.3.6.3 Testen der Voraussetzungen

Damit statistische Verfahren zur Prüfung einer Hypothese geeignet sind, gibt es jeweils Voraussetzungen, welche erfüllt sein müssen. Bei sämtlichen statistischen Verfahren welche durchgeführt wurden, wurden somit im Vorfeld die Voraussetzungen überprüft.

4.3.6.3.1 Korrelation nach Pearson

Um die Korrelation nach Pearson zu berechnen, sollten die Daten normalverteilt sein. Da die Stichprobe jedoch N > 30 ist, wurde in Anlehnung an Hussy, Schreier und Echterhoff (2013, S. 69) die Korrelation nach Pearson auch dann verwendet, wenn keine Normalverteilung vorlag.

4.3.6.3.2 Regressionsanalysen

Bei den Berechnungen von Regressionen wurden jeweils vorgängig, die von Field (2013, S. 220) beschriebene Voraussetzungen geprüft und dabei die Werte zur Interpretation von Field (2013) beachtet. Dazu wurde zuerst mittels grafischer Darstellung geprüft, ob ein linearer Zusammenhang zwischen den Prädiktorvariablen und der Kriteriumsvariable gegeben ist. In einem nächsten Schritt wurde geprüft, ob die Prädiktorvariablen nicht zu stark miteinander korrelieren, also dass keine Multikollinearität vorliegt. Keine Multikollinearität wurde interpretiert, wenn der Toleranzwert nicht kleiner .10 und der Varianzinflationsfaktor nicht grösser als 10 war. Ebenfalls mittels grafischer Darstellung wurde geprüft, ob für jeden Wert der Prädiktorvariablen der Fehlerwert den Erwartungswert 0 und dieselbe Varianz beinhaltet. Mit dem Durbin-Watson-Test wurde berechnet, ob die Fehlerwerte nicht voneinander abhängen, also keine Autokorrelation vorliegt. Dass keine Autokorrelation vorliegt wurde interpretiert, wenn der Wert des Durbin-Watson-Tests im Bereich von 2, nicht aber in der

Nähe von 0 oder 4 lag. Als letztes wurde anhand eines Histogrammes überprüft, ob die Fehlerwerte näherungsweise normalverteilt waren.

4.3.6.4 Frequenzanalyse

Um die Verbindung zwischen der Anwendung kognitiver Strategien und der Vollständigkeit mentaler Modelle zu untersuchen, wurde eine Frequenzanalyse durchgeführt. Diese wurde als gut geeignet erachtet, da laut Rack, Zahn und Mateescu (in press), Frequenzanalysen verwendet werden können, um komplexe Verhaltensweisen in kleine, kategoriale Einheiten niederzubrechen und zu quantifizieren. Da die Erhebung der Beschreibung des mentalen Modells als offene Frage mittels des Fragebogens, zusammen mit den geschlossenen Items zu der Anwendung kognitiver Strategien stattgefunden hat, handelt es sich bei der Frequenzanalyse um ein *concurrent embedded Design* (vgl. Rack et al., in press).

4.3.6.4.1 Kategorisierung

Rack et al. (in press) machen darauf aufmerksam, dass die Güte der Frequenzanalyse stark von der Qualität des Kategoriensystems abhängig ist. Es wurde daher darauf geachtet, dass das Kategoriensystem, dargestellt in Tabelle 10, möglichst detailliert und nachvollziehbar ist.

Tabelle 10. Kategoriensystem der durchgeführten Frequenzanalyse

Aspekte des mentalen Modells	Beschreibungen	Mögliche Begriffe	Ankerbeispiele
Konzeptuelles Modell	Sämtliche Beschreibungen zu Objekten und Gegenständen werden dieser Kategorie zugeordnet.	Objekt, Laptop, Programm, Ganzzahlen, ...	*„Die Zahlen repräsentieren Noten, welche auch maximal eine Dezimalstelle beinhalten können (4.6, sowie 5 oder 5.0).“*
Strukturelles Modell	Sämtliche Beschreibungen zu Abläufen, Deklarationen und Prozeduren werden dieser Kategorie zugeordnet.	Kopfzeile, Schleife, Variable deklarieren, Templates, Ablauf, Plan, zuletzt, ganz oben...	*„Für diese Aufgabe kannst du einen for-loop verwenden.“*
Kausales Modell	Sämtliche Beschreibungen zu Auswirkungen und natürlichen Beziehungsprozessen werden dieser Kategorie zugeordnet.	Ursache, auslösen, passieren, Fehlermeldung, weil, ...	*„Pass auf, dass beim Einlesen etwaige Falscheingaben das Programm nicht abstürzt.“*

Als mentales Modell diente das Domänenmodell nach van Merriënboer und Kirschner (2018), welches konzeptuelle, strukturelle und kausale Modelle beinhaltet. Diese drei Arten der Modelle wurden als Aspekte des mentalen Modells erachtet. Es wurde untersucht, ob

jeweils Aussagen zu den Aspekten konzeptuelle, strukturelle und kausale Modelle in den Beschreibungen der Teilnehmenden enthalten waren. Dazu wurden die Beschreibungen gemäss den Kategorienbeschreibungen aus dem Kategoriensystem untersucht, ob Aussagen zu den Aspekten in den Beschreibungen enthalten sind. Für jeden der drei Aspekte wurde eine 1 eingetragen, wenn mindestens eine Aussage dazu beinhaltet war, oder eine 0, wenn eine Aussage dazu fehlte. Die Stärke der Aspekte und mehrmaliges Auftreten wurde nicht gewertet. Es wurde einzig untersucht, ob die drei Aspekte jeweils beinhaltet sind, was für jeden Aspekt zu einer Maximalzahl von 1 führte. Zuletzt wurde die Summe der beinhalteten Aspekte zusammengezählt. Ein vollständiges mentales Modell erhielt somit den maximalen Wert 3. Für die Darstellung wurde eine Kreuztabelle verwendet, welche sich für Frequenzanalysen gut eignet (vgl. Rack et al., in press).

4.3.6.5 Alternative Auswertungsmethode

Um die Vorkenntnisse von Novizen mit deren Anwendung kognitiver Strategien zu untersuchen, wurde auch die Durchführung einer Clusteranalyse in Erwägung gezogen. Mittels einer Clusteranalyse hätte die Möglichkeit bestanden, Novizen anhand deren Vorkenntnissen und deren Anwendung kognitiver Strategien zu möglichst ähnlichen Teilmengen, sogenannten *Clustern*, zusammenzufassen (Wentura & Pospeschill, 2015). Mit dem strukturaufdeckenden Verfahren der Clusteranalyse hätte man somit die Novizen anhand deren Vorkenntnissen und deren Anwendung kognitiver Strategien gruppieren können. Da die Fragestellung zu mentalen Modellen und kognitiven Strategien jedoch nach der Verbindung der beiden Konstrukte sucht und sich konkrete Hypothesen ableiten liessen, wurde eine multiple Regression als besser geeignet angesehen.

5 Ergebnisse

Nachfolgend werden die in der qualitativen und quantitativen Studie gewonnenen Ergebnisse dargestellt und erläutert.

5.1 Studie 1

Die vorgehende qualitative Studie wies einen explorativen Charakter auf und hatte zum Ziel, kognitive Strategien und deren Herkunft zu von Novizen aufzudecken und einen Einblick dazu zu erhalten, wie der Programmierunterricht für Lernende optimal gestaltet werden könnte. Zudem sollte aus den Interviews hervorgehen, welche Voraussetzungen, oder Kenntnisse von den Novizen als wichtig empfunden werden. In den folgenden Unterkapiteln werden die gewonnenen Resultate dargestellt.

5.1.1 Kognitive Strategien zur Programmgeneration

Die Teilnehmenden gaben unterschiedliche Strategien an, welche sie zum Lösen der von ihnen beschriebenen, erinnerten Aufgabestellung eingesetzt haben. Als ersten Schritt gaben mehrere Teilnehmende an, die Aufgabenstellung sorgfältig durchzulesen, um eine erste Idee und genauere Überlegungen zur Aufgabe anstellen zu können.

> *„Also, das Vorgehen ist eigentlich immer gleich, indem man versucht die Aufgaben zu verstehen.“* (TN 9)

Mehr als die Hälfte der Teilnehmenden gaben an, dass sie vor der eigentlichen Programmierung, also der Implementierung selbst, die Lösung schriftlich auf Papier planen.

> *„Also, bevor ich anfange mit Programmieren, überlege ich mir den Ablauf, wie das Programm aufgebaut sein soll. Ich schaue zuerst mal. Erstelle zuerst mal auch von Hand auf Papier schnell eine kleine Zeichnung wie der Algorithmus aussieht. [...]“* (TN 4)

Bei der Planung auf Papier wurden das Erstellen eines Ablaufdiagrammes und das Schreiben von Pseudocode genannt. Eine der befragten Personen führte Strategien mit vorgehender Planung auf Papier, direkt auf das Lösen komplexer Aufgabenstellungen zurück.

> *„Meistens wenn ich sehe, dass es eine komplexe Aufgabe ist, mache ich eine Skizze, zeichne etwas auf das die Aufgabenstellung zeigt und dann schreibe mal von Hand*

auf, was heraus kommen sollte am Schluss. Und dann schreibe ich Pseudocodes und dann gebe ich am Schluss den richtigen Code ein und teste dann aus.“ (TN 8)

Eine weitere Strategie, welche von den Teilnehmenden genannt wurde ist die Trial-and-Error-Strategie. Eine der befragten Personen gab an, dass das Ausprobieren von verschiedenen Möglichkeiten dann gemacht wird, wenn zuvor angewendete Strategien sich nicht als zielführend erwiesen.

„[...] Und dann danach fange ich eigentlich an herum zu töggeln. Das ist dann halt nicht mehr so schön, wie man es machen sollte.“ (TN 8)

Eine weitere kognitive Strategie, welche bei den Interviews beschrieben wurde, war das schrittweise Abarbeiten der Programmierung. Dabei wurde der Programmcode vorzu von oben nach unten abgearbeitet.

„Ich fange immer zuerst an mit der Grundstruktur. Also jetzt bei dem Programm habe ich zuerst angefangen mit Eingaben die wir machen mussten und nachher habe ich für jedes Produkt eine eigene Methode geschrieben und dort bin ich einfach von Methode zu Methode gegangen. Also zuerst die erste Methode ganz fertig geschrieben und geschaut ob sie geht und dann die zweite Methode und ja, bin eigentlich wirklich Schritt für Schritt durchgegangen.“ (TN 5)

Des Weiteren wurde die Durchführung eines Schreibtischtests als Strategie für das Testen von geschriebenem Programmcode beschrieben.

[...] Dann mache ich vielleicht den Schreibtischtest, ob es überhaupt geht. Wenn es funktioniert, fange ich an zu programmieren, also den Code zu schreiben. [...]“ (TN 4)

Auf Nachfrage, beschrieb die befragte Person was genau bei der Durchführung eines Schreibtischtests gemacht wird.

„Also ein Schreibtischtest ist, du erstellst eine Tabelle mit deinen Variablen und du fängst an einzusetzen, nimmst ein paar Beispiele und dann schaust du ob es aufgeht. Du gehst deinen ganzen Algorithmus ab und wenn immer es zu einem Ergebnis kommt, kannst du davon ausgehen, es ist richtig.“ (TN 4)

Die meisten Teilnehmenden haben nicht lediglich eine der beschriebenen Strategien verwendet, sondern zur Lösung der individuell genannten Aufgabe mehrere Strategien angewendet.

5.1.1.1 Itementwicklung zu kognitiven Strategien der Programmgeneration

Aus den erhobenen kognitiven Strategien zur Programmgeneration konnten anschliessend die, in Tabelle 11 dargestellten Items, entwickelt werden.

Tabelle 11. Entwickelte Items zu kognitiven Strategien der Programmgeneration

Zum Lösen der Aufgabenstellung gehe ich folgendermassen vor...
Ich erstelle ein Ablaufdiagramm
Ich schreibe Pseudocode
Ich mache einen Schreibtischtest
Ich arbeite den Code von oben nach unten ab
Ich versuche Zusammenhänge zu verstehen
Ich probiere verschiedene Möglichkeiten aus

Um die kognitiven Strategien möglichst differenziert erheben zu können, wurden die Strategien mit Papierplanung jeweils als eigenständige Items formuliert.

5.1.1.2 Hypothesen

Aus den Ergebnissen zu den kognitiven Strategien zur Programmgeneration kann somit folgende Hypothese abgeleitet werden:

Hypothese 3b: *Die schulische Leistung lässt sich anhand der Prädikatoren Anwendung konzeptueller kognitiver Strategien, Anwendung sequentieller kognitiver Strategien und Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie vorhersagen.*

5.1.2 Kognitive Strategien zum Programmverständnis

Bei der Erhebung kognitiver Strategien zum Programmverständnis wurde von den meisten Teilnehmenden angegeben, bisher noch keine fremde Programmcodes erweitert oder angepasst zu haben.

> *„Das kann ich nicht genau beantworten. Ich habe bis jetzt nur eigene Programme geschrieben. Ich habe noch nie etwas von jemandem übernommen und umschreiben müssen.“* (TN 4)

> *„[...] Aber ich habe damit keine grosse Erfahrung ich habe noch nie ein Programm abgeändert“* (TN 9)

Die Teilnehmenden versuchten trotz der ihnen fehlenden Erfahrung mit fremdem Programmiercode nachzuvollziehen, wie sie beim Lösen einer solchen Programmieraufgabe vorgehen würden. So wurde genannt, dass versucht werden würde, den fremden Programmiercode zuerst zu verstehen. Manche der Teilnehmenden hielten sich bei der Beschreibung eher vage.

> *„Wenn ich jetzt einfach so von jemanden einen Code analysieren müsste und ich dann schauen müsste wieso er nicht funktioniert oder es allgemein formuliert wäre was ich machen müsste, dann würde sich halt unterscheiden, dass ich zuerst herausfinden müsste, was genau bis jetzt schon da ist und was ich noch quasi ändern muss oder überhaupt neu schreiben muss. Aber so eine Aufgabe habe ich noch nie gehabt und von dem her würde ich sagen, bei so einer Aufgabe ist der Code halt eben gegeben und dann ist dann so der Kommentar, da müssen sie noch die und die Methode implementieren und dann ist eigentlich die Vorgehensweise nicht sehr anders, als wenn ich von Grund auf etwas selber schreibe."* (TN 2)

Teilnehmende, welche detaillierter auf die Strategien beim Verstehen eines fremden Programmcodes eingegangen sind, haben insbesondere beschrieben, wonach sie sich in dem Programmcode orientieren würden. Als Orientierungshilfe nannten die Teilnehmenden Dokumentationen und Kommentare im Programmcode.

> *„Ich denke, wenn der der es geschrieben hat, richtig gute Kommentare gegeben hat, kann ich vielleicht nachvollziehen und dort auch Anpassungen machen. [...]"* (TN 4)

Auch wurde das Überprüfen der bestehenden Syntax auf Fehler als Vorgehensweise beschrieben.

> *„[...] Liegt es an einem Semikolon, oder ob man etwas nicht initialisiert hat, oder keinen Wert gegeben, oder ja. Dem würde ich eigentlich sagen, eigentlich Flüchtigkeitsfehler. Man weiss ja, dass man dies machen sollte, aber sie sind halt doch oftmals da. Oder ganz wichtig sind auch Klammern. Dass man die richtig setzt. Denn wenn eine Klammer nicht richtig gesetzt ist, dann geht einfach gar nichts mehr."* (TN 1)

Von Teilnehmenden wurde oftmals angefügt, dass das Schreiben eines eigenen Codes einfacher wäre und weniger Zeit in Anspruch nehmen würde als mit fremd verfasstem Code zu arbeiten.

5.1.2.1 Hypothese

Da die Teilnehmenden grösstenteils über keine Erfahrung mit dem Verstehen von fremden Programmcode verfügten, wurden aus den Ergebnissen dazu keine Items entnommen. Die befragten Programmierlernenden verfügten bis hin zu maximal 1.5 Jahre Erfahrung in der Programmierung, weshalb folgende Hypothese möglich ist:

Hypothese 4: *Novizen mit bis zu 2,5 Jahre Programmiererfahrung unterscheiden sich signifikant betreffend Erfahrung mit Aufgaben zum Programmverständnis, gegenüber Novizen mit 2,5 bis 5 Jahre Programmiererfahrung.*

5.1.3 Voraussetzungen zum Erlernen der Programmierung

Wenn die Teilnehmenden gefragt wurden, welche Voraussetzungen zum Erlernen der Programmierung von Bedeutung sind, nannten diese sowohl persönliche Eigenschaften und die Motivation der Lernenden, als auch deren Vorkenntnisse. Bei den persönlichen Eigenschaften wurden Ausdauer, Fleiss, Geduld, Interesse an der Programmierung und Selbstdisziplin genannt.

> *„Ja, ein bisschen Hartnäckigkeit, dass man nicht aufgibt, wenn man bei einem Problem ansteht. Weil man nicht weiss ob, man eine Klammer falsch gesetzt hat oder so.“* (TN 5)

Auch wurde logisches Denken als gute Voraussetzung zum Erlernen der Programmierung wahrgenommen. Zudem wurden Kenntnisse in Mathematik als wichtigen Aspekt erachtet.

> *„Vielleicht mindestens so Grundkenntnisse... Nein vielleicht schon etwas mehr Kenntnis von Mathematik für die Algorithmen aufzustellen.“* (TN 4)

Neben den mathematischen Kenntnissen wurden auch Sprachkenntnisse als notwendige Voraussetzung angesehen.

> *„Ich glaube die Rechtschreibung ist auch so ein Punkt. Die meisten Programmiersprachen sind ans Englische angelehnt. Es wäre von Vorteil, wenn man die Wörter richtig schreibt, weil sonst gibt es immer Fehlermeldungen.“* (TN 4)

Zusätzlich wurde auch genannt, dass auch Kenntnisse in Informatik zu den Voraussetzungen gehören.

5.1.3.1 Itementwicklung zu den Vorkenntnissen

Da keine persönlichen Eigenschaften sowie auch nicht Aspekte der Motivation Gegenstand dieser Arbeit sind, wurden Ergebnisse dazu nicht für den Fragebogen verwendet. Es wurden stattdessen Items zur Erhebung der Kenntnisse in Mathematik, Sprache und Informatik entwickelt.

Tabelle 12. Entwickelte Items zu den Vorkenntnissen

Wie gut sind Ihre Kenntnisse in...?
Mathematik allgemein [1]
Algebra [1]
Schreiben [2]
Lesen [2]
Englisch [2]
Programmiergrundlagen (Syntax, Arrays, Schleifen, etc.) [3]
Bedienung von Computer/Software [3]
Rechnerarchitekturen [3]

Anmerkungen. Die Items haben folgende Kategorienzugehörigkeit: 1 = Mathematikkenntnisse; 2 = Sprachkenntnisse; 3 = Informatikkenntnisse

Um eine bessere Qualität der Daten zu erhalten, wurden wie in Tabelle 12 dargestellt, zu den verschiedenen Kategorien der Vorkenntnisse mehrere Items operationalisiert.

5.1.3.2 Hypothesen

Aus den Ergebnissen zu den Voraussetzungen können somit folgende Hypothesen abgeleitet werden:

Hypothese 1a: *Die Anwendung konzeptueller kognitiver Strategien lassen sich anhand der Prädikatoren Mathematik-, Sprach- und Informatikkenntnisse vorhersagen.*

Hypothese 1b: *Die Anwendung sequentieller kognitiver Strategien lassen sich anhand der Prädikatoren Mathematik-, Sprach- und Informatikkenntnisse vorhersagen.*

Hypothese 1c: *Die Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie lässt sich anhand der Prädikatoren Mathematik-, Sprach- und Informatikkenntnisse vorhersagen.*

5.1.4 Herkunft kognitiver Strategien

Auf die Frage, wie die Teilnehmenden gelernt haben, die von ihnen beschriebene Vorgehensweise zu verwenden, konnten verschiedene Quellen erhoben werden. Manche der Teilnehmenden haben ihre kognitiven Strategien auf den Unterricht und die dazugehörigen Lehrmittel zurückgeführt.

> *„[Der Dozent] hat es uns so beigebracht. Also die Vorgehensweise ist ein ganz wesentlicher Teil seines Unterrichtes gewesen.“* (TN 9)

Andere Teilnehmende hingegen haben als Herkunft gemeinsames Erarbeiten oder auch das eigenständige Ausprobieren genannt.

> *„Und sonst habe ich es nicht gelernt und habe einfach rausgefunden, dass es für mich am besten funktioniert, wenn ich einfach mal anfange und schaue, wo gibt es Probleme. Weil eigentlich theoretisch hätte ich gelernt, dass man anders vorgeht, dass man alles auf Papier, alles systematisch macht. Aber das ist für mich, das Ganze auf Papier aufschreiben, ist schon zu viel Aufwand. Ich schaue lieber wo gibt es Probleme, ja.“ (TN 2)*

Die Antworten zu der Frage nach der Herkunft der angewendeten Strategie, bezog sich einzig auf die Programmgeneration.

5.1.4.1 Itementwicklung zur Herkunft kognitiver Strategien

Aus den Ergebnissen, konnten somit die in Tabelle 13 dargestellten Items, entwickelt werden.

Tabelle 13. Entwickelte Items zur Herkunft kognitiver Strategien

Meine Strategien hinsichtlich der Programmierung habe ich...
... aus dem Unterricht
... durch Ausprobieren entdeckt
... zusammen mit anderen Lernenden erarbeitet
... aus Lehrmitteln
... Andere:

Bei der letzten Option hatten die Teilnehmenden die Möglichkeit, weitere Quellen zu nennen.

5.1.5 Ideen zur Vermittlung der Programmierung

Um eine Einsicht darin zu erhalten, was Programmierlernende beim Erlernen der Programmierung unterstützt, wurden die Teilnehmenden gefragt, wie sie jemandem das Programmieren beibringen würden und in einer weiteren Frage, welche Verbesserungsvorschläge sie für den aktuellen Programmierunterricht geben möchten. Die Aussagen dazu wurden in einer Kategorie zugeordnet, welche für die Handlungsempfehlungen aussagekräftig ist.

Auf die Frage, wie die Teilnehmenden anderen Lernenden das Programmieren lehren würden, wurden unterschiedliche Methoden genannt. So wurde von mehreren Teilnehmenden auf verschiedene online Medien genannt, welche zur Unterstützung Aufgaben und Tutorials anbieten, welche selbstständig bearbeitet werden können. Zudem wurde von mehreren der Teilnehmenden ausgesagt, dass sie als Erstes vermitteln würden, wie die Aufgaben vorab auf Papier geplant werden können. Die Teilnehmenden empfanden es als wichtig Schritt für Schritt vorzugehen.

> *„Ja, dass man eben Schritt für Schritt anfängt. Dass man vielleicht bei den ersten Programmieraufgaben auch so ein Ablaufdiagramm zeichnet, dass man drein kommt vom Gedankengang, was man machen muss und, ja.“* (TN 5)

Es wurde aber auch von Teilnehmenden darauf hingewiesen, dass das erfolgreiche Erlernen der Programmierung ein individueller Prozess ist, da nicht alle mit denselben Vorhergehensweisen optimal programmieren.

> *„[...] und dass man dann von dort her anfängt und aber, dass man mit der Zeit herausfinden muss was für einem selber am besten ist, und ja.“ (TN 2)*

Bei der Frage nach den Verbesserungen für den Unterricht fielen die Antworten sehr unterschiedlich aus. Manche der Teilnehmenden schätzten es, die Aufgaben Schritt-für-Schritt mit dem Dozierenden gelöst zu haben und empfanden es als zu schwierig, eine Aufgabe von Beginn her selbst zu lösen. Andere Teilnehmende hingegen, sahen einen grösseren Lerneffekt darin, die Aufgabenstellungen jeweils komplett eigenständig oder zusammen mit anderen Lernenden zu erarbeiten.

> *„[...] der Nachteil ist, wenn einer vor dir steht und sagt, jetzt schreiben wir das und das. Du schreibst eben nur ab. Ich habe das Gefühl, ich habe nicht viel gelernt.“* (TN 4)

> *„Dort war das Problem, dass man es nicht mit dem Dozenten angeschaut hat, wie man die Aufgaben löst. [...]“* (TN 5)

Als Input zu den Verbesserungswünschen haben mehrere Teilnehmende auch die Möglichkeit zum gemeinsamen Erarbeiten genannt. Ebenfalls Unterstützung bei der Programmierung, wünschten sich Teilnehmenden durch eine geeignete Entwicklungssoftware, welche beispielsweise die Programmierenden auf allfällige Fehler aufmerksam macht.

5.2 Studie 2

Die quantitative Studie wies einen verallgemeinernden Charakter auf und hatte zum Ziel, die in Kapitel 3.2 abgeleiteten Hypothesen, mit den Erkenntnissen der qualitativen Studie, zu prüfen. In den folgenden Unterkapiteln werden die gewonnenen Ergebnisse dargestellt.

5.2.1 Vorkenntnisse und kognitive Strategien

Zunächst soll die Verbindung zwischen Vorkenntnissen und den kognitiven Strategien näher untersucht werden. Dazu wurden drei Hypothesen formuliert und anschliessend geprüft. Im Folgenden werden die Ergebnisse erläutert.

5.2.1.1 Hypothese 1a

Die Hypothese 1a beschäftigte sich mit der Frage nach dem Zusammenhang konzeptueller kognitiver Strategien mit den Vorkenntnissen in Mathematik, Sprache und Informatik. Es wurde somit eine multiple lineare Regression berechnet, um eine Vorhersage zur Anwendung konzeptueller kognitiver Strategien machen zu können, basierend auf den Vorkenntnissen in Mathematik, Sprache und Informatik. Es konnte keine signifikante Regressionsgleichung gefunden werden ($F(3,102) = 1.397$, $p = .25$), mit einem R^2 von .039. Die vorhergesagte Anwendung konzeptueller kognitiver Strategien der Teilnehmenden beträgt 1.504 + .001 (Mathematikkenntnisse) + .185 (Sprachkenntnisse) + .041 (Informatikkenntnisse), wobei die Mathematik-, Sprach- und Informatikkenntnisse mittels der Ausprägung von 1 = sehr schlecht, bis 5 = sehr gut, gemessen wurden. Die Anwendung konzeptueller kognitiver Strategien der Teilnehmenden erhöhte sich um .001 in der Ausprägung für jeden Ausprägungspunkt in Mathematikkenntnissen, .185 für jeden Ausprägungspunkt in Sprachkenntnissen und .041 für jeden Ausprägungspunkt in Informatikkenntnissen. Weder Mathematikkenntnisse, Sprachkenntnisse, noch Informatikkenntnisse waren signifikante Prädikatoren für die Anwendung konzeptueller kognitiver Strategien. Die Hypothese wurde somit abgelehnt.

5.2.1.2 Hypothese 1b

Die Hypothese 1b ging der Frage nach dem Zusammenhang sequentieller kognitiver Strategien mit den Vorkenntnissen in Mathematik, Sprache und Informatik nach. Es wurde auch hier eine multiple lineare Regression berechnet, um eine Vorhersage zur Anwendung sequentieller kognitiver Strategien machen zu können, basierend auf den Vorkenntnissen in Mathematik, Sprache und Informatik. Es konnte keine signifikante Regressionsgleichung gefunden werden ($F(3,102) = .643$, $p = .59$), mit einem R^2 von .019. Die vorhergesagte Anwendung sequentieller kognitiver Strategien der Teilnehmenden beträgt 2.475 + .050 (Mathematikkenntnisse) + .028 (Sprachkenntnisse) + .172 (Informatikkenntnisse), wobei die Mathematik-, Sprach- und Informatikkenntnisse mittels der Ausprägung von 1 = sehr schlecht, bis 5 = sehr gut, gemessen wurden. Die Anwendung sequentieller kognitiver Strategien der Teilnehmenden erhöhte sich um .050 in der Ausprägung für jeden Ausprägungspunkt in Mathematikkenntnissen, .028 für jeden Ausprägungspunkt in Sprachkenntnissen und .172 für jeden Ausprägungspunkt in Informatikkenntnissen. Weder Mathematikkenntnisse, Sprachkenntnisse, noch Informatikkenntnisse waren signifikante Prädikatoren für die Anwendung sequentieller kognitiver Strategien. Die Hypothese wurde abgelehnt.

5.2.1.3 Hypothese 1c

Die Hypothese 1c beschäftigte sich mit der Frage nach dem Zusammenhang des Abarbeitens des Codes von oben nach unten als kognitive Strategie mit den Vorkenntnissen in Mathematik, Sprache und Informatik. Es wurde eine multiple lineare Regression berechnet, um eine Vorhersage zur Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie machen zu können, basierend auf den Vorkenntnissen in Mathematik, Sprache und Informatik. Es konnte keine signifikante Regressionsgleichung gefunden werden ($F(3,102) = .447$, $p = .72$), mit einem R^2 von .013. Die vorhergesagte Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie der Teilnehmenden beträgt 2.417 + .109 (Mathematikkenntnisse) + .077 (Sprachkenntnisse) + .051 (Informatikkenntnisse), wobei die Mathematik-, Sprach- und Informatikkenntnisse mittels der Ausprägung von 1 = sehr schlecht, bis 5 = sehr gut, gemessen wurden. Die Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie der Teilnehmenden erhöhte sich um .109 in der Ausprägung für jeden Ausprägungspunkt in Mathematikkenntnissen, .077 für jeden Ausprägungspunkt in Sprachkenntnissen und .051 für jeden Ausprägungspunkt in Informatikkenntnissen. Weder Mathematikkenntnisse, Sprachkenntnisse, noch Informatikkenntnisse waren signifikante Prädikatoren für die Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie. Die Hypothese wurde somit abgelehnt.

5.2.1.4 Korrelationsanalyse

Neben den multiplen linearen Regressionen wurden zu den Vorkenntnissen und kognitiven Strategien zusätzlich die Korrelationen nach Pearson berechnet, um zu erkunden, welche Vorkenntnisse jeweils mit der Häufigkeit der Anwendung der unterschiedlichen kognitiven Strategien korrelieren (s. Tabelle 14).

Tabelle 14. Korrelationen und Signifikanz von Vorkenntnissen und kognitiven Strategien

	Konzeptuelle kognitive Strategien		Sequentielle kognitive Strategien		Abarbeitens des Codes von oben nach unten	
	r	*P*	*r*	*p*	*r*	*p*
Mathematikkenntnisse	-.013	.896	.044	.654	.092	.347
Sprachkenntnisse	.196*	.044	.042	.671	.055	.577
Informatikkenntnisse	.065	.507	.123	.210	.032	.748

Anmerkungen. (*) Korrelation auf dem Niveau von 0,05 (2-seitig) signifikant.

Die Informatikkenntnisse korrelieren leicht mit der Häufigkeit der Anwendung von sequentiellen kognitiven Strategien, jedoch nicht signifikant, r(104) = .123, p = .21. Hingegen korrelieren Sprachkenntnisse signifikant leicht mit der Häufigkeit der Anwendung von konzeptuellen kognitiven Strategien, r(104) = .196, p = .044.

5.2.2 Mentale Modelle und kognitive Strategien

Die nächste Frage bezieht sich auf die Verbindung zwischen mentalen Modellen und kognitiven Strategien. Hier wurden zwei Hypothesen formuliert und anschliessend geprüft. Damit eine gute Nachvollziehbarkeit gegeben ist, wurden für die Kreuztabellen und deren Berechnung, die Skalenwerte der Anwendung konzeptuellen Strategien auf Ganzzahlen gerundet, sodass die Werte wieder der ursprünglichen 5stufigen Likertskala von *nie* bis *immer* entsprechen. Im Folgenden werden die Ergebnisse dargestellt und erläutert.

5.2.2.1 Hypothese 2a

Als erstes wurde die Hypothese geprüft, welche besagt, dass Novizen mit vollständigen mentalen Modellen, sich in der Ausprägung der Anwendung konzeptueller kognitiver Strategien, gegenüber Novizen mit nicht vollständigen mentalen Modellen unterscheiden. Dazu wurde eine Kreuztabelle mit zeilenweise absoluten und relativen Werten berechnet (siehe Tabelle 15).

Tabelle 15. Kreuztabelle mit den absoluten und relativen Werten für die Anwendung konzeptueller kognitiver Strategien gegeben die Vollständigkeit mentaler Modelle mit den Aspekten konzeptuell, strukturell und kausal

Anwendung konzeptueller kognitiver Strategien	Vollständigkeit mentaler Modelle			
	1 von 3 Aspekte	2 von 3 Aspekte	3 von 3 Aspekte	Total
nie	7 (63.6%)	2 (18.2%)	2 (18.2%)	11 (100%)
selten	7 (24.1%)	15 (51.7%)	7 (24.1%)	29 (100%)
gelegentlich	11 (37.9%)	13 (44.8%)	5 (17.2%)	29 (100%)
oft	0 (0.0%)	2 (40.0%)	3 (60.0%)	5 (100%)
Total	25 (33.8%)	32 (43.2%)	17 (23.0%)	74 (100%)

Anmerkungen. Wertebereich der der Anwendung konzeptueller kognitiver Strategien (gerundet) 0 (nie) bis 5 (immer); N = 74.

Wie in Tabelle 15 ersichtlich, verfügten Teilnehmende, welche konzeptuelle kognitive Strategien nie verwenden vermehrt über unvollständige mentale Modelle. Jedoch die Teilnehmenden, welche angaben, konzeptuelle kognitive Strategien oft zu verwenden, vermehrt über vollständigere mentale Modelle. Um zu ermitteln, ob dieser Unterschied signifikant ist, wurde ein Chi²-Test durchgeführt. Das Ergebnis war knapp nicht signifikant ($\chi^2(6) = 11.17$, $p = .083$). Da das Ergebnis lediglich knapp nicht signifikant ausgefallen ist, wurde in einer weiteren Analyse, die gelegentliche Anwendung konzeptueller kognitiver Strategien nicht berücksichtigt. Die hatte den Vorteil die stärkeren Ausprägungen der Anwendung konzeptueller kognitiver Strategien besser zu unterscheiden. Die Ergebnisse können in der korrigierten Tabelle 16 entnommen werden.

Tabelle 16. Korrigierte Kreuztabelle mit den absoluten und relativen Werten für die Anwendung konzeptueller kognitiver Strategien gegeben die Vollständigkeit mentaler Modelle mit den Aspekten konzeptuell, strukturell und kausal

Anwendung konzeptueller kognitiver Strategien	Vollständigkeit mentaler Modelle			
	1 von 3 Aspekte	2 von 3 Aspekte	3 von 3 Aspekte	Total
nie	7 (63.6%)	2 (18.2%)	2 (18.2%)	11 (100%)
selten	7 (24.1%)	15 (51.7%)	7 (24.1%)	29 (100%)
oft	0 (0.0%)	2 (40.0%)	3 (60.0%)	5 (100%)
Total	14 (31.1%)	19 (42.2%)	12 (26.7%)	45 (100%)

Anmerkungen. Wertebereich der der Anwendung konzeptueller kognitiver Strategien (gerundet) 0 (nie) bis 5 (immer), wobei Teilnahmen mit Wert 3 (gelegentlich) nicht berücksichtigt wurden; N = 45.

Um zu eruieren, ob sich nun ein signifikanter Unterschied zeigt, wurde erneut ein Chi²-Test durchgeführt, auch hier ohne die Berücksichtigung der gelegentlichen Anwendung konzeptueller kognitiver Strategien. Es konnte dabei ein signifikanter Unterschied berechnet werden ($\chi^2(4) = 10.33$, $p = .035$). Die Hypothese konnte somit angenommen werden.

5.2.2.2 Hypothese 2b

Des Weiteren wurde angenommen, dass Novizen, welche über vollständige mentale Modelle verfügen, sich auch in der Ausprägung der Anwendung sequentieller kognitiver Strategien, gegenüber Novizen, welche nicht über vollständige mentale Modelle verfügen, unterscheiden.

Tabelle 17. Kreuztabelle mit den absoluten und relativen Werten für die Anwendung sequentieller kognitiver Strategien gegeben die Vollständigkeit mentaler Modelle mit den Aspekten konzeptuell, strukturell und kausal

Anwendung sequentieller kognitiver Strategien	Vollständigkeit mentaler Modelle			
	1 von 3 Aspekte	2 von 3 Aspekte	3 von 3 Aspekte	Total
nie	0 (0.00%)	1 (50.0%)	1 (50.0%)	2 (100%)
selten	2 (33.3%)	3 (50.0%)	1 (16.7%)	6 (100%)
gelegentlich	11 (36.7%)	10 (33.3%)	9 (30.0%)	30 (100%)
oft	9 (32.1%)	13 (46.4%)	6 (21.4%)	28 (100%)
immer	3 (37.5%)	5 (62.5%)	0 (0.0%)	8 (100%)
Total	25 (33.8%)	32 (43.2%)	17 (23.0%)	74 (100%)

Anmerkungen. Wertebereich der der Anwendung sequentieller kognitiver Strategien (gerundet) 0 (nie) bis 5 (immer); N = 74.

Betrachtet man die Ergebnisse, welche in der Tabelle 17 dargestellt sind, so lässt sich erkennen, dass Teilnehmende, welche sequentielle kognitive Strategien immer verwenden, über kein vollständiges mentales Modell berichteten. Bei der Berechnung mittels Chi²-Test, ob Unterschiede über die diversen Ausprägungen zur Anwendung sequentieller kognitiver Strategien nachgewiesen werden können, war das Ergebnis jedoch nicht signifikant ($\chi^2(8) = 5.57$, $p = .695$). Die Hypothese wurde daher abgelehnt.

5.2.2.3 In Überlegungen miteinbezogene Aspekte der Programmieraufgabe

Pennington (1987) beschrieb, dass die Anwendung kognitiver Strategien auch davon abhängt, welche Informationen den Programmierenden vorliegen. Wie die unterschiedlichen von Pennington (1987) beschriebenen Aspekte von den befragten Novizen zu Beginn der

Aufgabe in ihre Überlegungen miteinbezogen werden, kann der Abbildung 10 entnommen werden.

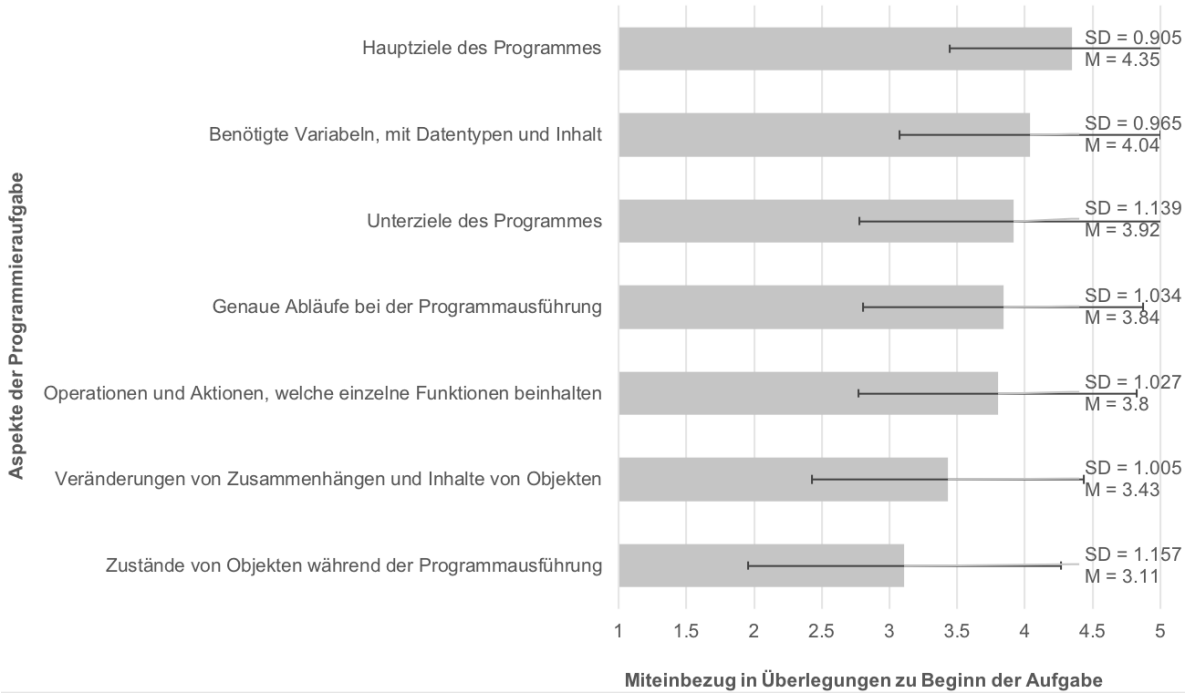


Abbildung 10. Mittelwerte und Standardabweichung zum Miteinbezug in Überlegungen zu Beginn der Aufgabe bei den Teilnehmenden (N=106) betreffend der unterschiedlichen Aspekten der Programmieraufgabe. Skalierung von 1 (nie) bis 5 (immer)

Insbesondere zu den Hauptzielen des Programmes und den benötigten Variablen wurde ein Miteinbezug angegeben. Hingegen Zustände und Zusammenhänge, welche bei der Ausführung des Programmes von Bedeutung sind, werden von den Befragten zu Beginn der Aufgabe weniger Beachtung geschenkt.

5.2.3 Herkunft kognitiver Strategien

Eine weitere Frage dieser Studie widmet sich der Herkunft kognitiver Strategien. Dazu wurden die Teilnehmenden befragt, wo sie sich die angewendeten kognitiven Strategien angeeignet haben. Wie in Abbildung 11 ersichtlich führten die Teilnehmenden die angewendeten Strategien vorallem auf Gelerntes im Unterricht und Entdecktes beim Ausprobieren zurück. Die Befragten konnten hingegen weniger die Lehrmittel als Herkunft der verwendeten Strategien identifizieren. Viele Teilnehmende haben im Zusatzfeld vermerkt, die kognitiven Strategien von weiteren Orten gelernt zu haben. Dabei wurde oftmals das Erlernen während der praktischen Umsetzung einer realen Aufgabenstellung genannt. Dies entweder selbstständig, oder gemeinsam mit erfahrenen Programmierenden. Auch oft wurde genannt, die Strategien wurden online gelernt. Dies beispielsweise mittels online Kursen, Videos auf Youtube, Austauschportalen wie Blogs und Foren, oder online angebotenen Beispielcodes.

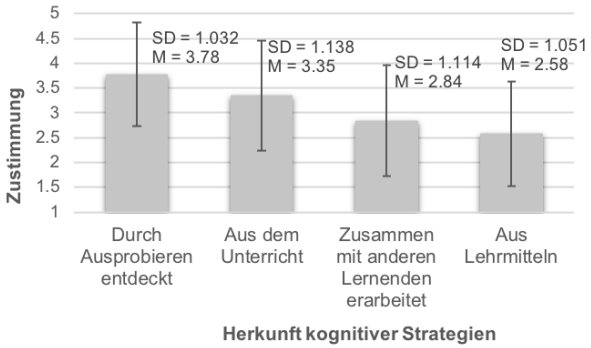


Abbildung 11. Mittelwerte und Standardabweichung der Zustimmung der Teilnehmenden zu der Herkunft kognitiver Strategien (N=106). Skalierung von 1 (trifft gar nicht zu) bis 5 (trifft völlig zu)

5.2.3.1 Hypothese 4

Da in der qualitativen Studie kaum kognitive Strategien entnommen werden konnten, wurde angenommen, dass Novizen mit bis zu 2,5 Jahre Programmiererfahrung sich gegenüber Novizen mit 2,5 bis 5 Jahre Programmiererfahrung, signifikant betreffend der Erfahrung mit Aufgaben zum Programmverständnis unterscheiden.

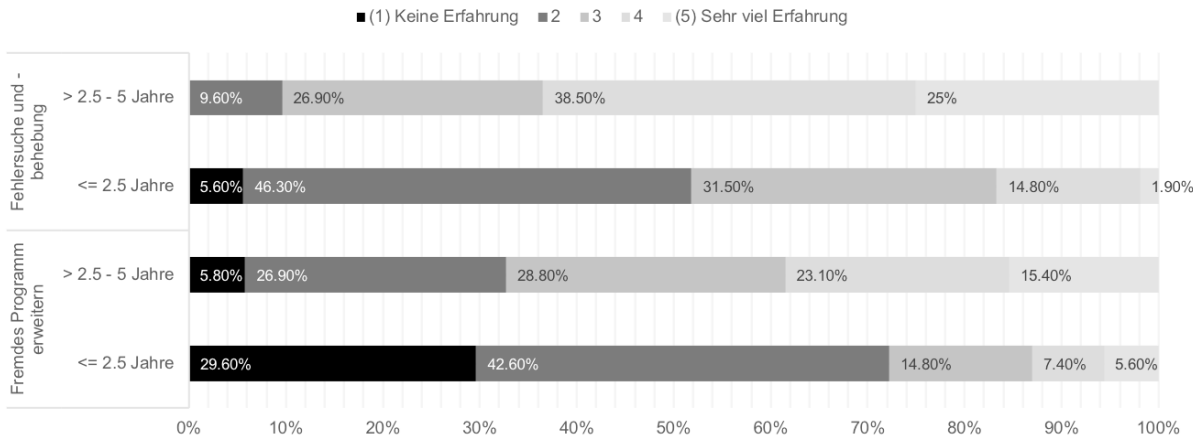


Abbildung 12. Relative Häufigkeiten der Ausprägung der Erfahrungen der Teilnehmenden bei der Fehlersuche und -behebung sowie der Erweiterung fremdem Programmcode nach deren Programmiererfahrung in Jahren (N=106)

Abbildung 12 lässt erkennen, dass die Erfahrungen betreffend den Aufgabenarten Fehlersuche und -behebung sowie der Erweiterung fremden Programmcodes, bei Novizen mit bis zu 2,5 Jahre Programmiererfahrung weniger stark ausgeprägt ist, als bei Novizen mit 2,5 bis 5 Jahre Programmiererfahrung. Um die Hypothese zu prüfen wurde für die beiden Aufgabenarten ein T-Test berechnet. Die Teilnehmenden mit bis zu 2,5 Jahre Programmiererfahrung (M = 2.61, SD = .878), hatten weniger Erfahrung betreffend Fehlersuche und -behebung, als die Teilnehmenden mit 2,5 bis 5 Jahre Programmiererfahrung, (M = 3.79, SD = .936). Der Unterschied war signifikant, $t(104) = 6.683$, $p < .001$. Die Effektstärke liegt bei $r = .30$ und entspricht damit einem mittleren Effekt. Die Teilnehmenden mit bis zu 2,5 Jahre Programmiererfahrung (M = 2.17, SD = 1.112), verfügten ebenfalls über weniger Erfahrung betreffend der Erweiterung fremden Programmcodes, als die Teilnehmenden mit 2,5 bis 5 Jahre Programmiererfahrung, (M = 3.15, SD = 1.161). Der Unterschied war auch hierbei signifikant, $t(104) = 4.472$, $p < .001$. Die Effektstärke liegt bei $r = .16$ und entspricht damit einem kleinen Effekt. Die Hypothese konnte angenommen werden.

5.2.4 Kognitive Strategien und schulischen Leistung

Die Hypothesen 3a und 3b beinhalten Annahmen zu den Zusammenhängen von kognitiven Strategien und der schulischen Leistung.

5.2.4.1 Hypothese 3a

Die Hypothese 3a enthält die Annahme, dass Novizen, welche konzeptuelle kognitive Strategien oft oder immer verwenden, eine höhere schulische Leistung erreichen als Novizen, welche konzeptuelle kognitive Strategien nie oder selten verwenden. Um diese Annahme zu prüfen wurde ein T-Test mit einer unabhängigen Stichprobe durchgeführt. Es wurden dabei nur die Teilnahmen berücksichtigt, bei welchen die Ausprägung der Anwendung konzeptueller kognitiver Strategien zwischen 1 und 2 oder zwischen 4 und 5 lag.

In Abbildung 13 ist ersichtlich, dass die Teilnehmenden, welche angegeben konzeptuelle kognitive Strategien oft oder immer zu verwenden (*M* = 5.1, *SD* = .894), haben eine höhere Note betreffend ihrer schulischen Leistung angegeben, als die Teilnehmenden, welche angegeben haben, konzeptuelle kognitive Strategien nie oder selten verwenden, (M = 4.845, *SD* =

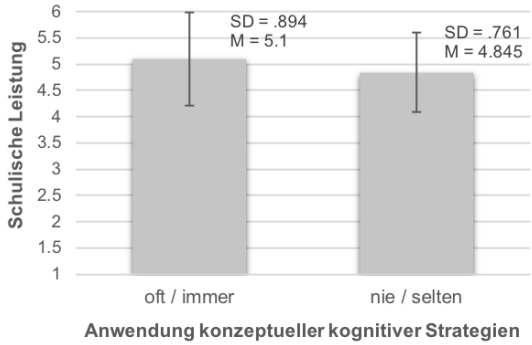


Abbildung 13. Mittelwerte und Standardabweichung der schulischen Leistung der Teilnehmenden nach deren Anwendung konzeptueller kognitiver Strategien (N=47)

.761). Der Unterschied war jedoch nicht signifikant, $t(45) = .696$, $p = .49$. Die Effektstärke liegt bei $r = .10$ und entspricht damit einem kleinen Effekt.

5.2.4.2 Hypothese 3b

Die Hypothese 3b beschäftigte sich mit der Frage nach dem Zusammenhang der schulischen Leistung mit der Anwendung kognitiver Strategien. Es wurde eine multiple lineare Regression berechnet, um eine Vorhersage zur schulischen Leistung machen zu können, basierend auf der Anwendung konzeptueller kognitiver Strategien, der Anwendung sequentieller kognitiver Strategien und der Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie. Es konnte keine signifikante Regressionsgleichung gefunden werden ($F(3,102) = .463$, $p = .71$), mit einem R^2 von .013. Die vorhergesagte schulische Leistung der Teilnehmenden beträgt 5.103 + .011 (Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie) + .013 (Anwendung konzeptueller kognitiver Strategien) - .083 (Anwendung sequentieller kognitiver Strategien), wobei die Anwendung kognitiver Strategien jeweils mittels der Häufigkeit von 1 = nie, bis 5 = immer, erhoben wurden. Die schulische Leistung der Teilnehmenden, erhöhte sich um .011 für jeden Ausprägungspunkt der Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie, .013 für jeden Ausprägungspunkt der Anwendung konzeptueller kognitiver Strategien und verringert sich um .083 für jeden Ausprägungspunkt der Anwendung sequentieller kognitiver Strategien. Weder die Anwendung des Abarbeitens des Codes von oben nach unten als kognitive Strategie, die Anwendung konzeptueller kognitiver Strategien, noch Anwendung sequentieller kognitiver Strategien waren signifikante Prädikatoren für die schulische Leistung. Die Hypothese wurde somit abgelehnt.

6 Diskussion

In diesem Kapitel werden die gewonnenen Ergebnisse im Hinblick auf die Fragestellung und Unterfragestellungen interpretiert und diskutiert, sowie Gestaltungs- und Handlungsempfehlungen für den Programmierunterricht abgeleitet. Zudem werden Limitationen dieser Studie thematisiert. Zuletzt wird noch ein abschliessender Ausblick auf mögliche weitere Forschungen gegeben.

Wie die Ergebnisse der qualitativen Studien aufzeigen, wendeten die Teilnehmenden unterschiedliche kognitive Strategien an, welche sich grösstenteils in konzeptuelle und sequentielle Strategien, nach der Unterteilung von Schwank (1993), kategorisieren lassen. Es konnte jedoch entnommen werden, dass Novizen hauptsächlich über kognitive Strategien zur Programmgeneration verfügen. Kognitive Strategien zum Programmverständnis wurden lediglich aus den Erfahrungen von der Entwicklung eines Programmcodes abgeleitet beschrieben. Die Teilnehmenden haben vermehrt ausgesagt, noch kein Programmcode betreffend Modifikation oder Erweiterung, im Unterricht behandelt zu haben. Auch die Resultate der quantitativen Befragung zeigte, dass die Erfahrung mit Aufgabenstellungen zum Programmverständnis zu Beginn des Erlernens der Programmierung noch sehr knapp ausfallen.

Novizen empfanden Vorkenntnisse in Mathematik, Sprachen und Informatik als wichtige Voraussetzungen, um Programmierung erfolgreich erlernen zu können. Robins et al. (2003) besagen jedoch, dass sich erfolgreiche gegenüber weniger erfolgreichen Novizen vor allem im Bezug auf die Anwendung kognitiver Strategien unterscheiden. Es hat sich jedoch kein Zusammenhang zwischen der Güte der Vorkenntnisse in Mathematik, Sprachen oder Informatik mit der Häufigkeit der Anwendung sequentieller und konzeptioneller Strategien in dieser Studie gezeigt. Auch das Abarbeiten des Codes von oben nach unten als kognitive Strategie, konnte nicht auf die Güte der Vorkenntnisse in Mathematik, Sprachen und Informatik zurückgeführt werden. Auch wenn die Ergebnisse keine Vorhersage zur Anwendung kognitiver Strategien aufgrund der Vorkenntnisse in Mathematik, Sprachen und Informatik aufzeigen, wäre es dennoch möglich, dass Vorkenntnisse und kognitive Strategien einen Zusammenhang aufweisen. Es wäre beispielsweise denkbar, dass mittels einer Clusteranalyse, eine Gruppierung der Anwendung kognitiver Strategien anhand der Vorkenntnisse möglich wäre. Mit der Korrelationsanalyse konnten jedoch zwei kleine Zusammenhänge aufgedeckt werden Die Korrelation von Informatikkenntnissen mit der Häufigkeit der Anwendung von sequentiellen kognitiven Strategien, könnte eventuell darin begründet sein, dass umso besser die Informatikkenntnisse von Novizen sind, sie sich im Umgang mit der

Technik desto sicherer fühlen und daher bei besseren Informatikkenntnissen ein positiveres Ergebnis mit dem Ausprobieren unterschiedlicher Möglichkeiten erwarten. Dass Sprachkenntnisse leicht mit der Häufigkeit der Anwendung von konzeptuellen kognitiven Strategien korrelieren, könnte darauf zurückzuführen sein, dass Personen mit mehr Affinität zu Schrift und Sprache, die Aufgabenstellung besser verschriftlicht planen können.

Als nächstes stellte sich die Frage nach der Verbindung mentaler Modelle und kognitiver Strategien. Es hat sich gezeigt, dass vollständigere mentale Modelle, betreffend den Aspekten konzeptuell, strukturell und kausal, zur Anwendung konzeptioneller kognitiver Strategien geführt haben. Dieses Ergebnis könnte aufzeigen, dass eine Planung der Lösung einer Problemstellung erschwert ist, wenn nicht Novizen über kein vollständiges mentales Modell der Aufgabenstellung verfügen. Die Planung ist jedoch eine sehr wichtige Phase beim Lösen einer Programmieraufgabe (vgl. Rogalski & Samurçay, 1993). Somit ist den mentalen Modellen im Programmierunterricht eine hohe Bedeutung zuzuschreiben. Es konnte auch entnommen werden, dass sich die befragten Novizen mehr Gedanken zu den Zielen und Variablen eines Programmes machen, weniger jedoch über Zusammenhänge und Veränderungen, welche während der Programmausführung erst tragend werden.

Um die Frage zu beantworten, wie kognitive Strategien entstehen, wurde deren Herkunft erforscht. Die befragten Novizen konnten vor allem den Unterricht und das selbstständige Ausprobieren als Quelle identifizieren, sowie auch das Lernen über online Medien. Auch die Wichtigkeit vom Arbeiten in der Praxis konnte entnommen werden. Es kann von den Resultaten abgeleitet werden, dass praktische Aufgabenstellungen, deren Umsetzung die Lernenden auch als sinnvoll ansehen, zu einem grösseren wahrgenommenen Mehrwert führen und zur Aneignung erfolgreicher Strategien beitragen. An dieser Stelle zeigten die Ergebnisse grosse Parallelen zu den Erkenntnissen von Paxton (2002). Denn auch Paxton (2002) sieht einen Vorteil bei Aufgabenstellungen, welche für Lernende einen grösseren Nutzen beinhalten und schlägt vor, mit Programmierlernenden grössere Projekte umzusetzen.

Entgegen der Aussage von Robins et al. (2003), dass sich erfolgreiche gegenüber weniger erfolgreichen Novizen, betreffend der Anwendung von kognitiven Strategien unterscheiden, konnte in dieser Studie kein Zusammenhang der schulischen Leistung mit der Anwendung kognitiver Strategien festgestellt werden. Es stellt sich jedoch die Frage, mit welcher Validität sich der Erfolg anhand der schulischen Leistung per Selbsteinschätzung erheben lässt. Es lässt sich vermuten, dass die soziale Erwünschtheit an dieser Stelle eine beeinflussende Störvariable darstellt, sodass sich die Teilnehmenden teilweise womöglich eine verbesserte Note angegeben haben. Zudem haben keine Teilnehmenden angegeben eine schulische

Leistung unter der Note 3 zu erbringen und der Mittelwert der schulischen Leistung war mit 4.88 eher hoch. Es ist somit fraglich, inwiefern den Resultaten zum Zusammenhang der schulischen Leistung und der Anwendung kognitiver Strategien, Beachtung geschenkt werden darf.

6.1 Handlungs- und Gestaltungsvorschläge

Zur Klärung der letzten Unterfragestellung, soll die Frage beantwortet werden, wie kognitive Strategien beim Liveprogramming gefördert werden können. Dies soll in diesem Unterkapitel in Form von abgeleiteten Handlungs- und Gestaltungsvorschlägen beantwortet werden.

Da der Unterricht und das Ausprobieren als Herkunft kognitiver Strategien identifiziert werden konnten, ist es wichtig, dass diesen beim Unterricht genügend Aufmerksamkeit geschenkt wird. Gerade kognitive Strategie zum Programmverständnis werden zu Beginn teils vernachlässigt. Doch sind kognitive Strategien zum Programmverständnis sehr wichtig, da bereits bei der Fehlersuche, spätestens aber bei der Instandhaltung eines Programmes (vgl. Rogalski & Samurçay, 1993), Strategien zum Programmverständnis genutzt werden müssen. Es ist daher sinnvoll, diese kognitiven Strategien bereits zu Beginn in den Unterricht zu integrieren. Dies könnte beispielsweise so gestaltet werden, dass die dozierende Person bereits einen kleinen Teil des Programmes im Vornherein programmiert und sich zusammen mit der Klasse in das Programm einliest und es anschliessend modifiziert oder erweitert.

Aufgrund der Aussagen der Befragten, wie sie jemandem das Programmieren beibringen würden und wodurch sie ihr Vorgehen gelernt haben, kann empfohlen werden, den Liveprogramming-Unterricht mit zusätzlichen online Medien und Methoden, wie beispielsweise gemeinsames Lösen einer Aufgabenstellung, zu ergänzen. Auch Paxton (2002) empfiehlt neben dem Liveprogramming weitere Methoden einzusetzen. Es wäre vielleicht aber auch vorstellbar, nicht lediglich fremderstellte online Materialien in den Unterricht zu integrieren, sondern zusätzlich auch eigene, von der dozierenden Person erstellte, online Materialien auf einer Plattform zur Verfügung zu stellen. Ein angedachtes Beispiel könnte sein, dass die dozierende Person wochenweise einen halbstündigen Bildschirmmitschnitt, der Planung und Umsetzung eines realen Projektes mit anschliessenden Aufgaben zur Verfügung stellt. Dies könnte dem Unterricht mit Liveprogramming einen grösseren Praxisbezug geben und zeitgleich würde man sich etwas von den lediglich zu Unterrichtszwecken gestellten Übungen wegbewegen.

Ein sehr wichtiger Punkt, der an dieser Stelle abschliessend zu den Handlungsempfehlungen noch Erwähnung finden sollte, ist das Fazit des Konferenzartikels von Luxton-Reilly (2016), bei welchem er darauf aufmerksam macht, dass der schulische Erfolg von Novizen nur sehr schwer erreichbar ist, wenn die Erwartung an deren Leistung von Dozierenden nicht zu hoch angesetzt wird. Es kann daher empfohlen werden, die Erwartungen im Vorfeld nochmals zu überprüfen und darauf zu achten, dass für das Lösen von Programmieraufgaben genügend Zeit eingerechnet wird.

6.2 Limitationen

Obwohl sich die CIT von Flanagan (1954) als Methode zur Exploration kognitiver Strategien gut geeignet hat, ist nicht auszuschliessen, dass auch implizite Strategien Anwendung finden, welche somit nicht erfasst werden konnten. Kognitive Strategien, welche Programmierenden nicht bewusst sind, können jedoch nur schlecht in einem verallgemeinernden Untersuchungsdesign erhoben werden. Wären also die kognitiven Strategien in der qualitativen Studie mittels der Methode des lauten Denkens (vgl. Pennington, 1987) verwendet, so wären eventuell zusätzliche, implizite Strategien miterhoben worden. Es wäre jedoch fraglich, ob eine Selbsteinschätzung der Teilnehmenden in einem Fragebogen möglich wäre und welche Reliabilität diese aufweisen würde. Mittels des verwendeten, verallgemeinernden Untersuchungsdesigns konnten somit Erkenntnisse über, den Novizen bewusste, kognitive Strategien gesammelt werden.

Das Mixed-Methods-Untersuchungsdesign war für die Beantwortung der Fragestellungen sehr geeignet und ermöglichte es, unterschiedliche Auswertungsmethoden, jeweils passend zu der Hypothese oder der Fragestellung, zu verwenden. Was sich jedoch als ein Nachteil herausstellte war, dass die Teilnehmenden der qualitativen Studie bis hin zu maximal 1.5 Jahre Erfahrung in der Programmierung aufwiesen und die gleiche Ausbildung, an derselben Hochschule absolvierten. Die qualitative, verallgemeinernde Studie hingegen, beinhaltete eine heterogene Stichprobe betreffend der Ausbildung der Teilnehmenden. Diese verfügten bis hin zu maximal 5 Jahren Programmiererfahrung und besuchten unterschiedliche Bildungsinstitute. Daher wäre es eventuell möglich, dass nicht alle kognitive Strategien für die verallgemeinernde Studie erhoben werden konnten.

6.3 Fazit und Ausblick

Die Leitfrage der vorliegenden Arbeit befasst sich mit der Frage, welche Rolle kognitive Strategien von Novizen im Programmierunterricht einnehmen. Diese Frage konnte zu einem Teil beantwortet werden. Auch wenn kein direkter Zusammenhang zwischen der schu-

lischen Leistung und der Anwendung kognitiver Strategien entdeckt werden konnte, gewährte die Studie dennoch einen tieferen Einblick betreffend kognitiven Strategien von Novizen und ermöglichte es, praktische Handlungs- und Gestaltungsvorschläge für der Programmierunterricht abzuleiten. Insbesondere zwei wichtige Erkenntnisse, konnten durch die Studie gewonnen werden. Zum Einen hat sich herauskristallisiert, dass die befragten Novizen zu Beginn des Erlernens der Programmieren über kognitive Strategien zur Programmgeneration verfügen, jedoch einen Mangel an kognitiven Strategien zum Programmverständnis aufweisen. Des Weiteren konnte ein Zusammenhang mit der Vollständigkeit mentaler Modelle und der Anwendung konzeptueller kognitiver Strategien aufgedeckt werden. Inwiefern die Ausprägung der einzelnen Aspekte mentaler Modelle, die Anwendung kognitiver Strategien beeinflussen, wäre eine interessante weitergehende Forschungsfrage. Die Ergebnisse dieser Studien könnten darauf hinweisen, dass mentale Modelle einen wichtigen Bestandteil in der Programmierung und deren Erlernen darstellen. Dies geht auch aus den Studien von Pennington (1987) und van Merriënboer und Kirschner (2018) hervor. Eine zusätzliche Studie mit dem Fokus auf mentale Modelle von Programmierlernenden könnte das Verständnis über das Verhalten von Novizen noch weiter verbessern.

7 Literaturverzeichnis

Arzarello, F., Chiappini, G.P., Lemut, E., Malara, N. & Pellerey, M. (1993). Learning Programming as a Cognitive Apprenticeship Through Conflicts. In E. Lemut, B. du Boulay & G. Dettori (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming.* (NATO ASI Series, Series F: Computer and Systems Sciences, Vol 111, pp. 284-298). Springer: Berlin, Heidelberg. https://dx.doi.org/10.1007/978-3-662-11334-9_25

Bringula, R.P., Tolentino, M.A.A., Manabat, G.M.A. & Torres, E.L. (2012). Effects of attitudes towards java programming on novice programmers' errors. *Philippine Information Technology Journal*, 5, 29-34.

Brooks, R. (1990). Categories of programming knowledge and their application. *Internation al Journal of Man-Machine Studies*, 33, 241-246. https://doi.org/10.1016/S0020-7373(05)80118-X

Bundesamt für Statistik (2017). Personen in Ausbildung - Ausgabe 2017. Neuchatel: BFS.

Buehner, M. (2011). *Einführung in die Test- und Fragebogenkonstruktion*. München: Pearson.

Carbone, A., Hurst, J.R., Mitchell, I.J., & Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. *ACE '09*, 1-10.

Corritore, L.C. & Wiedenbeck, S. (2001). An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human-Computer Studies*, 54, 1-23. https://doi.org/10.1006/ijhc.2000.0423

Davies, S.P. (1993). Models and theories of programming strategy. *International Journal of Man-Machine Studies*, 39, 237-267. https://doi.org/10.1006/imms.1993.1061

Dreyfus, S.E. (2004). The Five-Stage Model of Adult Skill Acquisition. *Bulletin of Science Technology & Society*, 24, 117-181. https://doi.org/10.1177/0270467604264992

Dutke, S. (1994). Mentale Modelle: Konstrukte des Wissens und Verstehens: kognitionspsychologische Grundlagen für die Software-Ergonomie. In M. Frese & H. Oberquelle (Hrsg.), *Arbeit und Technik: Praxisorientierte Beiträge aus Psychologie und Informatik* (Bd. 4). Göttingen: Verlag für Angewandte Psychologie.

Fachhochschule Nordwestschweiz (2018a). *EduNaT.* Verfügbar unter: https://www.fhnw.ch/de/die-fhnw/strategische-initiativen/edunat

Fachhochschule Nordwestschweiz (2018b). *Ein Werkzeug zur Unterstützung des Programmierunterrichts.* Verfügbar unter: https://www.fhnw.ch/ppt/content/prj/T999-0734

Field, A. (2013). Discovering statistics using IBM SPSS Statistics. London: Sage.

Fisler, K. (2014). The recurring rainfall problem. *Proceedings of the tenth annual conference on International computing education research*, 35-42. https://doi.org/10.1145/2632320.2632346

Flanagan, J.C. (1954). The critical incident technique. *Psychological Bulletin*, 51, 327-358.

Johnson-Laird, P.N. (1983). Mental Models: Towards a cognitive science of language, inference and consciousness. New York, NY: Cambridge University Press.

Helfferich, C. (2011). *Die Qualität qualitativer Daten. Manual für die Durchführung qualitativer Interviews.* Wiesbaden: Springer VS. https://dx.doi.org/10.1007/978-3-531-92076-4

Holling, H. & Gediga, G. (2011). *Statistik – Deskriptive Verfahren* (Reihe Bachelorstudium Psychologie, Bd. 4). Göttingen: Hogrefe.

Hussy, W., Schreier, M. & Echterhoff, G. (2013). Forschungsmethoden in Psychologie und Sozialwissenschaften für Bachelor. Heidelberg: Springer.

Kuckartz, U. (2014). Mixed Methods Methodologie, Forschungsdesigns und Analyseverfahren. Wiesbaden: Springer VS. http://dx.doi.org/10.1007/978-3-531-93267-5

Kuckartz, U., Ebert, T., Rädiker, S. & Stefer, C. (2009). *Evaluation online. Internetgestützte Befragung in der Praxis.* Wiesbaden: Springer VS. http://dx.doi.org/10.1007/978-3-531-91317-9

Linn, M.C. & Dalbey, J. (1989). Cognitive Consequences of Programming Instruction. In E. Soloway & J.C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 57-81). Hillsdale, NJ: Lawrence Erlbaum Associates.

Luxton-Reilly, A. (2016). Learning to Program is Easy. *ITiCSE '16 Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 284-289. https://doi.org/10.1145/2899415.2899432

Matthews, R., Hin, H.S. & Choo, K.A. (2015). Comparative Study of Self-test Questions and Self-assessment Object for Introductory Programming Lessons. *Procedia - Social and Behavioral Sciences*, 176, 236-242. https://doi.org/10.1016/j.sbspro.2015.01.466

Mummendey, H.D. & Bolten, H. (1981). Die Veränderung von Social-Desirability-Antworten bei erwarteter Wahrheitskontrolle (Bogus-Pipeline-Paradigma). *Zeitschrift für Differentielle und Diagnostische Psychologie*, *2*, 151-156.

Oberquelle H. (1984). On models and modelling in human-computer co-operation. In: G.C. van der Veer, M.J. Tauber, T.R.G. Green & P. Gorny (Eds.), *Readings on Cognitive Ergonomics - Mind and Computers* (Lecture Notes in Computer Science, Vol. 178, S. 26-43). Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-13394-1_3

Ormerod, T. (1990). Human Cognition and Programming. In J.-M. Hoc, T.R.G. Green, R. Samurçay & D.J. Gilmore (Eds.), *Psychology of Programming.* (Computers and People Series, pp. 63-82). San Diego, CA: Academic Press Inc. https://doi.org/10.1016/B978-0-12-350772-3.50009-4

Pennington, N. (1987). Comprehension strategies in programming. In G. Olson, S. Sheppard & E. Soloway (Eds.), *Empirical Studies of Programmers: 2nd Workshop* (pp. 100-113). Norwood, NJ: Ablex.

Paxton, J. (2002). *Live programming as a lecture technique. Journal of Computing Sciences in Colleges*, 18, 51-56.

Rack, O., Zahn, C. & Mateescu, M. (in press). Coding and Counting-Frequency Analysis for Group Interaction. In: Brauner, E., Boos, M., & Kolbe, M. (Eds.). *The Cambridge Handbook of Group Interaction Analysis.* Cambridge, UK: Cambridge University Press.

Robins, A., Rountree, J. & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Journal of Computer Science Education*, 13, 137-172. http://dx.doi.org/10.1076/csed.13.2.137.14200

Rogalski J. & Samurçay R. (1993). Task Analysis and Cognitive Model as a Framework to Analyse Environments for Learning Programming. In E. Lemut, B. du Boulay & G. Dettori (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming.* (NATO ASI Series, Series F: Computer and Systems Sciences, Vol 111, pp. 6-19). Springer: Berlin, Heidelberg. https://dx.doi.org/10.1007/978-3-662-11334-9_2

Rubin, M.J. (2013). The effectiveness of live-coding to teach introductory programming. SIGCSE '13 Proceeding of the 44th ACM technical symposium on Computer science education, 651-656. https://doi.org/10.1145/2445196.2445388

Salleh, M.S., Shukur, S. & Judi, H.M. (2013). Analysis of Research in Programming Teaching Tools: An Initial Review. *Procedia - Social and Behavioral Sciences*, 103, 127-135. https://doi.org/10.1016/j.sbspro.2013.10.317

Schecker, H. (2014). Überprüfung der Konsistenz von Itemgruppen mit Cronbachs alpha. In D. Krüger, I. Parchmann, H. Schecker (Hrsg.), *Methoden in der naturwissenschaftsdidaktischen Forschung* (online-Zusatzmaterial). Springer: Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37827-0

Schwank, I. (1993). Cognitive Structures and Cognitive Strategies in Algorithmic Thinking. In E. Lemut, B. du Boulay & G. Dettori (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming.* (NATO ASI Series, Series F: Computer and Systems Sciences, Vol 111, pp. 249-259). Springer: Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-11334-9_22

Soloway, E., Bonar, J. & Ehrlich, K. (1989). Cognitive Strategies and Looping Constructs: An Empirical Study. In E. Soloway & J.C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 191-207). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sweller, J., van Merriënboer, J.J.G. & Paas, F.G.W.C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, 10, 251–296.

Wentura D., Pospeschill M. (2015). Clusteranalyse. In J. Kritz (Hrsg.), *Multivariate Datenanalyse. Basiswissen Psychologie* (S. 165-179). Springer, Wiesbaden. https://doi.org/10.1007/978-3-531-93435-8_11

Winslow, L.E. (1996). Programming pedagogy: A psychological overview. *ACM SIGCSE Bulletin*, 28, 17-22. http://dx.doi.org/10.1145/234867.234872

van der Veer, G.C. (1993). Mental Representations of Computer Languages - a Lesson from Practice. In E. Lemut, B. du Boulay & G. Dettori (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming.* (NATO ASI Series, Series F: Computer and Systems Sciences, Vol 111, pp. 20-33). Springer: Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-11334-9_3

van Dijk, T.A. & Kintsch, W. (1983). *Strategies of Discourse Comprehension*. New York, NY: Academic Press.

van Merriënboer, J.J.G. & Kirschner, P.A. (2018). *Ten Steps to Complex Learning, a systematic approach to four-component instructional design* (3rd ed.). New York, NY: Routledge.

van Merriënboer, J.J.G. (1997). *Training complex cognitive skills: a four-component instructional design model for technical training* (3rd ed.). Englewood Cliffs, NJ: Educational Technology Publications.

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

10 Anhang

A. Interviewleitfaden

Leitfaden für CIT-Interview

Interviewerin: Amanda Folie

Interviewte Person:

Datum: XX.XX.XXXX

Zeit: XX.XX – XX.XX Uhr

Ort: FHNW Olten, Raum.....

EINLEITUNG

- Begrüssung, Vorstellung und Dank für die Gesprächsbereitschaft
- Kurze Information über Ablauf, Ziel, Inhalt und Methodik des Interviews
- Bekanntgabe der ungefähren Dauer des Interviews (30 Minuten einrechnen)
- Zusicherung des Datenschutzes: Vertraulichkeit
- Fragen zum Werdegang und Erfahrung mit Programmieren

EINSTIEG:

Klärung des Aktivitätsziels

In dieser Studie geht es um das Programmieren. Ich befrage Sie, da Sie über Kenntnisse in der Programmierung verfügen.

1) Was würden Sie sagen, ist das Hauptziel beim Programmieren?

2) In wenigen Worten, wie würden Sie das allgemeine Ziel beim Programmieren zusammenfassen?

HAUPTTEIL:

Erfragen Strategien

Denken Sie zurück, an das letzte Mal, als Sie eine komplexe Programmieraufgabe (z.B. Funktion, Erweiterung, Programm, etc.) erhalten haben. Es kann sich dabei um eine Aufgabenstellung handeln, bei welcher Sie eine Lösung erzielt haben, oder um eine Aufgabenstellung welche Sie nicht lösen konnten.

1) Bitte beschreiben Sie die Aufgabenstellung.

2) Bitte beschreiben Sie so detailliert als möglich, wie Sie bei der Programmierung dieser Aufgaben vorgegangen sind.

3) Bitte beschreiben Sie, inwiefern Ihre Vorgehensweise zu der Lösung der Programmieraufgabe beigetragen hat.

4) Bitte beschreiben Sie, inwiefern die Programmieraufgabe am Schluss gelöst werden konnte?

5) Wie haben Sie gelernt, auf diese Weise vorzugehen?

6) Wie unterscheidet sich die Vorgehensweise während dem Programmieren, wenn Sie bestehenden Code anpassen, gegenüber, wenn Sie einen Code von null auf selber schreiben?

7) Auf welche Weise denken Sie, könnte das Vorgehen beim Programmieren, im Unterricht noch besser vermittelt werden?

8) Wenn Sie jemandem das Programmieren beibringen würden, welche Tipps würden Sie geben, welche Vorgehensweise vorschlagen?

9) Was denken Sie, ist die wichtigste Voraussetzung, um Programmieren zu lernen?

ABSCHLUSS

- Information Ende der Befragung
- Gibt es noch etwas, das Sie ergänzen möchten?
- Herzlichen Dank für Ihre Teilnahme

B. Case Summarys

PERSON 1

- 1 Jahr Programmiererfahrung
- Hat als Ziel der Programmierung das Lösen von Problemen angegeben
- Konnte Aufgabe nicht alleine lösen
- Ablaufdiagramm als kognitive Strategie
- Sieht Ablaufdiagramm als Strukturierungsmöglichkeit
- Hat aus Lehrbuch gelernt so vorzugehen
- Nur Programmgenerierung; kein Programmverständnis gelernt
- Beim Programmierverständnis würde die Person zuerst Hypothesen betreffend Fehlern in der Syntax bilden und diese suchen.
- Frühes Beginnen mit der Programmierung
- Findet wichtig um zu lernen: gemeinsames Programmieren; häufig zu programmieren; Verwendung von Online-Median
- Vorkenntnisse und persönliche Eigenschaften; die benötigt sind Mathematik, Computer-Kenntnisse, Geduld

PERSON 2

- 1,5 Jahre Programmiererfahrung
- Hat als Ziel der Programmierung angegeben, Anweisungen für einen Computer verständlich zu machen (Sprache)
- Konnte Aufgabe nicht alleine lösen; erst beim gemeinsamen Programmieren
- Ausprobieren als kognitive Strategie; macht kein Ablaufdiagramm
- Hat mit Programmier-Buddy-System gelernt so vorzugehen, durch ausprobieren
- Programmverständnis nicht im Unterricht gelernt
- Findet wichtig um zu lernen: jeder lernt anders; logischer Ablauf; verschiedene Methoden ausprobieren
- Vorkenntnisse und persönliche Eigenschaften; logisches Denken, Motivation, Durchhaltevermögen

PERSON 3

- Hat als Ziel die Umsetzung eines Gedankenmodells in ein ausführbares Programm
- Machte sich vor dem Lösen der Aufgaben Gedanken, um Schreibarbeit einzusparen
- Hat eine Strategie angewendet, bei welcher der Code von oben nach unten abgearbeitet wird
- Konnte die Aufgabe selbständig lösen
- Findet wichtig um zu lernen; schrittweiser Aufbau
- Programmverständnis nicht im Unterricht gelernt; hat aber gelernt Kommentare einzubauen für späteres Nachvollziehen
- Findet wichtig um zu lernen: Lehrmittel (Theorie); schrittweises Vorgehen; spielerisches Lernen; Pseudo-Code
- Vorkenntnisse und persönliche Eigenschaften; Interesse und Selbstdisziplin, Freunde,
- Vergleicht das Erlernen von Programmieren mit dem Erlernen einer Fremdsprache

PERSON 4

- 1 Jahr Programmiererfahrung
- Das Ziel der Programmierung ist das Lösen von Aufgaben
- Konnte die Aufgabe nicht lösen
- Macht sich zuerst Gedanken zum Ablauf, dann Ablaufdiagramm, dann Schreibtischtest. Fängt erst dann an mit Programmieren. Testet vorzu während dem Programmieren.
- Hat im Unterricht gelernt so vorzugehen.
- Programmverständnis nicht im Unterricht gelernt. Hat aber gelernt, Kommentare zu erfassen für spätere Nachvollziehbarkeit.
- Wichtig um zu lernen; Selbstausprobieren, gemeinsames Lernen, Dozent sollte nur für Fragen bereitstehen.
- Vorkenntnisse und persönliche Eigenschaften; Rechtschreibung, Englisch, Mathematik, Fleiss

PERSON 5

- 1 Jahr Programmiererfahrung
- Das Ziel der Programmierung: Eine Aufgabenstellung in Algorithmen umzuwandeln und das Übersetzen in eine andere Sprache
- Strategie angewendet, bei welcher der Code von oben nach unten abgearbeitet wird, macht kein Ablaufdiagramm
- Konnte die Aufgabe alleine lösen
- Hat im Unterricht gelernt so vorzugehen.
- Wichtig um zu lernen; Rücksprache mit der dozierenden Person, gute Erklärungen betreffend Funktionen der Programmiersprachen, Schritt für Schritt Vorgehen
- Vorkenntnisse und persönliche Eigenschaften; Ausdauer, Computerwissen

PERSON 6

- 1,5 Jahr Programmiererfahrung
- Das Ziel der Programmierung ist einen Ablauf für den Computer verständlich zu definieren; übersetzen, in die Programmiersprache
- Hat ein Ablaufdiagrammerstellt danach programmiert und anschliessend getestet
- Sieht Ablaufdiagramm als Strukturierungsmöglichkeit
- Konnte die Aufgabe selbständig lösen
- Hat die Vorgehensweise aus dem Unterricht
- Wichtig um zu lernen; Ablaufdiagramme, Schreibmaschinentest, Strategie angewendet, bei welcher der Code von oben nach unten abgearbeitet wird
- Vorkenntnisse und persönliche Eigenschaften; strukturiertes Denken, Geduld

PERSON 7

- 1,5 Jahr Programmiererfahrung
- Das Ziel der Programmierung ist das Verstehen der Computersprache und den richtigen Befehl zur richtigen Ausführung zu erstellen.
- Konnte Aufgabe nicht lösen
- Vergleicht das Programmieren mit dem Sprachverständnis von Kindern, wo man alles genau und einfach erklären muss.
- Hat Pseudocodes als kognitive Strategie verwendet.
- Vorgehen im Unterricht gelernt
- Wichtig um zu lernen; Individuelle Methoden, Unterstützung durch Software, Sprache zu lernen
- Vorkenntnisse und persönliche Eigenschaften; Interesse, Motivation, spezielle Fachkenntnisse, Informatikkenntnisse

PERSON 8

- 1,5 Jahr Programmiererfahrung im Unterricht, vorher Praxiserfahrung
- Das Ziel der Programmierung ist, dem Computer zu sagen, was er machen soll
- Konnte Aufgabe nicht alleine lösen
- Macht eine Skizze, schreibt Pseudocodes, schreibt am Code und testet anschliessend, ausprobieren
- Bezeichnet Pseudocodes als menschennäher
- Hatte die Vorgehensweise in der Lehre, in der Praxis gelernt
- Strategie angewendet, bei welcher der Code von oben nach unten abgearbeitet wird, beim Programmverständnis
- Wichtig um zu lernen; Praxisbeispiele, Selbstausprobieren, Erfahrungsaustausch, online Medien, verstehen wie der Computer denkt, Konzept der Programmiersprache
- Vorkenntnisse und persönliche Eigenschaften; Analytisches und logisches Denken, sequenzielles Denken und technisches Verständnis
- Merkt an, dass die Programmierung zurzeit zu viel gehypt wird. Findet wichtiger, dass zuerst der Computer und das Word richtig bedient werden kann.

PERSON 9

- 1 Jahr Programmiererfahrung
- Das Ziel des Programmierens ist den Computer für individuelle Zwecke nutzen können
- Versuchte zuerst die Aufgabe und das Problem zu verstehen. Versuchte anschliessend das Problem mit Pseudocode zu lösen. Hat dann ein Ablaufdiagramm hergestellt, danach Programmieren
- Findet es wichtig, die Syntax und die Regeln einer Programmiersprache zu kennen
- Hat die Aufgabe aus Zeitgründen nicht fertig gelöst.
- Hat im Unterricht gelernt so vorzugehen
- Programmverständnis nicht im Unterricht gelernt
- Wichtig um zu lernen; Schritt für Schritt Vorgehen, Verständnis dass Computer sehr genaue Anweisungen braucht, Syntax kennenlernen
- Vorkenntnisse und persönliche Eigenschaften; Interesse, Technikaffinität, Motivation, Mathekenntnisse

C. Codierregeln

Regeln zur Codierung qualitativer Daten

Umfang der Codierungen:
Grundsätzlich wird jeder Ganzsatz auf eine Frage codiert. Ausnahmen werden dann gemacht, wenn eine Aufzählung die Antwort ist, welche unabhängige Aspekte beinhaltet. In diesem Fall wird nur der betreffende Teil des Satzes codiert.

Doppelcodierung:
Jede Information pro Person nur einmal codiert, um Doppelcodierungen zu vermeiden.

Verteilte Codierung:
Es werden immer alle Antworten codiert. Wenn eine Antwort nicht in die vorgesehene Kategorie passt, wird sie direkt einer passenden Kategorie zugeordnet. Dies jedoch nur, wenn dadurch keine Doppelcodierung entsteht.

Behandlung von fehlenden Werten („Missing“):
Studie 1: Da keine Zählung der qualitativen Daten erfolgt, werden fehlende Werte oder Antworten, welche einzig einen Gedankenstrich beinhalten nicht codiert.

Studie 2: Missings werden als solche codiert. Sie sind als -66 zu erkennen.

D. Kategoriensystem

Studie 1: Kategoriensystem

Hauptkategorien	Subkategorien	Beschreibungen	Ankerbeispiele
Programmiererfahrung		Sämtliche Kommentare zu den eigenen Erfahrungen mit Programmierung werden in dieser Hauptkategorie zugeordnet.	*„Wir haben das zweite Modul im Studium und jetzt programmieren wir mit Java. Also, ja, einfach einmal das Programmieren kennenlernen, weil, ja, wir schreiben noch keine weltbewegenden Programme.“* *(TN 1)*
Hauptziele beim Programmieren		In den Subkategorien dieser Kategorie werden alle Aussagen zugewiesen, welche sich auf die individuell wahrgenommenen Hauptziele beim Programmieren beziehen.	
	Bereitstellung moderner/individueller Technik	In diese Kategorie kommen alle Kommentare, bei denen das Bereitstellung moderner/ individueller Technik als Hauptziel nennen.	*„Das Ziel sind verschiedene, einerseits Abläufe automatisieren, andererseits Probleme die von Hand unlösbar wären und zu aufwändig, wo ein Computer eben mit seiner Rechenpower gut umsetzen kann, braucht es die Programmierung. Systemerstellung wo Komponenten zusammen schaffen, dann die Applikationen. Ja, es ist noch schwierig, weil die Frage so selbstverständlich ist. Programmierung ist eigentlich das Nutzen vom Computer. In dem er genau die Aufgaben*

			geben kann. Die Aufgaben, wie man es von ihm braucht, dass er sie macht und so genau kann man das ja nicht mit Fertigprogrammen, denn häufig braucht man, ist das so individuell dann muss man programmieren, um das anzupassen und neu entwickeln. Würde ich mal sagen.“ (TN 9)
	Übersetzen von Anweisungen für den Computer	In diese Kategorie kommen alle Kommentare, bei denen das Übersetzen von Anweisungen für den Computer als Hauptziel nennen.	*„Ja, also das Ziel ist ja meistens, dass man irgendeine Anweisung nimmt und das dann für den Computer verständlich macht. Und ja, dann halt dass man das wie in eine Maschinensprache übersetzen kann. Einem Menschen könnte man es ganz anders erklären, aber für einen Computer muss man es halt in einer anderen Sprache schreiben.“ (TN 2)*
	Lösen von Problemen	In diese Kategorie kommen alle Kommentare, bei denen das Lösen von Problemen als Hauptziel nennen.	*„Probleme lösen, eigentlich.“ (TN 1)*
Programmieraufgaben		In dieser Kategorie werden alle Beschreibungen von Programmieraufgaben gesammelt.	*„Das ist eine Aufgabenstellung gewesen, wo man drei Vektoren eingeben konnte und nachher aus denen haben Sparprodukte, wo also einfach... Jetzt muss ich gerade überlegen. Wo man also einfach drei Produkte ausrechnen musste mit denen und dann hat einfach nachher das Programm das Resultat rausgegeben und mit welchem Vektor das mit welchem verrechnet wurde.“ (TN 5)*

Kognitive Strategien der Programmgeneration		Sämtliche Kommentare zu den kognitiven Strategien beim Erstellen von Programmen werden in den Subkategorien dieser Hauptkategorie zugeordnet.	
	„Abarbeiten des Codes von oben nach unten“-Strategie	In dieser Kategorie werden alle genannten Strategien gespeichert, bei denen der Code von oben nach unten bearbeitet wird.	*„Mein allgemeiner Vorgang ist, ich versuche mal etwas, so eine Grundidee und dann schaue ich was nicht funktioniert. Und man lernt es ja so dass, man sollte sich alles auf Papier aufschreiben so ein Ablaufdiagramm oder so. Ich bin immer so, ich fange dann halt mit dem Grundgerüst an, was brauche ich überhaupt und dann schaue ich welche von denen ich ganz einfach schreiben kann... Also zum Beispiel kann ich oft... Die Variabledeklaration ist dann einfach gemacht. Und dann habe ich so verschiedene Methoden zum Beispiel und dann geh ich also eigentlich Stück...Ja, dann arbeite ich diese einfach ab und schaue wo es ein Problem gibt und dort wo es Probleme gibt... ja, das Problem ist, oft findet man einfach keine Lösung. Dann hilft es oft eine Pause zu machen und dann nochmals quasi neu darauf zu schauen, ja.“ (TN 2)*
	Trial-and-Error	In dieser Kategorie werden alle genannten Strategien gespeichert, bei denen ohne Konzept vorgegangen wird.	*„Und dann schreibe ich Pseudocodes und dann gebe ich am Schluss den richtigen Code ein und teste dann aus. Und dann danach fange ich eigentlich an herum zu töggeln. Das ist dann halt nicht mehr so schön, wie man es machen sollte.“ (TN 8)*

	Planung auf Papier	In diese Kategorie kommen alle Kommentare, bei denen das Ablaufdiagramm, oder der Schreibtischtest bei der Erstellung genannt werden.	*„Meistens wenn ich sehe, dass es eine komplexe Aufgabe ist, mache ich eine Skizze, zeichne etwas auf das die Aufgabenstellung zeigt und dann schreibe mal von Hand auf, was heraus kommen sollte am Schluss. Und dann schreibe ich Pseudocodes und dann gebe ich am Schluss den richtigen Code ein und teste dann aus.“ (TN 8)*
Kognitive Strategien des Programmverständnisses		Kommentare zu kognitiven Strategien der Evaluation und Erweiterung werden in den Subkategorien dieser Hauptkategorie zugeordnet.	
	Lesen von Kommentaren	In diese Kategorie kommen alle Kommentare, bei denen das Lesen von Kommentaren bei der Programmevaluation/-erweiterung genannt werden.	*„Also ich erinnere mich noch, in der zweiten Unterrichtsstunde als der Dozent meinte, man solle doch Kommentare einfügen, da Kommentare wichtig sind für Leute die den Code nicht geschrieben haben, damit sie ihn eben verändern oder anpassen können. Ich denke es ist einfacher den Code von Grund auf selber zu schreiben. Vorausgesetzt der andere hat quasi nicht ausreichend Kommentare in sein Programm eingefügt. Also ich weiss nicht genau wie sein Programm aufgebaut ist, dann dauert es erstmal eine Zeit lang, um sich damit auseinanderzusetzen und das Programm vollends zu verstehen. Von dem her würde ich sagen, aus meiner Sicht, ich fände es einfacher den Code von vorneherein selber zu schreiben.“ (TN 3)*
	Schrittweise-Abarbeiten-Strategie	In dieser Kategorie werden alle genannten Strategien gespei-	*„Mein allgemeiner Vorgang ist, ich versuche mal etwas, so eine Grundidee und dann schaue ich was nicht funktioniert. Und man lernt*

		chert, bei denen der Code von oben nach unten bearbeitet wird.	*es ja so dass, man sollte sich alles auf Papier aufschreiben so ein Ablaufdiagramm oder so. Ich bin immer so, ich fange dann halt mit dem Grundgerüst an, was brauche ich überhaupt und dann schaue ich welche von denen ich ganz einfach schreiben, also zum Beispiel kann ich oft variable Deklaration ist dann einfach gemacht. Und dann habe ich so verschiedene Methoden zum Beispiel und dann geh ich also eigentlich Stück, ja, dann arbeite ich diese einfach ab und schaue wo es ein Problem gibt und dort wo es Probleme gibt... ja, das Problem ist, oft findet man einfach keine Lösung. Dann hilft es oft eine Pause zu machen und dann nochmals quasi neu darauf zu schauen, ja." (TN 3)*
	Prüfen der Syntax	In diese Kategorie kommen alle Kommentare, bei denen das Prüfen der Syntax bei der Programmevaluation/-erweiterung genannt werden.	*„Wir haben zweier Zweiergruppen die zusammenarbeiten und teilweise lösen wir denn halt Aufgaben unabhängig voneinander und es kommt dann heraus, dass man ganz ein anderes Resultat hat am Schluss. Und wenn du dann halt siehst, bei deinem Kollegen läuft irgendetwas nicht und dann musst du da den Fehler finden, dann ist nicht so schwierig als wenn Sachen anders formuliert sind, als dass man es selbst macht, dann braucht es echt lange bis man versteht, ah, was wollte er eigentlich machen. Wir schauen dann meistens eigentlich jeweils die offensichtlichen Themen an. Liegt es an einem Semikolon, oder ob man etwas nicht initialisiert hat, oder keinen Wert gegeben, oder ja. Dem würde ich eigentlich sagen eigentlich Flüchtigkeitsfehler. Man weiss ja, dass man dies machen sollte, aber sie sind halt doch oftmals da. Oder ganz wichtig sind auch Klammern. Dass man die richtig setzt. Denn wenn eine Klammer nicht richtig*

			gesetzt ist, dann geht einfach gar nichts mehr." (TN 1)
Erfolgreiche kognitive Strategien		In dieser Kategorie werden alle Kommentare zum Vorteil eingesetzter kognitiver Strategien gesammelt.	*„Es hilft halt einfach einen Überblick über die Fragestellung zu bekommen. Weil, wenn man einfach nur halt den ausformulierten Text hat, was genau gemacht werden soll, dann ist es schwierig zu wissen grad direkt, wie man daraus einen Code macht. Und wenn man halt ein Ablauf-Diagramm hat wo steht, also zuerst die Ausgangssituation, nächster Schritt und so, dann hilft es einfach die Sache besser zu strukturieren." (TN 5)*
Nicht erfolgreiche kognitive Strategien		In dieser Kategorie werden alle Kommentare zum Nichtgelingen eingesetzter kognitiver Strategien gesammelt.	*„Also, ich... Es ist am Anfang wirklich nicht gegangen und dann habe ich einfach so fünf Minuten so Auszeit gemacht und dann nochmals probiert und dann..." (TN 2)*
Fehlendes Wissen betreffend Vorgehensweisen		In diese Kategorie kommen alle Kommentare, bei denen fehlendes Wissen bezüglich Vorgehensweise genannt wird.	*„Also wir haben zum Beispiel gelernt was Schleifen sind, ja. Also es ist meistens so, dass wir den Code entwickeln und nicht den Code anschauen und denn wissen was machen. Wobei es ja logisch ist, dass man das macht, wenn man etwas programmiert." (TN 1)"*
Herkunft kognitiver Strategien		In den Subkategorien dieser Kategorie werden alle Kommentare zur Herkunft eingesetzter kognitiver Strategien gesammelt.	

	Ausprobieren	In diese Kategorie kommen alle Kommentare, bei denen Ausprobieren als Herkunft genannt werden.	*„Und sonst habe ich es nicht gelernt und habe einfach rausgefunden, dass es für mich am besten funktioniert, wenn ich einfach mal anfange und schaue, wo gibt es Probleme. Weil eigentlich theoretisch hätte ich gelernt, dass man anders vorgeht, dass man alles auf Papier, alles systematisch macht. Aber das ist für mich, das Ganze auf Papier aufschreiben, ist schon zu viel zu Aufwand. Ich schaue lieber wo gibt es Probleme, ja.“ (TN 2)*
	Gemeinsamer Austausch	In diese Kategorie kommen alle Kommentare, bei denen der gemeinsame Austausch als Herkunft genannt werden.	*„Also, wir haben im jetzigen Kurs sowieso ein Programmier-Buddy-System, das heisst eigentlich, dass wir immer zu zweit programmieren sollten und von daher ist es so, dass man andere Leute auch immer fragt und den Code austauscht.“ (TN 2)*
	Lehrmittel aus dem Unterricht	In diese Kategorie kommen alle Kommentare, bei denen Lehrmittel aus dem Unterricht als Herkunft genannt werden.	*„Das Ablaufdiagramm? Im Unterricht einfach. Wir haben ein Lehrbuch und dort drin ist es ziemlich gut erklärt.“ (TN 1)*
	Unterricht	In diese Kategorie kommen alle Kommentare, bei denen der Unterricht als Herkunft genannt werden.	*„Wir haben es bei unserem Dozenten schon so gelernt Schritt für Schritt vorzugehen. Er zeichnet sogar noch am Anfang so ein Ablauf-Diagramm. Das mache ich eigentlich meistens nicht, obwohl es eigentlich sehr gut wäre, wenn man so etwas machen würde. Aber so Schritt für Schritt haben wir bei ihm im Unterricht gelernt. Wir haben auch Programmier-Aufgaben mit ihm zusammen gelöst, wo er gesagt hat, wie er halt Schritt für Schritt vorgeht.“ (TN 5)*

Voraussetzungen um Programmieren zu lernen		In den Subkategorien dieser Kategorie werden alle Kommentare zur Voraussetzungen um Programmieren zu lernen gesammelt.	
	Logisches Denken	In dieser Kategorie werden Aussagen gespeichert, welche logisch zu Denken als Voraussetzung nennen.	*„Ja, dann analytisches und logisches Denken. Und ja, technisches Verständnis und so ein wenig sequentielles Denken.“ (TN 8)*
	Sprachkenntnisse	In dieser Kategorie werden Aussagen gespeichert, welche Sprachkenntnisse als Voraussetzung nennen.	*„Ich glaube die Rechtschreibung ist auch so ein Punkt. Die meisten Programmiersprachen sind ans Englische angelehnt. Es wäre von Vorteil, wenn man die Wörter richtig schreibt, weil sonst gibt es immer Fehlermeldungen.“ (TN 4)*
	Mathematikkenntnisse	In dieser Kategorie werden Aussagen gespeichert, welche Mathematikkenntnisse als Voraussetzung nennen.	*„Ich denke, es braucht nicht Mal so viel... Ja, je nach dem... Ich nehme an, für viele Leute die programmieren... Es braucht ja nicht eine wahnsinnig hohe Intelligenz, also dass man irgendwie überintelligent sein muss. Zum Beispiel, mein Mathelehrer, der programmiert irgendwie Bildbearbeitung. Da muss man... Es ist zwar schon noch kompliziert. Er sagte, man braucht ein Mathe-Studium, damit man es machen kann, um es zu verstehen, die Algorithmen und so. Und ich denke aber, weiss nicht... Für irgendein Webapp... Es kommt einfach ein wenig darauf an, was man will.“ (TN 9)*

	Persönliche Eigenschaften	In dieser Kategorie werden Aussagen gespeichert, welche persönliche Eigenschaften als Voraussetzung nennen.	*„[...] und auch eine gewisse Geduld haben, weil oft wenn man etwas eingegeben hat, fehlt vielleicht ein Komma und bis man das gefunden hat, man muss Geduld haben. Und sonst, ich weiss, wenn man logisch vorgeht, dann wäre das machbar.“ (TN 6)*
	Motivation	In dieser Kategorie werden Aussagen gespeichert, welche Motivation als Voraussetzung nennen.	*„Was sehr wichtig ist, dass man wirklich unbedingt Interesse am Programmieren haben muss. Es muss dich wirklich interessieren, denn sonst bringt es nichts, wenn du denkst „äh, das brauche ich ja doch nicht“ dann hast du auch keine Motivation zu lernen.“ (TN 7)*
	Informatikkenntnisse	In dieser Kategorie werden Aussagen gespeichert, welche Informatikkenntnisse als Voraussetzung nennen.	*„Manchmal sind es ja wirklich ganz kleine Sachen. Und ja, dass man allgemein gerne mit dem Computer umgeht und so.“ (TN 5)*
Ideen zur Vermittlung der Programmierung		In den Subkategorien dieser Kategorie werden sämtliche Ideen zum Vermitteln der Programmierung gesammelt.	
	Frühes Beginnen mit dem Programmierunterricht	In dieser Kategorie werden alle Kommentare gesammelt, bei denen ein früher Lernbeginn genannt wurde.	*„Es ist halt schwierig, weil es braucht so eine gewisse Art an Probleme heranzugehen. Es ist nicht wie im normalen Alltag wo man die vorgegebenen Schemen hat, sondern man muss wie umdenken. Und ich denke, das ist vor allem das Schwierigste. Und ja, wenn halt früh bereits anfängt, in der Schule bringt es sicher schon viel. Ich glaube das wird jetzt auch so gemacht, dass man früher anfängt.*

			Weil ja, man soll ja einfach reinkommen in die Denkensweise.“ (TN 1)
	Gamification	In dieser Kategorie werden alle Kommentare gesammelt, bei denen das Lehren auf spielerische Weise genannt wurde.	*„Ich würde wahrscheinlich sagen, fang einfach und langsam an, sonst wirst Du ziemlich schnell die Lust daran verlieren, wenn es gleich zu komplex wird. So ich denk, ein spielerischer Umgang mit Programmiersprachen trägt dazu bei, eben die Lust, sag ich mal, am Programmieren zu erhalten auch wenn die Programme unübersichtlich und komplexer werden. Ja, ich würde wahrscheinlich auch darauf hinweisen, dass es eben mit einem eigenen Programm anfängt und sich so sukzessive ranarbeitet. Dann... ich weiss nicht, kennen Sie Turtle-Graphics? [...]“ (TN 3)*
	Unterschiedliche Strategien aufzeigen	In dieser Kategorie werden alle Kommentare gesammelt, bei das Aufzeigen von unterschiedlichen Strategien genannt wurde.	*„Aber ich finde beim Programmieren das Wichtigste, dass man herausfindet was für einem selber funktioniert und dann, ja, vielleicht dass man im Unterricht sagen würde, ja, es gibt die und die Methode wie man es machen kann, also die und die Herangehensweise und dann, dass man für sich selber herausfindet und ein wenig experimentieren muss, was für einem am Angenehmsten ist. Weil ich finde es schwierig, wenn einem im Unterricht gesagt wird, sie müssen so oder so vorgehen. Andererseits es muss ja dann oft auf eine Artgeprüft werden und wir haben jetzt zum Beispiel oft geschriebene Prüfungen beim Programmieren und dann muss man ja fast sagen sie müssen so oder so vorgehen, weil sie wollen ja dann die Vorgehensweise auch an der Prüfung haben. Und das finde ich auch... weil ja, es wäre ja nicht unbedingt hilfreich, wenn sie für jeden an-*

			ders gut funktioniert.“ (TN 2)
	Planung auf Papier aufzeigen	In dieser Kategorie werden alle Kommentare gesammelt, bei das Aufzeigen von Planung auf Papier wurde.	*„Ich würde einfach sagen, das was für mich am besten funktioniert und dann würde ich sagen, dass für den Anfang, würde ich aber schon empfehlen, dass man ganz so Schritt für Schritt vorgeht und eben auch auf Papier mal plant, und dass man dann von dort her anfängt und aber, dass man mit der Zeit herausfinden muss was für einem selber am besten ist, und ja.“ (TN 2)*
	Logik und Abläufe aufzeigen	In dieser Kategorie werden alle Kommentare gesammelt, bei das Aufzeigen von Logik und Abläufen genannt wurde.	*„Dass man wirklich Schritt für Schritt denken muss. Man hat manchmal das Gefühl, dass man das und das machen muss, noch jeden, und alles gleichzeitig und das finde ich am Anfang fast nicht möglich. Man muss wirklich für jedes Ding einen eigenen Abschnitt schreiben und dann zusammensetzen, nicht alles in eines rein wursteln.“ (TN 6)*
	Selbstständiges Erarbeiten	In diese Kategorie kommen alle Kommentare, bei denen das selbstständige Erarbeiten positiv und negativ genannt wird.	*„Gut, da würde ich mal sagen, in dem Kurs, bei dem Dozenten... Er macht es so, zuerst haben wir halbe Zeit Theorie und halbe Zeit bleiben wir noch in der Schule und haben Zeit zu programmieren. Und Vorteil ist, man kann ein bisschen probieren und wenn man Probleme hat, kann man zu ihm nach vorne gehen und fragen. Ich habe das Gefühl, programmieren lernt man nur durch anwenden und selber schreiben, testen, herumtüfteln. Beim Kurs vorher habe ich gemerkt, der Nachteil ist, wenn einer vor dir steht und sagt, jetzt schreiben wir das und das. Du schreibst eben nur ab. Ich habe das*

			Gefühl, ich habe nicht viel gelernt.“ (TN 4)
	Gemeinsames Erarbeiten	In diese Kategorie kommen alle Kommentare, bei denen das gemeinsame Erarbeiten genannt wird.	*„Was mir auch oft hilft, ist es mit anderen Leuten zu besprechen. Ich habe es dann eigentlich noch mit einer Kollegin angeschaut und wir sind dann zusammendraufgekommen, ich haben einen Teil, den sie nicht konnte und sie hat mir dann noch den Teil gesagt, den ich noch brauchte eigentlich um es zu Programmieren.“ (TN 2)*
	Klassische Lernmedien	In dieser Kategorie werden alle Kommentare gesammelt, bei denen klassische Lernmedien genannt wurde.	*„Ich kann nur für mich sprechen. Ich fand das in allem sehr gut. Es gab zu jeder Unterrichts-Stunde ein Kapitel im Buch. Also die Theorie die man zuerst lesen sollte. Anschliessend 4 bis 5 Aufgaben die man mit der Theorie bearbeiten konnte und ich fand das schon ziemlich gut.“ (TN 3)*
	Online Lernmedien	In dieser Kategorie werden alle Kommentare gesammelt, bei denen online Lernmedien genannt wurde.	*„Ich würde einfach einmal die Websites, welche ich kenne, empfehlen, weil man da die Programmiersprache ein wenig kennenlernen kann. Und dann würde ich zum Beispiel das als Aufgabenstellung anschauen.“ (TN 8)*
	Hilfsmittel	In dieser Kategorie werden alle Kommentare gesammelt, bei denen Hilfsmittel beschrieben wurden.	*„Unterricht, es ist halt so, man kann es nicht von heute auf morgen. Mein Vorschlag für die Zukunft wäre so eine richtig gute Software zu entwickeln so für Anfänger.“ (TN 7)*

E. Feedbackformular Pretest

Feedback zum Fragebogen

Persönliche Angaben

Geschlecht: ☐ m ☐ w
Alter:
Beruf:

Zum Fragebogen

Wie viel Zeit haben Sie für die Durchführung des Fragebogens benötigt?: _____ Minuten

Haben Sie den Fragebogen komplett ausgefüllt?
☐ Ja ☐ Nein, weil

_____

Auf welchem Gerät haben Sie den Fragebogen ausgefüllt?
☐ Computer ☐ Laptop ☐ Tablet ☐ Smartphone ☐ Andere:

_____

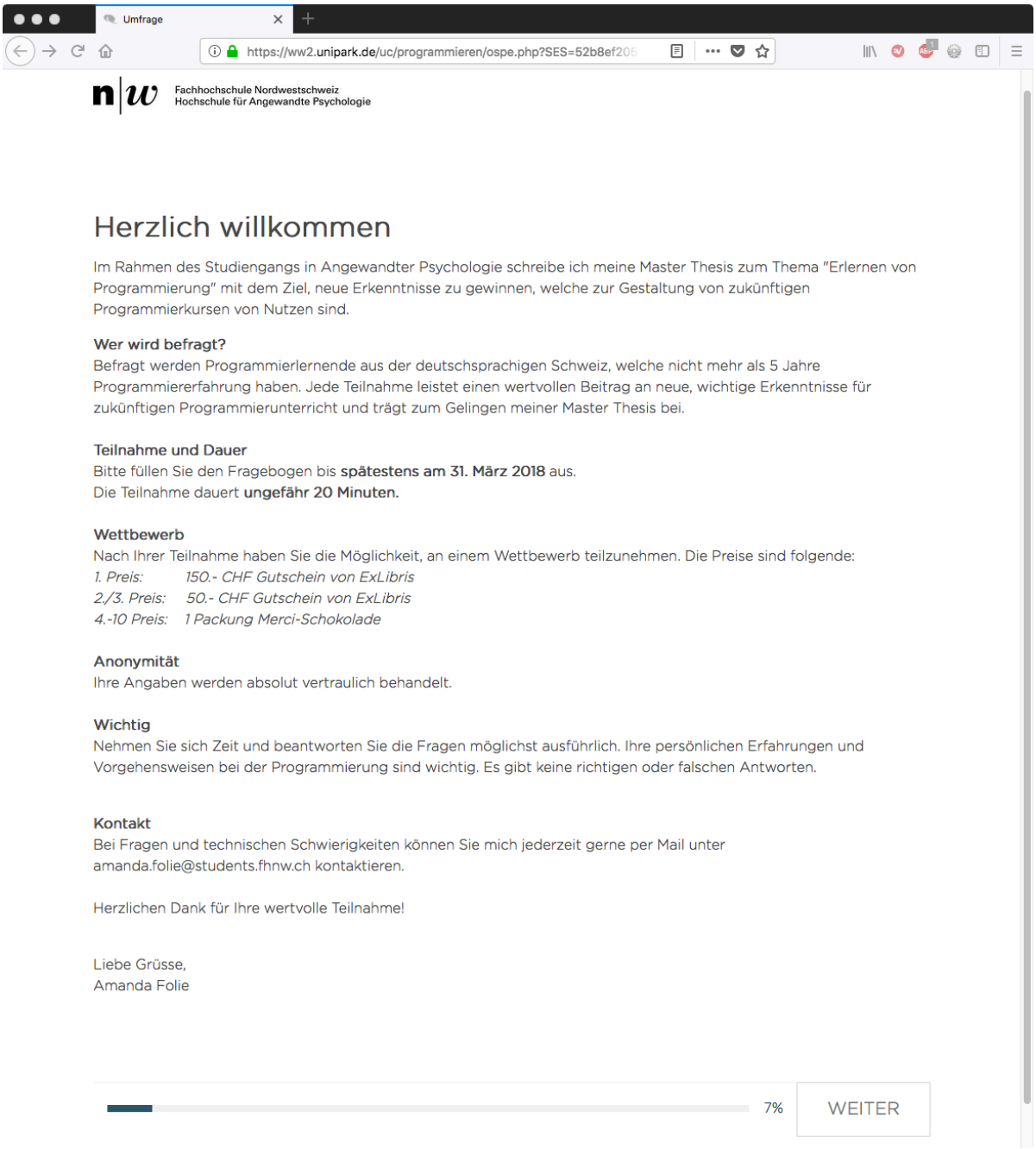
Hatten Sie technische Schwierigkeiten während der Durchführung des Fragebogens?:
☐ Ja ☐ Nein *Wenn ja*, welche?

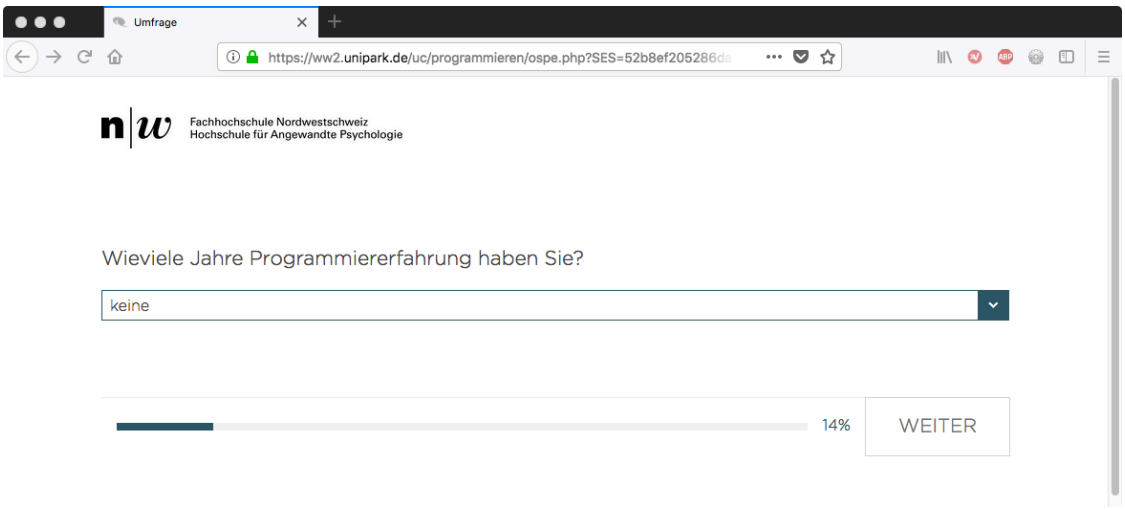
_____

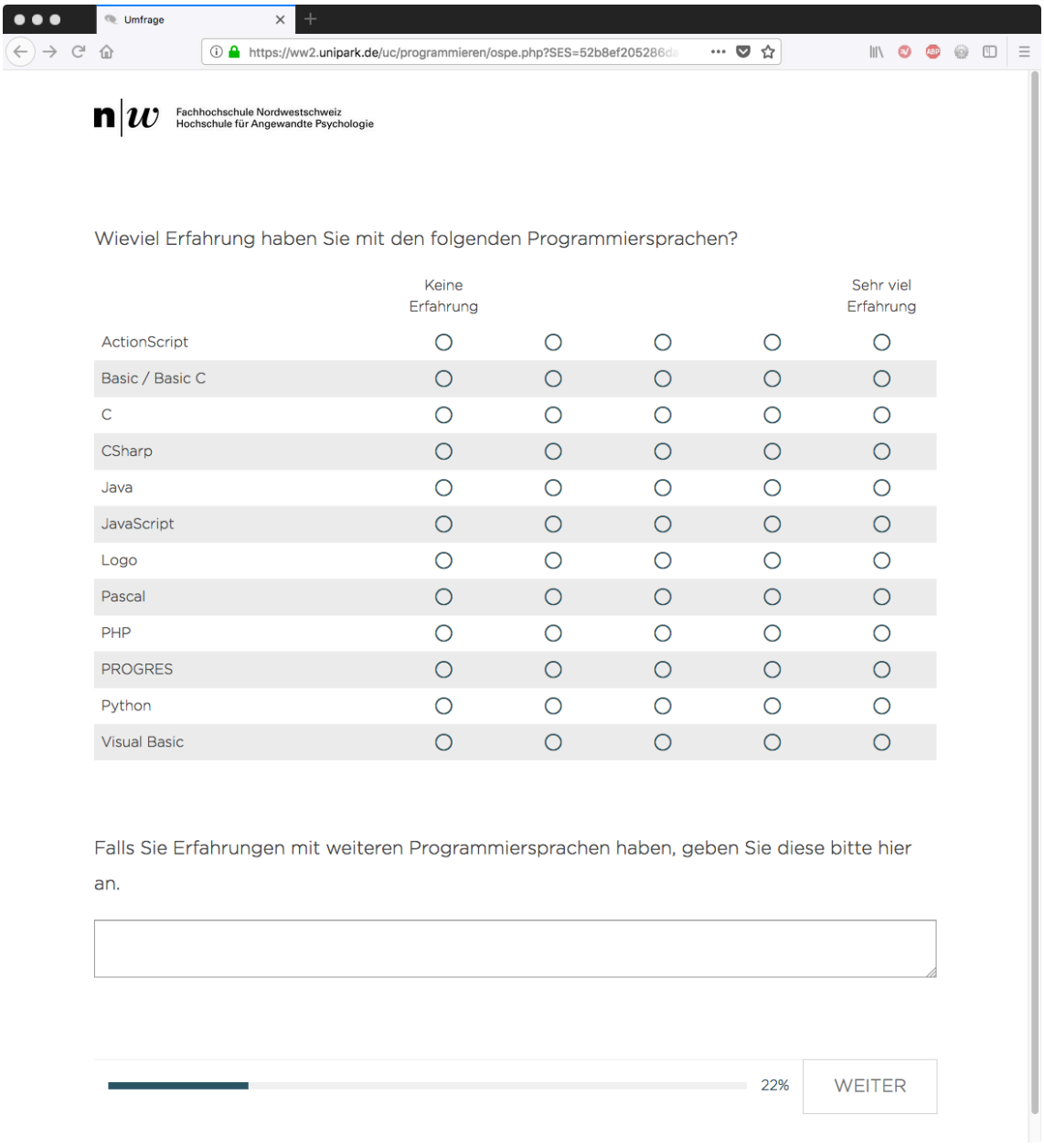
Waren alle Fragen verständlich?: ☐ Ja ☐ Nein
Falls nein, welche Fragen waren nicht gut verständlich und weshalb?

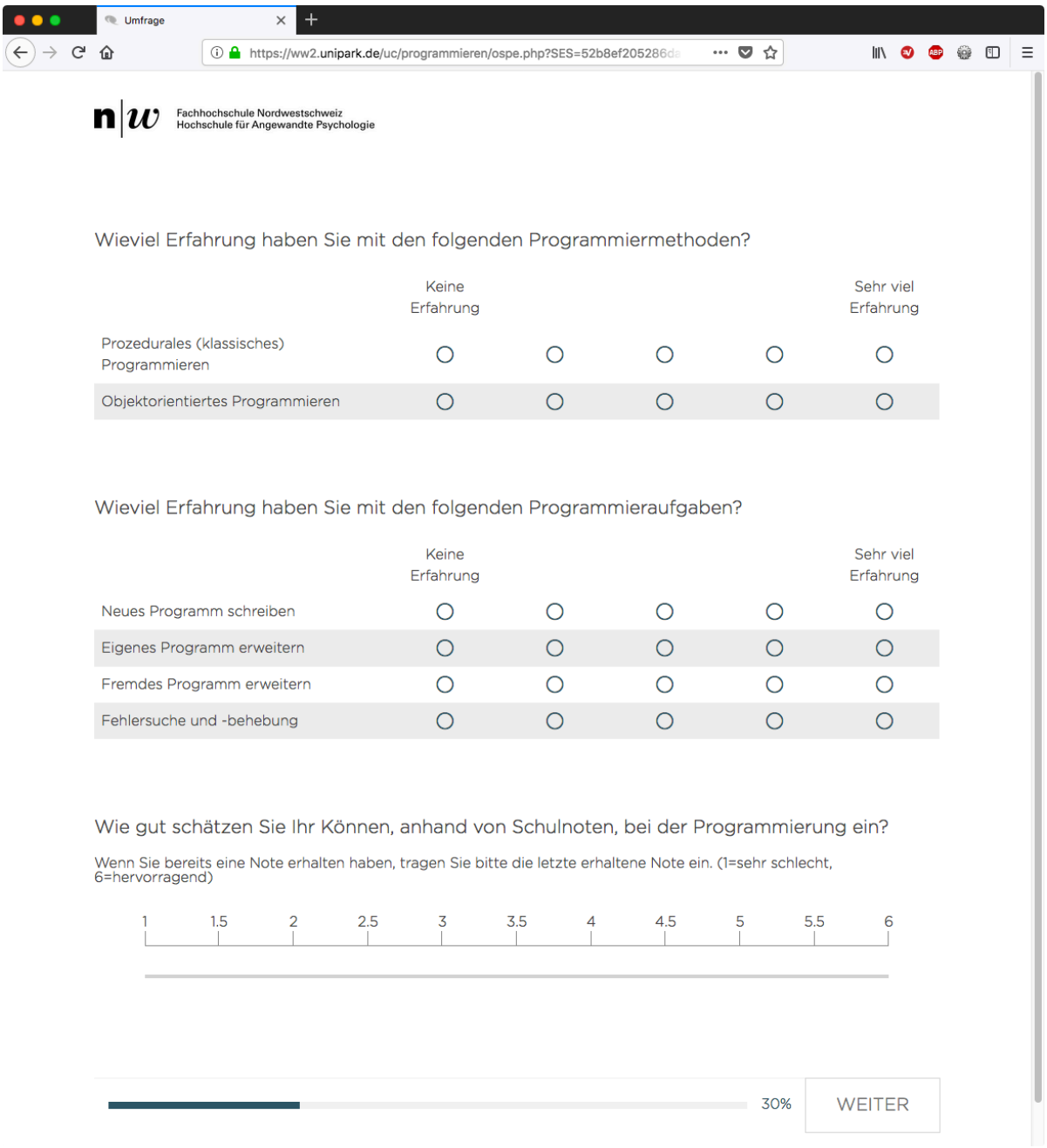
_____
_____
_____
_____
_____

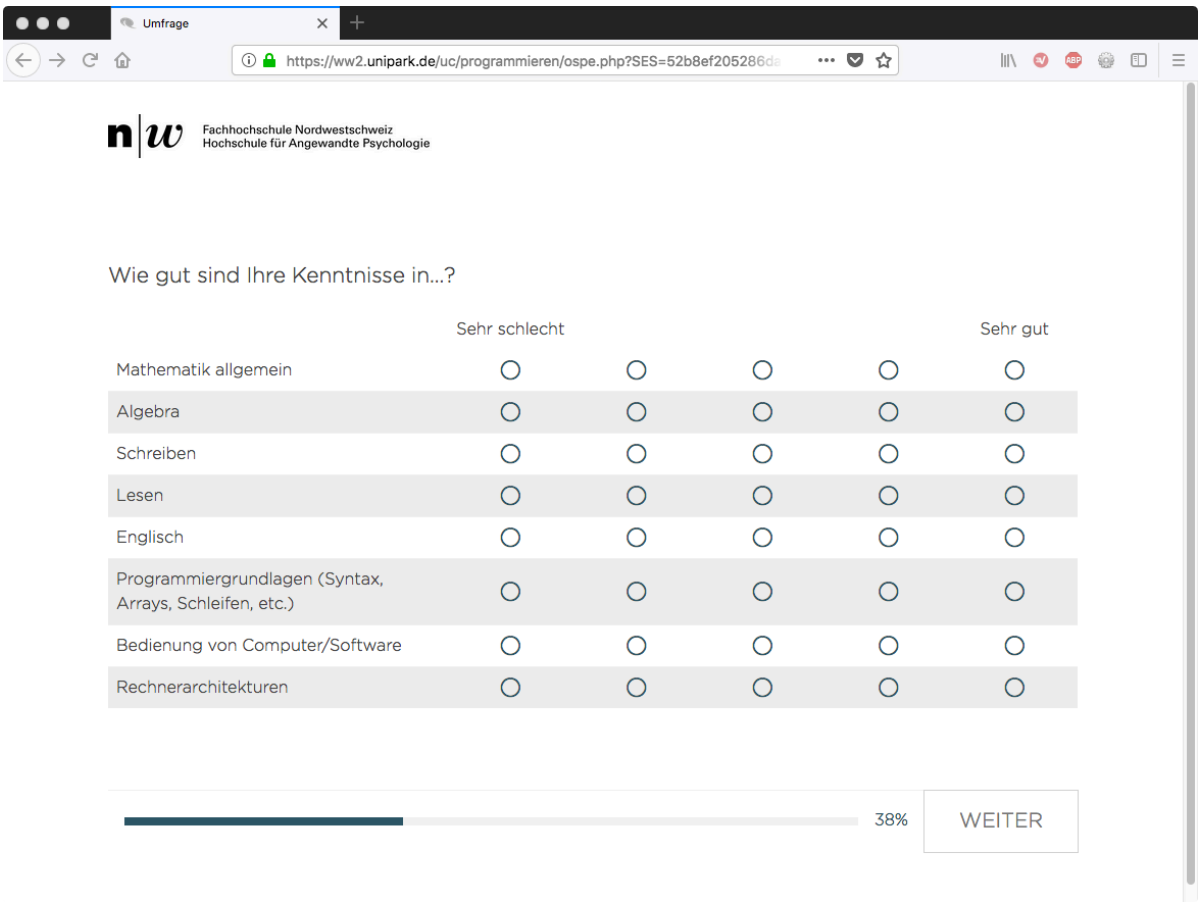
F. Online Fragebogen

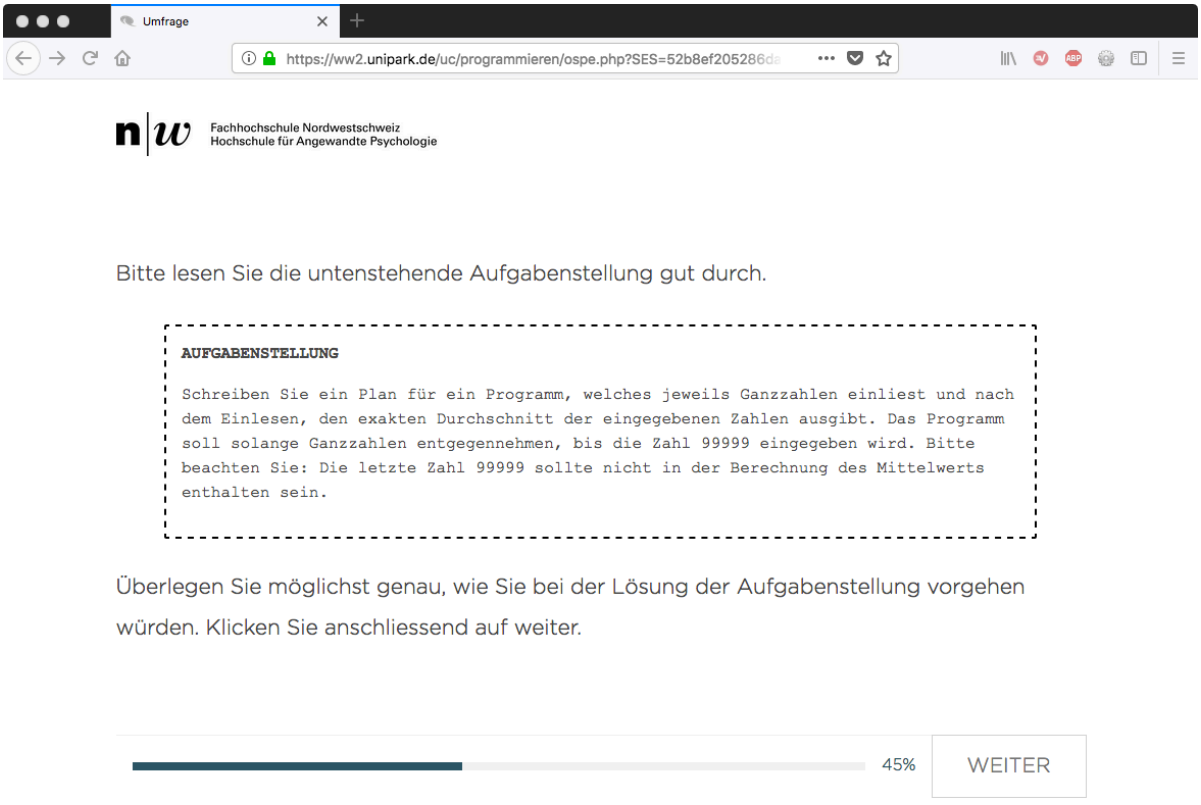


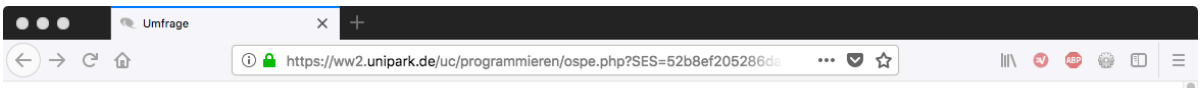

Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Wieviele Jahre Programmiererfahrung haben Sie?
keine
14%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Wieviel Erfahrung haben Sie mit den folgenden Programmiersprachen?
Keine Erfahrung
Sehr viel Erfahrung
ActionScript
Basic / Basic C
C
CSharp
Java
JavaScript
Logo
Pascal
PHP
PROGRES
Python
Visual Basic
Falls Sie Erfahrungen mit weiteren Programmiersprachen haben, geben Sie diese bitte hier an.
22%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Wieviel Erfahrung haben Sie mit den folgenden Programmiermethoden?
Keine Erfahrung
Sehr viel Erfahrung
Prozedurales (klassisches) Programmieren
Objektorientiertes Programmieren
Wieviel Erfahrung haben Sie mit den folgenden Programmieraufgaben?
Keine Erfahrung
Sehr viel Erfahrung
Neues Programm schreiben
Eigenes Programm erweitern
Fremdes Programm erweitern
Fehlersuche und -behebung
Wie gut schätzen Sie Ihr Können, anhand von Schulnoten, bei der Programmierung ein?
Wenn Sie bereits eine Note erhalten haben, tragen Sie bitte die letzte erhaltene Note ein. (1=sehr schlecht, 6=hervorragend)
1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6
30%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Wie gut sind Ihre Kenntnisse in...?
Sehr schlecht
Sehr gut
Mathematik allgemein
Algebra
Schreiben
Lesen
Englisch
Programmiergrundlagen (Syntax, Arrays, Schleifen, etc.)
Bedienung von Computer/Software
Rechnerarchitekturen
38%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Bitte lesen Sie die untenstehende Aufgabenstellung gut durch.
AUFGABENSTELLUNG
Schreiben Sie ein Plan für ein Programm, welches jeweils Ganzzahlen einliest und nach dem Einlesen, den exakten Durchschnitt der eingegebenen Zahlen ausgibt. Das Programm soll solange Ganzzahlen entgegennehmen, bis die Zahl 99999 eingegeben wird. Bitte beachten Sie: Die letzte Zahl 99999 sollte nicht in der Berechnung des Mittelwerts enthalten sein.
Überlegen Sie möglichst genau, wie Sie bei der Lösung der Aufgabenstellung vorgehen würden. Klicken Sie anschliessend auf weiter.
45%
WEITER



Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie

Zum Lösen der Aufgabenstellung gehe ich folgendermassen vor...

Bitte kreuzen Sie jeweils die zutreffenste Antwort an.

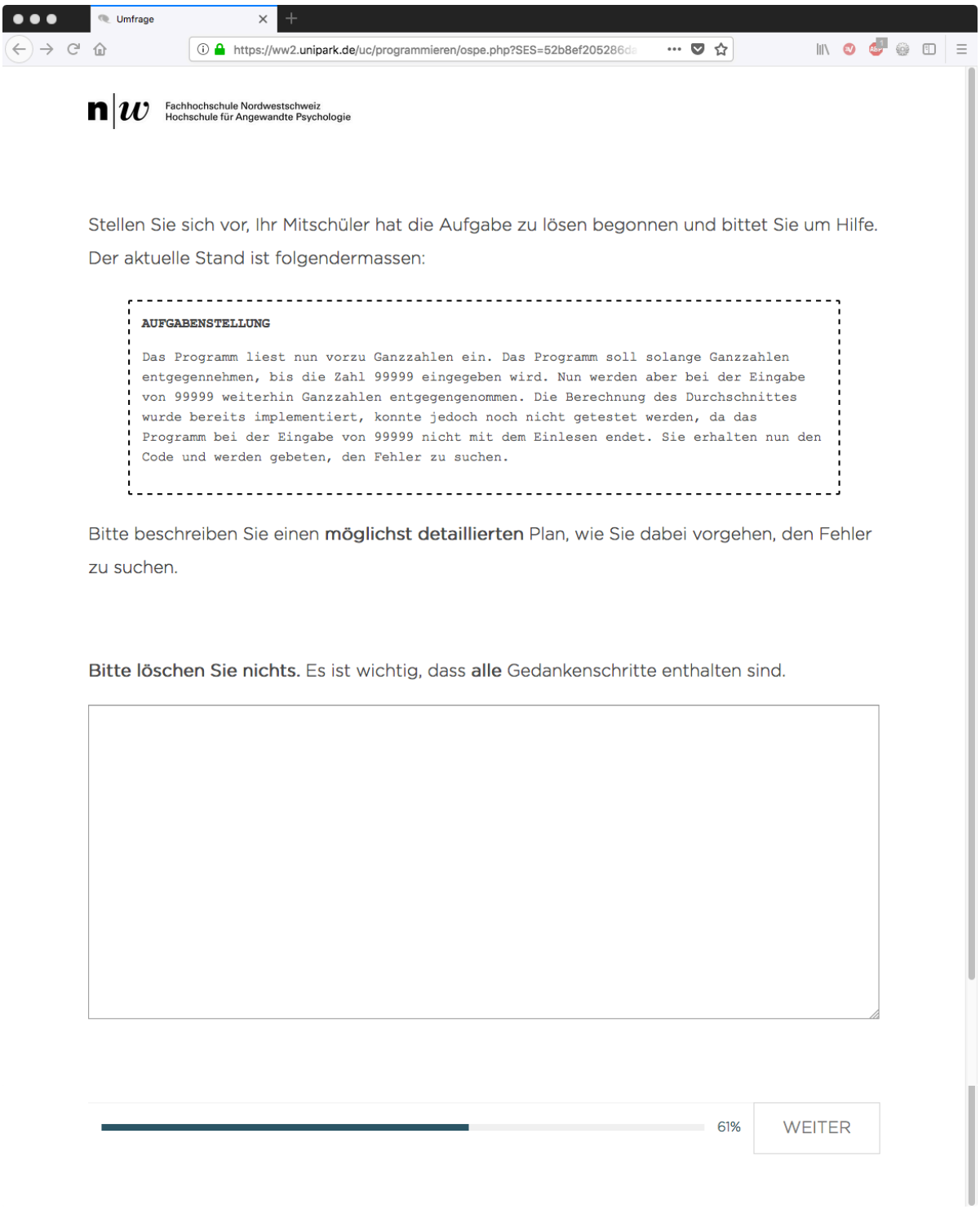
	nie	selten	gelegentlich	oft	immer
Ich erstelle ein Ablaufdiagramm	○	○	○	○	○
Ich schreibe Pseudocode	○	○	○	○	○
Ich mache einen Schreibtischtest	○	○	○	○	○
Ich arbeite den Code von oben nach unten ab	○	○	○	○	○
Ich versuche Zusammenhänge zu verstehen	○	○	○	○	○
Ich probiere verschiedene Möglichkeiten aus	○	○	○	○	○

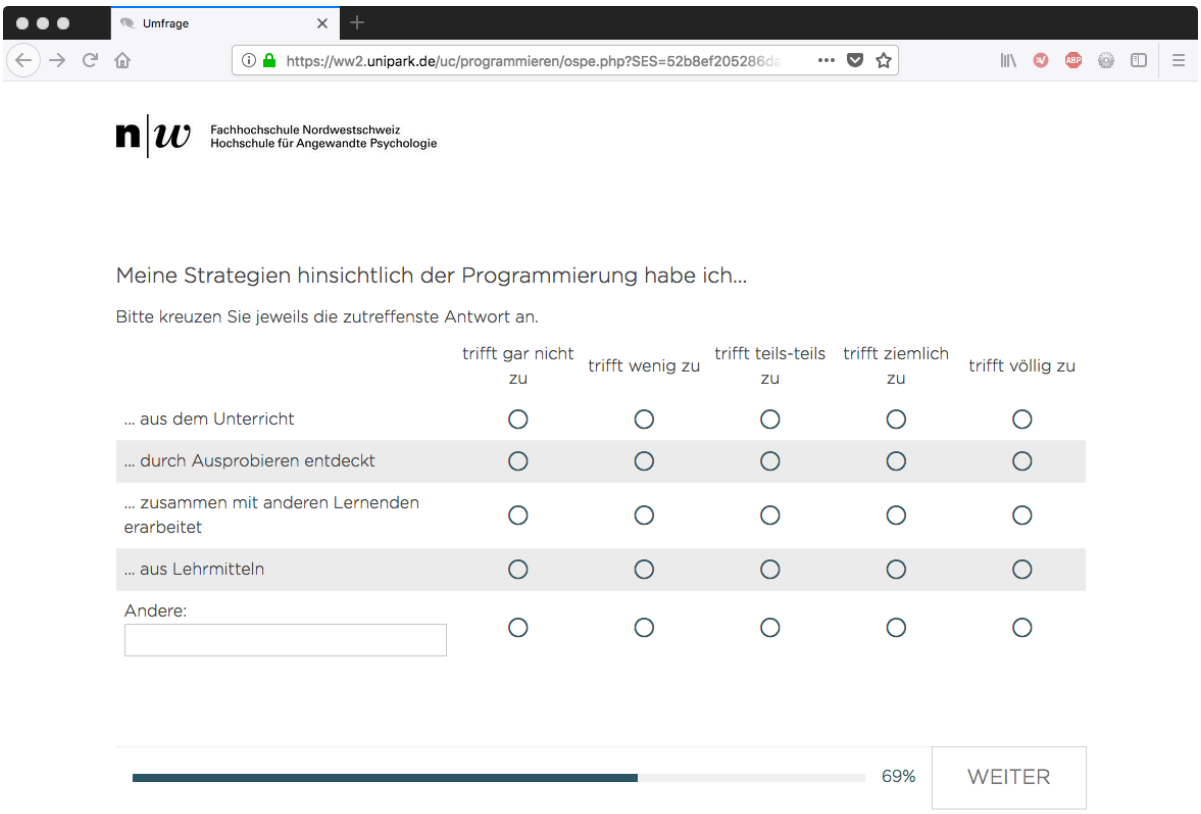
Zu Beginn beim Lösen der Aufgabe mache ich mir Gedanken zu...

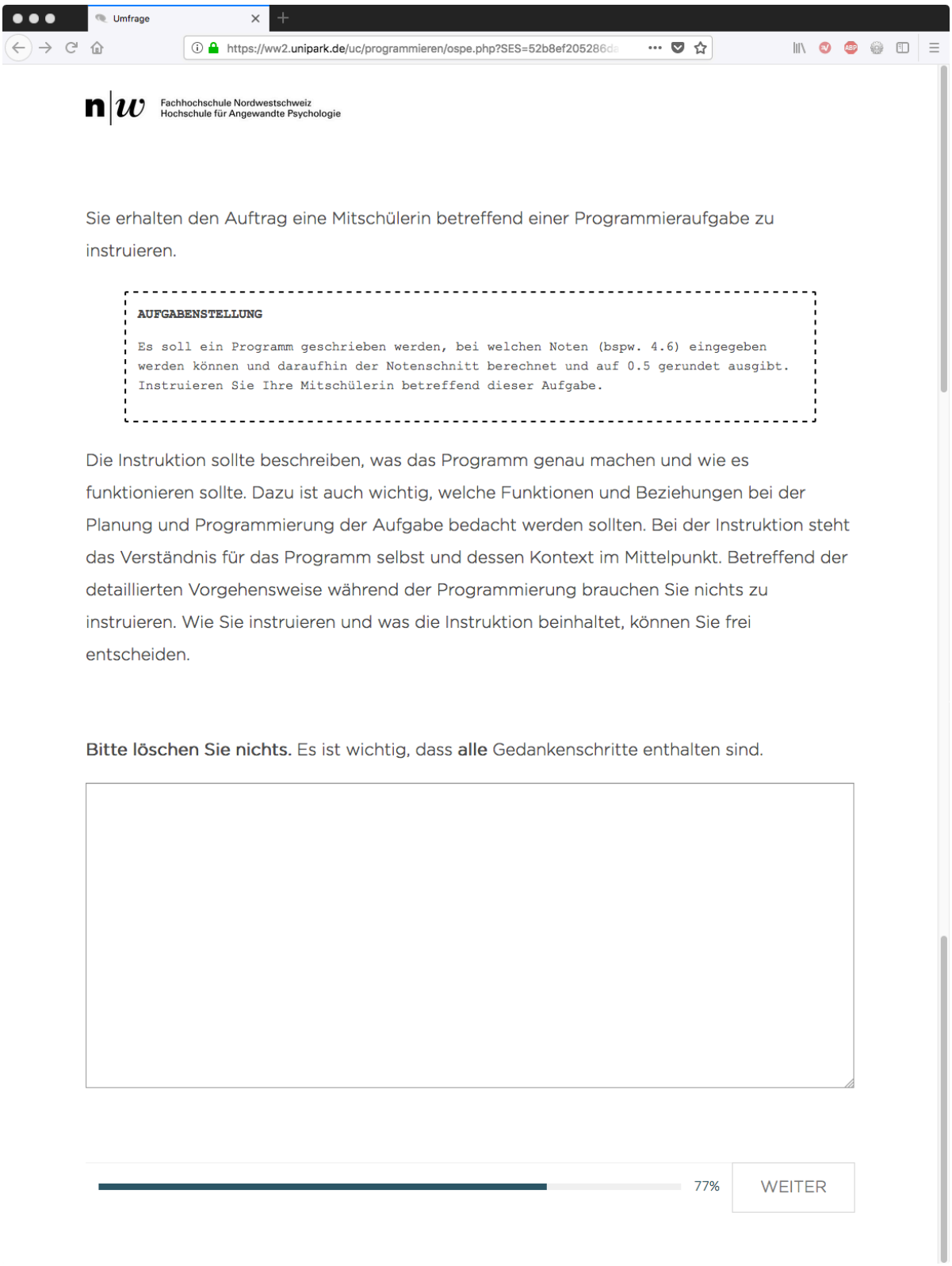
Bitte kreuzen Sie jeweils die zutreffenste Antwort an.

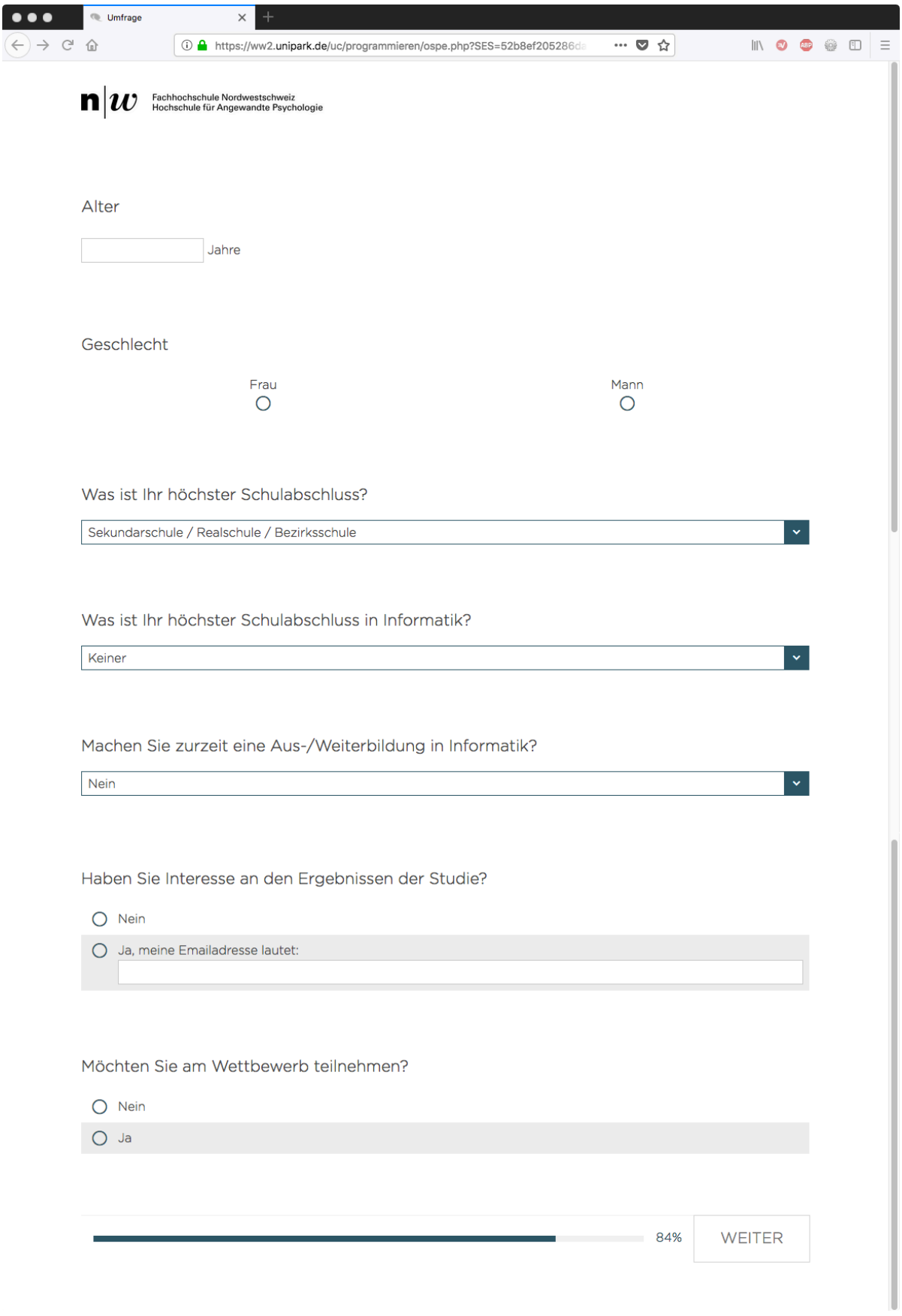
	nie	selten	gelegentlich	oft	immer
... den Operationen und Aktionen, welche einzelne Funktionen beinhalten	○	○	○	○	○
... den benötigten Variabeln, mit Datentypen und Inhalt	○	○	○	○	○
... den genauen Abläufen bei der Programmausführung	○	○	○	○	○
... den Veränderungen von Zusammenhängen und Inhalte von Objekten	○	○	○	○	○
... Zustände von Objekten während der Programmausführung	○	○	○	○	○
... den Hauptzielen des Programmes	○	○	○	○	○
... den Unterzielen des Programmes, welche zum Erreichen des Hauptziels nötig sind	○	○	○	○	○

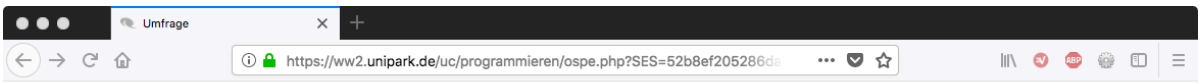
53% WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Stellen Sie sich vor, Ihr Mitschüler hat die Aufgabe zu lösen begonnen und bittet Sie um Hilfe. Der aktuelle Stand ist folgendermassen:
AUFGABENSTELLUNG
Das Programm liest nun vorzu Ganzzahlen ein. Das Programm soll solange Ganzzahlen entgegennehmen, bis die Zahl 99999 eingegeben wird. Nun werden aber bei der Eingabe von 99999 weiterhin Ganzzahlen entgegengenommen. Die Berechnung des Durchschnittes wurde bereits implementiert, konnte jedoch noch nicht getestet werden, da das Programm bei der Eingabe von 99999 nicht mit dem Einlesen endet. Sie erhalten nun den Code und werden gebeten, den Fehler zu suchen.
Bitte beschreiben Sie einen möglichst detaillierten Plan, wie Sie dabei vorgehen, den Fehler zu suchen.
Bitte löschen Sie nichts. Es ist wichtig, dass alle Gedankenschritte enthalten sind.
61%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Meine Strategien hinsichtlich der Programmierung habe ich...
Bitte kreuzen Sie jeweils die zutreffenste Antwort an.
trifft gar nicht zu
trifft wenig zu
trifft teils-teils zu
trifft ziemlich zu
trifft völlig zu
... aus dem Unterricht
... durch Ausprobieren entdeckt
... zusammen mit anderen Lernenden erarbeitet
... aus Lehrmitteln
Andere:
69%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Sie erhalten den Auftrag eine Mitschülerin betreffend einer Programmieraufgabe zu instruieren.
AUFGABENSTELLUNG
Es soll ein Programm geschrieben werden, bei welchen Noten (bspw. 4.6) eingegeben werden können und daraufhin der Notenschnitt berechnet und auf 0.5 gerundet ausgibt. Instruieren Sie Ihre Mitschülerin betreffend dieser Aufgabe.
Die Instruktion sollte beschreiben, was das Programm genau machen und wie es funktionieren sollte. Dazu ist auch wichtig, welche Funktionen und Beziehungen bei der Planung und Programmierung der Aufgabe bedacht werden sollten. Bei der Instruktion steht das Verständnis für das Programm selbst und dessen Kontext im Mittelpunkt. Betreffend der detaillierten Vorgehensweise während der Programmierung brauchen Sie nichts zu instruieren. Wie Sie instruieren und was die Instruktion beinhaltet, können Sie frei entscheiden.
Bitte löschen Sie nichts. Es ist wichtig, dass alle Gedankenschritte enthalten sind.
77%
WEITER


Umfrage
https://ww2.unipark.de/uc/programmieren/ospe.php?SES=52b8ef205286
Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie
Alter
Jahre
Geschlecht
Frau
Mann
Was ist Ihr höchster Schulabschluss?
Sekundarschule / Realschule / Bezirksschule
Was ist Ihr höchster Schulabschluss in Informatik?
Keiner
Machen Sie zurzeit eine Aus-/Weiterbildung in Informatik?
Nein
Haben Sie Interesse an den Ergebnissen der Studie?
Nein
Ja, meine Emailadresse lautet:
Möchten Sie am Wettbewerb teilnehmen?
Nein
Ja
84%
WEITER



Fachhochschule Nordwestschweiz
Hochschule für Angewandte Psychologie

Ende der Befragung

Sie sind nun am Ende der Befragung angelangt. Sie können das Fenster nun schliessen.

Ich danke Ihnen herzlich für Ihre wertvolle Teilnahme.

Beste Grüsse,
Amanda Folie

100%

G. Skalenbildung

Folgende Aufstellung zeigt die Skalen mit den aufgenommenen und weggelassenen Items für die Berechnungen.

Kategorie	Skala/Items	Cronbachs α	Verwendete Items	Weggelassene Items
Vorkenntnisse	Mathematikkenntnisse	.92	Mathematik allgemein	
			Algebra	
	Sprachkenntnisse	.77	Lesen	Englisch (.64)*
			Schreiben	
	Informatikkenntnisse	.61	Programmiergrundlagen (Syntax, Arrays, Schleifen, etc.)	
			Bedienung von Computer/Software	
			Rechnerarchitekturen	
Kognitive Strategien	Konzeptuelle Strategien	.42	Ich erstelle ein Ablaufdiagramm.	Ich versuche Zusammenhänge zu verstehen (.17)*
			Ich schreibe Pseudocode	
			Ich mache einen Schreibtischtest	
	Sequentielle Strategien		Ich probiere verschiedene Möglichkeiten aus	
	Abarbeiten des Codes von unten nach oben		Ich arbeite den Code von oben nach unten ab	

Anmerkungen. Der Wert in Klammern unterhalb der weggelassenen Items entspricht dem Wert von Cronbachs Alpha, wenn das Item in die Skala mitaufgenommen wird.

Verwendete SPSS-Syntax

```
compute Mathematik = mean(v_55, v_54).
execute.

compute Sprachen = mean(v_67, v_56).
execute.

compute Informatik = mean(v_58, v_65, v_66).
execute.

compute konzeptuelle_Strategien = mean(v_91, v_92, v_93).
execute.

compute sequentielle_Strategien = mean(v_104).
execute.
```