Universität Duisburg-Essen

Fakultät für Wirtschaftswissenschaften

Studiengang „Master Lehramt Informatik für Gymnasien/Gesamtschulen“

Konstruktion und Erprobung eines Bausteins zur Kompetenzmessung im Bereich der objektorientierten Programmierung in den Dimensionen Syntax und Semantik

Masterarbeit

zur Erlangung des akademischen Grades eines

Master of Education

vorgelegt von

Mike Barkmin
2272345

Bearbeitungszeitraum: 22.12.2016 - 29.05.2017

Erstgutachter: Prof. Dr. Torsten Brinda, Didaktik der Informatik,
Universität Duisburg-Essen

Zweitgutachter: Dr. David Tobinski, Institut für Psychologie,
Fakultät für Bildungswissenschaften, Universität Duisburg-Essen

Inhaltsverzeichnis

Kurzfassung

Eine der ersten Herausforderungen, denen NovizInnen der Programmierung gegenüberstehen, ist das Beherrschen der Syntax einer Programmiersprache. Um diese Kompetenz zu erfassen und zu bewerten, ist es daher ein lohnendes Ziel über Unterschiede zwischen ExpertInnen und NovizInnen aufzuklären. Erkenntnisse dazu sollen in das Kompetenzmodell der objektorientierten Modellierung einfließen, welches im Rahmen des COMMOOP-Projekts entwickelt wurde.
Diese Arbeit stellt die Konzeption und Erprobung eines Kompetenzmessinstruments für die Dimensionen Syntax und Semantik vor. Auf Basis vorangegangener Forschungen wurde dazu ein Aufgabentyp für die Kompetenzmessung konzipiert und in Form eine webbasierten Tools umgesetzt. Natürlichsprachliche Texte und Quelltexte müssen dabei selbstgesteuert memoriert und reproduziert werden. In einer empirischen Untersuchung mit 42 Testpersonen wurde der Aufgabentyp anschließend erprobt. Dazu wurden jeweils drei Items, die sich in ihrer Länge und kognitiven Komplexität ähneln, zu einer Gruppe zusammengefasst. Insgesamt wurden so drei Gruppen konstruiert, deren Items unterschiedlich lang und unterschiedlich kognitiv anspruchsvoll waren. Außerdem wurden der Einfluss des Arbeitsgedächtnisses und der Aufmerksamkeit sowie der Programmiererfahrung erfasst. Dazu wurden gängige psychologische Test und ein Fragebogen zur Selbstauskunft der Programmiererfahrung eingesetzt.
In der Studie konnten nur für kurze und kognitiv weniger anspruchsvolle Items signifikante Unterschiede zwischen ExpertInnen und NovizInnen in den Differenzen zwischen den natürlichsprachlichen und den informatischen Items bezogen auf die Tippgeschwindigkeit, die zeitlichen Gesamtlänge der Merkphasen und die Anzahl der Merkphasen festgestellt werden. Alleinig die Programmiererfahrung erwies sich als Prädiktor für die Fähigkeit bei der Bewältigung der Aufgabe. Zusätzlich wurde eine qualitative Analyse der Bearbeitung durchgeführt. Die Ergebnisse weisen darauf hin, dass ExpertInnen elaboriertere Merkstrategien sowohl syntaktischer als auch semantischer Art einsetzen. Die Strategien können dem Templating oder Chunking zugeordnet werden. Diese Erkenntnis deckt sich mit den psychologischen Theorien zur Ausbildung von Expertise. Insgesamt erweist sich der Aufgabentyp als geeignet für die Messung und Identifizierung von Unterschieden zwischen ExpertInnen und NovizInnen in den Bereichen Syntax und Semantik. Für ein einsatzfähiges Messinstrument gilt es im nächsten Schritt, Anforderungen an Items herauszuarbeiten, die zur Unterscheidung verschiedener Expertiseniveaus geeignet sind.

Abstract

One of the first challenges that novices of programming are confronted with is to handle the syntax of a programming language. In order to assess and evaluate this competency, it seems promising to clarify differences between experts and novices. The aim is to incorporate knowledge into the competency model of object-oriented modeling developed within the framework of the COMMOOP project.
This thesis presents the conception and testing of a competency measuring tool for the two dimensions syntax and semantics. On the basis of some fundamental research, a specific type of task for competency measurement was conceived and implemented in a web-based tool. Natural texts and source codes have to be memorized and reproduced. Afterwards this web-based tool was tested in an empirical study with 42 participants. For this purpose, three items, which are similar in terms of their length and their cognitive complexity, were grouped. Three groups were constructed, whose items differ in view of their length and cognitive complexity. In addition both, the influence of working memory and attention as well as the programmer's experience were recorded. For this purpose, common psychological tests and a questionnaire for self-evaluation of the programmer's coding experience were used.
In the study, only for short and cognitively less demanding items significant differences between experts and novices could be found between the natural language items and the source codes, in relation to the typing speed, the overall temporal length of the memory phases, and the number of memory phases. Thus, only the programming experience appears to be a reliable predictor of the ability to cope with the task. In addition, a qualitative analysis of the processings was carried out. The results underline that experts use more elaborate memory strategies – on the syntactic and the semantic level. The strategies can be related to templating or chunking theories. This knowledge coincides with the psychological theories for the training of expertise. Overall, the task seems suitable for the measurement and identification of differences between experts and novices in the fields of syntax and semantics. For an operational measuring instrument, the next step is to identify requirements for items which are suitable to distinguish different levels of expertise.

1 Einleitung

Im Folgenden werden die Motivation und der Hintergrund der vorliegenden Arbeit beschrieben. Die Ziele und die Methodik der Arbeit werden im Anschluss erörtert. Abgeschlossen wird dieses Kapitel mit der Vorstellung des Aufbaus der Arbeit.

1.1 Motivation und Hintergrund der Arbeit

Das Beherrschen der Syntax einer Programmiersprache ist eine der ersten Herausforderungen, der NovizInnen der Programmierung gegenüberstehen. Für die eigenständige und flexible Nutzung einer Programmiersprache, sind das Verständnis sowie die Fähigkeit zur Verwendung der Syntax jedoch von zentraler Bedeutung. ExpertInnen hingegen scheinen syntaktische Strukturen verinnerlicht zu haben, sodass ihnen genügend Raum bleibt, sich auf die eigentliche Problemstellung zu konzentrieren. Auch beim Lesen von Quelltexten scheinen ExpertInnen ein schnelles Auffassungsvermögen zu haben, was die Semantik des Quelltextes ist und dies zum Teil auch unabhängig von der Programmiersprache, solange ein bekanntes Programmierparadigma zugrunde liegt. Ist diese Beobachtung auf einen Kompetenzunterschied in den Bereichen Syntax und Semantik zurückzuführen? Oder spielen andere Kompetenzen oder gar kognitionspsychologische Konstrukte wie das Arbeitsgedächtnis eine wichtigere Rolle? Eine Übersicht über potentiell relevante Kompetenzen in der objektorientierten Programmierung kann ein Kompetenzmodell liefern.
Ein Kompetenzmodell stellt einen empirisch abgesicherten Vorschlag dar, wie Wissen und Können in einer Domäne strukturiert sind. Lernen wird in Kompetenzmodellen als aktiver, konstruktiver und selbstgesteuerter Prozess aufgefasst, der die Selbstbefähigung und Eigenverantwortung der Lerner in den Fokus nimmt. Insbesondere im Bereich der Hochschule, in der das eigenständige Organisieren von Lernprozessen ein zentrales Prinzip ist, bietet es sich an, ein Kompetenzmodell zur Orientierung zu entwickeln. Das Ziel des Projektes COMMOOP (siehe Kramer, Hubwieser und Brinda, 2016), in dessen Rahmen diese Masterarbeit erstellt wird, besteht darin, ein Kompetenzmodell für die objektorientierte Programmierung zu entwickeln. Bisher wurde aus der Theorie ein Vorschlag für ein Kompetenzmodell abgeleitet. Derzeit wird daran gearbeitet, diesen Vorschlag empirisch zu validieren. Ein erster Aufgabentyp und entsprechende Items wurden bereits konstruiert. Perspektivisch soll auf Grundlage des Kompetenzmodells ein adaptives Testinstrument entwickelt werden. Ein solches Instrument kann dazu dienen, den Lernstand der Lernenden ideal einzuschätzen und diesen somit die bestmögliche individuelle Förderung zu ermöglichen.

1.2 Ziele und Methodik der Arbeit

Im Rahmen dieser Arbeit soll ein weiterer Baustein zur Validierung des Kompetenzmodells aus dem COMMOOP-Projekt entwickelt und erprobt werden. Dieser Baustein soll die Kompetenz im Umgang mit Syntax und Semantik einer objektorientierten Programmiersprache erheben. Dazu sollen ein webbasierter Aufgabentyp und Items unter Berücksichtigung der Theorie und des Forschungsstands entwickelt werden. In einer empirischen Untersuchung werden zudem

mehrere Kandidaten für geeignete Kennwerte zur Erfassung von Kompetenzunterschieden getestet und verglichen. Für die Aufklärung über die genauere Gestalt der Dimensionen Syntax und Semantik von ExpertInnen, also Personen mit einer hohen Kompetenz in diesen Dimensionen, bedarf es einer qualitativen Komponente der Validierung. Ein Mixed-Methods-Design ermöglicht die Zusammenführung der verschiedenen Sichtweisen im Gesamtkontext der Kompetenzmodellierung. Um den Einfluss kognitionspsychologischer Größen auf die Performanz in der getesteten Kompetenzdimension im Blick zu behalten, werden diese in die Untersuchung miteinbezogen. Diese Arbeit ist daher an der Schnittstelle zwischen Kognitionspsychologie, Informatik und ihrer Didaktik einzuordnen.

1.3 Aufbau

In *Kapitel 2* werden die theoretischen Grundlagen für die Konzeption und Entwicklung des Bausteins gelegt. Zuerst wird im Unterkapitel 2.1 in die nötigen informatischen Grundlagen eingeführt. Dazu werden in Abschnitt 2.1.1 die Syntax und Semantik von Programmiersprachen im Allgemeinen beschrieben, bevor im Abschnitt 2.1.2 ein Überblick über Konzepte der objektorientierten Programmierung und ihre syntaktische Umsetzung in Java gegeben wird. Das Kompetenzmodell, zu dessen Validierung der Baustein entwickelt wird, wird in Abschnitt 2.1.3 in seiner Form und Genese beschrieben. Das Unterkapitel 2.2 ist den psychologischen Grundlagen gewidmet. In den ersten drei Abschnitten werden die Konzepte Arbeitsgedächtnis (2.2.1), Aufmerksamkeit (2.2.2) und Expertise (2.2.3) beschrieben. Anwendung finden diese Konzepte in der Organisation von Informationen im Arbeitsgedächtnis (2.2.4). Im Unterkapitel 2.3 werden verschiedene Metriken zur Bestimmung der kognitiven Komplexität von Quelltexten und natürlichsprachlichen Texten diskutiert. Diese werden bei der Generierung der Items (4.2.2) verwendet. Abgeschlossen wird das Kapitel mit einer Zusammenfassung und Schlussfolgerungen für den weiteren Verlauf dieser Arbeit.
In *Kapitel 3* wird der Forschungsstand zum Zusammenhang von Gedächtnisfähigkeiten und Expertise (3.1) über verschiedene Domänen hinweg vorgestellt. Fortgesetzt wird das Kapitel 3 mit den domänenspezifischen Gedächtnisfähigkeiten von ExpertInnen und NovizInnen der Programmierung. Bei dem in dieser Arbeit entwickelten Aufgabentyp sollen Quelltexte übertragen werden, daher wird in Unterkapitel 3.3 in den Forschungsstand zu kognitiven Prozessen beim Übertragen von Texten eingeführt. Im letzten Unterkapitel 3.4 werden die Kenntnisse aus dem theoretischen Hintergrund und dem Forschungsstand zusammengeführt und Forschungsfragen abgeleitet.
In *Kapitel 4* wird die Konzeption und Methodik der empirischen Untersuchung vorgestellt. Zuerst wird im Unterkapitel 4.1 der gewählte Forschungsansatz begründet, bevor in Unterkapitel 4.2 die verwendeten informatischen und psychologischen Messinstrumente und im Unterkapitel 4.3 die Strukturierung der Untersuchung vorgestellt werden. Wie mit den erhobenen Daten verfahren werden soll, wird im Unterkapitel 4.4 Auswertungsmethodik beschrieben. Zum Abschluss wird im Unterkapitel 4.5 die Entwicklung und Implementierung des informatischen Messinstruments präsentiert.
Das *Kapitel 5* enthält im ersten Unterkapitel eine Beschreibung der Durchführung der empirischen Untersuchung. Dabei wird auf die durchgeführten Pretests in 5.1.1, den Ablauf der Hauptuntersuchung in 5.1.2 und die getesteten Personen in 5.1.3 eingegangen. Im zweiten Unterkapitel 5.2 werden die Ergebnisse zu den einzelnen Forschungsfragen dargestellt. Zum Abschluss werden diese im Unterkapitel 5.3 diskutiert und interpretiert.
In *Kapitel 6* wird die Arbeit zunächst im Unterkapitel 6.1 zusammengefasst, dann wird im Unterkapitel 6.2 ein Fazit im Hinblick auf das Ziel der Arbeit (siehe Unterkapitel 1.2) gezogen, bevor im letzten Unterkapitel ein Ausblick für anknüpfende Forschungsprojekte gegeben wird.

2 Theoretische Grundlagen

In diesem Kapitel wird in die informatischen und psychologischen Grundlagen der Arbeit eingeführt. Dabei werden wesentliche begriffliche Grundlagen aus den Domänen, die in dem in den folgenden Kapiteln konstruierten Kompetenzerfassungsinstrument angesprochen werden, thematisiert. Zunächst werden Definitionen der Begriffe Syntax und Semantik als Arbeitsgrundlage vorgestellt. Die theoretischen Ausführungen zu Kontrollstrukturen und dem Paradigma der objektorientierten Programmierung erfüllen innerhalb der Arbeit verschiedene Zwecke: Sie dienen erstens als Grundlage zur Konstruktion der verwendeten informatischen Items, zweitens wird das nötige Vokabular für qualitative Betrachtungen von Quelltexten dargestellt. Im Unterkapitel zu den kognitionspsychologischen Grundlagen werden zunächst die Komponenten des Arbeitsgedächtnisses und ihre Funktionsweisen präsentiert. Vertieft betrachtet wird dabei das Konzept der Expertise und deren Auswirkung auf das Organisieren von Informationen im Arbeitsgedächtnis. Weitergeführt wird dieses Kapitel mit der Vorstellung von Methoden zur Einordnung der kognitiven Komplexität von Quelltexten und natürlichsprachlichen Texten, die für den empirischen Teil dieser Arbeit relevant sind. Dazu wird in diesem Kapitel die WCC-Metrik vorgestellt. Um eine Vergleichbarkeit zwischen Quelltexten und natürlichsprachlichen Texten erzielen zu können, werden verschiedene Lesbarkeitsindizes vorgestellt. In den weiteren Kapiteln der Arbeit werden diese Methoden für die Generierung von vergleichbaren Items genutzt (siehe Abschnitt 4.2.2). Abgeschlossen wird mit einer Zusammenfassung und Schlussfolgerungen für die weiteren Ausführungen.

2.1 Informatische Grundlagen

Eine Programmiersprache ist eine mögliche Schnittstelle zwischen Mensch und Computer, mit deren Hilfe ein Verfahren zur Problemlösung in einer für den Computer „verständlichen“ Form dargestellt werden kann. Programmiersprachen sind künstliche Sprachen, die durch ihre Syntax und ihre Semantik (2.1.1) definiert werden. Sie folgen verschiedenen Programmierparadigmen. Eines dieser Paradigmen ist die objektorientierte Programmierung, auf deren Konzepte im Abschnitt 2.1.2 vertieft eingegangen wird. Nachfolgend wird in 2.1.3 der Forschungsstand zur Einführung eines Kompetenzmodells für die objektorientierte Programmierung vorgestellt.

2.1.1 Syntax und Semantik von Programmiersprachen

Sowohl in den natürlichen als auch in den formalen Sprachen definieren die Regeln der Syntax den Aufbau von Texten aus den Zeichen einer Sprache. Ohne diese Regeln kann einer Zeichenfolge keine Bedeutung zugeschrieben werden. „Mit der syntaktisch falschen Gleichung ‚(a - (+ 3 = b +‘ wissen wir eben so wenig anzufangen wie mit dem Satz ‚Dem Augen schärft verklären‘.“ (Echtle und Goedicke, 2000, S. 11) Eine korrekte Syntax ist die Voraussetzung dafür, dass die Semantik, also die Bedeutung eines Textes oder eines Programms, festgelegt werden kann.

Syntax

Die formale Beschreibung der Syntax einer Programmiersprache ist im Wesentlichen durch eine kontextfreie Grammatik gegeben. Eine Grammatik ist ein 4-Tupel $G = \{N, T, P, S\}$ bestehend aus den Nichtterminalsymbolen N, den Terminalsymbolen T, den Produktionsregeln P und dem Startsymbol S. In dieser Arbeit werden Syntaxdiagramme zur Beschreibung des kontextfreien Anteils der Syntax verwendet. Dazu wird jedem Nichtterminalsymbol ein gerichteter Graph mit je einer zusätzlichen Eingangs- und Ausgangskante, die die Produktionsregeln darstellen, zugeordnet. Die Knoten des Graphen repräsentieren Symbole der Grammatik. Rechtecke stellen Nichtterminalsymbole und Kreise Terminalsymbole dar (vgl. Claus und Schwill, 2001, S.651f). Beim Übersetzen des Quelltextes in für den Computer ausführbare Befehle wird eine syntaktische Analyse durchgeführt. Dabei wird überprüft, ob der Quelltext der zugrundeliegenden Grammatik entspricht. Ein Quelltext kann allerdings die syntaktischen Überprüfung bestehen und trotzdem nicht ausführbare Befehle enthalten. Dies wäre zum Beispiel der Fall, wenn eine Variable im Programmablauf verwendet wird, bevor sie deklariert wurde. Um manche dieser Probleme zu erkennen, wird bei der Übersetzung eine semantische Analyse durchgeführt, in der zusätzliche Anforderungen an den Quelltext gestellt werden.

Semantik

Um eine zulässige syntaktische Struktur einer textuellen Beschreibung eine Bedeutung zu geben, wird eine Semantik zugrunde gelegt. Unter Semantik wird aus linguistischer Sicht die Lehre von der inhaltlichen Bedeutung einer Sprache verstanden, wobei in der Informatik Programmiersprachen oder mathematische Kalküle gemeint sind. Erst durch die formale und exakte Beschreibung der Semantik kann ein Programm unmissverständlich interpretiert werden (ebd., S. 587). Eine Semantik kann auf verschiedenen Wegen bestimmt werden, wobei die Wahl einer Definition durch den Kontext und Zweck der Definition nahegelegt wird. Echtle und Goedicke unterscheiden drei alternative Ansätze: die operationale Definition (Interpretersemantik), die denotationale Definition (Funktionensemantik) und die rein verbale Definition (Echtle und Goedicke, 2000, S. 17):

- Bei der *operationalen* Definition einer Semantik wird ein Interpreter angegeben, der aus den Eingabedaten die Ausgabedaten erzeugt, indem das Programm schrittweise abgearbeitet wird. Der Interpreter wird als Automat definiert, welcher das abzuarbeitende Programm, die aktuelle Position im Programm sowie den Zustand mitführt. Der operationale Ansatz ist einerseits zwar eindeutig, andererseits muss für jede Eingabe der Ablauf des Programms nachvollzogen werden. Insbesondere die formale Angabe der realisierten Funktion kann daher Schwierigkeiten bereiten.

- Die *denotationale* Definition einer Semantik unterscheidet sich dadurch, dass die Bedeutung einzelner Sprachkonstruktionen direkt festgelegt wird. Dabei wird von einem konkreten Maschinenmodell abstrahiert zu einer „Wirkungs-Abbildung“: $\mathbb{F}[a_0] : Z \to Z$. Sie beschreibt, welche Wirkung eine Anweisung a_0 auf einen Zustand hat, d.h. in welchen Folgezustand die Maschine nach dem Abarbeiten der Anweisung übergeht.

- Die *verbale* Definition verzichtet demgegenüber auf die strenge Formalisierung. Stattdessen liegt ein Schwerpunkt auf der Erläuterung der Intention eines Programms. Angesprochen wird vor allem die Vorstellungskraft (ebd., S. 17).

An anderer Stelle werden mit der Übersetzersemantik und der axiomatischen Semantik noch weitere Ansätze benannt (Claus und Schwill, 2001, S. 587). Das Prinzip der Übersetzersemantik besteht darin, eine neue Programmiersprache auf die Bedeutung einer bereits bekannten

Programmiersprache zurückzuführen. Unter Zuhilfenahme der Methoden der Automatentheorie oder der Theorie der formalen Sprachen wird ein Übersetzer für den Transfer konstruiert. Die axiomatische Semantik unterscheidet sich von den übrigen Semantiken durch ihr Abstraktionsniveau. Während die denotationale Semantik mit Funktionen arbeitet, um Zustandsänderungen zu beschreiben, werden in der axiomatischen Semantik nur noch Eigenschaften von Zuständen in Form von Programmformeln verwendet.

2.1.2 Konzepte der objektorientierten Programmierung

Programmiersprachen werden ausgehend von den erfüllten Programmierparadigmen in verschiedene Gruppen unterteilt: in funktionale, prädikative, imperative und objektorientierte Sprachen. Diese Arbeit befasst sich mit der Kompetenzmessung im Bereich der objektorientierten Programmierung. Die Merkmale anderer Programmierparadigmen werden nicht näher betrachtet. In der empirischen Untersuchung findet die objektorientierte Sprache „Java“ Anwendung, deren Syntax in den nachfolgenden Ausführungen daher im Fokus steht.

Objekte und Klassen

Eine grundlegende Definition für die Objektorientierung muss die des Objekts sein: „Ein Objekt ist eine Entität, die durch ihren Zustand und die Manipulationsmöglichkeiten auf diesem Zustand charakterisiert wird.“ (Doberkat und Dißmann, 2002, S. 292) Objekte werden in Java als Instanzen von Klassen gebildet. Mit der Programmierung von Klassen können beliebig viele gleichartige Objekte dynamisch erzeugt werden. Klassen sind durch die Vereinbarung von Variablen und Methoden sowie ihre Bezeichnung im Klassennamen charakterisiert und geben den Bauplan eines Objekts an. (Echtle und Goedicke, 2000, S. 86)
Der Aufruf einer Methode (siehe Abbildung 2.1) ist aus programmtechnischer Sicht als Manipulation eines Objekts aufzufassen. In Java bestehen Methoden aus einem Namen, d.h. einem beliebigen Bezeichner, einer Parameterliste in runden Klammern, dem Rückgabedatentyp des Ergebniswerts sowie einem durch geschweifte Klammern markierten Programmstück, genannt Methodenrumpf. Im Programmstück der Methode wird die Rückgabe des berechneten Ergebniswerts durch die *return*-Anweisung veranlasst. Wenn der Ergebniswert nicht weiter benötigt wird, wird *void* als Rückgabedatentyp gesetzt. Das Ergebnis des Aufrufs einer Methode kann die Veränderung des Zustandes eines Objekts sein.

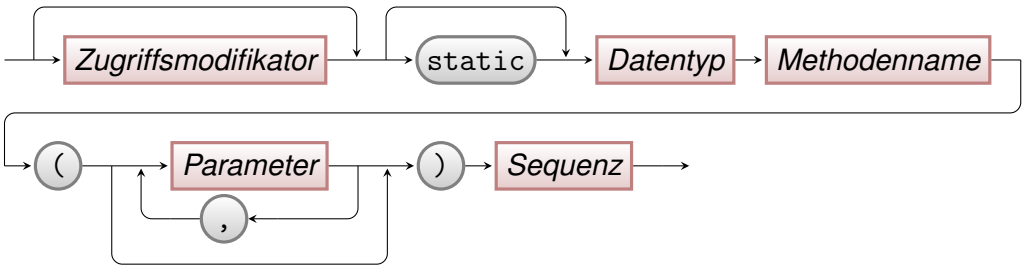


Abbildung 2.1: Syntaxdiagramm einer Methode in Java

Zusätzlich kann nach der Erzeugung eines Objekts ein Anfangszustand zugewiesen werden. Die dafür verwendete spezielle Methode zur Bildung des Initialzustands wird im Konstruktor (siehe Abbildung 2.2) realisiert. In Java bewirkt das Schlüsselwort *new*, dass ein Objekt erzeugt wird und dass die Konstruktor-Methode aufgerufen wird. Implizit ist in jeder Klasse ein leerer Konstruktor ohne Anweisung vorhanden, der bei Ausbleiben eines explizit deklarierten

Konstruktors ausgeführt wird. Sofern Klassen mehrere Konstruktoren haben, müssen diese durch die Anzahl oder den Datentyp der übergebenen Parameter wohl unterscheidbar sein. Anderenfalls kommt es zu Konflikten, da Konstruktoren namensgleich zu ihren Klassen sind.

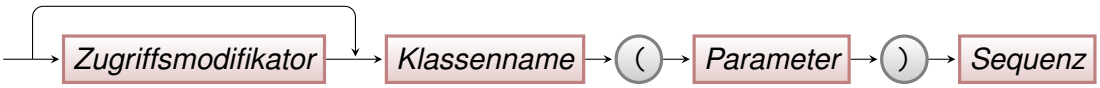


Abbildung 2.2: Syntaxdiagramm eines Konstruktors in Java

Zur spezifischen Datenstruktur in objektorientierten Programmiersprachen gehört auch die Einhaltung von Gestaltungsprinzipien der Klassen. Unter dem Prinzip der Kapselung wird ein spezielles Zusammenfassen des Zustands und des Verhaltens eines Objekts verstanden. Methoden und Attribute eines Objekts werden als Bestandteile eines Objekts betrachtet. Das Prinzip der Abschottung regelt die Zugriffsmöglichkeit von außen auf Objekte. Interne Strukturen und Abläufe können verborgen werden oder selektiv zur Nutzung in anderen Kontexten freigegeben werden durch Setzung der Zugriffsmodifikatoren *private* oder *protected*.
Neben der Generierung einer Vielzahl von strukturgleichen Objekten auf Grundlage einer Klasse, ist auch das Generieren verwandter Objekte wünschenswert. Das Prinzip der Vererbung bietet eine entsprechende Möglichkeit, indem aus einer gemeinsamen Oberklasse verschiedene Unterklassen abgeleitet werden. Der Prozess der Ableitung wird in Java über die *extends* Funktion realisiert. Unterklassen übernehmen alle Attribute und Methoden ihrer Oberklasse und erweitern diese gegebenenfalls um eigene Deklarationen.
Aus den vorgestellten Konstrukten, die eine Klasse ausmachen, ergibt sich in Java folgende syntaktische Beschreibung:

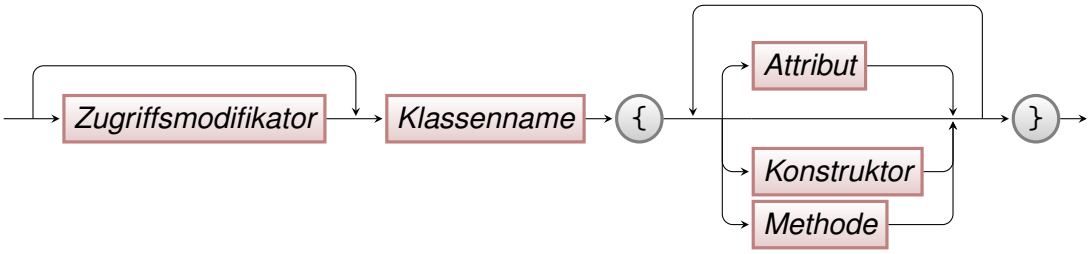


Abbildung 2.3: Syntaxdiagramm einer Klasse in Java

Kontrollstrukturen

Kontrollstrukturen bezeichnen Schemata, die festlegen, in welcher Reihenfolge Anweisungen ausgeführt werden sollen. Eine Anweisung wird in Java durch die Setzung eines schließenden Semikolons markiert. Die einfachste Art der Kontrollstruktur ist die Sequenz, die sich durch das Aneinanderreihen von Anweisungen ergibt. Anweisungen in einer Sequenz werden linear nacheinander abgearbeitet. Weitere Kontrollstrukturen in Java sind bedingte Anweisungen, die Mehrfachauswahl, Wiederholungen mt fester Anzahl, Wiederholungen mit Anfangsbedingung und Wiederholungen mit Endbedingung. Kontrollstrukturen werden in Java über bestimmte Schlüsselwörter umgesetzt. Im Hinblick auf die Verwendung in der empirischen Untersuchung werden in diesem Abschnitt die *for*-Schleife, die *while*-Schleife und die *if-else*-Fallunterscheidung vorgestellt.

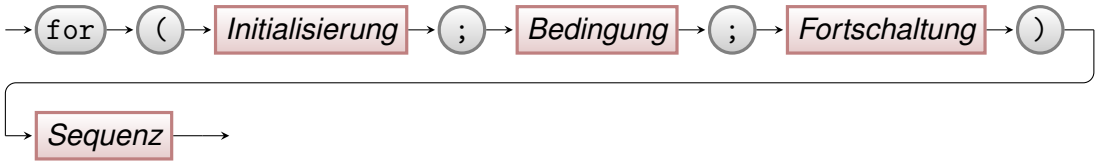


Abbildung 2.4: Syntaxdiagramm einer for-Schleife in Java

Die for-Schleife (siehe Abbildung 2.4) erzeugt eine Wiederholung mit fester Anzahl. Der Schleifenkopf einer *for*-Schleife enthält die Kontrollinformationen für die Anzahl der Schleifendurchläufe. Eingeleitet mit dem Schlüsselwort *for* folgen eine Initialisierung der Laufvariablen, eine Bedingung für die Laufvariable und die Fortschaltung der Laufvariablen. Die mehrfach zu durchlaufende Anweisungssequenz wird in einen Schleifenrumpf gefasst.
Die while-Schleife (siehe Abbildung 2.5) ist im Gegensatz zur for-Schleife keine Zählschleife sondern eine bedingte Schleife (Wiederholung mit Anfangsbedingung). Die Beendigung des Schleifendurchlaufs und damit die Anzahl der Wiederholungen wird bei der *while*-Schleife durch die Erfüllung der übergebenen Bedingung im Schleifenkopf gesteuert. Vor der Ausführung der Schleife wird die festgelegte Bedingung getestet, sodass nachfolgende Codesequenzen möglicherweise gar nicht ausgeführt werden. Eine Abwandlung der while-Schleife ist die do-while-Schleife, die mindestens einmal ausgeführt wird, sie wird in dieser Arbeit nicht verwendet und daher auch nicht näher betrachtet.

Abbildung 2.5: Syntaxdiagramm einer while-Schleife in Java

Die if-else-Fallunterscheidung (siehe Abbildung 2.6) ist eine bedingte Anweisung, d.h. ihre Ausführung hängt vom Wahrheitswert eines booleschen Ausdrucks ab. Sie besteht aus den Schlüsselwörtern *if* und *else*, einer Bedingung, die dem *if*-Operator übergeben wird und mindestens einer Anweisung bzw. zwei Sequenzen. Nach der Auswertung der Bedingung wird in Abhängigkeit vom Ergebnis entweder die erste Anweisung (bzw. Sequenz) oder die zweite Anweisung (bzw. Sequenz) ausgeführt. Durch die Verwendung des Schlüsselworts *else if* kann die Fallunterscheidung um einen weiteren Fall erweitert werden. Dieser Fall wird nur ausgeführt, wenn die vorherigen Fälle nicht eingetroffen sind und die Bedingung des Falls erfüllt wird.

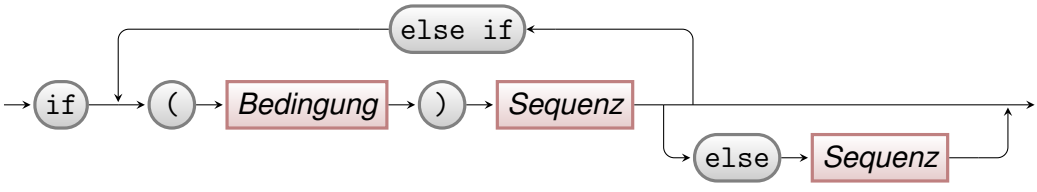


Abbildung 2.6: Syntaxdiagramm einer if-else-Fallunterscheidung in Java

Algorithmen

Eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann, wird als Algorithmus bezeichnet. (Claus und Schwill, 2001, S. 43) Ein Algorithmus beschreibt die Umwandlung von (zulässigen) Eingabedaten E in Ausgabedaten A. Die Abfolge der einzelnen Verarbeitungsschritte geht

dabei eindeutig aus der Darstellung hervor. Jeder Schritt umfasst genau eine wohldefinierte Elementaroperation. Algorithmen erfüllen die Eigenschaft der Finitheit, d.h. erstens, dass die Beschreibung der Umwandlung von Eingabe- in Ausgabedaten eine endliche Länge hat und zweitens, dass zur Ablage der Zwischenergebnisse nur endlich viel Speicherplatz benötigt wird. In der Sprache der Abbildungen ist ein Algorithmus eine Abbildung $f : E \to A$. Andersherum lässt sich jedoch nicht jede Abbildung $f : E \to A$ durch einen Algorithmus realisieren. Die Zuordnung der Algorithmen zur Menge der Abbildungen zwischen Eingabe- und Ausgabedaten ist daher nicht surjektiv.
Die Beschreibungsmächtigkeit von Algorithmen bestimmt die automatisierte Lösbarkeit von Problemen: Die Lösung solcher Probleme, die grundsätzlich nicht über einen Algorithmus abgebildet werden können, kann auch theoretisch nicht von Computern oder anderen Maschinen übernommen werden.
Die Umsetzung eines Algorithmus in eine Programmiersprache, die maschinell gelesen werden kann, geschieht in Form einer Implementierung. Zur Konstruktion eines Algorithmus stehen verschiedene Strukturelemente wie Sequenzen, bedingte Anweisungen und Wiederholungen zur Verfügung. Um Beziehungen herzustellen und Abläufe umzusetzen, kommen bei der Implementierung der Strukturelemente die vorangehend dargestellten sprachspezifischen Kontrollstrukturen zum Einsatz.

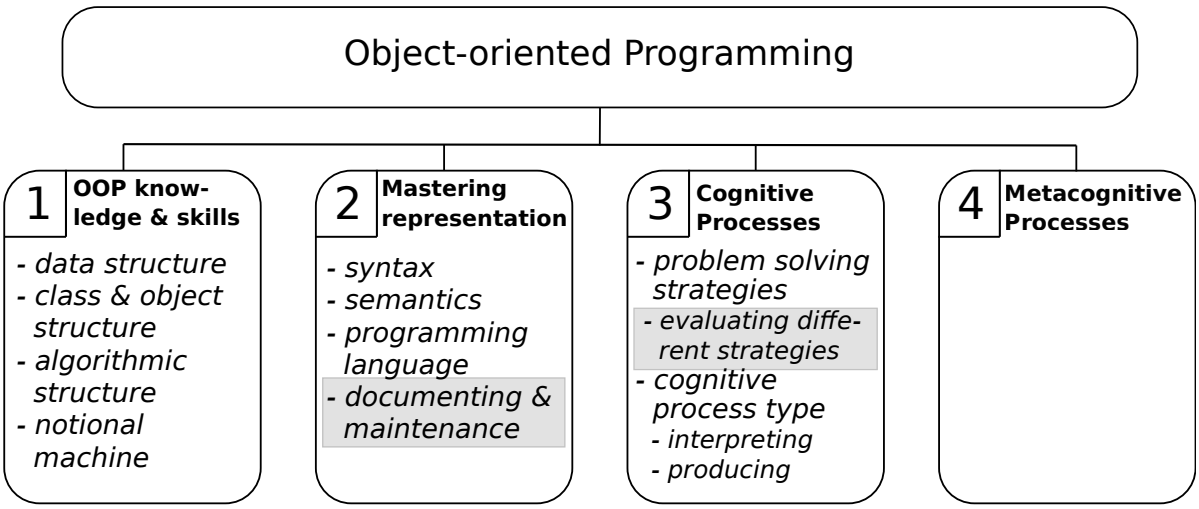
2.1.3 Kompetenzmodell für die objektorientierte Programmierung

Fächer- bzw. domänenspezifische Kompetenzmodelle klären über die Dimensionen und Stufen von empirisch überprüfbaren Kompetenzen auf. Unter „Kompetenzen“ werden nach der häufig zitierten Definition von Weinert „die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten, um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können“, verstanden (Weinert, 2001, S. 27). Kompetenzmodelle „beschreiben auf der Basis fachdidaktischer Konzepte die Komponenten und Stufen der Kompetenzen und stützen sich dabei auf pädagogisch-psychologische Forschungen zum Aufbau von Wissen und Können“ (Klieme u. a., 2003). Sie stellen damit die modelltheoretische Grundlage zur Bestimmung von Bildungsstandards dar. Bildungsstandards konkretisieren wiederum die Bildungsziele in Form von Kompetenzanforderungen, die im Schulumfeld für einzelne Jahrgänge festlegen, welche Niveaustufen die SchülerInnen erreichen sollen. Dabei geben sie aber keinen Weg vor, der zur Erreichung der Bildungsziele führt. Sie beinhalten also keine Vorgaben für die Thematik und Methodik des Unterrichts. Die Erreichung der Bildungsziele soll mittels standardisierter Verfahren überprüfbar sein. Die in den Standards zugrunde gelegten Kompetenzstufen sollen durch Aufgabenbeispiele veranschaulicht werden (vgl. ebd., S. 50).
Kompetenzen können in ihren Teildimensionen und Niveaustufen in Form von Kompetenzmodellen ausgewiesen werden. Klieme sieht in ihnen eine „pragmatische Antwort auf die Konstruktions- und Legitimationsprobleme traditioneller Bildungs- und Lehrplandebatten“ (ebd., S. 9). Kompetenzmodelle sind domänenspezifisch und das Produkt psychologischer sowie fachdidaktischer Überlegungen und Untersuchungen zu einem abgrenzbaren Lernbereich.

Für den Lernbereich Informatik haben Kramer, Hubwieser und Brinda (2016) damit begonnen, ein Kompetenzstrukturmodell für die Einführung in die objektorientierte Programmierung (OOP) im schulischen und universitären Umfeld zu entwickeln. Das Kompetenzmodell soll eine Reihe von Kompetenzen und ein Kategoriensystem enthalten, welches die Kompetenzen strukturiert. Der erste Vorschlag für ein Modell (siehe Abbildung 2.7) wurde nach einer Literaturrecherche

aus dem theoretischen Hintergrund abgeleitet. Dazu wurde zunächst der Kompetenzbegriff geklärt, bevor Kompetenzmodelle aus verschiedenen Domänen analysiert wurden. Diese Modelle sollten einen Anhaltspunkt dafür liefern, welche potentiellen Dimensionen für das Modell infrage kommen könnten. Bei der Analyse wurde festgestellt, dass schon existierende Kompetenzmodelle meist eine dreigliedrige Struktur aufweisen. Sie bestehen aus: Professionswissen bzw. Inhalten („professional knowledge and/ or content“), Darstellungsweisen der Inhalte („content representation“), kognitiven Prozessen („cognitive process aspect“) und teilweise nicht-kognitiven Fähigkeiten („non-cognitive abilities“). Für diese Bereiche wurde jeweils recherchiert, welche empirischen Arbeiten für die Domäne OOP vorliegen. Ausgehend vom bisherigen Forschungsstand haben die Autoren einen ersten Modellentwurf formuliert, der die domänenspezifischen Besonderheiten der OOP berücksichtigt. Als Hauptdimensionen wurden „Objektorientiertes Wissen und Fertigkeiten“ („OOP knowledge and skills“), „Umgang mit Darstellungen“ („mastering representation“), „kognitive Prozesse“ („cognitive processes“) und „meta-kognitive Prozesse“ („metacognitive processes“) genannt. Die Hauptdimensionen Fertigkeiten und Kognitionen wurden noch weiter aufgeschlüsselt in Teilaspekte. Dieses erste Modell wurde in drei Schritten und auf zwei Ebenen validiert. Einerseits wurden die Curricula der 16 Bundesländer, nationale Curricula aus Kanada, Großbritannien, Indien und den USA sowie Empfehlungen verschiedener Organisationen auf Kompetenzfacetten in der Domäne OOP analysiert. Es wurden insgesamt 911 Kompetenzfacetten gefunden, die auf ihre Passung mit dem Modell überprüft wurden. Im Ergebnis wurde das bestehende Modell um die Unterdimensionen „Quelltext-Konventionen“ („Code Conventions“) und „Dokumentieren und Wartung“ („Documenting and Maintenance“) erweitert. In der Dimension „Umgang mit Darstellungen“ und in der Unterdimension „Problemlösefähigkeit“ („Problem solving skills“) wurde die Kompetenzfacette „Verschiedene Strategien gegeneinander abwägen“ („Evaluating different strategies“) ergänzt. Beim Abgleich mit 119 Kompetenzfacetten, die von einer ITiCSE-Arbeitsgruppe in dreizehn Fallstudien zu nationalen Bildungsstandards in der Domäne OOP identifiziert wurden, zeigten sich keine weiteren Lücken des Entwurfs.
Zudem wurde das Kompetenzmodell mit der Sichtweise von Lehrenden in der schulischen Informatik auf die Domäne abgeglichen. LehrerInnen und Lehramtsstudierenden im Masterstudium wurden gefragt, welche Schritte erfahrungsgemäß beim Erlernen von Programmiersprachen am schwierigsten sind. Die Autoren gingen davon aus, von erlebten Schwierigkeiten auf Kompetenzen schließen zu können. Aus den Antworten haben die Autoren 123 Kompetenzbeschreibungen extrahiert. Die Einordnung in ein Kategoriensystem ergab, dass alle genannten Aspekte mit dem vorgeschlagenen Kompetenzmodell erfasst werden konnten.
Nach der Validierung steht derzeit das Modell in Abbildung 2.7 zur Diskussion. Zur empirischen Validierung ist die Konstruktion von Items ein wichtiger Schritt. Daher beabsichtigen Kramer, Hubwieser und Brinda (2016) Kompetenzkandidaten (Kombinationen einzelner Kompetenzfacetten) mit Items abzubilden. Dazu werden für jede Kombination etwa zehn bis zwölf Items angestrebt, die idealerweise von mehreren hundert Personen bearbeitet werden. Für die Weiterentwicklung des Modells wird zudem, Expertise aus der pädagogischen Psychologie eingeholt, insbesondere wenn es darum geht, das Kompetenzstrukturmodell in ein psychometrisches Messmodell zu transformieren.

Um die empirische Validierung zu starten, haben Kramer, Tobinski und Brinda (2016) Items in der Kombination der Dimensionen „kognitive Prozesse“ und „Objektorientiertes Wissen und Fertigkeiten“ entwickelt. Die Aufgabe bestand darin, bekannte syntaktische Elemente in objektorientierten Quelltexten wiederzuerkennen. Da zu diesem Zeitpunkt noch keine computerbasierte Umsetzung für den Aufgabentyp existierte, wurde der Item-Stamm in Form einer klassischen schriftlichen Erhebung umgesetzt. Für weitere Erhebungen im Rahmen der



Neuerungen des zweiten Entwurfs sind grau hinterlegt.

Abbildung 2.7: Kompetenzstrukturmodell der objektorientierten Programmierung nach Kramer, Hubwieser und Brinda (2016)

Validierung ist der Umstieg auf computerbasierte Items vorgesehen.
Für den ersten Itemtyp wurden zwölf zufällige Quelltexte mit jeweils 30 Zeilen aus dem offiziellen BlueJ Projekt (Barnes und Kölling, 2016) entnommen. Für jeden Quelltext wurden die TeilnehmerInnen aufgefordert, alle Stellen, an denen ein bestimmtes syntaktisches Element vorkommt, zu markieren. Außerdem sollten sie eine Regel formulieren, wie Elemente dieses Typs erkannt werden können. Insgesamt haben 29 Studierende, welche die Veranstaltung „Einführung in die Programmierung“ besuchten, an der Studie teilgenommen. Zusätzlich wurde die Studie drei Tage später mit fünf Studenten eines Masterstudiengangs Informatik wiederholt. Die Autoren haben anhand der selbst formulierten Regeln darauf schließen können, dass Programmieranfänger den Quellcode oberflächlich nach Schlüsselwörter durchsuchen, wohingegen Programmierexperten abstrakte Strukturen erkennen.

In dieser Arbeit wird die empirische Validierung des Modells mit der Konstruktion eines neuen Aufgabentyps fortgesetzt. Dazu werden die Dimensionen „class & object structure“, „algorithmic structure“, „syntax“, „semantics“ und „interpreting“ miteinander verknüpft. Verbunden mit der Kompetenzmodellierung sind Fragen nach dahinter liegenden kognitiven Strukturen und Einflussfaktoren.

2.2 Psychologische Grundlagen

In diesem Unterkapitel werden zunächst in 2.2.1 die Komponenten des Arbeitsgedächtnisses beschrieben, bevor in den Abschnitten 2.2.2 und 2.2.3 die Konzepte Aufmerksamkeit und Expertise näher erklärt werden. Das Unterkapitel schließt in Abschnitt 2.2.4 mit der Beschreibung der Organisation von Informationen im Arbeitsgedächtnis.

2.2.1 Arbeitsgedächtnis

Das Arbeitsgedächtnis ist ein zeitlich begrenzter Speicher, in dem die bewusste Speicherung und Verarbeitung von Informationen stattfindet. Das bekannteste Modell wurde von Baddeley und Hitch (1974) formuliert und später von Baddeley (2000) erweitert (siehe Abbildung 2.8).

Nach Baddeley existieren im Arbeitsgedächtnis vier Komponenten: die sogenannte zentrale

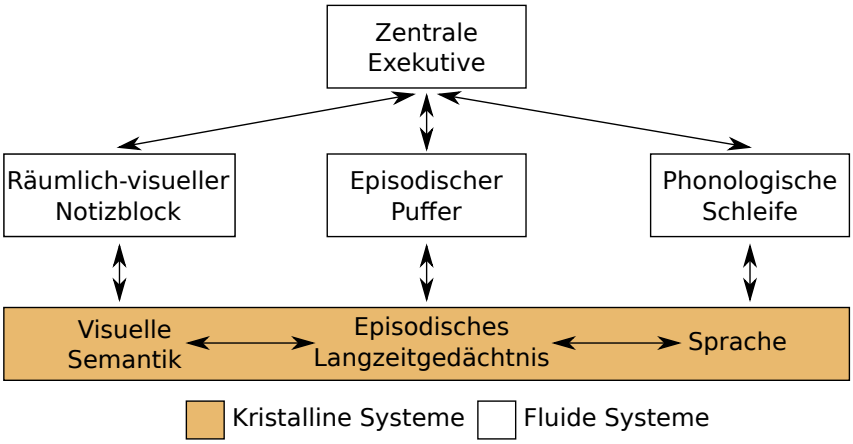


Abbildung 2.8: Arbeitsgedächtnismodell nach Baddeley (2000)

Exekutive, eine Steuerungseinheit, die die anderen drei Komponenten kontrolliert: die phonologische Schleife, der räumlich-visuelle Notizblock und der episodische Puffer. Wenngleich es sich bei allen Subsystemen um fluide Systeme handelt, ist die Funktion des Arbeitsgedächtnisses an kristalline Systeme geknüpft, aus denen die Inhalte bezogen werden. Fluide Systeme werden im Gegensatz zu den kristallinen Systemen durch Lernen nicht verändert. Die Unterscheidung stammt aus der Intelligenzforschung, die zwischen fluider und kristalliner Intelligenz unterschiedet. Die fluide Intelligenz betrifft allgemeine kognitive Tätigkeitsfelder wie das Problemlösen und die Fähigkeit zur Mustererkennung. Demgegenüber ist die kristalline Intelligenz an Wissen und Erfahrung gebunden und kann gezielt durch Lernprozesse ausgebaut werden. Nachfolgend werden die einzelnen Komponenten des Arbeitsgedächtnismodells kurz vorgestellt.

Zentrale Exekutive

Nach Baddeley (2012) ist die zentrale Exekutive die wichtigste Komponente des Arbeitsgedächtnismodells. Es handelt sich hierbei um das Gedächtnissystem, das die verschiedenen nachfolgend dargestellten Hilfssysteme koordiniert. Dazu stehen dem System folgende Funktionen zur Verfügung:

- Fokussierung der Aufmerksamkeit
- Aufteilen der Aufmerksamkeit auf verschiedene Stimuli
- Wechseln zwischen Aufgaben

Die Funktionen wurden zum Teil durch empirische Forschung nachgewiesen, jedoch besteht nach Baddeley (ebd.) die Möglichkeit, dass auch andere bzw. weitere Funktionen denkbar wären.

Phonologische Schleife

Die phonologische Schleife besteht aus einem phonologischen Speicher, welcher kurzzeitig ungefähr sieben Informationseinheiten halten kann, bevor diese verschwinden, und aus einem „rehearsal“-Prozess, bei dem durch inneres Sprechen die Informationen wiederholt werden. Die Informationseinheiten können somit durch Abrufen und erneutes Aufsagen länger als

die durchschnittlichen zwei Sekunden im phonologischen Speicher gehalten werden. Das assoziierte kristalline System ist die erlernte Sprache. Die phonologische Schleife ist die bisher am besten erforschte Komponente des Arbeitsgedächtnisses.

Räumlich-visueller Notizblock

Der räumlich-visuelle Notizblock ist das visuelle Äquivalent zur phonologischen Schleife. Auch er ist limitiert und kann typischerweise drei bis vier Objekte speichern. Nach Baddeley (2003) wird in dieser Komponente meistens zwischen dem visuellen und dem räumlichen Speicher unterschieden. Daneben gibt es auch andere Unterscheidungsformen, wie zum Beispiel die zwischen dem statischen und dem dynamischen Speicher (vgl. Pickering, 2001).

Episodischer Puffer

Der episodische Puffer wurde von Baddeley (2000) nachträglich dem Arbeitsgedächtnismodell hinzugefügt, da manche Phänomene, die eine Verknüpfung der phonologischen Schleife und des räumlich-visuellen Notizblocks verlangten, nicht mit dem Modell erklärt werden konnten. Diese Komponente soll eine limitierte Kapazität haben, welche Informationen zu sogenannten Episoden zu bündeln kann. Baddeley nimmt an, dass die Komponente bewusst von der zentralen Exekutive gesteuert wird. Dadurch sollen multidimensionale Kodierungen durch die Integration beider Komponenten möglich sein. Auch die Anreicherung von Episoden durch die Hinzunahme von Einheiten aus dem Langzeitgedächtnis soll möglich sein.

2.2.2 Aufmerksamkeit

Die Kontrolle der Aufmerksamkeit ist nach der Forschung von Baddeley eine wichtige Funktion der zentralen Exekutive. Nach Wentura und Frings (2012) lassen sich drei Hauptfunktionen der Aufmerksamkeit benennen: das „Planen/Kontrollieren“, das „Überwachen und das Selektieren“. Die Funktion „Planen/Kontrollieren“ ist dafür zuständig, „eine kontrollierte (also eine nicht automatisch ablaufende) Handlung vorzubereiten und auszuführen)“ (ebd., S. 84). Mit der Überwachungsfunktion wird „ein stetiges Überwachen der eigenen Umwelt“ realisiert (ebd., S. 84). Eine Veränderung der Umwelt würde Aufmerksamkeit auf sich ziehen. Die Funktion des Selegierens übernimmt das Filtern der Vielzahl von Informationen, um somit die relevanten Informationen hervorzuheben (vgl. ebd., S. 84).

Aufmerksamkeit und Arbeitsgedächtnis

Werden die Funktionen der Aufmerksamkeit mit der zentralen Exekutive des Arbeitsgedächtnisses verglichen, so können viele Gemeinsamkeiten festgestellt werden. Die zentrale Exekutive ist zum Beispiel für das Wechseln zwischen Aufgaben zuständig. Diese Funktion ist identisch mit der Funktion der Aufmerksamkeit Planen/Kontrollieren. Nach Wentura und Frings (ebd.) fusionieren die beiden Forschungsfelder nach und nach, sodass häufig der Terminus exekutive Kontrollfunktion anstelle des Terminus Aufmerksamkeit verwendet wird.

2.2.3 Expertise

Nach Anderson (1983) und Fitts und Posner (1967) besteht der Prozess, Fähigkeiten zu erwerben aus drei Phasen: der kognitiven Phase, der assoziativen Phase und der autonomen Phase. In der kognitiven Phasen werden Fakten, die für die zu erlernende Fähigkeit relevant

sind, eingeprägt. Diese werden typischerweise bei der Ausführung der Fähigkeit in Gedanken durchgegangen. In der sich daran anschließenden assoziativen Phase geschehen nach Anderson (2015) zwei Entwicklungen: zum einen werden Fehlvorstellungen ausgemacht und eliminiert, zum anderen werden die Verbindungen zwischen den einzelnen Elementen verstärkt, die ein erfolgreiches Ausführen der Fähigkeit ermöglichen. In der autonomen und letzten Phase des Fähigkeitserwerbs wird das Ausführen der Fähigkeit mehr und mehr automatisiert, wodurch sich auch das Tempo der Ausführung erhöht. Mit steigender Expertise werden die Kapazitäten des Arbeitsgedächtnisses immer weniger beansprucht, da die Zuständigkeit für die Ausführung einer Tätigkeit ins Unterbewusste verlagert wird.

Die Expertise zeigt sich zum Beispiel beim Memorieren und Reproduzieren von domänenspezifischen Gegenständen bzw. Inhalten. In diesem Bereich wurde ein großer Beitrag zur Forschung durch Untersuchungen rund um die Expertise von SchachspielerInnen erzielt (siehe u.a. Chase und Simon, 1973; De Groot, 1965; Gobet und Simon, 1996a,b,c). Dabei wurden Merkstrategien von Experten des Schachs untersucht, um Unterschiede zwischen ihnen und schwächeren Schachspielern aufzuzeigen. Daraus resultierten zwei Theorien, die die unterschiedlichen Merkfähigkeiten der beiden Gruppen erklären: die Chunking-Theorie und die darauf aufbauende Template-Theorie. Sie werden im nachfolgenden Abschnitt näher erläutert.

2.2.4 Organisation von Informationen im Arbeitsgedächtnis

Chunking-Theorie

Die von Chase und Simon (1973) formulierte Chunking-Theorie erklärt, wie ExpertInnen einer Domäne die ursprünglichen Grenzen von kognitiven Prozessen durch domänenspezifisches Wissen umgehen können. Dies geschieht, indem sie Informationen zu kleinen verbundenen Einheiten, sogenannten Chunks, zusammenfassen. Diese Chunks können entweder im Arbeitsgedächtnis gehalten werden oder im Langzeitgedächtnis gespeichert werden. Schon Miller (1956) hatte die Vermutung aufgestellt, dass Chunks ein korrektes Maß für die Informationen im kognitiven System bilden und dass 7 ± 2 Chunks im Arbeitsgedächnis gehalten werden können. Chase und Simon haben diese Vermutung aufgegriffen und formalisiert. Sie gehen davon aus, dass durch das Aneignen von Informationen einer Domäne mehr Chunks im Langzeitgedächtnis entstehen, welche mit mehr Expertise auch mehr Informationseinheiten bündeln. Während das Erlernen neuer Chunks eher langsam geschieht (ca. 10s per Chunk), funktioniert das Erkennen von Informationen, die in einem Chunk gespeichert sind, im Bruchteil einer Millisekunde. Weitergehende Forschungen deuten darauf hin, dass die Chunks mit weiteren Informationen verknüpft sind (vgl. Gobet und Lane, 2012). Jedoch konnten nach Gobet und Simon (1996c) nicht alle Phänomene, die in der empirischen Forschung mit Schachexperten erkannt wurden, durch die Chunking-Theorie erklärt werden. Leerstellen der Chunking-Theorie füllt die Template-Theorie von Gobet und Simon (ebd.).

Template-Theorie

Ein Template kann als eine Art Vorlage angesehen werden, welche befüllbare Lücken besitzt. Diese Lücken können mit Informationseinheiten oder Chunks gefüllt werden (vgl. ebd.). Der Unterschied zwischen Chunks und Templates besteht darin, dass Chunks einfache Muster sind, während Templates Konfigurationen sind, die mit zusätzlichen Informationen befüllt werden können (vgl. Guida u. a., 2012). Templates können als eine Art Prototyp aufgefasst werden, dessen Eigenschaften sich je nach Füllung leicht ändern können. Dadurch können im Schach

zum Beispiel verschiedene Schachpositionen zu einem Template zusammengefasst werden, wenn sie derselben Kategorisierung entsprechen.

2.3 Komplexität von Quelltexten und natürlichsprachlichen Texten

In dieser Arbeit werden Mittel zur Einordnung von Quelltexten und natürlichsprachlichen Texten benötigt (siehe Abschnitt 4.2.2). Neben der Länge (Anzahl der Zeichen) der jeweiligen Texte soll auch deren kognitive Komplexität berücksichtigt werden. Nachfolgend werden Methoden vorgestellt, die zur Bestimmung der kognitiven Komplexität verwendet werden können.

Komplexität von Quelltexten

Die Bestimmung der kognitiven Komplexität von Quelltexten ist ein wichtiger Faktor, um verstehen zu können, wie gut der Quelltext von ProgrammiererInnen verstanden werden kann. Die bis jetzt entwickelten Metriken für die Bestimmung der kognitiven Komplexität nehmen unterschiedliche Prämissen an, wodurch eine erhöhte kognitive Komplexität eines Quelltexts zustande kommt. Misra und Adewumi (2015) haben acht Metriken zur Bestimmung der Komplexität von objektorientierten Quelltexten analysiert. Alle analysierten Metriken bauen auf der Bewertung der Komplexität von grundlegenden Kontrollstrukturen, den „Basic Controll Structures (BCS)“, nach Shao und Wang (2003) auf (siehe Tabelle 2.1). Misra und Adewumi (2015) haben die

Kategorie	**BCS**	**Komplexität**
Sequenz	Sequenz	1
Verzweigung	If-Then-Else	2
	Case	3
Iteration	For-do	3
	Repeat-until	3
	While-do	3
Eingebettete Komponente	Funktionsaufruf	2
	Rekursion	3
Nebenläufigkeit	Parallel	4
	Interrupt	4

Tabelle 2.1: Basic Controll Structures (BCS) und ihre kognitive Komplexität nach Shao und Wang (2003)

analysierten Metriken weitergehend auf Praktikabilität und Validität untersucht. Die Weighted Class Complexity (WCC)-Metrik eignet sich aufgrund ihrer Eigenschaften besonders gut für die Anwendung auf die verwendeten Quelltexte in dieser Arbeit. Zur Bewertung der Komplexität werden neben Eigenschaften der vorkommenden Methoden auch Attribute und Nachrichten zwischen den Klassen berücksichtigt. Damit werden weitere wichtige Konzepte der OOP abgedeckt. Insbesondere ist die Metrik unabhängig von der verwendeten Programmiersprache. Neben der Eignung für mögliche Vergleichsstudien, gibt es auch aus statistischer Sicht Argumente für die Auswahl der WCC-Metrik. Die Ergebnisse der metrischen Auswertung liegen als Werte auf einer Verhältnisskala vor. Aus testtheoretischer Sicht ist das hohe Skalenniveau für die Möglichkeiten der statistischen Auswertung und die Qualität der möglichen Aussagen vorteilhaft. Lediglich für große Programme erweist sich der Einsatz der WCC als problematisch. Zum einen

sind hohe Ergebniswerte nur schwer interpretierbar, zum anderen lässt das Ergebnis keinen Rückschluss mehr zu auf die Ursachen der Schwierigkeit. Da es sich bei den eingesetzten Quelltexten um einfachste Programmstrukturen handelt, gibt es hier keine Indikationen, die gegen eine Verwendung der WCC sprechen.

Weighted Class Complexity

Die WCC nach Misra und Akman (2008) berücksichtigt die kognitive Komplexität von Operationen und Nachrichten innerhalb einer Methode und die kognitive Komplexität, die durch Attribute entsteht. Die Metrik wird in drei Schritten berechnet:

Schritt 1: Zunächst wird die kognitive Komplexität jeder Methode, genannt „Method Complexity (MC)“, berechnet. Diese ist definiert als die Summe der kognitiven Komplexität von q linearen Blöcken, zusammengesetzt aus einzelnen BCS. Jeder Block kann m Ebenen von verschachtelten BCS und jede Ebene wiederum n lineare BCS enthalten, sodass sich MC berechnen lässt durch:

$$MC = \sum_{j=1}^{q} \left[\prod_{k=1}^{m} \sum_{i=1}^{n} W_c(j,k,i) \right]$$

W_c ist die kognitive Komplexität einer Kontrollstruktur aus der Liste der BCS.

Schritt 2: Als nächstes muss die Anzahl der Attribute einer Klasse (N_a) gezählt werden.

Schritt 3: Jetzt wird die Komplexität einer Klasse, die „Weighted Class Complexity (WCC)“, berechnet.

$$WCC = N_a + \sum_{p=1}^{s} MC_p \tag{2.1}$$

Der Index s in Gleichung 2.1 gibt die Anzahl der Methoden in der Klasse an. Wenn mehr als eine Klasse im Quelltext vorhanden ist, d.h. insgesamt y Klassen, dann kann die Gesamtkomplexität, die „Total Weighted Class Complextiy (TWCC)“, berechnet werden durch:

$$TWCC = \sum_{x=1}^{y} WCC_x$$

Komplexität von natürlichsprachlichen Texten

Die kognitive Komplexität von natürlichsprachlichen Texten kann auf mehrere Arten dargestellt werden. Der inhaltliche Gehalt eines wissenschaftlichen Text kann dabei zum Beispiel viel höher sein als der eines Kinderbuchs, weshalb leicht argumentiert werden kann, dass mehr kognitive Ressourcen nötig sind, um den wissenschaftlichen Text zu verstehen. Andererseits könnte ein literarischer Text, der viele verschachtelte Satzkonstruktionen beinhaltet, deutlich schwieriger zu lesen sein als ein wissenschaftlicher Text. Aus dieser Perspektive kann man nachvollziehbar argumentieren, dass der literarische Text wahrscheinlich mehr kognitive Ressourcen beim Lesen benötigt als der wissenschaftliche Text. Die dargestellten Beispiele zeigen auf, dass die kognitive Komplexität von natürlichsprachlichen Texten von verschiedenen Aspekten abhängt. In dieser Arbeit soll die kognitive Komplexität von natürlichsprachlichen Texten durch ihre Lesbarkeit definiert werden, die inhaltliche Ebene soll dabei nicht mit in die Betrachtung einbezogen werden. Um die Lesbarkeit von natürlichsprachlichen Texten zu bestimmen, gibt es verschiedene Lesbarkeitsindizes. Zwei, die für die deutsche Sprache geeignet sind, sollen hier vorgestellt werden.

Wiener-Sachtextformel

Einem Vergleich von neun Lesbarkeitsindizes durch Bamberger (1984) zufolge, ist die erste neue Wiener-Sachtextformel (WS) die am besten geeignetste Formel zur Bestimmung der Lesbarkeit von Sachtexten. Da in dieser Arbeit ausschließlich Sachtexte verwendet werden, wurde diese Formel ausgewählt. Sie berechnet sich wie folgt:

$$WS = 0.1935 \cdot MS + 0.1672 \cdot SL + 0.1297 \cdot lW - 0.0327 \cdot ES - 0.875$$

Erklärung: MS $\widehat{=}$ Mehrsilber, SL $\widehat{=}$ durchschnittliche Satzlänge (in Wörtern), lW $\widehat{=}$ lange Wörter (mehr als sechs Buchstaben) und ES $\widehat{=}$ Einsilber.

Das Ergebnis der Auswertung ist als die Jahrgangsstufe zu interpretieren, ab der Heranwachsende in der Lage sind, den zu analysierenden Text flüssig zu lesen. Bei Ergebnissen über 12 wird von Schwierigkeitsstufen statt von Schuljahren gesprochen.

Flesch-Reading-Ease-DE

Da Lesbarkeitsformeln unterschiedliche Kriterien an Texte stellen, wird für die Bestimmung der Lesbarkeit noch eine zweite Formel herangezogen. Die Flesch-Reading-Ease-DE (FRE_{DE}) wurde für deutschsprachige Texte von Amstad (1978) aus der wohlbekannten Formel für englischsprachige Texte, der „Flesch-Reading-Ease“-Formel, abgeleitet. Amstad sah aufgrund der verschiedenen Satzstrukturen im Deutschen und im Englischen die Notwendigkeit, die Koeffizienten der englischen Version, welche von Flesch (1948) entwickelt wurde, anzupassen. Dazu analysierte er die Unterschiede zwischen der deutschen und der englischen Schriftsprache und kam auf folgende angepasste Formel:

$$FRE_{DE} = 180 - SL - 88.5 \cdot WL$$

Erklärung: SL $\widehat{=}$ durchschnittliche Satzlänge, WL $\widehat{=}$ durchschnittliche Wortlänge.

Es können Werte zwischen 0 und 100 erreicht werden, wobei der Wert 0 für einen sehr schweren für AkademikerInnen geeigneten Text steht und der Wert 100 auf einen sehr leichten Text für LeseanfängerInnen geeigneten Text hinweist (für weitere Abstufungen siehe Tabelle 2.2).

Wert	Lesbarkeit	Verständlich für
0–30	Sehr schwer	AkademikerInnen
30–50	Schwer	
50–60	Mittelschwer	
60–70	Mittel	13–15-jährige SchülerInnen
70–80	Mittelleicht	
80–90	Leicht	
90–100	Sehr leicht	11-jährige SchülerInnen

Tabelle 2.2: Flesch-Reading-Ease-Werte (Flesch, 1948)

2.4 Zusammenfassung, Schlussfolgerungen

In Unterkapitel 2.2 wurden die kognitionspsychologischen Konzepte Arbeitsgedächtnis, Aufmerksamkeit und Expertise vorgestellt. Zentral ist dabei die Erkenntnis, dass durch das Bündeln von Informationseinheiten zu Chunks oder das Verknüpfen mit Templates mehr Informationen im Arbeitsgedächtnis gespeichert werden können. Der Umfang eines Chunks und die Anzahl von Templates im Langzeitgedächtnis steigen mit wachsender Expertise. Chunks und Templates können je nach Domäne und Material geformt werden. Während Chunks spontan gebildet werden, prägen sich Templates durch regelmäßige Wiederholungen im Langzeitgedächtnis ein. In der Programmierung könnten Chunks gebildet werden, indem semantische Einheiten zusammengefasst werden. Ein Experte wird vermutlich aus einem sinnvollen Variablenname den zugehörigen Datentyp ableiten können, ohne sich ihn explizit merken zu müssen. Templates könnten dagegen die grammatikalischen Strukturen einer Programmiersprache abbilden. Beispiele für Templates könnten die in den Syntaxdiagrammen vorgestellten Strukturen sein, aber auch Konventionen und wiederkehrende algorithmische Strukturen. Templates und Chunks könnten eine Möglichkeit darstellen, ExpertInnen und NovizInnen in den Dimensionen Syntax und Semantik zu unterscheiden, weshalb ihre Untersuchung bei der Validierung des in diesem Kapitel vorgestellten Kompetenzmodells lohnenswert erscheint. Forschungsergebnisse zu diesem Bereichen werden im nächsten Kapitel vorgestellt.

3 Forschungsstand und Forschungsfragen

Kompetenzmodelle geben eine Auskunft darüber, wie Expertise in einer bestimmten Domäne beschrieben werden kann. Zur Überprüfung der Validität eines Kompetenzmodells der objektorientierten Programmierung werden Unterschiede zwischen ExpertInnen und NovizInnen der Domäne in einer aus dem Kompetenzmodell abgeleiteten Anforderungssituation untersucht.
Die Forschung in der Kongitionspsychologie zu kognitiven Unterschieden zwischen ExpertInnen und NovizInnen reicht bis in die 1960er Jahre zurück. Mehrfach repliziert wurde seitdem die Erkenntnis, dass ExpertInnen sich von NovizInnen dadurch unterscheiden, dass sie sich herausragend an inhaltlich sinnvolle Materialien ihrer spezifischen Domäne erinnern können. Dieses Ergebnis wurde zuerst von De Groot (1965) und Chase und Simon (1973) bei Studien mit Schachspielern ermittelt und konnte später auch in anderen Domänen bestätigt werden. In diesem Kapitel sollen eingangs die Erkenntnisse von De Groot und Chase und Simon vorgestellt werden. Danach wird auf Grundlage verschiedener empirischer Studien die Übertragbarkeit auf die Domäne Programmierung untersucht. Da in dieser Arbeit auch der Einfluss von kognitiven Fähigkeiten auf die Leistung beim Memorieren und Reproduzieren von Quelltext behandelt werden soll, werden im Unterkapitel 3.3 Erkenntnisse speziell aus diesem Bereich aufgezeigt.

3.1 Zusammenhang von Gedächtnisfähigkeiten und Expertise

Eine der ersten Domänen, die im Zusammenhang mit Expertise betrachtet wurde, ist die des Schachspiels. De Groot (1965) hat mit seinen Studien zum unterschiedlichen Verhalten von NovizInnen und ExpertInnen des Schachspiels herausgefunden, dass ExpertInnen nach einer fünfsekündigen Anschauphase eines Schachbretts ungefähr 90% von diesem reproduzieren können. Das Schachbrett zeigte dabei eine Situation aus der Mitte eines Spiels, in der noch 20 Schachfiguren vorhanden waren. Diese Studie wurde von Chase und Simon (1973) repliziert. Auch sie stellten fest, dass ExpertInnen diese Fähigkeit besitzen. Zusätzlich haben Chase und Simon die Arbeitsgedächtniskapazität der Testpersonen überprüft, um mögliche Verzerrungen der Ergebnisse aufzudecken. Da ihre Testpersonen keine überdurchschnittliche Arbeitsgedächtniskapazität aufwiesen, kamen sie zu dem Schluss, dass die Testpersonen offenbar in der Lage sind, ihre Chunks hierarchisch zu organisieren.
Weitere empirische Untersuchungen bestätigten Gobet und Simon (1996c), dass ExpertInnen des Schachspiels mehr Schachfiguren memorieren können, als es die Chunking-Theorie (siehe Abschnitt 2.2.4) von Chase und Simon erwarten lässt. In einer Studie konnten Experten neun Schachfelder mit einer Genauigkeit von 70% memorieren, was nach Ansicht von Gobet und Simon die zu erwartende durchschnittliche Anzahl von sieben Chunks deutlich übersteigt. Daher haben sie postuliert, dass SchachspielerInnen, wie auch ExpertInnen anderer „Memorieren-Aufgaben“, Strukturen oder Vorlagen aus ihrem Langzeitgedächtnis zusätzlich zu den Chunks im Kurzeitgedächtnis verwenden, um Informationen in kurzer Zeit zu erfassen und zu speichern. Daraus entstand die Template-Theorie (vgl. Abschnitt 2.2.4).

Auch in anderen Domänen wurde das Verhalten von ExpertInnen beim Memorieren von domänenspezifischen Gegenständen empirisch untersucht. Sala und Gobet (2017) haben eine Meta-Analyse von Studien, die das Verhalten von ExpertInnen und NovizInnen beim Memorieren von zufällig präsentierten Elementen eines strukturierten Materials untersuchen durchgeführt. Dabei wurden 45 Studien betrachtet, die die folgenden Kriterien erfüllen:

- Die untersuchte Domäne dient nicht direkt dem Trainieren des Arbeitsgedächtnisses.
- Die Performanz in der Memorieren- und Reproduzieren-Aufgabe wurde gemessen.
- Eine Art zufälliges Material wurde verwendet.
- Personen verschiedener Expertise-Niveaus nahmen teil.
- NovizInnen waren nicht komplett unvertraut mit dem Material.
- Das zufällige Material wurde erstellt durch Mischen eines strukturierten Materials.
- Die Daten haben eine Richtung des Effekts angegeben oder es konnte die Effektstärke berechnet werden.

Von den 45 betrachteten Studien wiesen 24 ausreichend große Fallzahlen auf, um eine Effektstärke berechnen zu können – darunter auch sechs Studien, die der Domäne „Programmierung“ zuzuordnen sind (siehe Abschnitt 3.2). Insgesamt kamen Sala und Gobet zu dem Ergebnis, dass ExpertInnen besser beim Memorieren und Reproduzieren von randomisierten Materialien auf einem bestimmten Themengebiet abschneiden als NovizInnen. Sie konnten außerdem zeigen, dass dieses Ergebnis signifikant ist. Wenngleich nur von einem moderaten Effekt gesprochen werden kann, ist davon auszugehen, dass ExpertInnen Zugriff auf mehr Strukturen im Langzeitgedächtnis haben. Diese können von ihnen genutzt werden, um Muster in dem randomisierten Material zu erkennen.
In der Informatik ist nach Sala und Gobet die Effektstärke geringer als in anderen Domänen. Dies führen sie auf die Länge der Merkphase von bis zu 10 Minuten zurück. In dieser Zeit können ExpertInnen wie auch NovizInnen neue Chunks im Langzeitgedächtnis speichern und damit ihre Performanz verbessern. Nachfolgend soll die Gedächtnisfähigkeiten von ExpertInnen und NovizInnen der Programmierung näher betrachtet werden.

3.2 Gedächtnisfähigkeiten von ExpertInnen und NovizInnen der Programmierung

In diesem Abschnitt werden die Methodik und die Ergebnisse einzelner Studien aus der Domäne Informatik, aus denen Schlüsse für die in dieser Arbeit verwendete Erhebungsmethode abgeleitet wurden, näher betrachtet. Die ausgewählten Studien beleuchten die Unterschiede zwischen ExpertInnen und NovizInnen der Programmierung beim Memorieren und Reproduzieren von Quelltexten. Deren Ergebnisse werden anschließend zusammengefasst und daraus Forschungsfragen für diese Arbeit abgeleitet. Die Studien (siehe Tabelle 3.1) wurden durch eine Literaturrecherche mit verschiedenen Kombinationen der Schlüsselwörter *working memory*, *programming*, *recall*, *Arbeitsgedächtnis*, *short-term memory*, *computer science*, *expert* und *Informatik* identifiziert. Dabei wurden die Datenbanken von Google Scholar, ACM Digital Library, ScienceDirect, Primo Universität Duisburg-Essen und Springer Link durchsucht. Außerdem wurde nach Literatur gesucht, in der die Arbeiten von Chase und Simon (1973) und De Groot (1965) zitiert wurden. Des Weiteren wurden die gefundenen Arbeiten im Bereich der Informatik,

wiederum auf Zitation in anderen Arbeiten analysiert. Dazu wurde die Datenbank von Scopus verwendet. Eine ausführliche Auflistung der Literaturrecherche ist im Anhang A.1 zu finden.

Die erste betrachtete Studie wurde von Shneiderman (1976) durchgeführt. In einem Experiment zum Memorieren von FORTRAN-Programmen wurden vier Erfahrungsgruppen unterschieden:

- Gruppe 1: 42 Personen ohne Programmiererfahrung
- Gruppe 2: 10 Personen, die einen Einführungskurs in FORTRAN absolviert hatten
- Gruppe 3: 18 Personen, die einen Einführungskurs und einen weiterführenden Kurs in FORTRAN absolviert hatten
- Gruppe 4: 9 Personen, die ein Studium der Informatik abgeschlossen oder als MitarbeiterInnen im Fachbereich Informatik gearbeitet hatten

Die Testpersonen mussten sich innerhalb von drei Minuten ein 20-zeiliges FORTRAN-Programm merken und anschließend innerhalb von vier Minuten das Gemerkte aufschreiben. Danach wurde der Vorgang mit einem 17-zeiligen durchmischten FORTRAN-Programm wiederholt. Um Lerneffekte auszuschließen, musste sich eine Hälfte der Testpersonen zuerst den ungeordneten und danach den geordneten Quelltext merken.
Die Ergebnisse haben gezeigt, dass mit steigender Programmiererfahrung die Anzahl der korrekt reproduzierten Programmzeilen von 7.1 auf 17.3 gestiegen ist, wenn der Quelltext geordnet war. Bei dem ungeordneten Quelltext wurden geringere Unterschiede festgestellt. Die Spannweite in der sich die Testpersonen bewegten, erstreckte sich hier von 4.1 bis 6.4 Zeilen. Shneiderman (ebd.) kam zu dem Schluss, dass Personen mit mehr Programmiererfahrung auf Grund ihrer Vertrautheit mit FORTRAN komplexere Chunks bilden und somit mehr Quelltextzeilen memorieren können.

In einer weiteren Studie von Adelson (1981) wurde die Leistung von fünf ExpertInnen – Dozenten der Einführungsvorlesung in PPL – und fünf NovizInnen – Studierenden der Einführungsvorlesung in PPL – beim Memorieren und Reproduzieren gegenübergestellt. Die Testpersonen sollten sich 16 Zeilen PPL-Quelltext merken. Dieser bestand aus drei einzelnen Programmen und wurde den Testpersonen in zufälliger Reihenfolge Zeile für Zeile präsentiert. Dabei wurde jede Zeile für eine Dauer von 20 Sekunden auf einem Bildschirm angezeigt.
Auch in dieser Studie haben die ExpertInnen (M = 12.73) mehr Quelltextzeilen reproduzieren können als die NovizInnen (M = 9.6). Im Rahmen dieser Studie wurden auch die Pausen aufgezeichnet, die während des Reproduzierens gemacht wurden, um mögliche Chunks identifizieren zu können. Bei der Auswertung zeigte sich, dass NovizInnen eher in syntaktischen Kategorien denken, wohingegen ExpertInnen semantische Kategorien formen. Mit steigender Expertise werden die Kategorien komplexer und verschieben sich von einzeiligen Operationen zu ganzen Methoden.

Aufbauend auf den Arbeiten von Adelson (ebd.) und Shneiderman (1976) hat Barfield (1986) eine weitere Studie in dem Bereich Memorieren und Reproduzieren von Quelltexten durchgeführt. Anders als in den vorherigen Studien wurde ein BASIC-Quelltext auf drei verschiedene Arten den Testpersonen präsentiert: Zunächst geordnet, dann komplett durchmischt und zuletzt zu logischen Einheiten (Chunks) zusammengefasst, welche durchmischt wurden. Die 221 studentischen Testpersonen wurden anhand der Kriterien, die auch Shneiderman verwendet hat, in vier Gruppen eingeteilt. Auch hier zeigten die Ergebnisse, dass mit steigender Expertise die Anzahl der memorierten Zeilen zunimmt – von 10.2 auf 24.0. Der Vergleich der drei

Präsentationsformen ergab, dass die Durchmischung der Chunks nur einen kleinen Einfluss auf die Anzahl der reproduzierten Quelltextzeilen im Vergleich zur geordneten Form hat. Am schlechtesten schnitten alle Gruppen bei der vollständig durchmischten Präsentationsform ab. Dabei ist außerdem aufgefallen, dass sich die ExpertInnen aus Gruppe 4 nicht mehr von den Testpersonen aus Gruppe 3 unterscheiden. Dieses Ergebnis unterstützt das Ergebnis von Shneiderman (1976), der bei durchmischten Quelltexten auch geringere Unterschiede festgestellt hatte. Auch Schmidt (1986) fand keine Korrelation zwischen Expertise und Performanz beim Memorieren von durchmischten Quelltexten und bestätigte damit vorangegangene Studien.

Studie	N	Merkphase	Programmiersp.
Shneiderman (1976)	79	180s	Fortran
Adelson (1981)	10	20s	PPL
Barfield (1986)	221	300s	BASIC
Schmidt (1986)	20	4s	PL/I
Bateson, Alexander und Murphy (1987)	50	180s	Fortran IV
Magliaro und Burtin (1987)	16	120s	BASIC
Guerin und Matthews (1990)	104	600s	COBOL

Tabelle 3.1: Übersicht über die Studien im Bereich Programmierung zu Unterschieden zwischen ExpertInnen und NovizInnen beim Memorieren und Reproduzieren von Quelltexten

Alle diskutierten Studien stimmen im Ergebnis darin überein, dass sich mit steigender Expertise die Fähigkeit, Quelltexte zu memorieren und zu reproduzieren, verbessert. Jedoch scheint diese Erkenntnis nur bei geordneten Quelltexten eindeutig zu sein. Die Präsentationsform, ob einzelne Quelltextzeilen oder der komplette Quelltext präsentiert werden, hat dagegen keine Relevanz. Bei durchmischten Quelltexten ist kein kohärentes Fazit aus den Studien zu ziehen. In einigen Studien (siehe Barfield, 1986; Guerin und Matthews, 1990; Shneiderman, 1976) konnten bei durchmischten Quelltexten keine Unterschiede festgestellt werden, in anderen Studien wurden dagegen Unterschiede sichtbar (siehe Adelson, 1981; Bateson, Alexander und Murphy, 1987; Magliaro und Burtin, 1987).
In den vorgestellten Forschungsdesigns wurde die Phase des Memorierens zeitlich begrenzt und der Phase des Reproduzierens vorangestellt. Abweichend davon wurde in der im Rahmen dieser Arbeit durchgeführten Studie die Verantwortlichkeit für die Strukturierung der memorierten Sequenz auf die Testpersonen, übertragen. Daraus resultieren veränderte kognitive Prozesse mit zusätzlichen Anforderungen, auf die im folgenden Abschnitt eingegangen wird.

3.3 Kognitive Prozesse beim Übertragen von Texten

Grundlegende kognitive Prozesse wie das Editieren eines natürlichsprachlichen Textes erfordern die Verwendung von kognitiven Systemen. Beim Schreibvorgang identifizierten Hayes und Flowers (1980) sechs grundlegende kognitive Prozesse:

- „Planning“: das Planen von Ideen
- „Translation“: das Übersetzen von Ideen in Sätze/Sprache
- „Programming“: das Programmieren von motorischen Abläufen (z.B.: Schreiben, Tippen oder diktieren)
- „Executing“: das Ausführen von motorischen Abläufen

- „Reading“: das Lesen des produzierten Textes
- „Editing“: das Editieren des produzierten Textes

Die Ergebnisse wurden von Kellogg (1996) aufgegriffen und auf Anforderungen an das Arbeitsgedächtnis geprüft. Die Ergebnisse sind in der Tabelle 3.2 aufgeführt.

Grundlegende Prozesse	Arbeitsgedächtniskomponenten		
	Räumlich-visueller Notizblock	Zentrale Exekutive	Phonologische Schleife
Planning	x	x	
Translating		x	x
Programming		x[1]	
Executing			
Reading		x	x
Editing		x	

[1] Für hoch trainierte motorische Abläufe sind diese Anforderungen klein bis nicht erkennbar.

Tabelle 3.2: Die Benutzung der Komponenten des Arbeitsgedächtnisses durch die sechs grundlegenden kognitiven Prozesse beim Schreiben nach Kellogg (1996)

Der Prozess „Programming“ beansprucht nach Kellogg (ebd.) mit steigender Expertise die Zentrale Exekutive in geringer werdendem Umfang, da diese Abläufe dann automatisch geschehen. Auch die von Anderson formulierten Ergebnisse zu Phasen der Expertise (siehe Abschnitt 2.2.3) weisen in die gleiche Richtung.
Die vorgelagerten Prozesses des kreativen Schreibens „Planning“ und „Translating“ sind für das Memorieren und Reproduzieren von Texten nicht relevant. Alle nachgelagerten Prozesse sind insbesondere für das Reproduzieren von Bedeutung. Das gedankliche Strukturieren des gemerkten Textes und die anschließende motorische Umsetzung entspricht dem „Programming“ und „Executing“. Nachfolgend wird das Geschriebene im Rahmen des Prozesses „Reading“ reflektiert und ggf. im Rahmen des Prozesses „Editing“ korrigiert. Das Memorieren beansprucht ebenfalls Ressourcen des Arbeitsgedächtnisses, daher kann es zu Konflikten kommen. Wie sich solche Konflikte auf das Reproduzieren auswirken zeigt die Studie von Hayes und Chenoweth (2006).
Sie haben überprüft, ob beim „Editing“ auch die phonologische Schleife verwendet wird. Dazu haben sie einen künstlichen Konflikt herbeigeführt, indem die Testpersonen das Wort „tap“ während der Bearbeitung sagen mussten. Dies führte zu reiner künstlicher Blockierung der phonologischen Schleife. Sie haben herausgefunden, dass das Arbeitsgedächtnis beim Übertragen und Editieren von Texten verwendet wird und sich durch künstliche Blockierung der phonologischen Schleife die Tippgeschwindigkeit verringert bzw. die Fehlerrate erhöht. Es ist zu vermuten, dass dieser Effekt auch beim Memorieren und Reproduzieren auftritt.

3.4 Forschungsfragen

In den betrachteten Studien folgt auf eine Phase des Memorierens eine des Reproduzierens der dargebotenen strukturierten Materialien. In allen vorgestellten Studien wurde die Phase des Memorierens zeitlich begrenzt. In dieser Arbeit soll der Prozess des Übertragens näher an eine Situation aus der Praxis angelehnt werden. Dies geschieht durch das dynamische

Ineinandergreifen der beiden Phasen. Die Testpersonen organisieren den Übertragungsprozess eigenständig, indem sie die Länge und Anzahl der Merkphasen selbst bestimmen.
Als eine weitere Abgrenzung von der bisherigen Forschung in diesem Bereich, wird die eingesetzte Programmiersprache verändert. Das Akquirieren von Testpersonen auf den Niveaustufen ExpertIn und NovizIn wäre bei der Auswahl einer Programmiersprache, die in den vorgestellten Studien zum Einsatz kam, durch veränderte Benutzerpräferenzen erschwert worden. Die Verbreitung der Programmiersprache Java (siehe TIOBE Index[1]) lässt die Erwartung zu, eine ausreichende Anzahl von ExpertInnen und NovizInnen gewinnen zu können.
Die Veränderung der methodischen Umsetzung und der Programmiersprache führt dazu, dass zunächst verifiziert werden muss, ob mit dem entwickelten Testinstrument (siehe Abschnitt 4.5) Unterschiede zwischen ExpertInnen und NovizInnen der Programmierung nachweisbar sind. Als Indikatoren für Unterschiede wurden die „Längen der Merkphasen“ und die „Anzahl der Merkphasen“ festgelegt. Außerdem wurde in Anlehnung an die Studie von Hayes und Chenoweth (2006) die Tippgeschwindigkeit und die Fehlerrate als weitere mögliche Indikatoren aufgenommen. Diese Überlegungen führen zu der ersten Forschungsfrage:

F1: *Welche Unterschiede zwischen ExpertInnen und NovizInnen der Programmierung sind mit Hilfe des entwickelten Messinstruments bezüglich*
(a) *ihrer Tippgeschwindigkeit,*
(b) *ihrer Fehlerrate,*
(c) *der Gesamtlänge der Merkphasen und*
(d) *der Anzahl der Merkphasen*
messbar?

Der Forschungsstand deutet daraufhin, dass ExpertInnen der Programmierung NovizInnen beim Memorieren und Reproduzieren von Quelltexten überlegen sein müssten. Es ist zu vermuten, dass diese Unterschiede durch voneinander abweichende Organisationsformen von Informationen im Arbeitsgedächtnis zu erklären sind.
Das Memorieren nimmt Ressourcen des Arbeitsgedächtnisses in Anspruch. Deshalb ist auf Basis der Forschung von Hayes und Chenoweth (ebd.) davon auszugehen, dass dies einen Einfluss auf die Tippgeschwindigkeit haben wird. Dieser sollte sich jedoch zwischen ExpertInnen und NovizInnen unterscheiden, da ExpertInnen auf Strukturen in der Form von Templates aus dem Langzeitgedächtnis zurückgreifen können und somit deren Arbeitsgedächtnis weniger beansprucht werden sollte. Daher kann die folgende Hypothese formuliert werden:

H1a: *Die Tippgeschwindigkeit von ExpertInnen beim Memorieren und Reproduzieren von Quelltexten ist höher als die von NovizInnen.*

Des Weiteren wird beim Memorieren und Reproduzieren der grundlegende Prozess des „Programming“ verwendet. Dieser benötigt Ressourcen der Zentralen Exekutive, jedoch sinkt der Bedarf bei steigender Expertise. Daraus kann gefolgert werden, dass ExpertInnen wahrscheinlich weniger Fehler beim Schreiben von Schlüsselwörtern begehen, da diese für sie weniger kognitiv anspruchsvoll sind als für NovizInnen.

H1b: *Die Fehlerrate von ExpertInnen beim Memorieren und Reproduzieren von Quelltexten ist geringer als die von NovizInnen.*

Da ExpertInnen nach Gobet und Simon (1996c) wahrscheinlich Templates verwenden, um Informationen im Arbeitsgedächtnis zu speichern, sollten sie sich auf Informationen, die nicht in

[1] `https://www.tiobe.com/tiobe-index//`, abgerufen am 20.04.2017

den Templates enthalten sind, konzentrieren können. Somit sollten ExpertInnen in kürzerer Zeit einen größeren Umfang an Informationen speichern können. Daraus leiten sich die folgenden Hypothesen ab:

> ***H1c:*** *Die Gesamtlänge der Merkphasen beim Memorieren und Reproduzieren von Quelltexten ist bei ExpertInnen geringer als bei NovizInnen.*

> ***H1d:*** *Die Anzahl der Merkphasen beim Memorieren und Reproduzieren von Quelltexten ist bei ExpertInnen geringer als bei NovizInnen.*

Die vorgestellten Forschungsergebnisse von Kellogg (1996) (siehe Abschnitt 3.3) lassen darauf schließen, dass ein nachgewiesener Unterschied zwischen ExpertInnen und NovizInnen nicht unbedingt vollständig durch die erhobene Programmiererfahrung erklärt werden kann, da verschiedene kognitive Prozesse beim Schreiben Ressourcen des Arbeitsgedächtnisses in Anspruch nehmen. Daher wurde folgende Forschungsfrage formuliert:

> ***F2:*** *Inwiefern erklären die Konstrukte Aufmerksamkeit, Arbeitsgedächtnis und Programmiererfahrung die Fähigkeit beim Memorieren und Reproduzieren von Quelltexten?*

Die natürliche Kapazität des Arbeitsgedächtnisse kann zwischen Personen variieren. Daher könnte die Leistung beim Memorieren und Reproduzieren von Quelltext durch die natürliche Kapazität des Arbeitsgedächtnisses beeinflusst werden.
Des Weiteren können unterschiedliche Ausprägungen der Aufmerksamkeitsspanne dazu führen, dass an den bearbeiteten Aufgaben nicht fokussiert gearbeitet wird. Somit kann zum einem die Fähigkeit beim Memorieren und Reproduzieren von Quelltext beeinflusst werden und zum anderen die Leistung des Arbeitsgedächtnisses, da die zentrale Exekutive, welche die Prozesse des Arbeitsgedächtnisses steuert, eng mit der Aufmerksamkeit verwandt ist.
Außerdem ist nach den Erkenntnissen aus den theoretischen Grundlagen und dem Forschungsstand davon auszugehen, dass die Programmiererfahrung die Fähigkeit, Quelltexten zu memorieren und zu reproduzieren beeinflusst.
Mit diesen Annahmen wurde das Modell in Abbildung 3.1 formuliert.

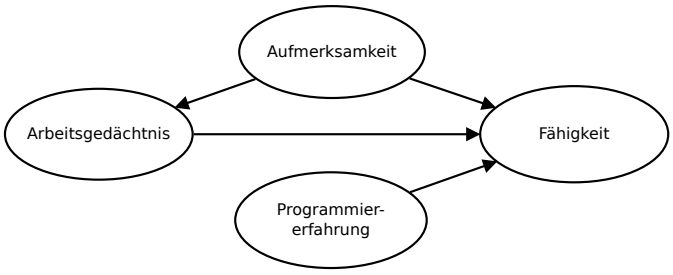


Abbildung 3.1: Strukturmodell des Einflusses der Kontrollgrößen auf die Fähigkeit im Test

Das Memorieren von strukturiertem Material wird in der kognitionspsychologischen Forschung mit der Chunking- und der Template-Theorie erklärt (siehe Abschnitt 2.2.4). Im Rahmen dieser Arbeit soll weitergehend geprüft werden, ob ExpertInnen und NovizInnen erkennbar auf diese Strategien zurückgreifen. Dazu wird folgende weitere Forschungsfrage gestellt:

> ***F3:*** *Welche Formen der Strukturierung wählen ExpertInnen und NovizInnen der Programmierung beim Übertragen und Memorieren von Quelltexten?*

4 Konzeption und Methodik der empirischen Untersuchung

4.1 Begründung des Forschungsansatzes

Das gewählte Untersuchungsdesign wird aus den bisherigen Forschungsarbeiten auf dem betrachteten Gebiet abgeleitet. Die Ansätze von de Groot und Adelson werden dabei kombiniert und um weitere Perspektiven ergänzt.
Für die empirische Untersuchung wurde eine Querschnittsstudie als Forschungsdesign ausgewählt. Die Querschnittsstudie zählt zu den Ex-post-facto-Designs, in denen mögliche Zusammenhänge zwischen Phänomenen als Resultat bestimmter Einflussgrößen untersucht werden. Zwei oder mehrere Stichproben werden bezüglich ihrer Ausprägung in einem bestimmten Faktor unterschieden. In der hier vorgestellten Untersuchung ist der interessierende Faktor und damit die unabhängige Variable die Programmiererfahrung der Testpersonen. Daraus werden, in Anlehnung an Adelson, NovizInnen und ExpertInnen als disjunkte Gruppen definiert.
In einer Querschnittstudie werden einmalig Daten gesammelt. Zur Beantwortung der formulierten Forschungsfragen wurde dieser Ansatz gewählt, da Kompetenzentwicklungen im zeitlichen Verlauf hier nicht betrachtet werden sollen.
Die Untersuchung gliedert sich in drei separate Module: Im ersten Modul der Untersuchung werden mit der Arbeitsgedächtnisleistung und der Aufmerksamkeit zwei kognitive Konstrukte erhoben. Im darauffolgenden Hauptmodul wird ein eigens konstruiertes Messinstrument zur Erfassung der Tippgeschwindigkeit und Merkfähigkeit bezüglich natürlichsprachlicher Texte und Quelltexte eingesetzt. Das dritte Modul der Untersuchung ist ein Kurzfragebogen, in dem neben der Selbstauskunft zur Programmiererfahrung auch soziodemografische Variablen erfasst werden.
Mit der Entwicklung dieses Messinstruments wird ein Baustein für die Kompetenzmessung im Rahmen des Projekts COMMOOP angestrebt. Ziel des Projekts COMMOOP ist die web-basierte Kompetenzmessung. Aus dieser Konstellation ergibt sich für diese empirische Untersuchung die Einschränkung auf eine web-basierte Lösung.
Für die Validierung des Messinstruments wurden zwei Pretests durchgeführt. Die Erkenntnisse aus den Pretests und die angestoßenen Modifikationen werden zur Nachvollziehbarkeit der Instrumentenentwicklung im Unterkapitel 5.1.1 beschrieben. Das folgenden Kapitel geht zunächst auf die finale Version der Messinstrumente ein.

4.2 Vorstellung der Messinstrumente

Das in der Arbeit verwendete Messinstrument besteht aus einem kognitionspsychologischen und einem informatischen Modul. Diese werden in den folgenden Abschnitten vorgestellt.

4.2.1 Kognitionspsychologisches Modul

Im Rahmen des kognitionspsychologischen Moduls werden die nachfolgend beschriebenen Tests durchgeführt.

d2 Aufmerksamkeits-Belastungs-Test

Der d2 Aufmerksamkeits-Belastungs-Test wurde entwickelt, um die Aufmerksamkeitsleistung zu ermitteln, die die Geschwindigkeit und die Genauigkeit einer Bearbeitung beeinflusst. Die Testpersonen sollen die Kombination aus dem Buchstaben „d“ und zwei Strichen aus einer Reihe mit ähnlich aussehenden Distraktoren heraussuchen und markieren. Dies geschieht unter einer Zeitrestriktion (20 Sekunden für 47 Zeichen), sodass die Testpersonen gezwungen sind, schnell zu arbeiten. Im Vorhinein wird den Testpersonen die Anweisung gegeben, Fehler möglichst zu vermeiden. Insgesamt sind hier 14 solcher Reihen zu bearbeiten, woraus sich eine Durchführungszeit von ca. 8 Minuten ergibt. Im Rahmen des Tests werden verschiedene

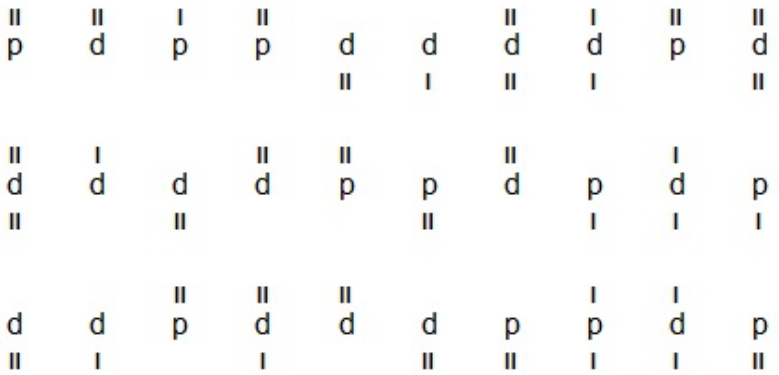


Abbildung 4.1: Ausschnitt d2 Aufmerksamkeits-Belastungs-Test

Kenngrößen erhoben. Die klassische Handanweisung sieht den D2-GZ-Wert (Gesamtzahl aller bearbeiteten Zeichen) als Kriterium für die Leistungsmenge, den D2-F-Wert (Fehlerrohwert) und den F%-Wert (Fehlerprozent-Wert) zur Bestimmung der Leistungsgüte, den D2-GZ-F-Wert (Gesamtzahl aller bearbeiteten Zeichen abzüglich Fehlerrohwert) als übergeordneten Gesamttestwert, die Schwankungsbreite (D2-SB) und die Fehlerverteilung als Maßzahlen für den Testverlauf vor (Brickenkamp, 1967, S. 12). In einer neueren Auflage (vgl. Brickenkamp, 1994) wurde der Konzentrationsleistungswert D2-KL (Anzahl der richtig durchgestrichenen Zeichen verringert um die Anzahl der Verwechslungsfehler), als eine weitere Kenngröße aufgenommen. Dieser gilt als aussagekräftig und robust gegenüber Verfälschungen durch abweichende Bearbeitungen und wurde deshalb zusammen mit dem D2-GZ-F zur Einordnung verwendet werden.

I-S-T 2000-R

Der I-S-T 2000-R-Intelligenztest (vgl. Amthauer u. a., 2006) besteht aus neun Modulen, die unterschiedliche Aspekte der Intelligenz prüfen, sowie aus zwei Zusatzmodulen zur Bestimmung der Merkfähigkeit. Da in dieser Arbeit nicht die Intelligenz, sondern nur die Leistung des Arbeitsgedächtnisses relevant ist, werden nur die Zusatzmodule eingesetzt. Das erste Zusatzmodul prüft die verbale Merkfähigkeit und somit die Kapazität der phonologischen Schleife. Das zweite Zusatzmodul misst die figurale Merkfähigkeit und somit den räumlich-visuellen

Notizblock. Je nach eingesetzter Merkstrategie der Testpersonen können die verwendeten Bereiche des Arbeitsgedächtnisses bei den Merkmodulen variieren, wodurch die beiden Bereiche nicht vollständig voneinander abgegrenzt werden können. In beiden Modulen haben die Testpersonen jeweils eine Minute Zeit, sich eine Wörterliste bzw. mehrere Figurenpaare einzuprägen, danach sollen sie in einem begrenzten Zeitraum das Gemerkte reproduzieren. Insgesamt dauert der Test somit ca. zehn Minuten.

Corsi-Block-Tapping-Test

Der Corsi-Block-Tapping-Test (vgl. Corsi, 1972) wird zur Bestimmung der Kapazität des räumlich-visuellen Notizblocks genutzt. Beim klassischen Test berührt ein Versuchsleiter eine Sequenz von bis zu neun Blöcken. Die Aufgabe der Testpersonen besteht darin, diese Sequenz nachzuahmen. Am Anfang enthält die Sequenz zwei Blöcke. Wenn die Testperson diese richtig nachgeahmt hat, wird die Sequenz um einen Block erhöht. Lag die Testperson falsch, wird eine weitere Sequenz mit der aktuellen Blockanzahl angezeigt. Wenn zwei aufeinanderfolgende Fehler auftreten, wird der Test beendet. Die Blockanzahl der zuletzt richtig nachgeahmten Sequenz ist als sogenannte Corsi-Spanne definiert. Da dieser Test jedoch von dem Versuchsleiter unbewusst beeinflusst werden kann, zum Beispiel durch unregelmäßiges Anzeigen der Sequenzen, wurde vom Autor der vorliegenden Arbeit eine computerbasierte Fassung des Tests implementiert.

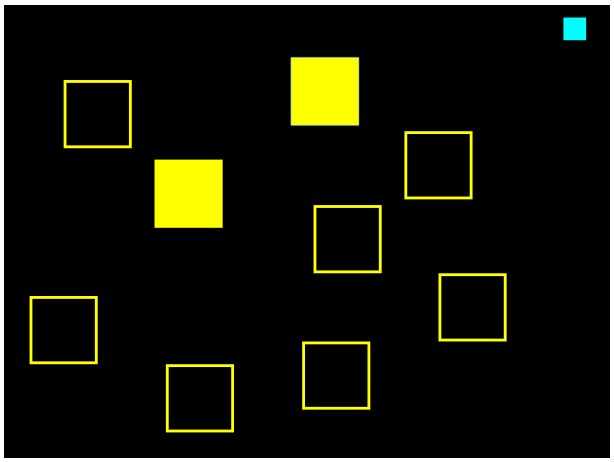


Abbildung 4.2: Computer-basierte Umsetzung des Corsi-Block-Tapping-Tests

Dazu wurde die Beschreibung von Brunetti, Gatto und Delogu (2014) zur Umsetzung eines computerbasierten Corsi-Block-Tapping-Tests als Vorlage verwendet. Die Anordnung und die Sequenzen wurden entsprechend der Darstellung von Kessels u. a. (2000) durchgeführt. Eine detaillierte Beschreibung der Implementierung ist auf der beigelegten CD zu finden.

4.2.2 Informatisches Modul

Im ersten Teilbereich des informatischen Moduls soll die Tippgeschwindigkeit der Testpersonen gemessen werden, um diese als möglichen Störfaktor bei der Bearbeitung des zweiten Teilbereichs kontrollieren zu können. Dazu wird auf der linken Seite des Bildschirms der abzuschreibende Quelltext angezeigt, welcher in ein Textfeld auf der rechten Seite des Bildschirms übertragen werden soll (siehe Abbildung 4.3). Im zweiten Teilbereich soll die Fähigkeit der Testperson getestet werden, Quelltext zu memorieren und zu reproduzieren. Zu diesem Zweck werden die Testpersonen aufgefordert, Quelltexte analog zum ersten Teilbereich in ein Textfeld

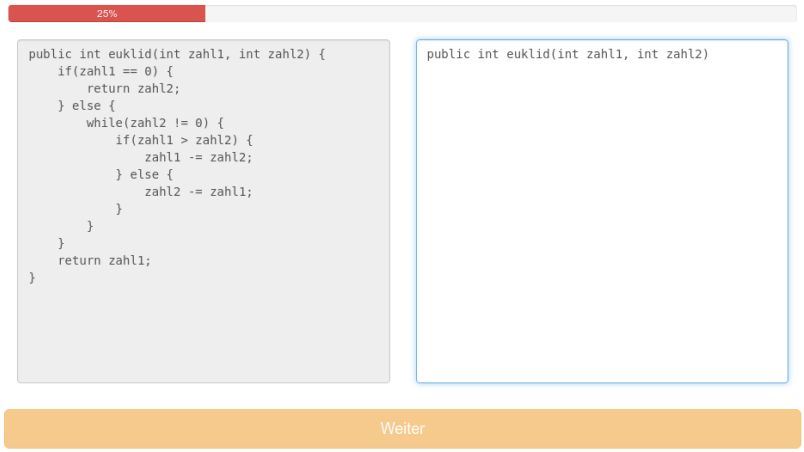


Abbildung 4.3: Aufgabe zur Erfassung der Tippgeschwindigkeit

zu übertragen. Der zu übertragende Text ist zunächst versteckt, kann jedoch durch Drücken von STRG, ALT und einer Schaltfläche oberhalb des Textfeldes aufgedeckt werden (siehe Abbildung 4.4). Durch die Kombination der Aktionen muss die Testperson beide Hände benutzen, sodass sie sich keine Notizen machen kann. Damit wird sichergestellt, dass ihre Fähigkeit, Quelltext zu memorieren, nicht verfälscht wird. Da jedoch weitere Störfaktoren auftreten können

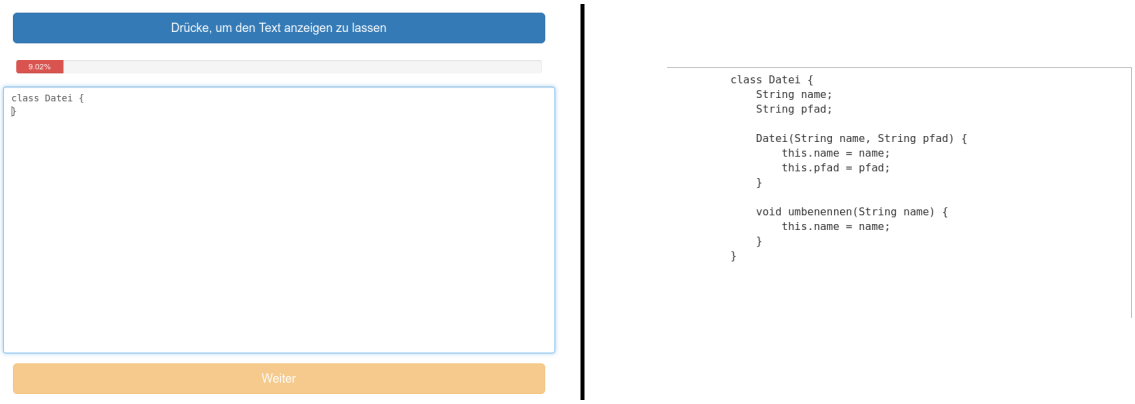


Abbildung 4.4: Aufgabe zur Erfassung der Merkfähigkeit

wie z.B. der Umgang mit der Maus und der Tastatur, wurde zu den Quelltexten jeweils ein natürlichsprachliches Item entwickelt. Durch Bestimmung der Differenz der Kennwerte der informatischen Items mit den natürlichsprachlichen Items können diese Störfaktoren kontrolliert werden. Die gemessenen Kennwerte werden nächst in Relation zur Testperson selbst gesetzt, bevor sie mit der Gesamtheit verglichen werden. Um Testpersonen mit diesem Instrument vertraut zu machen, wird für beide Teilbereiche jeweils ein Testitem eingesetzt.

Entwicklung der Items

Für den zweiten Teilbereich, die versteckte Variante, wurden drei Itemgruppen bestehend aus jeweils drei Items konstruiert. Der Test beschränkt sich auf neun Items, um die Gesamtdauer des Tests sinnvoll zu begrenzen. Jede Gruppe bestand aus einem Quelltext mit einer Klassenstruktur, einem Quelltext mit einem Algorithmus und einem natürlichsprachlichen Text. Die Gruppen wurden so konstruiert, dass sie in den Dimensionen Zeichenanzahl und kognitive Komplexität vergleichbar sind.

```
public class Haus {
    private int nummer;
    private String farbe;

    public void streiche(
        String farbe) {
        this.farbe = farbe;
    }
}
```

```
Lettland ist ein Staat
im Osten von Europa. Er
liegt an der Ostsee und
gehört zu den baltischen
Staaten. Die beiden
anderen sind Estland und
Litauen.
```

```
boolean istVielfaches(int
    zahl, int vielfaches)
    {
    if(vielfaches % zahl
        == 0) {
        return true;
    } else {
        return false;
    }
}
```

Abbildung 4.5: Erste Itemgruppe

Dazu wurde die kognitive Komplexität der Klassenstrukturen mit der WCC nach (Misra und Akman, 2008) berechnet. Für die kognitive Komplexität der Algorithmen wurde die MC verwendet, welche von der WCC für grundlegende Kontrollstrukturen verwendet wird (vgl. Unterkapitel 2.3). Die kognitive Komplexität der Texte wurde mit verschiedenen Lesbarkeitsindizes bestimmt. Die Auswahl der Lesbarkeitsindizes wurde in Kapitel 2.3 dargestellt. Aus diesen Überlegungen resultieren drei Gruppen. Die erste Itemgruppe enthält die Items mit der geringsten Zeichenanzahl und niedrigster kognitiver Komplexität. Die Items der ersten Gruppe sind in Abbildung 4.5 dargestellt. Die dritte Gruppe enthält die Items mit der höchsten Zeichenanzahl und kognitiven Komplexität (siehe Tabelle 4.1 — Erklärungen zu den Abkürzungen sind im Abbildungsverzeichnis zu finden). Für den ersten Teilbereich wurden zwei Items konstruiert, ein

	Klassenstruktur		Text			Algorithmus	
	Zeichen	WCC	Zeichen	FRE_{DE}	WS	Zeichen	MC
Gruppe 1	147	3	149	79	5	146	2
Gruppe 2	219	5	206	69	8	214	4
Gruppe 3	237	7	221	49	10	244	8

Tabelle 4.1: Itemgruppen

natürlichsprachlicher Text und ein Algorithmus, die vergleichbar in den Dimensionen Zeichenanzahl und kognitiver Komplexität zu den Items der letzten Gruppen waren. Eine Auflistung aller verwendeten Items ist im Anhang (A.3.1 und A.3.2) zu finden.

4.2.3 Fragebogen

Der Fragebogen (siehe Anhang A.1) besteht aus zwei Teilen. Zum einen werden allgemeine personenbezogene Daten erfragt, und zum anderen wird eine Selbsteinschätzung bezüglich der eigenen Programmiererfahrung verlangt. Zu den personenbezogenen Daten gehören zum einen der Studiengang, das Fachsemester, das Geschlecht und das Alter, aber zum andern auch die Händigkeit. Diese muss erhoben werden, da diese Angabe von dem verwendeten d2 Aufmerksamkeits-Belastungs-Tests verlangt wird.
Zur Erhebung der Programmiererfahrung wurden Selbsteinschätzungsfragen gewählt, da Siegmund u. a. (2014) herausgefunden haben, dass diese Fragen am besten die Programmiererfahrung bestimmen. Dazu haben die Autoren des Artikels Fragen aus Fragebögen gesammelt, die bereits eingesetzt wurden, um die Programmiererfahrung zu bestimmen. Diese wurde Studierenden zu Beantwortung geben. Des Weiteren mussten die Studierenden Programmier-

verständnisaufgaben bearbeiten. Auf diesem Wege haben die Autoren herausgefunden, dass Selbsteinschätzungsfragen am besten mit dem Abschneiden in den Programmierverständnisaufgaben korrelieren. Für den vorliegenden Fragebogen wurden fünf Items ausgewählt, die auf einer zehnstufigen Likert-Skala beantwortet werden sollten:

- Wie schätzen Sie ihre Programmiererfahrung ein?
- Wie erfahren sind Sie mit der Programmiersprache Java?
- Wie erfahren sind Sie mit dem Paradigma Objektorientierte Programmierung?
- Wie schätzen Sie Ihre Programmiererfahrung verglichen mit ExpertInnen mit 20-jähriger Berufserfahrung ein?
- Wie schätzen Sie Ihre Programmiererfahrung verglichen mit Ihren KommilitonInnen ein?

Insgesamt nimmt die Bearbeitungsdauer des Fragebogens ca. fünf Minuten in Anspruch. Eine detaillierte Beschreibung der Implementierung kann auf der beigelegten CD gefunden werden.

4.3 Strukturierung der Untersuchung

Um die Reproduzierbarkeit dieser Arbeit gewährleisten zu können, wurde aus den gewonnenen Erfahrungen aus den Pretests (siehe Unterkapitel 5.1.1) und aus der Analyse der Messinstrumente ein Verlaufsplan der Untersuchung (siehe Tabelle 4.2) erstellt. Dadurch kann auch bei mehrmaliger Durchführung der Untersuchung garantiert werden, dass alle nach dem gleichen Schema durchgeführt wurden. Damit die Anleitung der Testpersonen in den einzelnen Testphasen einen hohen Grad an Objektivität aufweist, wurde zusätzlich eine Präsentation erstellt, die alle notwendigen Anweisungen enthält. Die Präsentation ist auf der beigelegten CD zu finden.

Zeit (in min)	Phase	Medium
2	Begrüßung und Präsentation des Ablaufs der Studie	Präsentation
2	Vorstellung des d2 Aufmerksamkeits-Belastungs-Test	Präsentation
2	d2 Aufmerksamkeits-Belastungs-Test: Übungszeile und Klärung von Verständnisproblemen	Testmodul 1
4:40	d2 Aufmerksamkeits-Belastungs-Test	Testmodul 1
1	Vorstellung des I-S-T 2000-R	Präsentation
1	Einprägen (verbal)	Testmodul 2
2	Reproduktion (verbal)	Testmodul 2
1	Einprägen (figural)	Testmodul 2
3	Reproduktion (figural)	Testmodul 2
2	Vorstellung des Corsi-Block-Tapping-Tests (vorwärts) und Probeitem	Präsentation
3	Corsi-Block-Tapping-Test (vorwärts)	Notebook
2	Vorstellung des Corsi-Block-Tapping-Tests (rückwärts) und Probeitem	Präsentation
3	Corsi-Block-Tapping-Test (rückwärts)	Notebook
2	Vorstellung des Tippgeschwindigkeits-Tests und Probeitem	Präsentation, Notebook
6	Tippgeschwindigkeits-Test	Notebook

Zeit (in min)	Phase	Medium
2	Vorstellung des Merk-Tests und Probeitem	Präsentation, Notebook
16	Merk-Test	Notebook
5	Fragebogen	Notebook
1	Verabschiedung	

Tabelle 4.2: Verlaufsplan der Untersuchung

4.4 Auswertungmethodik

Die statistische Auswertung der Daten wurde mit Hilfe der Software R 3.3.3 (R Core Team, 2017) realisiert. Unvollständige Bearbeitungen wurden vor der quantitativen Analyse aus dem Datensatz entfernt.

Beschreibung der Stichprobe

Die soziodemografischen Daten der Testpersonen wurden mittels deskriptiver Statistik und Häufigkeitsanalysen ausgewertet. Dabei wurden der Studiengang, das Fachsemester, das Alter und das Geschlecht berücksichtigt.

Zur Einteilung der Gruppen ExpertInnen und NovizInnen wurde auf die Selbsteinschätzung der Testpersonen zurückgegriffen. Mit dem Paket „psych“ (Revelle, 2016) und der Funktion „alpha“ wurde der Wert für Cronbachs Alpha berechnet, der die innere Konsistenz der Items angibt. Cronbachs Alpha gilt als Indikator für die Eindimensionalität eines Konstrukts. Wenn diese gegeben ist, können die einzelnen Werte der fünf Likert-Items aufsummiert werden. Der Median dieses Scores kann als Trennlinie zwischen ExpertInnen und NovizInnen angenommen werden.

Auswertungsmethodik erste Forschungsfrage

Zur Beantwortung der ersten Forschungsfrage wurden Gruppenvergleiche zur Feststellung signifikanter Differenzen durchgeführt.
Sofern die Bedingung der Normalverteilung der interessierenden Variablen in der Grundgesamtheit gegeben war, wurde ein t-Test eingesetzt. Ausschlaggebend war hierfür das Ergebnis des Kolmogorov-Smirnov-Anpassungstests. Anderenfalls wurde auf den Mann-Whitney-U-Test zurückgegriffen, der zwar ohne die Annahme der Normalverteilung auskommt, dafür jedoch eine geringere Teststärke aufweist.
Bei der Durchführung der Tests wurden anstelle der Rohwerte die Differenzen zwischen den informatischen und den natürlich sprachlichen Items einer Gruppe verwendet. In einer Gruppe mit je drei Items wurden für jeden interessierenden Indikator zwei Differenzen ermittelt: Die Differenz zwischen dem Item mit der Klassenstruktur und dem natürlichsprachlichen Item sowie die Differenz zwischen dem Item mit dem Algorithmus und dem natürlichsprachlichen Item. Durch die Normierung gegenüber den individuellen verschiedenen Parametern in der natürlichsprachlichen Umgebung wurde die Vergleichbarkeit der Leistungsfähigkeit der Testpersonen erreicht.

Auswertungsmethodik zweite Forschungsfrage

Zur Beantwortung der zweiten Forschungsfrage wurde das aus der Theorie hergeleitete Strukturmodell durch ein Messmodell erweitert (siehe Abbildung 4.6). Für die Faktoren des Konstrukts Fähigkeit wird der Kennwert aus Forschungsfrage 1 verwendet, der am besten die Unterschiede zwischen ExpertInnen und NovizInnen misst. Diese beiden Modelle wurden dann in einem zweistufigen Prozess analysiert, wobei zuerst das Messmodell und danach das Strukturmodell untersucht wurde (vgl. Chin, 2010, S. 669).

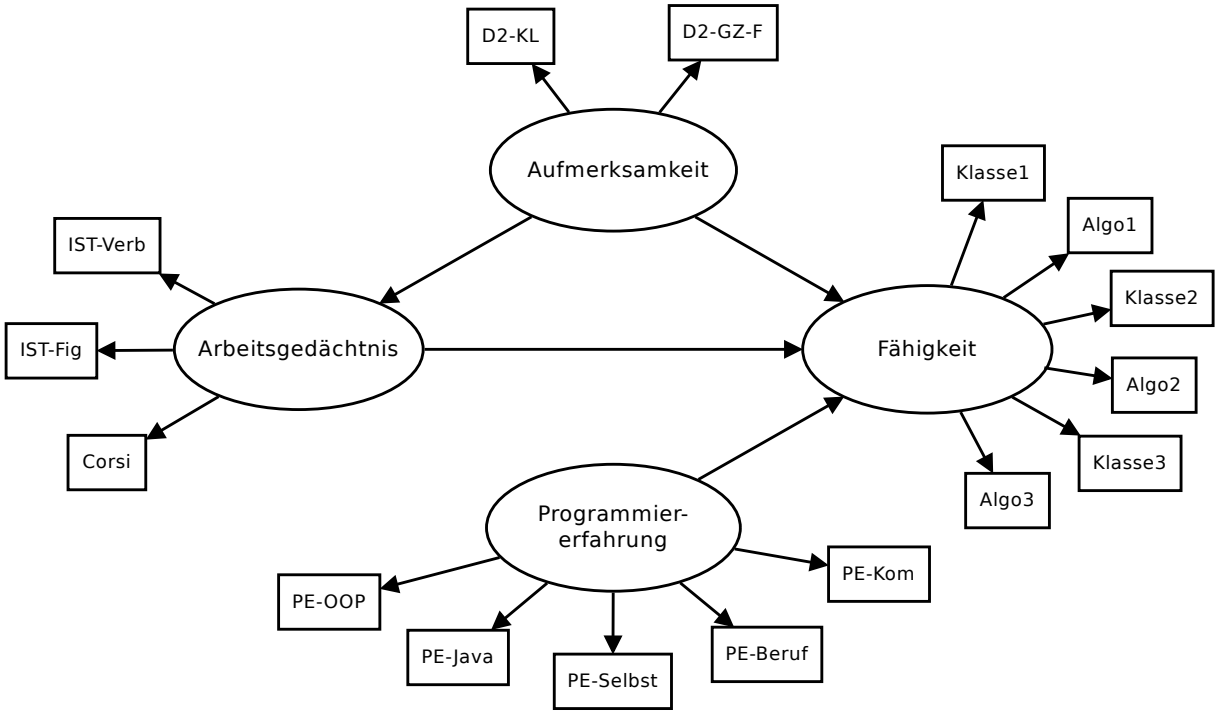


Abbildung 4.6: Strukturmodell mit Messmodell

Grundsätzlich gibt es dabei zwei Ansätze der Analyse. Zum einen kann ein varianz-basierter Ansatz PLS-SEM verwendet werden, zum anderen ein kovarianz-basierter Ansatz CB-SEM. Jeder dieser Ansätze ist für einen bestimmten Anwendungskontext geeignet. Um zwischen den Ansätzen entscheiden zu können, wurden von Hair (2013, S. 19) Faustregeln definiert. Da in dieser Studie kleine Stichproben zu erwarten sind, wurde die PLS-SEM Methode gewählt. Mit dieser Methode kann das Modell auf drei Ebenen validiert werden (Tenenhaus u. a., 2005, S. 172): Auf der Ebene der Qualität des Messmodells, der Qualität des Strukturmodells und der strukturellen Regressionsgleichungen.

Qualität des Messmodells

Die Qualität des Messmodells wird durch eine Analyse der Beziehung zwischen den gemessenen Daten und den Konstrukten bewertet. Die Konstrukte werden üblicherweise auf interne Konsistenz und auf Validität geprüft. In dieser Arbeit werden Cronbachs alpha und Dillon-Goldsteins rho dazu verwendet, die Eindimensionalität eines Konstrukts und somit die interne Konsistenz, zu testen. Der Wertebereich erstreckt sich von 0 bis 1, wobei ein höherer Wert eine höhere interne Konsistenz bedeutet. Bei Werten größer als 0.7 darf von der Eindimensionalität eines Konstrukts ausgegangen werden (vgl. ebd.). Zur Prüfung der Validität werden Konvergenz- und Diskriminanzvalidität herangezogen. Konvergenzvalidität wird durch die Analyse der Ladungen der Items auf ein Konstrukt überprüft. Dabei gilt als Richtgröße,

dass Items akzeptiert werden sollten, die eine Ladung von 0.7 oder größer haben (vgl. Hair, 2013, S. 103). Als ein weiteres Kriterium zur Bestimmung der Konvergenzvalidität wird die durchschnittlich erfasste Varianz betrachtet. Diese sollte größer als 0.5 sein (vgl. ebd., S. 103). Die Diskriminanzvalidität kann durch Betrachtung der Cross-loadings zwischen den Faktoren und den Konstrukten analysiert werden. Cross-loadings sind die Korrelation eines Faktors mit den anderen Konstrukten des Modells. Die Diskriminanzvalidität ist gegeben, wenn alle Ladungen der Faktoren auf ein Konstrukt größer sind als alle Cross-loadings (vgl. ebd., S. 107).

Qualität des Strukturmodells

Die Analyse des Strukturmodells zeigt, inwiefern die prognostizierten Beziehungen zwischen den Konstrukten empirisch nachgewiesen werden können. Dazu wird zum einen der Determinationskoeffizient, welcher die geteilte Varianz zwischen den Variablen aufzeigt, berechnet. Für diesen Wert kann keine Akzeptanzregel formuliert werden, jedoch werden Werte über 0.20 als hoch angesehen (ebd., S. 175). Zum anderen wird die Stärke der Pfadkoeffizienten berechnet.

Bootstrapping-Methode

Um die Signifikanz der Pfadkoeffizienten zu testen, wurde die Bootstrapping-Methode eingesetzt. Beim Bootstrapping wird eine große Anzahl an zufälligen Stichproben aus dem originalen Datensatz gezogen, die jeweils wieder zurückgelegt werden. Das heißt, dass eine zufällige Stichprobe eine Beobachtung mehrmals enthalten kann oder dass eine Beobachtung in keiner zufälligen Stichprobe enthalten sein kann. Hair (ebd., S. 132) empfiehlt, 5000 zufällige Stichproben zu generieren. Jede dieser Stichproben wird dazu verwendet, das Modell zu schätzen. Die Bootstrapping-Methode gibt die Signifikanz der Parameter durch ein Bootstrap-Konfidenzintervall an. Wenn das Intervall nicht die Null enthält, dann ist der Parameter signifikant. Außerdem kann über die Breite des Bootstrap-Konfidenzintervalls die Stabilität eines Parameters bestimmt werden. Je breiter das Intervall ist, desto geringer ist die Stabilität der Schätzung des Parameters (ebd., S. 136).

Vergleich von ExpertInnen und NovizInnen

Zusätzlich soll geprüft werden, ob sich die Pfadkoeffizienten zwischen ExpertInnen und NovizInnen unterscheiden. Dazu wird eine Stichprobenwiederholungsmethode verwendet. Das Paket „plspm“ bietet zum einen die Möglichkeit, eine Bootstrap-t-Test-Methode zu verwenden oder eine Permutations-Methode. Um die Bootstrap-t-Test-Methode anwenden zu können, müssen die Voraussetzungen des t-Tests erfüllt sein. Das heißt, dass die Daten in der Gesamtheit normalverteilt und dass die Gruppen gleich groß sein müssen. Die Permutations-Methode gibt keine Voraussetzungen bezüglich der Verteilung vor. Die Bootstrap-t-Test-Methode wird verwendet, wenn der Kolmogorov-Smirnov-Anpassungstest positiv bezüglich der Normalverteilung ausfällt.

Auswertungsmethodik dritte Forschungsfrage

Für die Auswertung der dritten Forschungsfrage werden zunächst Transkripte von den Bearbeitungen erstellt. Die Umwandlung der dynamischen in eine statische Form vereinfacht die Diskussion sowie die Auswertung der Bearbeitungen. Es gibt zwei Ansätze wie eine qualitative Analyse durchgeführt werden kann – den deduktiven und den induktiven Ansatz (Spencer und Snape, 2003). Beim deduktiven Ansatz wird ein aus der Theorie hergeleitetes Gerüst verwendet, um die erhobenen Daten zu analysieren. Beim induktiven Ansatz hingegen werden die

erhobenen Daten dazu verwendet, die Analyse zu strukturieren. Da in dieser Arbeit die Formen der Strukturierung von ExpertInnen und NovizInnen beim Memorieren und Reproduzieren von Quelltexten explorativ erschlossen werden sollen und somit hypothesenbildend gearbeitet wird, wird der induktive Ansatz gewählt.

4.5 Funktionalität, Datenentwurf und Implementierung des Messinstruments

Dieser Abschnitt beschreibt die Planung, Entwicklung und Implementierung des Messinstruments. Dieses wurde als ein neuer Aufgabentyp mit dem Namen „copy-task“ für die Web-Anwendung des Projekts COMMOOP entwickelt. Die bestehende Web-Anwendung wurde für die hier durchgeführte Studie erweitert und überarbeitet. Die Planung, Entwicklung und Implementierung der überarbeiteten Web-Anwendung ist auf der beigelegten CD zu finden. Um für zukünftige Aufgabentypen eine anschlussfähige Ausgangslage zu haben, wurde eine modulare Softwarearchitektur verwendet.

4.5.1 Funktionalitäten des Messinstruments

Bei der Entwicklung eines neuen Aufgabentyps muss zunächst geklärt werden, was mit diesem erhoben werden soll. Dazu muss aus den Forschungsfragen abgeleitet werden, welche relevanten Daten zum einen für die Definition einer Aufgabe des Aufgabentyps und zum anderen als Ergebnis einer Bearbeitung benötigt werden. Des Weiteren besteht ein Aufgabentyp, nach Vorgabe der Schnittstellen der Web-Anwendung, immer aus einer Ansicht für die Testperson und einer Ansicht zur Auswertung der Bearbeitungen. Daher besteht der Aufgabentyp aus drei funktional abgrenzbaren Bereichen. Diese werden im Folgenden beschrieben:

Erstellung einer neuen Aufgabe

Ein berechtigter Benutzer kann im Admin-Bereich neue Aufgaben des Aufgabentyps anlegen. Eine neue Aufgabe kann dann einer neuen Sammlung von Aufgaben hinzugefügt werden.

Bearbeitung einer Aufgabe durch eine Testperson

Die Testpersonen können während der Bearbeitung einer Aufgabe zwischen zwei Phasen wechseln. Wenn sie sich in der Schreibphase befinden, können sie Text in ein Textfeld schreiben oder in die Merkphase wechseln. In der Merkphase können sich die Testpersonen einen Text, der in der Schreibphase nicht sichtbar ist, merken und zurück in die Schreibphase wechseln. Damit die Testpersonen sich während der Merkphase keine Notizen, kein Bildschirmfoto o.ä. machen können, was ihnen einen Vorteil verschaffen würde, müssen sie eine Tastenkombination drücken. Durch Loslassen dieser Kombination wechseln sie automatisch zurück in die Schreibphase. Die Pretests (siehe Abschnitt 5.1.1) haben gezeigt, dass eine hundertprozentige Übereinstimmung des abzuschreibenden Textes und des geschriebenen Textes nicht zu erwarten ist. Daher wurde eine Funktionalität hinzugefügt, die es erlaubt, ein Fertigstellungskriterium in der Aufgabendefinition festzulegen. Entweder kann ein Zeitrahmen, eine Übereinstimmung in Prozent oder eine maximale Anzahl der Merkphasen vorgegeben werden. Damit die Testperson über den Fortschritt im Fertigstellungskriterium informiert ist, sind Fortschrittsbalken vorgesehen. Während der Bearbeitung werden alle Tastenanschläge und Zeitpunkte der Merkphasen gespeichert, um alle relevanten Informationen zur Beantwortung der Forschungsfragen zu erfassen.

Auswertung einer Bearbeitung

Die Bearbeitungen können von einem berechtigten Benutzer im Admin-Bereich der Web-Anwendung ausgewertet werden. Dazu wird eine Wiederholung der Bearbeitung angezeigt, die über eine Zeitleiste kontrolliert werden kann. Dadurch kann der Benutzer leichter zu interessanten Stellen zu springen. Zusätzlich bekommt der Benutzer durch eine farbliche Kodierung einen Überblick über die Bearbeitung. Zur Beantwortung der dritten Forschungsfrage hat der Benutzer außerdem die Möglichkeit Transkripte automatisch generieren zu lassen.

4.5.2 Datenbankentwurf

Bisher wurde dargestellt, welche Funktionalität der Aufgabentyp haben soll und wie diese von den unterschiedlichen Benutzern verwendet werden kann. In diesem Abschnitt wird aufgezeigt, in welcher Form die Daten gespeichert werden sollen. Aufgrund der Anbindung an die Web-Anwendung des Projekts COMMOOP müssen zwei Tabellen, „copy_task“ und „copy_task_results“, in der Datenbank angelegt werden. Die erste Tabelle enthält alle Definitionen der Aufgaben. Die zweite Tabelle beinhaltet alle Daten, die zu den Bearbeitungen gehören. Zunächst muss geklärt werden, welche Attribute in beiden Tabellen angelegt werden müssen und welche Datentypen erforderlich sind.
Hierfür wurde zunächst eine Auflistung der Attribute zur Definition einer Aufgabe erstellt und diesen ein Datentyp zugeordnet (siehe Tabelle 4.3). Dabei ist anzumerken, dass nach der Konvention der Web-Anwendung kleingeschriebene englische Bezeichner verwendet werden sollen. Die Informationen, die für eine Bearbeitung gespeichert werden sollen, leiten sich zum

Attribut	Beispiel	Datentyp
name	Zählschleife	varchar
description	Gibt alle Zahlen von 1 bis 100 aus.	text

text	`for (int i = 1; i < 101; i++) {`
	`System.out.println(i);`
	`}`
		text

type	Java	varchar
max_time	2	int
min_levenshtein	80	int
max_lookups	5	int

Tabelle 4.3: Beispiel eines Datensatzes zu einer Aufgabendefinition

einem aus den Forschungsfragen und zum anderen aus der hinzugefügten Funktionalität des Fertigstellungskriteriums ab. Das Resultat ist in Tabelle 4.4 zu sehen. Die Attribute lookups, keystrokes und categories werden im Abschnitt Implementierung genauer beschrieben. Da bei dem ersten Pretest die Speicherkapazität des Datentyps text (ca. 65 Kilobytes) überschritten war, wurde für diese Attribute der Datentyp mediumtext ausgewählt. Die bereitgestellte Speicherkapazität von ca. 16 Megabytes sollte ausreichend sein, da eine Bearbeitung von ca. 5 Minuten 196 Kilobytes in Anspruch nimmt. Natürlich ist dies auch von der generierten Informationsmenge in den 5 Minuten abhängig, jedoch ist die Differenz von 196 Kilobytes und 16 Megabytes ausreichend groß dimensioniert.

Attribut	Beispiel	Datentyp
lookups	Array von Merkphasen	mediumtext
keystrokes	Array von Tastenanschlägen	mediumtext

text	`for (int i = 1; i < 101; i++) {`
	`System.out.println(i`
	`}`
		text

levenshtein	2	int
typing_rate	2.34	float
error_rate	0.012	float
count_lookups	5	int
count_backspace	3	int
transkript	Array von Kategorien	mediumtext
time	29104	int

Tabelle 4.4: Beispiel eines Datensatzes zu einer Bearbeitung

4.5.3 Implementierung

In diesem Abschnitt wird die Umsetzung der beschriebenen Funktionalität und des Datenentwurfs beschrieben. Da für die Web-Anwendung server-seitig das Framework CodeIgniter[1], welches in der Programmiersprache PHP geschrieben ist, verwendet wurde, wurde es auch für diesen Aufgabentyp benutzt. Für die client-seitige Umsetzung wurde in Anlehnung an bestehende Aufgabentypen die Programmiersprache Javascript im Zusammenspiel mit HTML und CSS verwendet. Um die Interaktion des Benutzers mit der client-seitigen Umsetzung plattformunabhängig zu gestalten, wurde die Javascript Bibliothek JQuery[2] verwendet. Dadurch kann sichergestellt werden, dass der entwickelte Aufgabentyp in den Webbrowsern Firefox, Google Chrome, Safari und Edge funktionstüchtig ist.
Gemäß der Beschreibung der Funktionalität des Aufgabentyps lässt sich die Implementierung in drei Hauptteile gliedern: die Erstellung einer neuen Aufgabe, die Bearbeitung des Aufgabentyps durch eine Testperson und die Auswertung einer Bearbeitung.
Nach den Vorgaben der Web-Anwendung wurde durch die Verwendung des vorgegebenen Scripts „create_task.sh“ eine Dateistruktur, wie sie in Abbildung 4.7 zu sehen ist, erzeugt.

Erstellung einer neuen Aufgabe

Für die Erstellung einer neuen Aufgabe wurden die Dateien „new_page.php“ und „CopyTextNew.js“ bearbeitet. In „new_page.php“ wurde das HTML-Interface der Seite zum Erstellen einer neuen Aufgabe implementiert (siehe Abbildung 4.8). Die JavaScript-Datei „CopyTextNew.js“ ist auf dieser Seite als Skript eingebunden. Die Validierung und die Übermittlung der Eingaben an den Server werden von Methoden in dieser Datei übernommen. Die Übermittlung an den Server wird über einen Ajax-Request vom Typ POST an die URL „admin/tasks/copy-text/create/“ realisiert. Dabei werden die eingegeben Daten in einem JSON-Objekt zusammengefasst und dann übertragen. Um die Daten zu speichern, wurde die Methode „create“ in der Datei „admin/tasks/CopyText.php“ implementiert, in der die Daten zunächst umgeformt und dann in der Datenbank gespeichert werden.

[1] `https://codeigniter.com/`, abgerufen am 29.04.2017
[2] `https://jquery.com/`, abgerufen am 29.04.2017

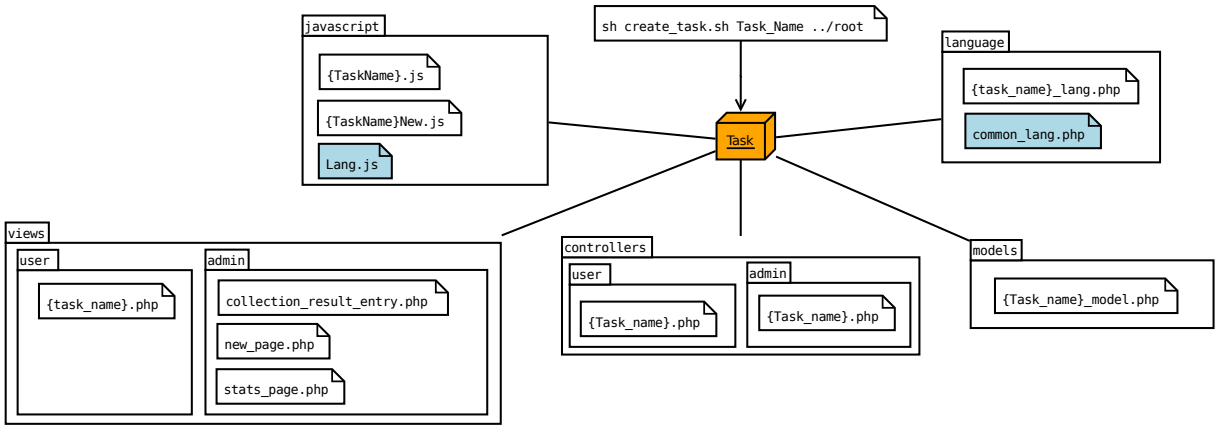


Abbildung 4.7: Allgemeine Struktur eines Aufgabentyps der Web-Anwendung aus dem Projekt COMMOOP

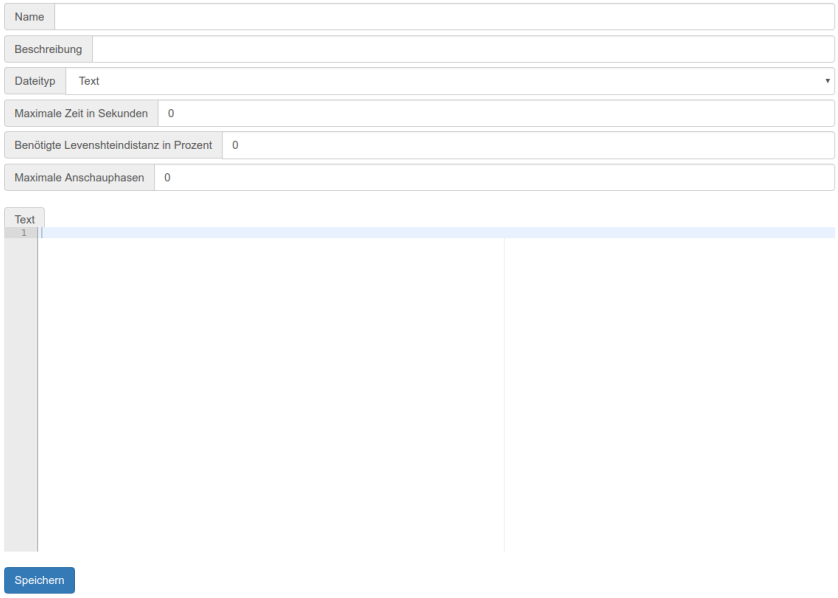


Abbildung 4.8: Aufbau der Seite zum Erstellen einer neuen Aufgabe des Typs copy-text

Bearbeitung einer Aufgabe durch eine Testperson

Für die Bearbeitung einer Aufgabe wurden die Dateien „views/copy_text.php“ und „CopyText.js“ modifiziert. Die Seite ist aus zwei Ansichten zusammengesetzt – der Schreibphasen-Ansicht und der Merkphasen-Ansicht (siehe Abbildung 4.9). Die Testperson sieht zuerst die Schreibansicht. Diese besteht aus einem Textfeld, zwei Schaltflächen und bis zu drei Fortschrittsanzeigen. Das Textfeld dient dazu, die Eingaben der Testpersonen entgegenzunehmen und diese aufzuzeichnen. Dazu wurde ein EventListener implementiert, der auf keydown-Events im Textfeld reagiert. Dieser überprüft zunächst das Event auf die Gültigkeit der Eingabe. Wenn der Text markiert wurde oder die STRG-Taste zusammen mit der Backspace- oder Entf-Taste gedrückt wurde, wird das Event verworfen, da in diesen Fälle mehr als ein Zeichen gleichzeitig gelöscht werden würde. Handelt es sich um ein gültiges Event, wird ein JSON-Objekt wie in Abbildung 4.10 in einem Array gespeichert. Zusätzlich zu den aus dem Event gewonnen Informationen wird es um die aktuelle Levenshtein-Distanz und mittels „new Date().getTime()“ um die Zeit ergänzt. Die Levenshtein-Distanz (Levenshtein, 1966) wird als Maß der Übereinstimmung der beiden Texte verwendet. Durch die Verwendung eines Back-Tracking-Verfahrens wird

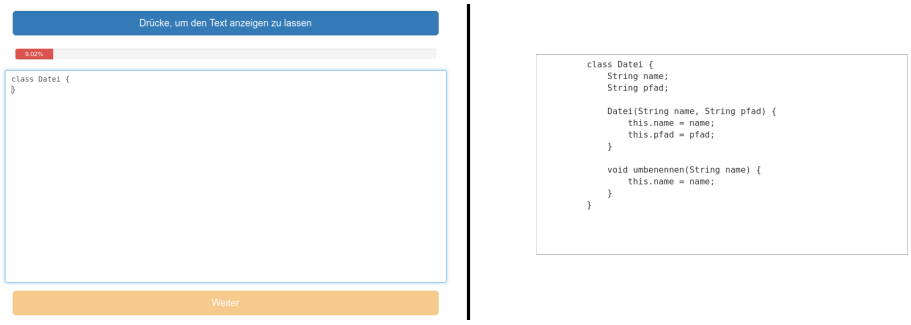


Abbildung 4.9: Aufbau der Schreibphasen-Ansicht (links) und der Merkphasen-Ansicht (rechts)

```
{
    'key': 42,
    'alt': true,
    'ctrl': false,
    'time': 1493542625013,
    'levenshtein': 112
}
```

Abbildung 4.10: JSON-Objekt eines Tastenanschlags

ermittelt, wie viele Einfüge-, Lösch- und Ersetzungsschritte mindestens notwendig sind, um einen Text in einen anderen Text zu überführen. Hierzu wurde die JavaScript Implementierung von gf3[3] verwendet. Um von der Schreibphasen-Ansicht in die Merkphasen-Ansicht wechseln zu können, wurde eine Schaltfläche oberhalb des Textfelds implementiert. Wenn diese aktiviert und gleichzeitig die STRG- und ALT-Taste gedrückt wird, werden die Elemente der Schreibphasen-Ansicht ausgeblendet und die Merkphasen-Ansicht wird angezeigt. Wird eine Taste dieser Kombination losgelassen, wechselt das Tool zurück zur Schreibphasen-Ansicht und fokussiert das Textfeld, um die Zeit zwischen Merkphase und Schreibphase möglichst gering zu halten. Die Tastenkombination wurde gewählt, damit die Testpersonen beide Hände in Anspruch nehmen müssen und somit keine Notizen oder ähnliche Aufzeichnungen anfertigen können. Weitere Maßnahmen, um eine Täuschung durch die Testpersonen zu verhindern, sind zum einen die Entfernung der Merkphasen-Ansicht aus der DOM, wenn die Schreibphasen-Ansicht angezeigt wird und die Enkodierung des zu merkenden Textes in Base64. Damit wird verhindert, dass der zu merkende Text aus der Quelltext-Ansicht kopiert werden kann. Um die Aufzeichnung der Bearbeitung zu vervollständigen, wird ein JSON-Objekt für jede Merkphase mit dem Start- und dem Endzeitpunkt erstellt und einem Array hinzugefügt.

Die Testperson kann durch das Drücken der Schaltfläche unterhalb des Textfeldes die Bearbeitung einreichen. Die Aktivierung kann an ein bis drei Fertigstellungskriterien zu einer Aufgabe geknüpft sein. Diese haben jeweils einen unterschiedlichen Einfluss auf den Ablauf der Bearbeitung:

Übereinstimmung Der zu merkende Text und der geschriebene Text müssen zu einer bestimmten Prozentzahl übereinstimmen, bevor die Bearbeitung eingereicht werden kann. Dabei werden doppelte Leerzeichen als einzelne gewertet und Umbrüche außer Acht gelassen. Dazu werden der geschriebene Text und der zu merkende Text zuerst mittels der Methode in Abbildung 4.11 umgewandelt, bevor diese verglichen werden.

[3] `https://github.com/gf3/Levenshtein`, abgerufen am 30.04.2017

Zeit Für die Bearbeitung steht ein begrenzter Zeitrahmen zu Verfügung. Wenn dieser überschritten wird, wird die Bearbeitung automatisch eingereicht — unabhängig von dem Fortschritt der anderen Kriterien.

Merkphasen Während der Bearbeitung darf nur begrenzt oft in die Merkphasen-Ansicht gewechselt werden. Ist dieses Kontingent verbraucht, kann nicht mehr in die Merkphasen-Ansicht gewechselt werden. Die Bearbeitung wird nicht automatisch eingereicht.

```
stripWhitespace: function(string) {
  // enferne Umbrüche
  string = string.replace(/\n|\r/g, ' ');
  // ersetzte mehrere Leerzeichen mit einem
  string = string.replace(/\s{2,}/g, ' ');
  // entferne führende und nachfolgende Leerzeichen
  string = string.replace(/^[\s\xA0]+|[\s\xA0]+$/g, '');
  // entferne Leerzeichen ringsum von Sonderzeichen
  string = string
    .replace(/\s*([\(\)\{\}\<\>\;\:\,\[\]\+\=\-\*\/\%\!])\s*/g, '$1');
  return string;
},
```

Abbildung 4.11: Methode zum Entfernen von nicht benötigten Leerzeichen und Umbrüchen

Damit die Testperson über den aktuellen Status eines oder mehrerer Fertigstellungskriterien informiert ist, wird über dem Textfeld für jedes Fertigstellungskriterium eine Fortschrittsanzeige sichtbar (siehe Abbildung 4.12). Wenn die Bearbeitung eingereicht wird, wird ein AJAX-Request

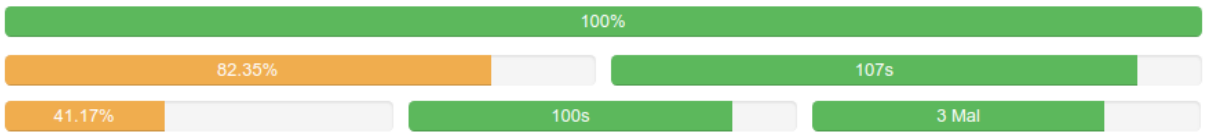


Abbildung 4.12: Varianten der Fortschrittsanzeigen

vom Typ POST an die URL „tasks/copy-text/check“ gesendet. Dieser enthält die Daten der Bearbeitung als JSON-Objekt (siehe Abbildung 4.13). Auf dem Server werden die Daten

```
{
    'text': Eingereichter Text,
    'lookups': Array von JSON-Objekten der Merkphasen,
    'keystrokes': Array von JSON-Objekten der Tastenanschläge,
    'time': Bearbeitungsdauer in Millisekunden,
    'levenshtein': Levenshtein-Distanz zum Einreichezeitpunkt
}
```

Abbildung 4.13: JSON-Objekt einer Bearbeitung

weiterverarbeitet. Dabei wird zunächst die berechnete Levenshtein-Distanz des Clients serverseitig überprüft. Danach werden aus den übermittelten Tastenanschlägen und Merkphasen

verschiedene Kennwerte wie die Anzahl der Merkphasen, die Gesamtlänge der Merkphasen, die Tippgeschwindigkeit und die Fehlerrate berechnet. Anschließend werden die Daten in der Datenbank gespeichert.

Auswertung einer Bearbeitung

Für die Implementierung der Auswertung einer Bearbeitung wurden die Dateien „controllers/admin/tasks/CopyText.php“, „js/CopyTextReplay.js“ und „views/admin/tasks/replay_page.php“ bearbeitet. Die erstellte Wiederholungsansicht einer Bearbeitung ist aus einem Textfeld, einer Zeitleiste und einer Tabelle zum Hinzufügen von Kategorien aufgebaut (siehe Abbildung 4.14). In dem Textfeld wird der Text zu einem gegebenem Zeitpunkt rekonstruiert. Dazu werden die

Abbildung 4.14: Ansicht der Wiederholung einer Bearbeitung

gespeicherten Tastenanschläge verwendet. Unterhalb des Textfelds befindet sich die Zeitleiste. Auf ihr werden die Ereignisse während der Bearbeitung repräsentiert (Tastenanschläge grün, Korrekturen rot, Navigationstasten orange und Merkphasen blau). Durch Bewegen des Zeitmarkers kann zu einem beliebigen Zeitpunkt gesprungen werden. Außerdem kann durch Verwendung der Kontrollschaltflächen auf der linke Seite unterhalb der Zeitleiste der Ablauf der Wiederholung gestartet, pausiert und zurückgesetzt werden. Auf der rechten Seite unterhalb der Zeitleiste können zudem die Abspielgeschwindigkeit und die Skalierung angepasst werden.

Am unteren Ende der Seite wird ein Transkript der Bearbeitung angezeigt, wenn auf die Generate Schaltfläche gedrückt wurde. Beim Generieren eines Transkripts werden die Abschnitte der Bearbeitung automatisch in drei verschiedene Phasen eingeteilt: Eine Merkphase (M) ist definiert als eine Phase, in der der zu übertragende Text angeschaut wird. Eine Schreibphase (S) ist definiert als eine zusammenhängende Phase, in der Eingaben getätigt werden. Wenn über einen Zeitraum von zwei oder mehr Sekunden keine Eingabe getätigt wird, wird dieser Abschnitt als Pausenphase (P) definiert. Für die Schreibphasen wird zusätzlich die überwundene Levenshtein-Distanz (LS) und die Anzahl der gelöschten Zeichen (G) ermittelt. Außerdem wird der Inhalt des Textfeldes nach der Schreibphase angezeigt (siehe Markierung 2). Dabei werden Zeichen, die neu hinzugekommen sind blau, Zeichen, die gelöscht wurden rot und nicht geänderte Zeichen grau eingefärbt. Diese Veränderung zeigt den Unterschied vom Beginn der Schreibphase zum Ende der Schreibphase auf. Bevor eine neue Merkphase beginnt, wird eine Zusammenfassung des letzten Abschnittes angezeigt (siehe Markierung 1). Hier wird die verwendete Zeit für die Schreibphasen und die Pausenphasen, sowie die überwundene Levenshtein-Distanz und die Anzahl der gelöschten Zeichen aufsummiert.

5 Durchführung und Auswertung der empirischen Untersuchung

5.1 Durchführung

5.1.1 Pretests

Um die Funktionsfähigkeit der Messinstrumente, die Eignung der generierten Items und den reibungslosen Untersuchungsablauf zu testen, wurden zwei instrumentelle Pretests durchgeführt.

Erster Pretest

An einem ersten Pretest nahmen ein Experte und eine Novizin teil. Getestet wurde das informatische Modul, dass zu diesem Zeitpunkt aus drei Items zur Ermittlung der Tippgeschwindigkeit und neun Items zum Memorieren und Reproduzieren bestand. Bei allen Items wurde eine hundertprozentige Übereinstimmung verlangt, um zum nächsten Item zu gelangen. Diese hohe Anforderung stellte sich bei der Durchführung des Pretests jedoch als Hindernis im Hinblick auf den eigentlichen Untersuchungsgegenstand heraus. Am Ende eines Items wurde jeweils viel Zeit darauf verwendet, kleine Fehler wie „Vertipper“ und ausgelassene Zeichen zu finden, sodass sich nur durch Unterstützung des Untersuchungsleiters bei der Fehlersuche die Durchführungszeit in einem akzeptablen Rahmen hielt. Dies führte zu der Entscheidung, die Levenshtein-Distanz in das Messinstrument als Maß für die Übereinstimmung aufzunehmen. Außerdem zeigte sich, dass eine ausführliche Formulierung der Instruktionen notwendig ist. Die Instruktionen wurden deshalb auf Grundlage des Feedbacks der beiden Testpersonen angepasst.

Zweiter Pretest

An dem zweiten Pretest nahmen drei ExpertInnen und fünf NovizInnen teil. In diesem Pretest wurden der d2 Aufmerksamkeits-Belastungs-Test, der Corsi-Block-Tapping-Test und das informatische Modul getestet. Im informatischen Modul wurden die Erkenntnisse des ersten Pretests eingearbeitet, sodass die Testpersonen nun eine Fortschrittsanzeige über dem Textfeld sehen konnten, welche die Übereinstimmung ihres Textes zum Originaltext in Prozent angezeigt hat. Das aktuelle Item konnte erst eingereicht werden, wenn eine Übereinstimmung von 98% erreicht wurde. Eine Schlussfolgerung aus dem zweiten Pretest betrifft die visuelle Abbildung der Testinstruktionen. Da in der Erhebung verschiedene Instrumente mit unterschiedlichen Abläufen und teilweise mehreren Phasen auftreten, hat sich die mündliche Kommunikation der Instruktionen als unvorteilhaft erwiesen. Zur besseren Darstellung der Instruktionen wurde deshalb für die Hauptuntersuchung eine Präsentation ausgearbeitet.
Beim Übertragen der Quelltexte zeigte sich, dass den Testpersonen im Pretest häufig Flüchtigkeitsfehler unterliefen. Die Beseitigung dieser verlängerte die Zeitspanne erheblich, die benötigt wurde, um die nächste Aufgabe freizuschalten. Daher wurde eine weitere Absenkung der geforderten Übereinstimmung auf 95% vorgenommen. Außerdem hat sich gezeigt, dass die

gewählte Kombination der Tasten „a“ und „b“ mit dem Drücken der Schaltfläche zum Anzeigen des verstecken Textes von einigen Testpersonen als zu umständlich kritisiert wurden. Daher eine neue Kombination der Tasten „STRG“ und „ALT“ mit dem Drücken der Schaltfläche oberhalb des Textfeldes beschlossen.

5.1.2 Ablauf der empirischen Untersuchung

Beschreibung der idealen Testbedingungen

Alle Testpersonen sollten idealerweise unter den gleichen Bedingungen getestet werden. Das heißt, dass sie die gleiche Tastatur, die gleiche Maus und den gleichen Bildschirm benutzen sollten, um mögliche Verzerrungen durch unterschiedliche Geräte zu vermeiden. Zum Einsatz kommen sollen Notebooks mit einem 15.6 Zoll großen Bildschirm, die eine Auflösung von 1366x768 Pixeln ermöglichen, sowie Mäuse als externe Eingabegeräte. Die Größe der Bildschirme und deren Auflösung ist besonders für den computerbasierten Corsi-Block-Tapping-Test wichtig, da dieser für diese Bedingungen implementiert wurde. Die computerbasierten Module der Untersuchung sollten im Browser Chrome Version 55 durchgeführt werden. Weiterhin sollten die Notebooks per Netzwerkkabel an das Internet angebunden werden, um eine möglichst schnelle und stabile Verbindung zu gewährleisten. Außerdem sollte die Untersuchung in einer ruhigen Atmosphäre durchgeführt werden, um die Konzentration der Testpersonen nicht zu beeinflussen. Eine Gruppengröße von maximal acht Personen ist erstrebenswert.

Abweichung von den idealen Testbedingungen

Bei der praktischen Durchführung der Tests wurden Notebooks des Modells Lenovo B50 und Mäuse des Modells Logitech M185 verwendet. Durch Einschränkungen in der verfügbaren Hardware konnte eine Gruppe von zehn Personen den Test nicht mit dem zuvor genannten Notebooks und Mäusen durchführen, wodurch es Unterschiede in der Bildschirmgröße und Auflösung gab. Diese Abweichung könnte vor allem die oben angesprochene Auswirkung auf den Corsi-Block-Tapping-Test haben, da dadurch die Blöcke in einer anderen Größe erscheinen als auf den Notebooks. Außerdem könnte diese Abweichung einen Einfluss auf die Tippgeschwindigkeit haben, da andere Tastaturen verwendet wurden.

5.1.3 Testpersonen

Anforderungen an die Testpersonen

Für die Untersuchung werden gleich viele NovizInnen wie ExpertInnen benötigt, um die statistischen Verfahren zur Klärung der Gruppenunterschiede durchführen zu können. Im Idealfall können zwei klare Gruppen gebildet werden. Das heißt, dass bezogen auf das Level der Programmiererfahrung um den Median wenig Testpersonen liegen sollen. Damit die gewünschte Verteilung erzielt werden kann, sollen zum einen Studierende aus dem ersten Semester eines Bachelorstudiengangs der Informatik und Studierende aus einem Masterstudiengang der Informatik für die Untersuchung gewonnen werden. Um die Gruppe der NovizInnen deutlicher von den der ExpertInnen abzugrenzen, wurde versucht auch Studierende zu akquirieren, die gar keine Programmiererfahrung haben. Außerdem wurden ExpertInnen gesucht, die täglich mit Quelltexten arbeiten, um die Gruppe der ExpertInnen weiter auszudifferenzieren. Diese Ausdifferenzierung dient vor allem der Beantwortung der dritten Forschungsfrage.
Es wurde angestrebt, insgesamt 40 Studierende zu gewinnen. Fünf Studierende ohne Programmiererfahrung, 15 Studierende mit geringer bis mittlerer Programmiererfahrung, 15 Studierende

mit mittlerer bis hoher Programmiererfahrung und fünf Studierende mit hoher bis höchster Programmiererfahrung.

Beschreibung der Stichprobe

Sozio-demografische Angaben

Insgesamt haben 42 Personen an der Untersuchung teilgenommen, davon haben 3 Personen die Untersuchung nicht vollständig abgeschlossen, sodass sie von der quantitativen Untersuchung ausgenommen werden müssen. Das Durchschnittsalter der 39 Testpersonen betrug zum Zeitpunkt der Testdurchführung ca. 24 Jahre. In Bezug auf die Verteilung der Geschlechter ist eine stärkere Beteiligung des männlichen Geschlechts mit ca. 69% (n = 27) gegenüber dem weiblichen mit ca. 31% (n = 12) festzuhalten. Die Studierenden kamen aus acht verschiedenen Studiengängen (siehe Abbildung 5.1), wobei der Anteil der Studierenden aus informatischen Studiengängen mit ca. 74% (n = 29) überwog.

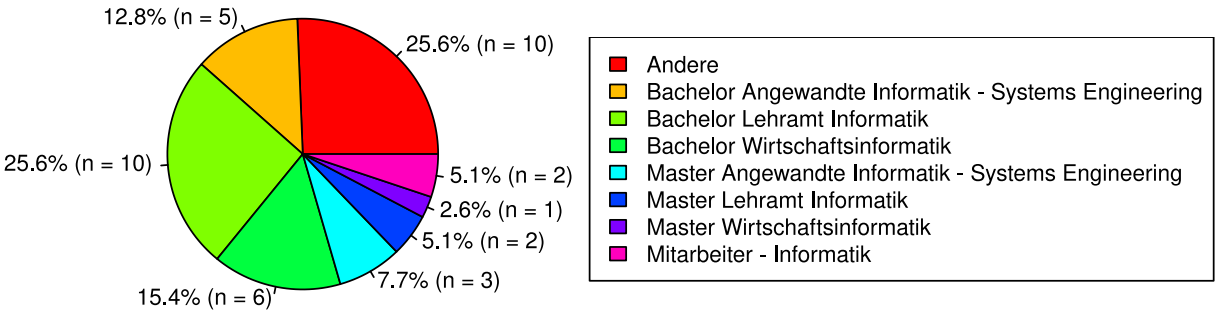


Abbildung 5.1: Verteilung der Studiengänge der Testpersonen

Corsi-Block-Tapping-Test

Im Corsi-Block-Tapping-Test wurde in der Vorwärtsvariante im Durchschnitt eine Spanne von 6.46 erreicht. In der Rückwärtsvariante war die Spanne ca. 0.4 Einheiten geringer (siehe Tabelle 5.1). Verglichen mit den Ergebnissen von jungen Erwachsenden (Vorwärts: M = 6.11, SD = 0.80, Rückwärts: M = 5.29, SD = 1.20) aus der Studie von Brunetti, Gatto und Delogu (2014) erreichten die Testpersonen bessere Ergebnisse. Außerdem ist auffällig, dass der Abstand zwischen der Vorwärtsspanne und der Rückwärtsspanne deutlich geringer ist. Dabei ist zu beachten, dass in dieser Studie Mäuse als Eingabegeräte verwendet wurden, wohingegen Brunetti, Gatto und Delogu (ebd.) Touchscreens einsetzten. Inwiefern die unterschiedlichen Eingabegeräte Auswirkungen auf das Testergebnis haben, wurde bisher nicht erforscht.

	Min	Max	Median	M	SD
Vorwärts	4	9	6	6.46	1.30
Rückwärts	4	9	6	6.08	1.01

Tabelle 5.1: Ergebnisse des Corsi-Block-Tapping-Tests

I-S-T 2000-R

Im verbalen Merkmodul wurden von möglichen 10 Punkten im Durchschnitt 9.11 Punkte erreicht. Im figuralen Merkmodul lag der Durchschnitt relativ gesehen mit 9.78 von 13 maximal

möglichen richtigen Antworten etwas niedriger. Im Vergleich mit den durchschnittlichen Werten von Gymnasiasten zwischen 15 und 25 in diesem Test (IST-Verb: M = 8.68, SD = 1.87, IST-Fig: M = 9.12, SD = 2.89) ist festzustellen, dass die Testpersonen in beiden Merkmodulen bessere Ergebnisse erreichten (siehe Amthauer u. a., 2006, S. 135).

	Min	Max	Median	M	SD
IST-Verb	4	10	10	9.11	1.33
IST-Fig	1	13	10	9.78	2.58

Tabelle 5.2: Ergebnisse der I-S-T 2000R Merkmodule

d2 Aufmerksamkeits-Belastungs-Test

Im d2 Aufmerksamkeits-Belastungs-Test wurden für die Konzentrationsleistung (KL) im Durchschnitt 202.5 (SD = 42.75) Punkte erzielt. Im Gesamttestwert (GZ-F) wurden im Durchschnitt 499.5 (SD = 76.83) Punkte erzielt. Bezogen auf die Normtabelle für Personen im Alter von 19 bis 39 Jahren (Brickenkamp, 1967, siehe) liegen somit die meisten Testpersonen auf dem oder über dem Prozentrang 90 (siehe Abbildung 5.2).

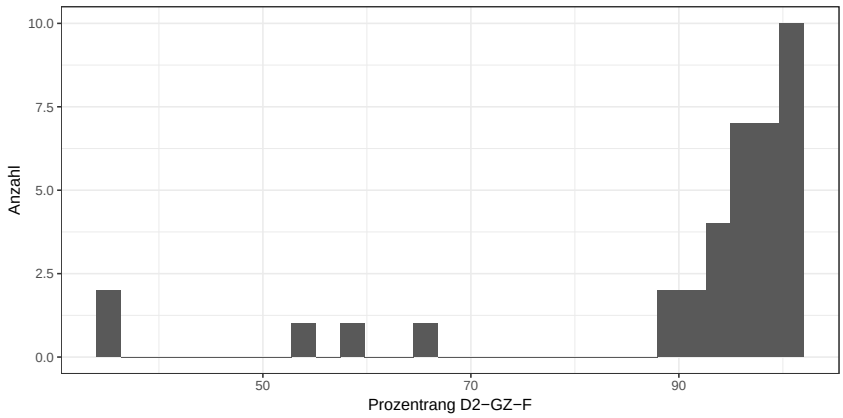


Abbildung 5.2: Histogramm der Prozentränge des GZ-F

Insgesamt bleibt festzuhalten, dass die Testpersonen in den kognitionspsychologischen Test überdurchschnittliche Ergebnisse erzielt haben und somit als eher leistungsstark in diesen Bereichen einzuordnen sind.

Im Folgenden wird die deskriptive Auswertung der fünf Items zur Programmiererfahrung vorgestellt. Zur Übersicht der Beantwortung der Items dient das Boxplotdiagramm in Abbildung 5.3. Alle Items wurden auf einer 10-stufigen Likert-Skala (1 = unerfahren, 10 = sehr erfahren) von den Testpersonen bewertet.

Selbsteingeschätzte Programmiererfahrung (PE-Selbst)

Die Fragen „Wie schätzen Sie Ihre Programmiererfahrung ein?“ wurde im Durchschnitt mit 5.08 (SD = 2.65) beantwortet. Das Maximum lag bei 9, das Minimum bei 1.

Programmiererfahrung in der Programmiersprache Java (PE-Java)

Die Frage „Wie erfahren sind Sie mit der Programmiersprache Java?“ wurde im Durchschnitt mit 5.27 (SD = 2.73) beantwortet. Das Maximum lag bei 9, das Minimum bei 1.

Programmiererfahrung in der objektorientierten Programmierung (PE-OOP)

Die Frage „Wie erfahren sind Sie mit dem Paradigma Objektorientierte Programmierung?“ wurde im Durchschnitt mit 5.72 (SD = 3.04) beantwortet. Das Maximum lag bei 10, das Minimum bei 1.

Programmiererfahrung im Vergleich zu einem Berufstätigen (PE-Beruf)

Die Frage „Wie schätzen Sie Ihre Programmiererfahrung verglichen mit ExpertInnen mit 20-jähriger Berufserfahrung ein?“ wurde im Durchschnitt mit 3.31 (SD = 2.28) beantwortet. Das Maximum lag bei 8, das Minimum bei 1. Dieses Item weist den geringsten Durchschnittswert auf. Auch ist zu sehen, dass 75% der Testpersonen mit einem Wert von 1 - 5 geantwortet haben. Das heißt, dass sich 75% der Testpersonen verglichen mit einem/einer ExpertIn mit 20-jähriger Berufserfahrung als eher unerfahren ansehen.

Programmiererfahrung im Vergleich zu den Kommilitonen (PE-Kom)

Anders verhält es sich bei der Frage „Wie schätzen Sie Ihre Programmiererfahrung verglichen mit Ihren KommilitonInnen ein?“. Bei diesem Item lag der Durchschnitt bei 5.90 (SD = 2.82). Somit wurde dieses Item im Durchschnitt am höchsten bewertet.

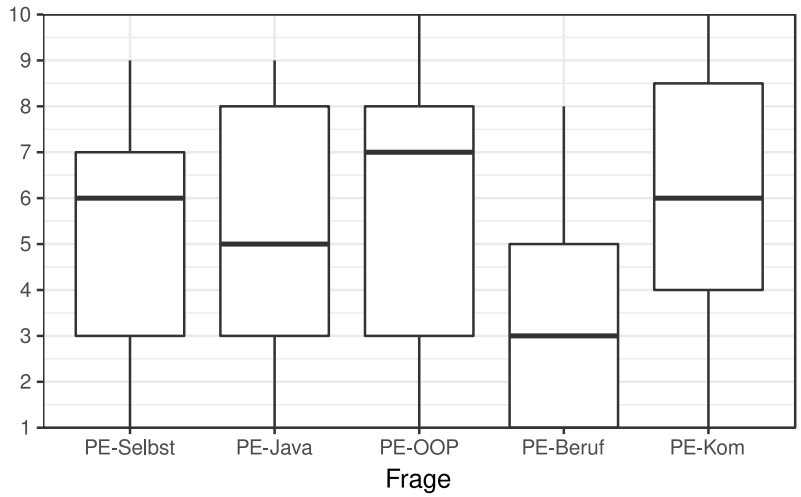


Abbildung 5.3: Boxplotdiagramme mit den Ergebnissen der Fragen zur Selbsteinschätzung der Programmiererfahrung

Score der Programmiererfahrung

Die für die Überprüfung der empirischen Fragestellungen notwendige Zuordnung von Testpersonen zu den Gruppen ExpertInnen und NovizInnen wird nachfolgend vorgenommen. Dazu wird wie in der Auswertungsmethodik beschrieben (siehe Abschnitt 4.4) die interne Konsistenz der Fragen zur Selbsteinschätzung der Programmiererfahrung überprüft. Cronbach’s alpha für

die fünf Fragen betrug 0.96, somit ist die interne Konsistenz als gut zu bewerten (vgl. Bortz und Döring, 2006, S. 725). Daher wurde aus den Scores der einzelnen Fragen ein gemeinsamer Score durch Addition gebildet. Dieser Score (max = 50, min = 5) stellt die selbsteingeschätzte Programmiererfahrung der Testpersonen dar. Der Median der selbsteingeschätzten Programmiererfahrung lag bei 29. Die Verteilung der selbsteingeschätzten Programmiererfahrung ist in Abbildung 5.4 zu sehen.

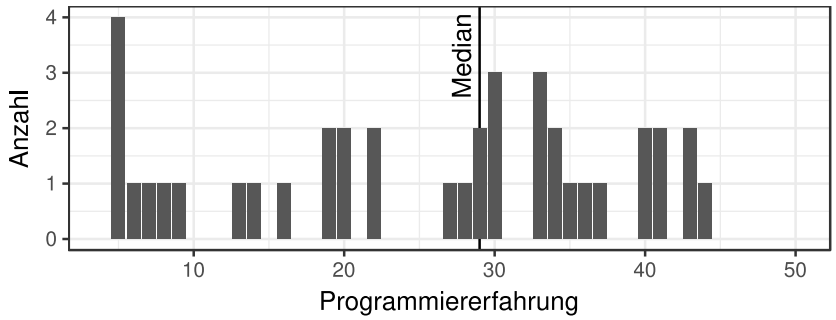


Abbildung 5.4: Verteilung der selbsteingeschätzten Programmiererfahrung

5.2 Darstellung der Ergebnisse

In diesem Unterkapitel werden die Ergebnisse der Untersuchung getrennt nach Forschungsfragen präsentiert.

5.2.1 Ergebnisse zur ersten Forschungsfrage

Die Unterschiede in den erhobenen Größen wurden mit den Mann-Whitney-U-Test auf Signifikanz geprüft. Auf die Anwendung des t-Tests musste aufgrund nicht erfüllter Voraussetzungen (signifikanter Kolmogorov-Smirnov-Test) verzichtet werden.

Tippgeschwindigkeit

Die Ergebnisse des Mann-Whitney-U-Tests zeigen, dass sich die Tippgeschwindigkeitsdifferenz nur in den ersten beiden Items (Klasse1 und Algo1) zwischen den NovizInnen und ExpertInnen signifikant unterscheidet (siehe Tabelle 5.3).

	M_N	M_E	W	p-Wert
Klasse1	1.25	0.52	245.00	0.02
Algo1	0.83	0.01	246.00	0.02
Klasse2	0.38	0.04	221.00	0.12
Algo2	0.96	0.86	184.00	0.68
Klasse3	0.81	0.70	176.00	0.87
Algo3	1.12	0.97	188.00	0.60

Tabelle 5.3: Gruppenunterschiede zwischen ExpertInnen und NovizInnen im Bezug auf die Tippgeschwindigkeitsdifferenz (gemessen in Tastenanschläge pro Sekunde)

Das heißt mit Betrachtung der mittleren Tippgeschwindigkeitsdifferenz, dass NovizInnen bei

dem Item Klasse1 mit 1.25 Tastenanschläge pro Sekunde langsamer waren, als beim dazugehörigen natürlichsprachlichen Text. Auch die Tippgeschwindigkeit von ExpertInnen hat sich bei diesem Item verringert, sie waren 0.52 Tastenanschläge pro Sekunde langsamer. Bei dem Item Algo1 hat sich gezeigt, dass dieses keinen Einfluss auf die Tippgeschwindigkeit der ExpertInnen genommen hat. Dagegen waren NovizInnen im Durchschnitt 0.83 Tastenanschläge pro Sekunde langsamer als bei dem entsprechenden natürlichsprachlichen Item. Das Item Klasse2 zeigt diesbezüglich nur die Tendenz, dass auch hier der Quelltext von ExpertInnen mit einer ähnlichen Tippgeschwindigkeit wie der natürlichsprachliche Text abgeschrieben wurde. Die Tippgeschwindigkeit der NovizInnen war auch bei diesem Item geringer.

Fehlerrate

Die Fehlerratendifferenz unterscheidet sich bei keinem Item signifikant zwischen den beiden Gruppen (siehe Anhang A.2).

Gesamtlänge der Merkphasen

Die Auswertung der zeitlichen Differenz der Merkphasen hat ergeben, dass sich die Items Klasse1, Algo1 und Klasse3 signifikant zwischen den ExpertInnen und NovizInnen unterscheiden (siehe Tablle 5.5). Auffällig ist, dass die durchschnittliche zeitliche Differenz der Merkphasen für

	M_N	M_E	W	p-Wert
Klasse1	-6.296	2.009	73	0.00
Algo1	-7.893	3.051	42	0.00
Klasse2	0.295	8.644	116	0.10
Algo2	-12.825	-7.542	140	0.37
Klasse3	-9.603	-4.643	96	0.02
Algo3	-18.450	-8.577	120	0.13

Tabelle 5.4: Gruppenunterschiede zwischen ExpertInnen und NovizInnen im Bezug auf die zeitliche Differenz der Merkphasen (gemessen in Sekunden)

die Items Klasse1 und Algo1 der ExpertInnen positiv ist. Das heißt, dass sie sich bei diesen Items die Quelltexte über eine kürzere Dauer angeschaut haben als die natürlichsprachlichen Texte. Eine interessante Beobachtung ist, dass die NovizInnen sich vom Item Klasse1 zum Item Klasse3 hinsichtlich der zeitlichen Differenz der Merkphasen weniger verschlechtert haben als die ExpertInnen.

Anzahl der Merkphasen

Für die Differenz der Anzahl der Merkphasen lässt sich feststellen, dass sich ExpertInnen in den Items Klasse1, Algo1 und Klasse3 signifikant von NovizInnen unterscheiden. Die Unterschiede zwischen den beiden Gruppen zeigen sich besonders beim Item Algo1, bei dem der Unterschied mit ca. 3.6 Merkphasen deutlich größer ist als bei dem Items Klasse1 (ca. 2 Merkphasen) bzw. Klasse3 (ca. 1.8 Merkphasen). Im Vergleich von Klasse1 mit Klasse3 entwickeln sich die beiden Gruppen gleich. Beide brauchen im Durchschnitt ca. 2 Merkphasen mehr im Item Klasse3. Die einzig positive Differenz zugunsten der Programmieritems zeigt sich bei den ExpertInnen im Item Algo1.

	M_N	M_E	W	p-Wert
Klasse1	-2.47	-0.40	91.00	0.015
Algo1	-3.24	0.40	61.00	0.001
Klasse2	-0.71	0.75	154.50	0.643
Algo2	-4.29	-4.10	159.00	0.748
Klasse3	-4.24	-2.40	104.00	0.045
Algo3	-8.12	-4.80	133.00	0.265

Tabelle 5.5: Gruppenunterschiede zwischen ExpertInnen und NovizInnen im Bezug auf die Differenz der Anzahl der Merkphasen

5.2.2 Ergebnisse zur zweiten Forschungsfrage

Zur Beantwortung der zweiten Forschungsfrage wurden die erhobenen Kennwerte zu Konstrukten zusammengefasst. Zum einen wurden die Kennwerte des d2 Aufmerksamkeits-Gedächtnis-Tests zum Konstrukt Aufmerksamkeit (A), die Kennwerte des I-S-T 2000-R und des Corsi-Block-Tapping-Tests zum Konstrukt Arbeitsgedächtnis (AG), die Items des Fragebogens zur selbsteingeschätzten Programmiererfahrung zum Konstrukt Programmiererfahrung (PE) und die gemessene zeitliche Differenz der Gesamtlänge der Merkphasen zum Konstrukt Fähigkeit (F) zusammengefasst. Aus dieser Strukturierung und der Erkenntnis aus den theoretischen Grundlagen und des Forschungsstands wurde ein Strukturgleichungsmodell erstellt (siehe Abbildung 4.6).

Qualität des Messmodels

Wie in Abschnitt 4.4 vorgestellt, wurde die Reliabilität und die Validität des Messmodels analysiert. Im Folgenden werden die zur Bestimmung der Qualität des Messmodells erforderlichen Ergebnisse vorgestellt.

Interne Konsistenz

Ein Messmodell erfüllt die interne Konsistenz, wenn Cronbachs alpha und Dillon-Golsteins rho für alle Konstrukte größer sind als 0.7. Mit Ausnahme des Arbeitsgedächtnisses erfüllen alle Konstrukte diese Bedingung. Daher wurde nach Untersuchung der Ladungen der Faktor Corsi, der mit 0.47 auf das Konstrukt Arbeitsgedächtnis lädt, aus dem Messmodell entfernt. Nach dieser Maßnahme können alle Konstrukte als eindimensional angenommen werden (vgl. Tabelle 5.6).

Konstrukt	Faktoren	C.alpha	DG.rho
Aufmerksamkeit	2	0.98	0.99
Arbeitsgedächtnis	2	0.71	0.87
Programmierung	5	0.96	0.97
Fähigkeit	6	0.84	0.88

Tabelle 5.6: Interne Konsistenz der Konstrukte

Konvergenzvalidität

In dieser Studie wurde die Konvergenzvalidität mittels der durschnittlich erfasste Varianz (DEV) und der Ladungen der Faktoren auf die Konstrukte analysiert. In Tabelle 5.7 ist zu sehen, dass die durchschnittlich erfasste Varianz aller Konstrukte über den Richtwert von 0.5 liegt. Demnach genügt das Messmodells bezogen auf diesen Indikator der Konvergenzvalidität. Bezogen auf

Konstrukt	DEV
Aufmerksamkeit	0.98
Arbeitsgedächtnis	0.77
Programmiererfahrung	0.88
Fähigkeit	0.53

Tabelle 5.7: DEV der Konstrukte

die Ladungen der Faktoren auf die Konstrukte (siehe Tabelle 5.8) ist zu erkennen, dass der Richtwert von 0.7 von den Faktoren der Konstrukte Aufmerksamkeit, Arbeitsgedächtnis und Programmiererfahrung erreicht wird. Beim Konstrukt Fähigkeit liegen die Faktoren Algo2 mit 0.66 und Klasse3 mit 0.58 unterhalb dieser Grenze, jedoch deutlich oberhalb der Entfernungsgrenze von 0.4. Daher wird auch aufgrund der inhaltlichen Nähe der Faktoren zu den anderen davon abgesehen, diese Faktoren zu streichen.

Faktor	Konstrukt	Gewicht	Ladung	Kommunalität	Redundanz
D2-KL	A	0.48	0.99	0.98	0.00
D2-GZ-F	A	0.53	0.99	0.99	0.00
IST-Fig	AG	0.57	0.88	0.78	0.19
IST-Verb	AG	0.56	0.88	0.77	0.19
PE-Selbst	PE	0.24	0.98	0.95	0.00
PE-Java	PE	0.20	0.97	0.93	0.00
PE-OOP	PE	0.24	0.97	0.93	0.00
PE-Beruf	PE	0.19	0.85	0.72	0.00
PE-Kom	PE	0.20	0.92	0.85	0.00
Klasse1	F	0.28	0.79	0.62	0.25
Algo1	F	0.35	0.82	0.67	0.27
Klasse2	F	0.16	0.75	0.56	0.23
Algo2	F	0.09	0.66	0.44	0.18
Klasse3	F	0.13	0.58	0.34	0.14
Algo3	F	0.32	0.75	0.56	0.23

Tabelle 5.8: Deskriptive Statistiken der Faktoren

Diskriminanzvalidität

Die Diskriminanzvalidität wird in dieser Studie über die Cross-loadings analysiert. In Tabelle 5.10 sind die Cross-loadings abgebildet. Es ist zu erkennen, dass die Ladungen der Faktoren auf dem zugewiesenen Konstrukt größer sind als alle Cross-loadings, daher ist die Diskriminanzvalidität des Messmodells gegeben.

Faktor	Konstrukt	A	AG	PE	F
D2-KL	A	**0.99**	0.49	0.14	0.15
D2-GZ-F	A	**0.99**	0.48	0.13	0.21
IST-Fig	AG	0.35	**0.88**	0.52	0.47
IST-Verb	AG	0.51	**0.88**	0.41	0.29
PE-Selbst	PE	0.10	0.52	**0.98**	0.67
PE-Java	PE	0.15	0.56	**0.97**	0.58
PE-OOP	PE	0.16	0.52	**0.97**	0.70
PE-Beruf	PE	0.03	0.37	**0.85**	0.56
PE-Kom	PE	0.18	0.52	**0.92**	0.58
Klasse1	F	0.12	0.31	0.53	**0.84**
Algo1	F	0.12	0.36	0.71	**0.87**
Klasse2	F	0.00	0.14	0.39	**0.73**
Algo3	F	0.27	0.45	0.38	**0.69**

Tabelle 5.10: Cross-loadings

Qualität des Strukturmodells

Die Qualität des Strukturmodells wurde, wie in Abschnitt 4.4 beschrieben, anhand des Determinationskoeffizienten und der Pfadkoeffizienten bestimmt. Der Determinationskoeffizient gibt die Varianz der Konstrukte an, die durch die Faktoren erklärt werden kann. Ein größerer Determinationskoeffizient entspricht einer größeren Qualität des Strukturmodells. Der Determinationskoeffizient (siehe Tabelle 5.11) des Konstrukts Fähigkeit ist als hoch anzusehen, der Determinationskoeffizient des Konstrukts Arbeitsgedächtnis hingegen als akzeptabel. Die

Konstrukt	Typ	R^2
Aufmerksamkeit	Exogen	0.00
Arbeitsgedächtnis	Endogen	0.24
Programmiererfahrung	Exogen	0.00
Fähigkeit	Endogen	0.45

Tabelle 5.11: Determinationskoeffizienten R^2 der Konstrukte

Effekte der Konstrukte sind in Tabelle 5.12 abgebildet. Zum einen werden direkte Effekte gezeigt, die mit den Pfadkoeffizienten gleichzusetzen sind, zum anderen werden der indirekte und der kombinierte Effekt abgebildet. Es ist zu erkennen, dass die Aufmerksamkeit einen Gesamteffekt von 0.10 auf die Fähigkeit hat. Das Arbeitsgedächtnis hat einen Gesamteffekt von 0.08 auf die Fähigkeit. Diese beiden Effekte werden von der Programmiererfahrung, die mit 0.61 den größten Effekt auf die Fähigkeit hat, deutlich übertroffen. Daraus ist abzuleiten, dass die psychologischen Komponenten nur einen kleinen Teil der Fähigkeit erklären, wohingegen die Programmiererfahrung für den größten Teil verantwortlich ist.

Bootstrapping-Methode

Die Signifikanz der ermittelten Größen wird in dieser Arbeit durch die Bootstrapping-Methode analysiert. Dazu wurden 5000 zufällige Stichproben generiert und diese dazu genutzt das Modell zu schätzen.

Beziehung	direkt	indirekt	gesamt
A -> AG	0.49	0.00	0.49
A -> PE	0.00	0.00	0.00
A -> F	0.06	0.04	0.10
AG -> PE	0.00	0.00	0.00
AG -> F	0.08	0.00	0.08
PE -> F	0.61	0.00	0.61

Tabelle 5.12: Effekte der Konstrukte

Signifikanz der Ladungen

Alle Ladungen der Faktoren auf die Konstrukte sind signifikant, da die Bootsrap-Konfidenzintervalle keine Null enthalten (siehe Tabelle 5.13). Die Schätzung Ladungen der Faktoren des Konstrukts Aufmerksamkeit sind sehr stabil, da die Bereite des Boostrap-Konfidenzintervalls sehr gering ist. Die Schätzung der Faktoren des Konstrukts Arbeitsgedächtnis weisen dagegen eine geringere Stabilität auf. Bei der Betrachtung der Faktoren des Konstrukts Programmiererfahrung fällt auf, dass die Schätzung der Ladungen der Faktoren PE-Beruf und PE-Kom eine geringer Stabilität aufweisen als die Schätzung der Ladungen der restlichen Faktoren dieses Konstrukts. Die Schätzung der Ladungen Faktoren des Konstrukts Fähigkeit weisen die geringsten Stabilitäten auf. Gerade die Schätzung der Ladungen der Faktoren Algo2 und Klasse3 erweist sich als wenig stabil.

	Original	Mean.Boot	Std.Error	perc.025	perc.975
A-D2-KL	0.99	0.99	0.00	0.98	1.00
A-D2-GZ-F	0.99	0.99	0.00	0.99	1.00
AG-IST-Fig	0.88	0.88	0.07	0.75	0.97
AG-IST-Verb	0.88	0.87	0.07	0.72	0.95
PE-PE-Selbst	0.98	0.97	0.01	0.95	0.99
PE-PE-Java	0.97	0.96	0.02	0.91	0.99
PE-PE-OOP	0.97	0.97	0.01	0.94	0.98
PE-PE-Beruf	0.85	0.85	0.03	0.78	0.91
PE-PE-Kom	0.92	0.92	0.03	0.84	0.96
F-Klasse1	0.79	0.78	0.12	0.53	0.93
F-Algo1	0.82	0.82	0.09	0.70	0.93
F-Klasse2	0.75	0.73	0.15	0.32	0.90
F-Algo2	0.66	0.62	0.19	0.13	0.86
F-Klasse3	0.58	0.56	0.18	0.13	0.82
F-Algo3	0.75	0.72	0.14	0.38	0.88

Tabelle 5.13: Bootstrap-Methode: Ladungen

Signifikanz der Determinationskoeffizienten

Die Determinationskoeffizienten der beiden endogenen Konstrukte Arbeitsgedächtnis und Fähigkeit sind signifikant. Die Stabilität der Schätzungen ist für beide Konstrukte ist ähnlich.

	Original	Mean.Boot	Std.Error	perc.025	perc.975
AG	0.24	0.25	0.11	0.04	0.48
F	0.41	0.48	0.09	0.31	0.67

Tabelle 5.14: Bootstrap-Methode: Determinationskoeffizienten

Signifikanz der Gesamteffekte

Die Analyse der Boostrap-Konfidenzintervalle (siehe Tabelle 5.15) zeigt, dass die Gesamteffekte von dem Konstrukt Aufmerksamkeit zu Fähigkeit und Arbeitsgedächtnis zu Fähigkeit bezogen auf eine fünfprozentige Irrtumswahrscheinlichkeit nicht signifikant sind. Daher müssen diese beiden Hypothesen verworfen werden. Die Gesamteffekte von dem Konstrukt Programmiererfahrung zu Fähigkeit und Aufmerksamkeit zu Arbeitsgedächtnis hingegen sind signifikant.

	Original	Mean.Boot	Std.Error	perc.025	perc.975
A -> AG	0.49	0.49	0.13	0.21	0.69
A -> PE	0.00	0.00	0.00	0.00	0.00
A -> F	0.09	0.09	0.13	-0.19	0.32
AG -> PE	0.00	0.00	0.00	0.00	0.00
AG -> F	0.07	0.06	0.17	-0.30	0.39
PE -> F	0.59	0.62	0.14	0.36	0.88

Tabelle 5.15: Bootstrap-Methode: Gesamteffekte

Vergleich von ExpertInnen und NovizInnen

Für den Vergleich der Pfadkoeffizienten der ExpertInnen und NovizInnen wird die Permutations-Methode verwendet, da schon bei der Vorstellung der Ergebnisse zur ersten Forschungsfrage ersichtlich wurde, dass die Daten nicht normalverteilt sind.
Die Tabelle 5.16 zeigt, dass sich die Pfadkoeffizienten nur für den Pfad vom Konstrukt Programmiererfahrung zum Konstrukt Fähigkeit signifikant voneinander unterscheiden.

	global	E	N	diff.abs	p.value	sig.05
A->AG	0.49	0.56	0.50	0.06	0.82	nein
A->F	0.06	0.02	-0.01	0.02	0.93	nein
AG->F	0.08	0.40	-0.07	0.46	0.15	nein
PE->F	0.61	0.29	0.83	0.54	0.02	ja

Tabelle 5.16: Gruppenvergleich der Pfadkoeffizienten

5.2.3 Ergebnisse zur dritten Forschungsfrage

In diesem Abschnitt wird die Auswertung der Transkripte der Bearbeitungen vorgestellt. Der Aufbau der Transkripte wurde in Abschnitt 4.5.3 beschrieben. Es werden nur Bearbeitungen der Items Klasse1 und Algo1 betrachtet, da sich die ExpertInnen für diese Items bezüglich der

Tippgeschwindigkeit, der Länge der Merkphasen und der Anzahl der Merkphasen signifikant von den NovizInnen unterscheiden. Bei den ExpertInnen wird dabei zwischen semantischen und syntaktischen Strukturierungen unterschieden. Die Transkripte der NovizInnen wurden dazu verwendet, mögliche Hürden und Fehlerquellen zu identifizieren. Dabei wurden Bearbeitungen jener NovizInnen betrachtet, bei denen davon auszugehen ist, dass sie keinen oder kaum Kontakt mit objektorientierter Programmierung oder Java hatten (Programmiererfahrung kleiner 8). Das Kapitel schließt mit einer Betrachtung untypischer Bearbeitungen, die die Grenzen der Dichotomisierung aufzeigen.

Semantische Strukturierung der ExpertInnen beim Merken und Übertragen

Die Bearbeitungen von ExpertInnen weisen häufig zwei Strukturmerkmale auf, die hier semantisch interpretiert werden sollen - es handelt sich dabei um die Bildung von Blöcken und das kontextuelle Ableiten.

Blöcke

Bei mehreren ExpertInnen (alle Programmiererfahrung größer 33) kann festgestellt werden, dass sie gleiche Sequenzen bündeln, die sie zwischen zwei Merkphasen notieren. Dabei stellen die zusammengefassten Anweisungen jeweils „programmlogisch“ klar abgrenzbare Blöcke dar. Die Testpersonen 2, 13 und 31 zeigen dieses Verhalten im Item Klasse1. Die Deklaration der Klasse mit ihren Attributen und die Klassenmethode bilden bei allen drei Personen jeweils einen in sich geschlossenen Block. Die Deklaration umfasst neben dem Zugriffsmodifikator und dem Klassennamen `Haus` noch zwei Attribute von verschiedenen Variablentypen, den Integer `nummer` sowie den String `farbe`. Aus semantischer Perspektive wird hier beschrieben, wodurch sich ein Haus in der gewählten Modellierung auszeichnet. Die Methode `streichen` enthält nur eine Anweisung, nämlich jene, die bewirkt, dass die Farbe des zu streichenden Objekts geändert wird. Die Merkphasen unmittelbar vor der Reproduktion von einem der beiden Blöcke dauerten bei den Probanden jeweils zwischen 3.8 und 6.8 Sekunden. Besonders deutlich wird die „Block-Perspektive“ bei Testperson 13 (siehe Tabelle 5.17), die nach der ersten Schreibphase zunächst eine weitere Merkphase absolviert (Z.3), in der der erste Block kontrolliert wird. Diese Interpretation liegt nahe, da im Anschluss an diese - mit 1.4 Sekunden auffällig kurze - Merkphase ein Zeichen korrigiert wird (Z.4).

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.12			
2	S	12.23	-56	1	␣␣␣␣public␣class␣Haus{ ␣␣␣␣private␣int␣nummer ␣␣␣␣private␣string␣farbe ␣␣␣␣
		12.23 (0.00)	-56	1	
3	M	1.41			
4	S	4.52	-1	1	␣␣␣␣public␣class␣Haus{ ␣␣␣␣private␣int␣nummer ␣␣␣␣private␣sString␣farbe; ␣␣␣␣
		4.52 (0.00)	-1	1	
5	M	3.78			

	Phase	Zeit in s	LS	G	Inhalt

6	S	14.02	-51	2	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣int␣nummer
					␣␣␣␣private␣String␣farbe;
					
					␣␣␣␣public␣void␣streiche(String␣farbr){
					␣␣␣␣this.farbe=farbe;
					␣␣␣␣

		14.02 (0.00)	-51	2	

Tabelle 5.17: Transkript zum Item Klasse1 von Testperson 13

Auch in den Bearbeitungen des Items Algo1 kann ein Verhalten der ExpertInnen beobachtet werden, dass ein Strukturieren des Quelltexts in Blöcke durch die Testpersonen nahelegt. Die Testpersonen 1, 2, 13 und 30 gehen in ihren Bearbeitungen jeweils in zwei Schritten vor: Im ersten Schritt wird der Block „Methodenkopf“ übertragen, im zweiten Schritt der „Methodenrumpf“. Im Methodenkopf werden zwei Werte vom Typ Integer übergeben: eine Zahl sowie ihr potentielles Vielfaches. Es handelt sich aus semantischer Perspektive daher zunächst um die Klärung der Frage, welche Zahlen auf ihre Relation hin untersucht werden sollen. Die eigentliche Prüfung der Beziehung findet erst im Methodenrumpf statt. Dabei handelt es sich um eine Standardumsetzung durch Prüfung des Divisionsrests. Mit Längen von 3.3 bis 6.7 Sekunden sind die Merkphasen mit denen im Item Klasse1 vergleichbar. Wieder wird der Eindruck, dass die Testpersonen den Quelltext tatsächlich in diese beiden Blöcke einteilen, von Testperson 13 unterstrichen. Analog zu ihrem Verhalten im Item Klasse1, schaltet sie der Bearbeitung des zweiten Blocks wieder eine Kontrolle vor (Z.3): nach einer 0.8-sekündigen Pause, auf die eine Korrektur folgt (Z.4), wendet sie sich dem Methodenrumpf zu (Z.6-9).

Ableiten

Die genauere Betrachtung der ausgewiesenen Pausen zwischen den Schreibphasen eines Merkzyklus und den anschließend produzierten Textelementen legt nahe, dass die Pausen für inhaltliche Reflexionen genutzt wurden. Ausgehend vom reproduzierten Quelltext werden durch semantische Überlegungen anschließende Anweisungen abgeleitet. Das Ableiten kann daher als inhaltliches Memorieren interpretiert werden. Beispiele für mögliches inhaltliches Memorieren im Item Klasse1 sind in den Bearbeitungen der Testpersonen 18, 26, 28 und 33 zu finden. Für das Item Algo1 finden sich ebenfalls Anhaltspunkte für inhaltliches Memorieren bei den Testpersonen 8 und 20.
Testperson 18 (Programmiererfahrung von 30) beginnt nach einer knapp dreisekündigen Merkphase zunächst mit der Deklaration der Klasse `Haus` (Z.2) und wechselt dann – möglicherweise zur Kontrolle der Deklaration – in eine kurze Merkphase (0.1 Sekunden) (Z.3), bevor mit der Auflistung der Attribute begonnen wird (Z.6). Den Konventionen entsprechend werden diese untereinander aufgelistet beginnend mit dem Integer `nummer`. Nachdem das erste Attribut vollständig und korrekt angegeben wurde, beginnt die Testperson mit der Notation des zweiten Attributs. Nachdem der Zugriffsmodifikator (private) und der Variablentyp aufgeführt wurden, hält die Testperson für 2.4 Sekunden inne (Pausenphase) (Z.7), bevor als Variablenname statt dem eigentlich richtigen Namen `farbe` der Name `Straße` notiert wird (Z.8). Man kann vermuten, dass die Testperson nach sinnvollen Eigenschaften gesucht hat, die einem Haus zugeschrieben werden können. Dabei musste erstens der Datentyp String berücksichtigt werden. Ein ausschlaggebendes Kriterium für die Wahl von `Straße` als Bezeichner könnte dann das Motiv der Vervollständigung der `nummer` - aufgefasst als Hausnummer - durch die Angabe einer Straße sein. Die beiden Attribute könnten gemeinsam als Adresse interpretiert werden

und verfügen somit über eine inhaltliche Verbindung. Diese Vermutung ist auch plausibel, wenn davon ausgegangen wird, dass in den bis dahin erfolgten Merkphasen (Z.1, Z.3 und Z.5) die Methode, in der der Kontext „Streichen eines Hauses“ aufgebaut wird, noch nicht berücksichtigt wurde. Nach der nächsten Merkphase wird der Name des Attributes zu `farbe` korrigiert. Die skizzierte Handlungsabfolge ist in Tabelle 5.18 nachzuvollziehen.

	Phase	Zeit in s	LS	G	Inhalt
5	M	2.50			
6	S	5.71	-33	0	␣␣␣␣public␣class␣Haus␣{ ␣␣␣␣private␣int␣nummer; ␣␣␣␣private␣String␣ ␣␣␣␣
7	P	2.43			
8	S	3.01	-6	1	␣␣␣␣public␣class␣Haus␣{ ␣␣␣␣private␣int␣nummer; ␣␣␣␣private␣String␣Straße; ␣␣␣␣ ␣␣␣␣
9	P	2.43			
		8.72 (4.86)	-39	1	
10	M	2.15			
11	P	2.19			
12	S	5.67	0	8	␣␣␣␣public␣class␣Haus␣{ ␣␣␣␣private␣int␣nummer; ␣␣␣␣private␣String␣Stfaraßbe; ␣␣␣␣ ␣␣␣␣ ␣␣␣␣

Tabelle 5.18: Ausschnitt aus dem Transkript zum Item Klasse1 von Testperson 18

Ein Beispiel dafür, dass Vervollständigungen aus semantischen Überlegungen heraus auch Schlüsselwörter betreffen könnten, zeigt die Bearbeitung von Testperson 26 (Programmiererfahrung 37). Nach einer zweisekündigen Merkphase beginnt die Testperson mit der Deklaration der Klasse (Z.2). Dabei setzt sie den Zugriffsmodifikator des Attributs `Number` auf `public` statt auf `private`, wie in Tabelle 5.19 zu erkennen ist. Diese abweichende Modellierung zieht keine syntaktischen Fehler nach sich und führt zu keinen Einschränkungen im Programmablauf. So ist es möglicherweise zu erklären, dass das Schlüsselwort erst im letzten Schritt vor der Abgabe zu `private` geändert wird (Z.20).

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.00			
2	S	10.19	-32	0	␣␣␣␣public␣class␣Haus␣{␣public␣int␣Number ↪ ; ␣␣␣␣
3	P	2.24			

Tabelle 5.19: Ausschnitt aus dem Transkript zum Item Klasse1 von Testperson 26

Hinweise auf semantische Überlegungen finden sich auch beim Prozess des Memorierens der Methode `streiche` von Testperson 28 (Programmiererfahrung 35). Nach sechs Merkphasen beginnt die Testperson damit, die Methode aufzuschreiben (Z.13). Dabei bleibt die Parameterliste im Methodenkopf noch leer, während der Methodenrumpf schon vollständig und korrekt implementiert wird. Nach einer zweisekündigen Pausenphase (Z.14) wird der `String␣farbe` in der Parameterliste ergänzt (Z.15) (siehe Tabelle 5.20). Die Abfolge der Phasen und Modifkationen des Quelltexts legt nahe, dass die Testperson sich nicht mehr explizit an die übergebene Parameterliste erinnern konnte. Dies könnte daran liegen, dass sie sich die Methode selbst z.B. nur inhaltlich - als „Hier wird das Haus gestrichen.“ - gemerkt hat. Die Pause in Z.14 dient dann dazu, aus den Anweisungen im Methodenrumpf den benötigten Parameter abzuleiten. Da es sich nur um eine einzelne Anweisung handelt, genügt der Testperson eine kurze Merkphase.

	Phase	Zeit in s	LS	G	Inhalt
12	M	0.94			

13	S	24.52	-32	3	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣String␣farbe;
					␣␣␣␣private␣int␣nummer;
					
					␣␣␣␣public␣void␣Streiche(){
					␣␣␣␣this.farbe␣=␣farbe;
					␣␣␣␣}
					␣␣␣␣

14	P	2.21			

15	S	1.65	-11	0	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣String␣farbe;
					␣␣␣␣private␣int␣nummer;
					
					␣␣␣␣public␣void␣Streiche(String␣farbe){
					␣␣␣␣this.farbe␣=␣farbe;
					␣␣␣␣}
					␣␣␣␣

16	P	2.66			

Tabelle 5.20: Ausschnitt aus dem Transkript zum Item Klasse1 von Testperson 28

Auch die Bearbeitung von Testperson 33 (Programmiererfahrung 33) enthält Anzeichen für das semantische Ableiten von Quellcode. Die Testperson beginnt mit einer eher langen Merkphase von über 10 Sekunden, auf die in einer ersten Schreibphase (Z.2) die Deklaration der Klasse mit ersten Teilen der Deklaration der Variablen `nummer` und `farbe` folgt. In der zweiten Schreibphase (Z.5) wird die Deklaration abgeschlossen. Darüber hinaus beginnt die Testperson mit der Implementation des Konstruktors `Haus`, dem sie den Parameter `String␣farbe` übergibt. Das Öffnen der geschweiften Klammern am Ende des Ausdrucks deutet an, dass der Konstruktor weiter ausgebaut werden soll. Es folgt eine Pause über 2.3 Sekunden, bevor die Testperson in einen neuen Merkzyklus wechselt. Nach zwei Merkphasen und zwei weiteren Pausen (alle mit einer Dauer von unter 3 Sekunden) wechselt die Testperson wieder in eine Schreibphase, in der der vermutete Konstruktur in eine Methode umgewandelt wird (Z.11). Dazu wird `Haus` durch `void␣streiche` ersetzt, bevor im gleichen Zyklus noch der Methodenrumpf befüllt und die Bearbeitung komplett abgeschlossen wird. Die Testperson hatte vermutlich erwartet, im Quelltext einen Konstruktor `Haus` vorzufinden. Das mehrmalige Wechseln zwischen Merk- und Pausenphasen gibt Hinweise darauf, dass das Fehlen eines Konstruktors bei der Testperson auf Verwunderung stößt. Sie versichert sich daher möglicherweise genau, dass dieser nicht bloß übersehen wurde. Der genaue Ablauf der angesprochenen Merkzyklen ist in der Tabelle 5.21

dargestellt.

	Phase	Zeit in s	LS	G	Inhalt
4	M	1.58			

5	S	17.02	-52	1	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣int␣nummer;
					␣␣␣␣private␣String␣farbe;
					
					␣␣␣␣public␣Haus(String␣farbe){
					␣␣␣␣
					␣␣␣␣

6	P	2.27			
		17.02 (2.27)	-52	1	
7	M	1.29			
8	P	1.44			
		0.00 (1.44)	0	0	
9	M	1.45			
10	P	2.90			

11	S	5.13	-11	8	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣int␣nummer;
					␣␣␣␣private␣String␣farbe;
					
					␣␣␣␣public␣Hauvoid␣streiche(String␣farbe)
					↪ {
					
					␣␣␣␣

12	P	2.02			

13	S	9.29	-19	5	␣␣␣␣public␣class␣Haus{
					␣␣␣␣private␣int␣nummer;
					␣␣␣␣private␣String␣farbe;
					
					␣␣␣␣public␣void␣streiche(String␣farbe){
					␣␣␣␣this.farbe␣=␣farbe;
					␣␣␣␣}
					␣␣␣␣}
					␣␣␣␣

Tabelle 5.21: Ausschnitt aus dem Transkript zum Item Klasse1 von Testperson 33

Anzeichen für semantisches Ableiten finden sich auch bei der Reproduktion des Algorithmus im Item Algo1. Die beiden Testpersonen 8 (Programmiererfahrung 41) und 20 (Programmiererfahrung 40) verhalten sich diesbezüglich ähnlich. Nach einer Merkphase von 14.7 bzw. 8.9 Sekunden (beide Z.1) beginnen die Testpersonen mit dem Notieren der memorierten Inhalte. Testperson 8 beginnt die if-else-Anweisung, implementiert jedoch die Bedingung nur unvollständig, was möglicherweise zunächst unbemerkt bleibt, da die Klammer am Ende der Anweisung geschlossen wird (Z.2). Nach einer Pause über 2.3 Sekunden wird die bisherige Bedingung `vielfaches␣%␣zahl` ergänzt um `==0` (Z.4). In der Pause könnte die Testpersonen den Algorithmus noch einmal im Kopf durchlaufen haben, um festzustellen, wie die Bedingung tatsächlich aussehen muss, damit in der Kontrollstruktur eine Entscheidung (mit boolean-Rückgabe) getroffen werden kann: Wie müssen sich die beiden Integer-Werte zueinander verhalten, damit es

sich um eine Zahl-Vielfaches-Bedingung handelt? Das Ergebnis dieser Überlegung könnte dann gewesen sein, dass die modulo-Prüfung notwendigerweise 0 ergeben muss. Auch Testperson 20 zögert zunächst, bevor sie die Bedingung in der if-else-Anweisung einsetzt (Z.2). Nach einer mit einer Dauer von fünf Sekunden etwas längeren Pause wird die komplette Bedingung und der Rest der Methode ohne weitere Pausen notiert (Z.4). Da beide Testpersonen nach eigener Auskunft über eine hohe Programmiererfahrung verfügen, dürfte ihnen die Prüfung, ob es sich bei einer Zahl um ein Vielfaches einer anderen Zahl handelt, als Standardbaustein gut bekannt sein. In Tabelle 5.22 ist beispielhaft ein Teil der zuletzt beschriebenen Bearbeitung zu sehen.

	Phase	Zeit in s	LS	G	Inhalt
1	M	14.68			
2	S	17.21	-50	1	`boolean␣istVielfaches(␣int␣zahl,␣int␣` ↪ `vielfaches){` `if(`
3	P	5.09			
4	S	17.71	-48	1	`boolean␣istVielfaches(␣int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl␣==␣0)␣return␣true;` `else␣return␣false;`

Tabelle 5.22: Ausschnitt aus dem Transkript zum Item Algo1 von Testperson 20

Syntaktische Strukturierung der ExpertInnen beim Merken und Übertragen

Neben semantisch aufgeladenen Handlungsweisen beim Übertragen von Quelltext können auch Elemente der syntaktischen Strukturierung ExpertInnen ausweisen. Testperson 7 (Programmiererfahrung 43) demonstriert im Item Klasse1 sowie im Item Algo1 Formen der syntaktischen Strukturierung.

Klammern

Im Item Klasse1 gliedert Testperson 7 ihren Quelltext im Voraus durch Klammern. Im Zuge der Deklaration der Klasse `Haus` mit ihren Attributen, die sich über zwei Merkzyklen erstreckt (Z.1 bis Z.4), wird der Klassenrumpf nicht nur durch eine geöffnete Klammer eingeleitet, sondern bereits im Voraus geschlossen (Z.4). Dabei wird eine Lücke gelassen, da die Testperson vermutlich bereits hier plant, den weiteren Quelltext zu einem späteren Zeitpunkt einzusetzen. Die Vorstellung von der Klasse als umfassendes Konstrukt, in das (hier nicht vorhandene) Konstruktoren und Methoden eingebettet werden, wird hier auf syntaktischer Ebene repräsentiert. Anschließend beginnt die Testperson in Z.6 in der dritten Schreibphase mit der Implementierung der Methode `Streiche`. Dabei beschränkt sie sich zunächst nur auf den Methodenkopf bis einschließlich der Parameterliste. In dem Folgenden Methodenrumpf wird wiederum ein Freiraum zwischen der öffnenden geschweiften Klammer und der schließenden geschweiften Klammer gelassen. In Z.9 braucht dieser Rumpf dann nur noch befüllt zu werden. Möglicherweise fußt das stark vorstrukturierende Verhalten von Testperson 7 auf bisherigen Programmiererfahrungen, nach denen das Auffinden vergessener Klammern oder anderer Satzzeichen als umständlich und ineffizient empfunden wird. Darüber hinaus kann das Verhalten auch als meta-kognitive Strategie ausgelegt werden: Durch vorausschauendes Strukturieren in einem aktuellen Merkzyklus entlastet die Testpersonen sich selbst in weiteren Merkzyklen, da weniger Informationen parallel zu neuen Inhalten laufend mit im Arbeitsgedächtnis gehalten werden müssen. Tabelle 5.23 verdeutlicht die Wirkung syntaktischer Elemente.

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.80			
2	S	9.89	-37	0	␣␣␣␣public␣class␣Haus␣{ ␣␣␣␣private␣int␣nummer; ␣␣␣␣
		9.89 (0.00)	-37	0	
3	M	1.88			
4	S	12.29	-23	3	␣␣␣␣public␣class␣Haus␣{ ␣␣␣␣private␣int␣nummer; ␣␣␣␣private␣String␣farbe; ␣␣␣␣} ␣␣␣␣
		12.29 (0.00)	-23	3	
5	M	2.28			

6	S	18.96	-34	13	␣␣␣␣public␣class␣Haus␣{
					␣␣␣␣private␣int␣nummer;
					␣␣␣␣private␣String␣farbe;
					
					␣␣␣␣public␣void␣Streiche(String␣farbe)␣{
					␣␣␣␣␣␣␣␣
					␣␣␣␣␣␣␣␣}}

Tabelle 5.23: Ausschnitt aus dem Transkript zum Item Klasse1 von Testperson 7

Die gleiche Technik zur Strukturierung des Quelltextes verwendet die Testperson im Item Algo1. Wieder werden Klammern immer paarweise gesetzt und ein Freiraum gelassen, in den nachfolgende Passagen später eingetragen werden. So entsteht eine Lücke für die if-else-Anweisung in Z.2, die in Z.4 mit dem Gerüst der if-else Anweisung gefüllt wird. Dort entsteht sodann eine weitere Lücke, die in Z.8 mit entsprechenden Anweisungen ausgefüllt wird. Das konsequente Befolgen dieser Vorgehensweise in der Klassenstruktur und im Algorithmus könnte dafür sprechen, dass es sich um eine bewährte, universell beim Programmieren verwendete Strategie der Testperson handelt.

Konventionen

Ebenfalls am Beispiel von Testperson 7 wird die Bedeutung von Konventionen für die syntaktische Strukturierung von Quelltext ersichtlich. Im Item Algo1 beginnt die Testperson mit dem Methodenkopf in Z.2. Im nächsten Merkzyklus wird die if-else-Anweisung in Z.4 teilweise implementiert. Nach dem Erstellen des Gerüsts und dem Befüllen der Bedingung wartet die Testperson für 2.7 Sekunden, bevor eine Korrektur an einem schon in der ersten Schreibphase produzierten Quelltext-Stück vorgenommen wird: Der Integer `Zahl`, den der Algorithmus als Eingabewert erhält, wird, wie in Tabelle 5.24 dargestellt ist, zum Integer `zahl` korrigiert, der in der Bedingung der if-else-Anweisung zuvor schon korrekt gesetzt worden war. Das Verhalten der Testperson könnte als Orientierung an Standard-Konventionen aufgefasst werden. Wenngleich es sich bei `zahl` nicht um die orthografisch korrekte Vokabel handelt, wird der Konvention

bei der Änderung der Vorzug gegeben.

	Phase	Zeit in s	LS	G	Inhalt
3	M	4.11			

4	S	12.22	-23	0	␣␣␣␣boolean␣istVielfaches(int␣Zahl,␣
					↪ int␣vielfaches)␣{
					␣␣␣␣if(vielfaches␣%␣zahl␣==␣0)␣{
					␣␣␣␣
					␣␣␣␣}
					␣␣␣␣}
					␣␣␣␣

5	P	2.70			

6	S	2.91	-2	1	␣␣␣␣boolean␣istVielfaches(int␣Zzahl,␣
					↪ int␣vielfaches)␣{
					␣␣␣␣if(vielfaches␣%␣zahl␣==␣0)␣{
					␣␣␣␣␣␣␣␣␣␣
					␣␣␣␣}
					␣␣␣␣}
					␣␣␣␣

Tabelle 5.24: Ausschnitt aus dem Transkript zum Item Algo1 von Testperson 7

Das Einhalten der Konventionen zeigt sich durchgängig über alle Testpersonen mit hoher Programmiererfahrung.

Fehlerquellen und Hürden der NovizInnen beim Merken und Übertragen

Bei der Analyse der Transkripte konnten spiegelbildlich zu den semantisch und syntaktisch (erfolgreichen) Strukturierungen der ExpertInnen auch Fehlerquellen und Hürden der NovizInnen beim Merken und Übertragen der Quelltexte in den Items Klasse1 und Algo1 ausgemacht werden. Dies soll exemplarisch an drei identifizierten Problembereichen aufgezeigt werden.
NovizInnen bereitet die Verarbeitung von syntaktisch komplexen Ausdrücken Schwierigkeiten. Die ungewohnte Kombination einzelner Zeichen, die nicht semantisch interpretiert werden kann, führt dazu, dass die Bildung von Chunks erschwert bzw. verhindert wird. Ein Beispiel, das dabei besonders heraus sticht, ist der Ausdruck `vielfaches%zahl==0`. Gerade Testpersonen mit keiner bis geringer Programmiererfahrung, wie die Testpersonen 10 (Programmiererfahrung 6) und 15 (Programmiererfahrung 7), aber auch Testpersonen mit einer etwas höheren Programmiererfahrung wie Testperson 37 (Programmiererfahrung 20), müssen den Ausdruck in mehrere Teile aufsplitten.
Als Hürde erweist sich auch die wahrscheinlich eingeschränkte Möglichkeit zur semantischen Überprüfung der Implementierung. Bei NovizInnen mit keiner bis geringer Erfahrung ist dies wenig verwunderlich: Beispiele hierfür sind die Testperson 19 (Programmiererfahrung 5), die das Attribut `farbe` als Integer definiert und die Testperson 10 (Programmiererfahrung 6), die dem Algorithmus `istVielfaches` die Variablen `intzahl` und `intvielfaches` übergibt. Aber auch NovizInnen mit Programmiererfahrung unterlaufen semantische Fehler. Testperson 11 (Programmiererfahrung 28) deklariert in der ersten Schreibphase die Klasse `Hausp` mit einem schwer entschlüsselbaren Attribut `private␣int␣Nublic␣class␣Haus`, welches neben semantischen Schwierigkeiten auch syntaktische Defizite aufweist. Syntaktisch korrekt, aber semantisch nicht korrekt ist auch das Attribut `private␣String␣Haus`, welches Testperson 36 (Programmiererfahrung 20), innerhalb der Klasse `Haus` deklariert.

Die syntaktische Strukturierung des Quelltexts durch richtige Klammerung erweist sich ebenfalls für ProgrammiernovizInnen als Hürde: Testperson 15 liefert hierfür ein Beispiel. Anstelle der Klammerung der Anweisungen in der if-else-Anweisung des Algorithmus wurde zunächst das Schlüsselwort `else` geklammert, bevor ein aufwendiger Suchprozess zum Auffinden der richtigen Klammerpositionen initiiert wurde.

Untypische Bearbeitungen

Einige Bearbeitungen erwiesen sich bei der Durchsicht der Bearbeitungen als untypisch bezogen auf das Matching von angegebener Programmiererfahrung und gezeigter Performance. Dabei gab es Abweichungen in beide Richtungen. Testperson 22 (Programmiererfahrung 14) benötigt für die Übertragung der Klasse und des Algorithmus jeweils nur genau so viele Merkphasen wie für den vergleichbaren natürlich sprachlichen Text. Dabei ist die Länge der Merkphasen insgesamt in beiden Items jedoch deutlich länger (4.8 Sekunden bzw. 11.2 Sekunden). Insgesamt ist die Testperson als guter „Merker“ einzustufen.
Demgegenüber zeigen die Testpersonen 14 (Programmiererfahrung 40) und 29 (Programmiererfahrung 34) Schwierigkeiten beim Memorieren und Reproduzieren der Quelltexte. Testperson 14 erweckt im Item Algo1 den Eindruck, wenig vertraut mit der Syntax von Java zu sein. Dazu unterläuft ein semantischer Fehler, dass da die Bedingung der if-Anweisung ungültig ist. Verglichen mit anderen ExpertInnen auf ähnlichem Niveau fällt der Umfang der übertragenen Blöcke zudem relativ gering aus. Schwächen auf semantischer Ebene zeigt auch Testperson 29. Hier fällt auf, dass die Testperson eventuell den Kontext zu wenig beachtet. Eine mögliche Ursache liegt darin liegen, dass die Klasse `haut` bis zuletzt im Item Klasse1 nicht zur Klasse `Haus` korrigiert wird. Als semantische Schwäche ist außerdem die Wahl des falschen Datentyps `String` bei der Deklaration der Variablen `nummer` anzusehen.
Die gefundenen Diskrepanzen zwischen erwarteter und tatsächlicher Performanz werden in der Diskussion der Ergebnisse noch einmal kritisch aufgegriffen.

5.3 Diskussion der Ergebnisse

In diesem Unterkapitel werden die Ergebnisse zu den drei Forschungsfragen auf Grundlage des theoretischen Hintergrunds und des Forschungsstands diskutiert.

In einem ersten Schritt wurden verschiedene Kennwerte auf Unterschiede zwischen ExpertInnen und NovizInnen hin getestet. Bei der Tippgeschwindigkeitsdifferenz unterscheiden sich ExpertInnen von NovizInnen nur in bei den Items Klasse1 und Algo1 signifikant voneinander. In den anderen Items konnten keine signifikanten Unterschiede festgestellt werden. Die Hypothese (H1a), dass ExpertInnen beim Memorieren und Reproduzieren von Quelltexten eine höhere Tippgeschwindigkeitsdifferenz haben als NovizInnen, kann auf dieser Grundlage nicht eindeutig bestätigt oder abgelehnt werden. Dass die Unterschiede nur bei den ersten beiden Items nachgewiesen werden konnten, weist darauf hin, dass die Hypothese möglicherweise nur bei kognitiv weniger anspruchsvollen Items tragfähig sein könnte.
Bezüglich der Fehlerratendifferenz konnten keine signifikanten Unterschiede nachgewiesen werden, damit muss die Hypothese (H1b), dass ExpertInnen eine geringer Fehlerrate beim Memorieren und Reproduzieren von Quelltexten haben, verworfen werden. Diese Hypothese wurde davon ausgehend getroffen, dass Schlüsselwörter für NovizInnen kognitiv anspruchsvoller und das Merken und Wiedergeben daher fehlerbehafteter sein würde. Da es sich bei vielen Schlüsselwörtern (`private`, `public`, `class`, `for` usw.) um bekannte englische Wörter handelt, konnten die NovizInnen diese im Bezug auf die Fehlerrate nicht messbar schlechter

memorieren und reproduzieren.
Die zeitliche Differenz der Gesamtlängen der Merkphasen und die Differenz der Anzahl der Merkphasen unterschied sich in den Items Klasse1, Algo1 und Klasse3 signifikant zwischen ExpertInnen und NovizInnen. Die Interpretation ist in diesem Fall ähnlich wie bei der Tippgeschwindigkeitsdifferenz. Auch hier werden die Hypothesen H1c und H1d nur in den weniger kognitiv anspruchsvollen Items bestätigt. Interessant ist jedoch, dass in dem Item Klasse3, welches zu den vermeidlich anspruchsvollsten Items gehört, ebenfalls ein Unterschied nachgewiesen werden konnte. Dies kann womöglich darauf zurückzuführen sein, dass der Quelltext des Items Klasse2 auch für NovizInnen übersichtlich erscheint. Darüber hinaus enthält das Item Klasse3 eine höhere Dichte an ungewohnten Schlüsselwörtern wie `float`, `super` und `extends`, sodass der Quelltext für NovizInnen wahrscheinlich schwieriger zu merken ist, was in einer höheren Gesamtlänge der Merkphasen und höheren Anzahl an Merkphasen für die informatischen Items resultiert.

Die Ergebnisse zur zweiten Forschungsfrage haben gezeigt, dass die im Strukturmodell angenommenen Beziehungen nur teilweise angenommen werden können. Die Programmiererfahrung erweist sich als signifikanter Prädiktor für die Fähigkeit in den Programmieritems (operationalisiert durch die Differenz der Gesamtlänge der Merkphasen). Damit wurde festgestellt, dass die Prognosekraft der Programmiererfahrung auch für Formate, in denen nicht nur die Fähigkeit zum eigenständigen Programmieren abgefragt wird, gegeben ist. Außerdem konnte festgestellt werden, dass die kognitiven Größen Aufmerksamkeit und Arbeitsgedächtnis keinen signifikanten Einfluss auf die Fähigkeit haben. Dieser Befund erweist sich als vielversprechend für die Eignung des Messinstruments zur Kompetenzmessung. Mit dem Messinstrument werden also Kompetenzunterschiede im Bereich der Informatik gemessen und nicht nur kognitionspsychologische Niveauunterschiede repliziert. Ein Vergleich der Pfadkoeffizienten zwischen den ExpertInnen und NovizInnen zeigt, dass der Pfadkoeffizient und damit der Einfluss von der Programmiererfahrung auf die Fähigkeit bei NovizInnen stärker ist als bei ExpertInnen. Daraus lässt sich schließen, dass mit dem Messinstrument vermutlich vor allem im Anfängerbereich Kompetenzunterschiede gemessen werden können.

In der qualitativen Analyse haben sich deutliche Unterschiede in den Bearbeitungsstrategien von ExpertInnen und NovizInnen in der Programmierung gezeigt. Es konnten vier strukturierende Vorgehensweisen identifiziert werden, die von ExpertInnen verwendet werden. Semantische Strukturierungen umfassen zum einen die Bildung von Blöcken. Daher werden mehrere semantisch zusammenhängende Quelltextzeilen verbunden, sodass hier davon ausgegangen werden kann, dass dabei Chunks gebildet werden. Gerade der Klassenkopf wurde oft zusammen mit den Attributen in einer Schreibphase übertragen, sodass diese Elemente wahrscheinlich mittels Chunking gemerkt wurden. Die Bewertung, welche Elemente ein Chunk beinhaltet, war nicht Bestandteil dieser Arbeit. Jedoch kann die Vermutung angestellt werden, dass die Zugriffsmodifikatoren, sowie die Datentypen von den ExpertInnen komprimiert werden können, zum Beispiel indem der Datentyp vom Attributnamen abgeleitet wird oder davon ausgegangen wird, dass der Zugriffsmodifikator nach den Standardkonventionen gewählt wurde. Als zweite Form der semantischen Strukturierung legt das Verhalten des Ableitens nahe, dass ExpertInnen auf Strukturen im Langzeitgedächtnis zurückgreifen, da sie versuchen zu erschließen, welche Inhalte fehlen bzw. welche Inhalte semantisch passen könnten. Dieses Verhalten legt die Vermutung nahe, dass sie den aktuell geschrieben Quelltext mit Quelltexten, die sie schon einmal geschrieben haben oder bekannten syntaktischen Strukturen abgleichen. Dies spricht wiederum dafür, dass sie eine Art von Templating verwenden. Auch die gefundenen syntaktischen Strukturierungen durch Klammern und Konventionen unterstützen diese Vermutung.

6 Zusammenfassung, Fazit und Ausblick

6.1 Zusammenfassung

Um einen Baustein zur Kompetenzmessung in den Dimensionen Syntax und Semantik entwickeln zu können, wurden im Rahmen dieser Arbeit in Kapitel 2 zunächst Grundlagen zu Syntax und Semantik der objektorientierten Programmiersprache Java gelegt, bevor im weiterem Verlauf des Kapitels in die kognitionspsychologischen Konzepte des Arbeitsgedächtnisses, der Aufmerksamkeit und der Expertise eingeführt wurde. Die Grundlagen aus diesen beiden Unterkapitel wurden dann verwendet, um aufzuzeigen, wie Kompetenzunterschiede in den Dimensionen Syntax und Semantik aussehen können.
In Kapitel 3 wurden bereits durchgeführte Studien in diesem Bereich vorgestellt. In der Mehrheit der Studien mussten Quelltextfragmente zunächst memoriert und danach reproduziert werden. Dabei wurde die Zeitspanne, in derer die Quelltextfragmente angezeigt wurden, vorher festgelegt. Die Ergebnisse dieser Studien deuteten darauf hin, dass ExpertInnen der Programmierung mehr Quelltextzeilen memorieren können als NovizInnen. In dieser Arbeit wurde eine Weiterentwicklung von diesem methodischen Vorgehen vorgenommen. Die Verantwortung dafür, wann und wie oft der Quelltext angeschaut wird, wurde den Testpersonen übertragen. Das Ziel dieser Veränderung im Untersuchungsdesign lag darin, unterschiedliche Vorgehensweisen von ExpertInnen und NovizInnen zu identifizieren.
In Kapitel 4 wurden Konzeption und Methodik der empirischen Untersuchung vorgestellt. Die Erhebung bestand aus drei unabhängigen Modulen: einem kognitionspsychologischen Modul, einem informatischen Modul und einem Fragebogen. Das kognitionspsychologische Modul enthielt einzelne Tests zur Aufmerksamkeit und zu Komponenten des Arbeitsgedächtnisses. Der Einsatz dieses Moduls diente der Kontrolle der unterschiedlichen kognitiven Fähigkeiten der Testpersonen. Das informatische Modul enthielt den konstruierten Baustein zur Kompetenzmessung. Innerhalb des Bausteins wurden drei Gruppen aus jeweils drei Items gebildet, wobei jede der drei Gruppen aus einer Klassenstruktur, einem Algorithmus und einem natürlichsprachlichen Text bestand. Die Items wurden hinsichtlich ihrer kognitiven Komplexität und ihrer Länge bewertet, sodass die Items einer Gruppe jeweils aus theoretischer Sicht ein ähnliches Anforderungsniveau aufweisen sollten. Für die Einschätzung der kognitiven Komplexität der informatischen Items wurde die WCC-Metrik herangezogen, für die Einschätzung der natürlichsprachlichen Texte wurden die erste neue Wiener-Sachtextformel und der Flesch-Reading-Ease-DE verwendet. Die Gründe für die Auswahl dieser Bewertungsinstrumente sind in den theoretischen Grundlagen dargelegt.
Im Vorfeld der empirischen Untersuchung wurden drei Forschungsfragen formuliert. Die Darstellung und Diskussion der Ergebnisse zu den Forschungsfragen sind in Kapitel 5 enthalten. Zuerst wurde geprüft, mit welchen Kenngrößen Unterschiede zwischen ExpertInnen und NovizInnen gemessen werden können. In der Untersuchung wurden für diese Forschungsfrage die Tippgeschwindigkeit, die Fehlerrate, die Gesamtlänge der Merkphasen und die Anzahl der Merkphasen erhoben. Um die Vergleichbarkeit der Testpersonen zu ermöglichen, wurde für die einzelnen Kenngrößen jeweils die Differenzen zwischen den informatischen und natürlichsprachlichen Items berechnet. Zusammenfassend fällt auf, dass ExpertInnen und NovizInnen sich bzgl. der Kennwerte vorwiegend in den Items auf der niedrigsten Komplexitäts-

stufe unterscheiden. Die Fehlerrate kann als einzige Kenngröße auf Basis der Bearbeitungen ausgeschlossen werden, da die Gruppenunterschiede hier nie signifikant wurden.
Außerdem wurde der Zusammenhang zwischen kognitionspsychologischen Kenngrößen, der Programmiererfahrung und der Fähigkeit, Quelltexte zu Memorieren und Reproduzieren (operationalisiert durch die Differenz der Gesamtlängen der Merkphasen) analysiert. Demnach haben kognitionspsychologischen Kenngrößen keinen signifikanten Einfluss auf die Fähigkeit. Die Programmiererfahrung erweist sich dagegen als guter Indikator der Fähigkeit, Quelltexte zu memorieren und zu reproduzieren. Wenn der Vergleich zwischen ExpertInnen und NovizInnen gezogen wird, dann fällt aus, dass die Programmiererfahrung für die NovizInnen ein mehr als doppelt so großer Prädiktor für die Fähigkeit ist, als für die Gruppe der ExpertInnen. Daraus kann gefolgert werden, dass Kompetenzunterschiede zwischen ExpertInnen wahrscheinlich schwerer mit dem Messinstrument zu identifizieren sind, als solche zwischen NovizInnen. Daher sollte das Messinstrument eher im Umfeld von NovizInnen eingesetzt werden.
Zuletzt wurden die Bearbeitungen der Testpersonen auf Bearbeitungsstrategien bei den ExpertInnen und Hürden bei den NovizInnen hin qualitativ untersucht. Anhand verschiedener Fallbeispiele aus der Gruppe der NovizInnen wurden mögliche kognitive Hürden erkannt werden. Bei den ExpertInnen konnten im Ergebnis beim Übertragen vier Strategien identifiziert werden. Zwei davon sind der Kategorie semantische Strategien zuzuordnen und zwei den syntaktischen Strategien. Aus den Bearbeitungen geht zwar hervor, dass Merkstrategien verwendet werden, aber eine eindeutige Aufklärung über den Inhalt dieser Merkstrategien war nicht möglich. Die Vermutung liegt aber nahe, dass ExpertInnen auf Strukturen aus dem Langzeitgedächtnis zurückgreifen, da sie sich oftmals nach längeren Pausen noch fehlende Quelltextbausteine erschließen konnten.

6.2 Fazit und Ausblick

Das Ziel dieser Arbeit lag darin, einen weiteren Baustein zur Validierung des Kompetenzmodells aus dem COMMOOP-Projekt zu entwickeln und zu erproben. Dieser Baustein sollte die Kompetenz in den Dimensionen Syntax und Semantik messen. Insgesamt hat sich der Baustein als Messinstrument bewährt. Für den Einsatz in der Kompetenzmessung sind jedoch Optimierungen vorzunehmen bzw. Einschränkungen zu treffen: Zwar können Kompetenzunterschiede zwischen ExpertInnen und NovizInnen gemessen werden, jedoch müssen dafür solche Items konstruiert werden, die vergleichbar sind zu denen, die in der durchgeführten Untersuchung der Gruppe 1 zugehörten. Durch detaillierte Analysen z.B. von Transkripten aus Phasen des lauten Denkens könnte weiter erforscht werden, weshalb sich in den übrigen beiden Item-Gruppen keine signifikanten Unterschiede zwischen den ExpertInnen und NovizInnen messen ließen. Das hier gewählte Design der Items lässt offen, ob hierbei die Länge der Items oder deren Komplexität eine größere Rolle einnimmt.
Bevor das Messinstrument zur Kompetenzmessung verwendet werden kann, ist die Erprobung weiterer Varianten zwecks Optimierung ratsam. Das entwickelte Tool bietet bereits noch weitere Varianten an, die im Rahmen einer nächsten Forschungsarbeit getestet werden könnten. Die Beschränkung der Bearbeitungszeit oder der Anzahl der Merkphasen haben möglicherweise einen Einfluss auf die gewählten Strukturierungsformen.
Darüber hinaus liegt eine sinnvolle Ergänzung des bisherigen Designs in der weitergehenden Untersuchung, ob der Inhalt der Quelltexte den Testpersonen in Erinnerung bleibt. Dadurch könnte eventuell darauf geschlossen werden, wie stark auf semantischer Ebene agiert wird.
Schließlich könnte durch einen Wechsel von Java zu einer anderen ebenfalls objektorientierten Programmiersprache erforscht werden, ob die Kompetenz in den Dimensionen Syntax und Semantik nicht nur sprachspezifisch, sondern auch sprachübergreifend ausgeprägt wird.

A Anhang

A.1 Literaturrecherche

Datenbank	Suchwörter
Google Scholar	Working Memory, Working Memory CS, Working Memory Computer Science, Working Memory Programming, Short-term Memory, Short-term Memory CS, Short-term Memory Programming, Arbeitsgedächtnis Programmieren, Programming Recall
ACM Digital Library	Working Memory, Working Memory CS, Working Memory Computer Science, Working Memory Programming, Short-term Memory (1-4), Short-term Memory CS, Short-term Memory Programming, Programming Recall
Primo Universität Duisburg-Essen	Working Memory, Working Memory CS, Working Memory Computer Science, Working Memory Programming, Short-term Memory, Short-term Memory CS, Short-term Memory Programming, Arbeitsgedächtnis Programmieren, Programming Recall
Spinger Link	Working Memory, Working Memory CS, Working Memory Computer Science, Working Memory Programming, Short-term Memory, Short-term Memory CS, Short-term Memory Programming, Arbeitsgedächtnis Programmieren, Programming Recall
ScienceDirect	Working Memory, Working Memory CS, Working Memory Computer Science, Working Memory Programming, Short-term Memory, Short-term Memory CS, Short-term Memory Programming, Arbeitsgedächtnis Programmieren, Programming Recall
Scopus	Zitationssuche von Adelson (1981), Barfield (1986), Bateson, Alexander und Murphy (1987), Chase und Simon (1973), Guerin und Matthews (1990), Magliaro und Burtin (1987), Shneiderman (1976)

Tabelle A.1: Literaturrecherche zur unterschiedlichen Verwendung des Arbeitsgedächtnisses von Novizen und Experten beim Programmieren (Stand: 15.04.2017)

A.2 Fragebogen Items

Abbildung A.1: Fragebogen zur Erfassung personenbezogener Daten

A.3 Informatische Items

A.3.1 Erster Teilbereich

Text

Zeichen: 290, FRE_{DE}: 48, WS: 10
Frühling ist eine deutsche Filmreihe, die seit Anfang 2010 produziert wird. Durch ihr Talent, den Menschen zuzuhören, wird die Dorfhelferin dabei mehr zu einer Art Sozialarbeiterin, die es versteht, den Frühling in Familien zu tragen, die einen Schicksalsschlag erleben oder erlitten haben.

Algorithmus

Zeichen: 292, MC: 9

```
public int euklid(int zahl1, int zahl2) {
    if(zahl1 == 0) {
        return zahl2;
    } else {
        while(zahl2 != 0) {
            if(zahl1 > zahl2) {
                zahl1 -= zahl2;
            } else {
                zahl2 -= zahl1;
            }
        }
    }
    return zahl1;
}
```

A.3.2 Zweiter Teilbereich

Klassenstruktur 1

Zeichen: 147, WCC: 3

```
public class Haus {
    private int nummer;
    private String farbe;

    public void streiche(String farbe) {
        this.farbe = farbe;
    }
}
```

Text 1

Zeichen: 149, FRE_{DE}: 79, WS: 5
Lettland ist ein Staat im Osten von Europa. Er liegt an der Ostsee und gehört zu den baltischen Staaten. Die beiden anderen sind Estland und Litauen.

Algorithmus 1

Zeichen: 146, MC: 2

```
boolean istVielfaches(int zahl, int vielfaches) {
    if(vielfaches % zahl == 0) {
        return true;
    } else {
        return false;
    }
}
```

Algorithmus 2

Zeichen: 219, MC: 5

```
public String einruecken(String text, int anzahl, String symbol) {
    if(breite < 0) {
        return null;
    }
    for(int index = anzahl; index > 0; index--) {
        text = symbol + text;
    }
    return text;
}
```

Text 2

Zeichen: 206, FRE_{DE}: 69, WS: 8
Meerschweinchen heißen „Schweinchen“, weil sie wie Schweine quieken. „Meer“ kommt daher, dass sie von Südamerika aus über den Atlantik nach Europa gebracht worden sind. In Europa sind sie beliebte Haustiere.

Klassenstruktur 2

Zeichen: 214, WCC: 4

```
class Datei {
    String name;
    String pfad;

    Datei(String name, String pfad) {
        this.name = name;
        this.pfad = pfad;
    }

    void umbenennen(String name) {
        this.name = name;
    }
}
```

Klassenstruktur 3

Zeichen: 237, WCC: 7

```
public class Dreieck extends Vieleck {
    public float a, b, c = 1;

    public Dreieck() {
        super(3);
    }
}

public class Vieleck {
    private int ecken;

    public Vieleck(int ecken) {
        this.ecken = ecken;
    }
}
```

Text 3

Zeichen: 221, FRE_{DE}: 49, WS: 10
Allerheiligen ist ein Fest. Möglicherweise haben die Christen im frühen Mittelalter das Fest von Nichtchristen übernommen. Viele Menschen hatten das Bedürfnis, wenigstens einmal im Jahr gemeinsam an Verstorbene zu denken.

Algorithmus 3

Zeichen: 244, WCC: 8

```
void istPrimzahl(int zahl) {
    boolean primzahl = true;
    for(int p = 2; p < zahl; p++) {
        if(( i % p) == 0) {
            primzahl = false;
            break;
        }
    }
    if(primzahl) {
        System.out.println(i);
    }
}
```

A.4 Weitere Auswertungen

	M_N	M_E	W	p-Wert
Klasse1	0.008	-0.003	197.000	0.424
Algo1	0.013	0.014	145.000	0.460
Klasse2	0.007	0.016	150.000	0.557
Algo2	0.007	-0.000	199.000	0.390
Klasse3	0.007	0.004	183.000	0.707
Algo3	-0.002	-0.015	212.000	0.209

Tabelle A.2: Gruppenunterschiede zwischen ExpertInnen und NovizInnen im Bezug auf die Differenz der Fehlerraten (gemessen in Form der Anzahl gelöschter Zeichen pro Gesamtzeichenanzahl)

A.5 Transkripte

A.5.1 Testperson 1

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	1.00	-1.33	-2.39	-6.62	-7.57	-6.04
Anzahl	1.00	2.00	1.00	-2.00	-2.00	-1.00

Tabelle A.3: Übersicht über die Differenzen der Merkphasen von Testperson 1

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.74			
2	S	12.10	-38	4	`public␣class␣Haus␣{␣private␣int␣nummer ↪ ;␣`
3	P	3.75			
		12.10 (3.75)	-38	4	
4	M	3.08			
5	S	7.39	-21	3	`public␣class␣Haus␣{␣private␣int␣nummer ↪ ;␣private␣string␣farbe;␣`
6	P	3.27			
		7.39 (3.27)	-21	3	
7	M	2.46			
8	P	2.17			
9	S	11.40	-32	3	`public␣class␣Haus␣{␣private␣int␣nummer ↪ ;␣private␣string␣farbe;␣public␣ ↪ void␣streiche(␣String␣farbe)`
10	P	2.23			
11	S	6.55	-14	1	`public␣class␣Haus␣{␣private␣int␣nummer ↪ ;␣private␣string␣farbe;␣public␣ ↪ void␣streiche(␣String␣farbe)␣␣ ↪ this.farbe␣?␣farbe;6`
12	P	2.76			
13	S	4.93	-5	1	`public␣class␣Haus␣{␣private␣int␣nummer ↪ ;␣private␣string␣farbe;␣public␣ ↪ void␣streiche(␣String␣farbe)␣␣ ↪ this.farbe␣?=␣farbe;␣}}6`
		22.88 (7.16)	-51	5	

Tabelle A.4: Transkript zur Bearbeitung 607 von Testperson 1 (Programmiererfahrung 44)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.90			

	Phase	Zeit in s	LS	G	Inhalt
2	S	13.44	-47	0	`boolean␣istVielfaches(int␣zahl,int␣` ↪ `vielfaches)␣{`
		13.44 (0.00)	-47	0	
3	M	6.73			
4	S	17.62	-54	0	`boolean␣istVielfaches(int␣zahl,int␣` ↪ `vielfaches)␣{␣if␣(vielfaches␣%␣` ↪ `zahl␣==␣0)␣{␣return␣true;␣}␣else` ↪ `␣{␣return␣false;`
		17.62 (0.00)	-54	0	

Tabelle A.5: Transkript zur Bearbeitung 614 von Testperson 1 (Programmiererfahrung 44)

A.5.2 Testperson 2

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	1.01	4.70	9.96	0.20	-4.71	-6.98
Anzahl	1.00	1.00	2.00	1.00	0.00	-2.00

Tabelle A.6: Übersicht über die Differenzen der Merkphasen von Testperson 2

	Phase	Zeit in s	LS	G	Inhalt
1	M	5.62			

2	S	12.18	-59	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`

3	P	2.02			
		12.18 (2.02)	-59	0	
4	M	6.64			

5	S	19.57	-52	7	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)␣{`
					`this.farbe␣=␣farbe;`
					`}`

6	P	2.34			

7	S	0.00	-1	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)␣{`
					`␣this.farbe␣=␣farbe;`
					`}`

8	P	2.13			

9	S	2.85	0	5	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)␣{`
					`␣this.farbe␣=␣farbe;`
					`}`

10	P	3.16			

	Phase	Zeit in s	LS	G	Inhalt

11	S	1.35	0	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)␣{`
					`this.farbe␣=␣farbe;`
					`}`

		23.76 (7.63)	-52	12	

Tabelle A.7: Transkript zur Bearbeitung 639 von Testperson 2 (Programmiererfahrung 43)

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.00			
2	S	9.24	-48	0	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ vielfaches)␣{`
3	P	2.06			
		9.24 (2.06)	-48	0	
4	M	4.57			

5	S	13.06	-54	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`
					`return␣false;`
					`}`

		13.06 (0.00)	-54	0	

Tabelle A.8: Transkript zur Bearbeitung 644 von Testperson 2 (Programmiererfahrung 43)

A.5.3 Testperson 7

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	3.24	2.46	3.94	-7.59	-3.69	-14.07
Anzahl	-1.00	0.00	0.00	-3.00	-2.00	-2.00

Tabelle A.9: Übersicht über die Differenzen der Merkphasen von Testperson 7

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.80			

2	S	9.89	-37	0	public␣class␣Haus␣{
					private␣int␣nummer;
					private␣String␣farbe;
					
					}

		9.89 (0.00)	-37	0	
3	M	1.88			

4	S	12.29	-23	3	public␣class␣Haus␣{
					private␣int␣nummer;
					private␣String␣farbe;
					
					}

		12.29 (0.00)	-23	3	
5	M	2.28			

6	S	18.96	-34	13	public␣class␣Haus␣{
					private␣int␣nummer;
					private␣String␣farbe;
					
					public␣void␣Streiche(String␣farbe)␣{
					␣␣␣␣
					␣␣␣␣}}

		18.96 (0.00)	-34	13	
7	M	1.42			
8	P	2.17			

9	S	3.30	-17	0	public␣class␣Haus␣{
					private␣int␣nummer;
					private␣String␣farbe;
					
					public␣void␣Streiche(String␣farbe)␣{
					␣␣␣␣this.farbe␣=␣farbe;
					}}

		3.30 (2.17)	-17	0	

Tabelle A.10: Transkript zur Bearbeitung 377 von Testperson 7 (Programmiererfahrung 43)

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.48			

2	S	17.38	-48	4	`boolean␣istVielfaches(int␣Zahl,␣int␣`
					`↪ vielfaches)␣{`
					
					`}`

		17.38 (0.00)	-48	4	
3	M	4.11			

4	S	12.22	-23	0	`boolean␣istVielfaches(int␣Zahl,␣int␣`
					`↪ vielfaches)␣{`
					`if(vielfaches␣%␣zahl␣==␣0)␣{`
					
					`}`
					`}`

5	P	2.70			

6	S	2.91	-2	1	`boolean␣istVielfaches(int␣Zzahl,␣int␣`
					`↪ vielfaches)␣{`
					`if(vielfaches␣%␣zahl␣==␣0)␣{`
					`␣␣␣␣␣␣␣␣`
					`}`
					`}`

		15.13 (2.70)	-25	1	
7	M	1.56			

8	S	11.04	-30	1	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches)␣{`
					`if(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`
					`␣␣␣␣␣␣␣␣return␣false;`
					`}`
					`}`

		11.04 (0.00)	-30	1	

Tabelle A.11: Transkript zur Bearbeitung 381 von Testperson 7 (Programmiererfahrung 43)

A.5.4 Testperson 8

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	1.30	6.12	16.39	-26.72	-35.92	-31.82
Anzahl	1.00	2.00	1.00	-2.00	-3.00	-7.00

Tabelle A.12: Übersicht über die Differenzen der Merkphasen von Testperson 8

	Phase	Zeit in s	LS	G	Inhalt
1	M	9.27			

2	S	9.70	-30	0	`public␣class␣Haus{`
					`int␣numemr;`
					
					`}`

3	P	3.56			

4	S	11.05	-31	1	`public␣class␣Haus{`
					`int␣nummemr;`
					`␣␣␣␣String␣farbe;`
					
					`public␣void␣streic`
					
					`}`

5	P	7.12			

6	S	12.24	-35	3	`public␣class␣Haus{`
					`int␣nummer;`
					`String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					
					`}`

7	P	2.64			
		32.98 (13.32)	-96	4	
8	M	4.42			
9	P	2.18			

10	S	3.82	-8	5	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					
					`}`

11	P	2.83			

	Phase	Zeit in s	LS	G	Inhalt

12	S	0.98	-8	0	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					
					`}`

		4.80 (5.00)	-16	5	

Tabelle A.13: Transkript zur Bearbeitung 417 von Testperson 8 (Programmiererfahrung 41)

	Phase	Zeit in s	LS	G	Inhalt
1	M	8.86			

2	S	23.76	-66	7	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl)`
					`}`

3	P	2.35			

4	S	0.69	-4	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)`
					`}`

5	P	2.95			

6	S	11.01	-31	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`␣␣␣␣return␣true;`
					`}`
					`else␣return␣false;`
					`}`

		35.46 (5.30)	-101	7	

Tabelle A.14: Transkript zur Bearbeitung 421 von Testperson 8 (Programmiererfahrung 41)

A.5.5 Testperson 10

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-4.31	-6.76	-18.82	-27.10	-11.76	-60.52
Anzahl	-3.00	-5.00	-6.00	-12.00	-12.00	-41.00

Tabelle A.15: Übersicht über die Differenzen der Merkphasen von Testperson 10

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.45			
2	S	5.35	-16	0	`public␣clas␣Haus`
3	P	2.38			
		5.35 (2.38)	-16	0	
4	M	1.84			
5	P	3.36			
6	S	0.00	-1	0	`public␣class␣Haus`
7	P	3.55			
8	S	0.00	-1	0	`public␣class␣Haus{`
9	P	2.34			
10	S	1.31	0	0	`public␣class␣Haus{`
		1.31 (9.25)	-2	0	
11	M	4.65			
12	S	6.61	-18	0	`public␣class␣Haus{` `private␣int␣nummer,`
		6.61 (0.00)	-18	0	
13	M	2.13			

14	S	7.28	-21	0	`public␣class␣Haus{`
					`private␣int␣nummer,`
					`private␣string␣faber;`

		7.28 (0.00)	-21	0	
15	M	2.59			

16	S	5.08	-16	0	`public␣class␣Haus{`
					`private␣int␣nummer,`
					`private␣string␣faber;`
					`public␣void␣streiche`

17	P	2.60			
		5.08 (2.60)	-16	0	
18	M	3.43			

	Phase	Zeit in s	LS	G	Inhalt

19	S	8.80	-14	1	`public␣class␣Haus{`
					`private␣int␣nummer,`
					`private␣string␣faber;`
					`public␣void␣streiche(string␣farbe){`

		8.80 (0.00)	-14	1	
20	M	4.21			

21	S	7.52	-16	0	`public␣class␣Haus{`
					`private␣int␣nummer,`
					`private␣string␣faber;`
					`public␣void␣streiche(string␣farbe){`
					`this.farbe=farbe`

		7.52 (0.00)	-16	0	
22	M	2.47			

23	S	5.00	-3	0	`public␣class␣Haus{`
					`private␣int␣nummer,`
					`private␣string␣faber;`
					`public␣void␣streiche(string␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

		5.00 (0.00)	-3	0	

Tabelle A.16: Transkript zur Bearbeitung 496 von Testperson 10 (Programmiererfahrung 6)

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.65			
2	S	5.17	-8	0	`boolean␣`
		5.17 (0.00)	-8	0	
3	M	1.81			
4	S	3.15	-13	0	`boolean␣istVielfaches`
5	P	2.20			
		3.15 (2.20)	-13	0	
6	M	3.42			
7	S	11.82	-24	0	`boolean␣istVielfaches(intzahl,` `↪ intvielfaches)`
		11.82 (0.00)	-24	0	
8	M	1.17			
9	S	3.82	-1	0	`boolean␣istVielfaches(intzahl,` `↪ intvielfaches){`
		3.82 (0.00)	-1	0	
10	M	4.62			

	Phase	Zeit in s	LS	G	Inhalt
11	S	8.00	-17	0	`boolean istVielfaches(intzahl, intvielfaches){` `if (vielfaches%zahl`
12	P	2.29			
		8.00 (2.29)	-17	0	
13	M	2.02			
14	S	5.98	-5	1	`boolean istVielfaches(intzahl, intvielfaches){` `if (vielfaches%zahl==0)`
		5.98 (0.00)	-5	1	
15	M	1.26			
16	S	3.51	-1	0	`boolean istVielfaches(intzahl, intvielfaches){` `if (vielfaches%zahl==0){`
		3.51 (0.00)	-1	0	
17	M	3.70			

18	S	4.37	-12	0	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`

19	P	7.96			

20	S	1.86	-5	0	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`
					`{else}`

21	P	2.69			
		6.23 (10.65)	-17	0	
22	M	2.95			

23	S	3.53	0	1	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`
					`{}else}`

24	P	2.10			

25	S	1.65	-1	1	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`
					`}else}{`

26	P	2.22			

	Phase	Zeit in s	LS	G	Inhalt
		5.18 (4.32)	-1	2	
27	M	2.62			

28	S	4.26	-13	0	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`
					`}else{`
					`return false;`

29	P	3.26			

30	S	1.00	-1	0	`boolean istVielfaches(intzahl, intvielfaches){`
					`if (vielfaches%zahl==0){`
					`return true;`
					`}else{`
					`return false;`
					`}`
					`}`

		5.26 (3.26)	-14	0	

Tabelle A.17: Transkript zur Bearbeitung 517 von Testperson 10 (Programmiererfahrung 6)

A.5.6 Testperson 11

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-2.25	-1.45	4.53	-2.47	-5.23	-2.95
Anzahl	-1.00	-3.00	-1.00	-1.00	-6.00	-5.00

Tabelle A.18: Übersicht über die Differenzen der Merkphasen von Testperson 11

	Phase	Zeit in s	LS	G	Inhalt
1	M	5.36			
2	S	10.59	-30	5	`public␣class␣Hausp` `private␣int␣Nublic␣class␣Haus{`
		10.59 (0.00)	-30	5	
3	M	1.01			

4	S	13.29	-22	7	`public␣class␣Hausp`
					`private␣int␣Nnummer;`
					`private␣String␣ublic␣class␣Haus{`

		13.29 (0.00)	-22	7	
5	M	1.06			

6	S	1.77	-7	0	`public␣class␣Hausp`
					`private␣int␣nummer;`
					`private␣String␣farbe;ublic␣class␣Haus{`

7	P	2.09			

8	S	1.28	0	0	`public␣class␣Hausp`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`ublic␣class␣Haus{`

		3.04 (2.09)	-7	0	
9	M	1.53			

10	S	8.96	-35	6	`public␣class␣Hausp`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(String␣farbe){`
					`ublic␣class␣Haus{`

		8.96 (0.00)	-35	6	
11	M	0.87			

12	S	2.86	-16	0	`public␣class␣Hausp`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(String␣farbe){`
					`␣␣␣␣this.farbe=farbe;ublic␣class␣Haus{`

	Phase	Zeit in s	LS	G	Inhalt
		2.86 (0.00)	-16	0	

Tabelle A.19: Transkript zur Bearbeitung 287 von Testperson 11 (Programmiererfahrung 28)

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.98			
2	S	4.15	-22	0	`boolean␣istVielfaches(`
		4.15 (0.00)	-22	0	
3	M	0.55			
4	S	5.74	-26	0	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ vielfaches){`
		5.74 (0.00)	-26	0	
5	M	2.27			

6	S	10.45	-23	3	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl==0){`

		10.45 (0.00)	-23	3	
7	M	1.09			

8	S	6.03	-13	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl==0){`
					`return␣true;`
					`}`

		6.03 (0.00)	-13	0	
9	M	1.03			

10	S	4.61	-5	1	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl==0){`
					`return␣true;`
					`}␣else␣{`

		4.61 (0.00)	-5	1	
11	M	0.58			

12	S	4.76	-13	2	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl==0){`
					`return␣true;`
					`}␣else␣{`
					`return␣false`
					`}`
					`}`

	Phase	Zeit in s	LS	G	Inhalt
		4.76 (0.00)	-13	2	
13	M	0.53			
		0.00 (0.00)	0	0	

Tabelle A.20: Transkript zur Bearbeitung 291 von Testperson 11 (Programmiererfahrung 28)

A.5.7 Testperson 13

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	6.07	6.18	0.36	-8.29	-2.75	-2.11
Anzahl	6.00	5.00	-2.00	-10.00	-4.00	-5.00

Tabelle A.21: Übersicht über die Differenzen der Merkphasen von Testperson 13

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.12			

2	S	12.23	-56	1	`public␣class␣Haus{`
					`private␣int␣nummer`
					`private␣string␣farbe`

		12.23 (0.00)	-56	1	
3	M	1.41			

4	S	4.52	-1	1	`public␣class␣Haus{`
					`private␣int␣nummer`
					`private␣sString␣farbe;`

		4.52 (0.00)	-1	1	
5	M	3.78			

6	S	14.02	-51	2	`public␣class␣Haus{`
					`private␣int␣nummer`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbr){`
					`this.farbe=farbe;`

		14.02 (0.00)	-51	2	

Tabelle A.22: Transkript zur Bearbeitung 316 von Testperson 13 (Programmiererfahrung 41)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.33			
2	S	10.28	-44	5	`boolean␣istVielfaches␣(int␣zahl,␣int␣` `↪ Vielfaches`
		10.28 (0.00)	-44	5	
3	M	0.84			
4	S	6.32	-4	3	`boolean␣istVielfaches␣(int␣zahl,␣int␣` `↪ Vivelfaches){`
		6.32 (0.00)	-4	3	
5	M	0.12			
		0.00 (0.00)	0	0	

	Phase	Zeit in s	LS	G	Inhalt
6	M	4.90			

7	S	12.56	-33	4	`boolean␣istVielfaches␣(int␣zahl,␣int␣velfaches){`
					`if␣/(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true`

8	P	2.33			

9	S	5.58	-19	1	`boolean␣istVielfaches␣(int␣zahl,␣int␣velfaches){`
					`if␣/(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;}{else␣return␣false};`

		18.14 (2.33)	-52	5	

Tabelle A.23: Transkript zur Bearbeitung 319 von Testperson 13 (Programmiererfahrung 41)

A.5.8 Testperson 14

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-8.27	-1.58	0.89	-27.84	-5.51	-10.07
Anzahl	-1.00	1.00	1.00	-1.00	1.00	0.00

Tabelle A.24: Übersicht über die Differenzen der Merkphasen von Testperson 14

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.72			
2	S	10.10	-37	0	`public␣class␣Haus{private␣int␣nummer;`
		10.10 (0.00)	-37	0	
3	M	4.58			
4	S	10.81	-27	0	`public␣class␣Haus{private␣int␣nummer;␣` `↪ public␣String␣farbe;␣public␣void` `↪ ␣`
5	P	2.71			
		10.81 (2.71)	-27	0	
6	M	2.55			
7	S	17.94	-42	3	`public␣class␣Haus{private␣int␣nummer;␣` `↪ public␣String␣farbe;␣public␣void` `↪ ␣streiche(String␣farbe)␣{` `this.farbe␣=␣farbe;}}`
8	P	5.65			
9	S	0.00	0	0	`public␣class␣Haus{` `private␣int␣nummer;␣public␣String␣` `↪ farbe;␣public␣void␣streiche(` `↪ String␣farbe)␣{` `this.farbe␣=␣farbe;}}`
10	P	3.24			
		17.94 (8.89)	-42	3	
11	M	8.29			
12	P	2.22			
13	S	2.12	0	2	`public␣class␣Haus{` `private␣int␣nummer;␣private␣public␣` `↪ String␣farbe;␣public␣void␣` `↪ streiche(String␣farbe)␣{` `this.farbe␣=␣farbe;}}`
14	P	4.55			
15	S	1.25	-5	7	`public␣class␣Haus{` `private␣int␣nummer;␣private␣public␣` `↪ String␣farbe;␣public␣void␣` `↪ streiche(String␣farbe)␣{` `this.farbe␣=␣farbe;}}`

	Phase	Zeit in s	LS	G	Inhalt
		3.37 (6.77)	-5	9	

Tabelle A.25: Transkript zur Bearbeitung 608 von Testperson 14 (Programmiererfahrung 40)

	Phase	Zeit in s	LS	G	Inhalt
1	M	5.29			
2	S	14.83	-47	4	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{`
		14.83 (0.00)	-47	4	
3	M	7.17			
4	S	23.49	-32	18	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{if(vielfaches%zahl)␣return␣true;␣`
5	P	2.67			
6	S	0.00	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{if(vielfaches%zahl)␣return␣true;}`
7	P	3.13			
8	S	0.00	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{if(vielfaches%zahl)␣{␣return␣true;}`
9	P	3.06			
10	S	6.82	-19	2	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{if(vielfaches%zahl)␣{␣return␣true;}else{return␣false;}␣`
		30.32 (8.86)	-53	20	

Tabelle A.26: Transkript zur Bearbeitung 613 von Testperson 14 (Programmiererfahrung 40)

A.5.9 Testperson 15

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-11.79	-16.06	-15.59	-22.13	-23.27	-32.42
Anzahl	-4.00	-6.00	-8.00	-7.00	-11.00	-12.00

Tabelle A.27: Übersicht über die Differenzen der Merkphasen von Testperson 15

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.04			
2	S	6.53	-19	0	`public␣class␣Haus␣{`
3	P	2.12			
		6.53 (2.12)	-19	0	
4	M	5.21			

5	S	8.75	-24	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private`

		8.75 (0.00)	-24	0	
6	M	1.45			

7	S	4.32	-16	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`

8	P	2.59			
		4.32 (2.59)	-16	0	
9	M	3.32			

10	S	8.54	-32	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(string␣farbe)`

		8.54 (0.00)	-32	0	
11	M	4.03			

12	S	2.77	-1	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(string␣farbe)␣{`

13	P	3.84			
		2.77 (3.84)	-1	0	

	Phase	Zeit in s	LS	G	Inhalt
14	M	1.44			

15	S	7.58	-16	2	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(string␣farbe)␣{`
					`this.farbe␣=␣farbe;`

		7.58 (0.00)	-16	2	
16	M	1.25			

17	S	1.03	0	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(string␣farbe)␣{`
					`this.farbe␣=␣farbe;`

18	P	2.18			

19	S	1.98	-2	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(string␣farbe)␣{`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

		3.00 (2.18)	-2	0	

Tabelle A.28: Transkript zur Bearbeitung 495 von Testperson 15 (Programmiererfahrung 7)

	Phase	Zeit in s	LS	G	Inhalt
1	M	7.71			
2	S	4.87	-22	0	`boolean␣istVielfaches␣(`
3	P	3.32			
		4.87 (3.32)	-22	0	
4	M	2.66			
5	S	6.74	-24	2	`boolean␣istVielfaches␣(int␣zahl␣,␣int␣` `↪ vielfaches`
6	P	2.61			
		6.74 (2.61)	-24	2	
7	M	1.17			
8	S	5.75	-2	1	`boolean␣istVielfaches␣(int␣zahl␣,␣int␣` `↪ vielfaches)␣{`

	Phase	Zeit in s	LS	G	Inhalt
		5.75 (0.00)	-2	1	
9	M	3.94			
10	S	6.27	-12	2	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{` `if␣(vielfaches`
		6.27 (0.00)	-12	2	
11	M	0.99			
12	S	5.91	-9	0	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{` `if␣(vielfaches␣%␣zahl␣==␣0)`
		5.91 (0.00)	-9	0	
13	M	2.38			

14	S	6.20	-12	0	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true`

		6.20 (0.00)	-12	0	
15	M	1.06			

16	S	9.82	-7	3	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`{␣else␣}`

		9.82 (0.00)	-7	3	
17	M	1.46			

18	S	1.08	-1	1	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`{␣else␣}`

19	P	2.36			

20	S	0.00	1	0	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`{␣else␣{`

21	P	2.26			

22	S	0.73	0	1	`boolean␣istVielfaches␣(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`{}␣else␣{`

23	P	3.62			

	Phase	Zeit in s	LS	G	Inhalt

24	S	1.24	0	0	`boolean␣istVielfaches␣(int␣zahl␣,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`

		3.04 (8.25)	0	2	
25	M	2.66			

26	S	3.20	-13	0	`boolean␣istVielfaches␣(int␣zahl␣,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`
					`return␣false;`

27	P	2.09			

28	S	1.19	-2	0	`boolean␣istVielfaches␣(int␣zahl␣,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`
					`return␣false;`
					`}`
					`}`

		4.40 (2.09)	-15	0	

Tabelle A.29: Transkript zur Bearbeitung 510 von Testperson 15 (Programmiererfahrung 7)

A.5.10 Testperson 18

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-0.78	2.80	11.85	1.58	12.48	2.19
Anzahl	-2.00	-1.00	0.00	-8.00	-2.00	-9.00

Tabelle A.30: Übersicht über die Differenzen der Merkphasen von Testperson 18

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.86			
2	S	8.92	-19	5	`public␣class␣Haus␣{`
		8.92 (0.00)	-19	5	
3	M	0.16			
4	P	2.01			
		0.00 (2.01)	0	0	
5	M	2.50			

6	S	5.71	-33	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣`

7	P	2.43			

8	S	3.01	-6	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣Straße;`

9	P	2.43			
		8.72 (4.86)	-39	1	
10	M	2.15			
11	P	2.19			

12	S	5.67	0	8	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣Stfaraßbe;`

		5.67 (2.19)	0	8	
13	M	2.36			

14	S	6.51	-28	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`␣␣␣␣public␣void␣streiche(String␣`

15	P	2.33			

	Phase	Zeit in s	LS	G	Inhalt
		6.51 (2.33)	-28	1	
16	M	1.36			

17	S	4.33	-8	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)`
					`␣␣␣␣{`

18	P	2.46			
		4.33 (2.46)	-8	1	
19	M	0.88			

20	S	11.71	-17	7	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)`
					`{`
					`this.farbe␣=␣farbe`
					`␣␣␣␣}`
					`}`

21	P	2.99			

22	S	0.00	-1	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe)`
					`{`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

		11.71 (2.99)	-18	7	

Tabelle A.31: Transkript zur Bearbeitung 337 von Testperson 18 (Programmiererfahrung 30)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.93			
2	S	5.72	-23	2	`boolean␣vielfaches(int␣`
3	P	3.13			
		5.72 (3.13)	-23	2	
4	M	0.83			

	Phase	Zeit in s	LS	G	Inhalt
5	S	6.54	-20	1	`boolean␣vielfaches(int␣zahl,␣int␣` `↪ vielfaches){`
		6.54 (0.00)	-20	1	
6	M	1.24			

7	S	4.47	-16	0	`boolean␣vielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`␣␣␣␣if(vielfaches␣==␣`

		4.47 (0.00)	-16	0	
8	M	0.81			

9	S	9.25	-8	7	`boolean␣vielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)`
					`{`

		9.25 (0.00)	-8	7	
10	M	0.90			

11	S	10.03	-25	4	`boolean␣vielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)`
					`{`
					`return␣true;`
					`}`
					`return␣false;`

		10.03 (0.00)	-25	4	
12	M	0.99			

13	S	3.13	-2	0	`boolean␣vielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)`
					`{`
					`return␣true;`
					`}`
					`return␣false;`
					`}`
					`}`

14	P	2.52			

	Phase	Zeit in s	LS	G	Inhalt

15	S	2.51	-5	4	`boolean␣vielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)`
					`{`
					`return␣true;`
					`}`
					`r␣␣␣␣␣␣else␣eturn␣false;`
					`}`
					`}`

		5.64 (2.52)	-7	4	

Tabelle A.32: Transkript zur Bearbeitung 341 von Testperson 18 (Programmiererfahrung 30)

A.5.11 Testperson 19

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-18.22	-20.35	-15.98	-27.37	-22.52	-27.35
Anzahl	-3.00	-5.00	-5.00	-6.00	-6.00	-9.00

Tabelle A.33: Übersicht über die Differenzen der Merkphasen von Testperson 19

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.66			
2	S	6.66	-18	0	`public␣class␣Haus␣{`
3	P	2.59			
		6.66 (2.59)	-18	0	
4	M	3.83			
5	S	6.95	-18	1	`public␣class␣Haus␣{` `private␣int␣Nummer;`
6	P	2.44			

7	S	9.32	-18	2	`public␣class␣Haus␣{`
					`private␣int␣Nummer;`
					`private␣int␣farbe;,`

8	P	3.51			
		16.27 (5.95)	-36	3	
9	M	6.66			
10	P	3.89			

11	S	3.27	-5	3	`public␣class␣Haus␣{`
					`private␣int␣Nummer;`
					`private␣Strintg␣farbe;,`

12	P	2.84			

13	S	6.39	-16	1	`public␣class␣Haus␣{`
					`private␣int␣Nnummer;`
					`private␣String␣farbe;`
					`public␣voir␣farbe,`

14	P	2.21			

15	S	0.63	0	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣voird␣farbe,`

16	P	3.34			
		10.29 (12.27)	-21	5	
17	M	5.03			
18	P	2.83			

	Phase	Zeit in s	LS	G	Inhalt

19	S	12.10	-17	6	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣treiche(String␣farbe)␣{,`

20	P	2.93			

21	S	0.00	-1	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣streiche(String␣farbe)␣{,`

22	P	2.67			
		12.10 (8.42)	-18	6	
23	M	6.28			

24	S	9.04	-17	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣streiche(String␣farbe)␣{`
					`this.farbe␣=␣farbe;`
					`,`

25	P	2.17			
		9.04 (2.17)	-17	0	
26	M	0.88			
27	P	2.34			

28	S	2.01	-2	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣streiche(String␣farbe)␣{`
					`this.farbe␣=␣farbe;`
					`}`
					`},`

		2.01 (2.34)	-2	0	

Tabelle A.34: Transkript zur Bearbeitung 501 von Testperson 19 (Programmiererfahrung 5)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.33			
2	S	5.41	-21	0	`boolean␣istVierlfaches`
3	P	2.35			
4	S	0.00	-1	1	`boolean␣istVierlfaches`
5	P	3.41			
		5.41 (5.76)	-22	1	
6	M	3.56			
7	S	8.72	-23	2	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ Vielfaches)`
8	P	2.10			

	Phase	Zeit in s	LS	G	Inhalt
		8.72 (2.10)	-23	2	
9	M	2.16			
10	S	1.57	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `Vielfaches)␣`
11	P	2.15			
		1.57 (2.15)	-1	0	
12	M	6.23			
13	S	5.45	-12	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `Vielfaches)␣` `if␣(vielfaches`
14	P	2.22			
15	S	0.67	-1	1	`boolean␣istVielfaches(int␣zahl,␣int␣Vv` ↪ `ielfaches)` `if␣(vielfaches`
16	P	2.50			
17	S	0.00	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches)␣{` `if␣(vielfaches`
18	P	3.22			
		6.12 (7.93)	-14	1	
19	M	6.08			
20	S	9.71	-9	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches)␣{` `if␣(vielfaches␣%␣zahl␣==␣0)`
		9.71 (0.00)	-9	0	
21	M	2.84			
22	S	7.84	-13	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches)␣{` `if␣(vielfaches␣%␣zahl␣==␣0)␣{` `return␣true;`
		7.84 (0.00)	-13	0	
23	M	3.34			
24	P	3.54			
25	S	7.91	-19	2	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches)␣{` `if␣(vielfaches␣%␣zahl␣==␣0)␣{` `return␣true;` `}␣else␣{` `return␣false;`
26	P	2.17			
		7.91 (5.71)	-19	2	
27	M	0.94			

	Phase	Zeit in s	LS	G	Inhalt

28	S	3.19	-2	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{`
					`if␣(vielfaches␣%␣zahl␣==␣0)␣{`
					`return␣true;`
					`}␣else␣{`
					`return␣false;`
					`}`
					`}`

		3.19 (0.00)	-2	0	

Tabelle A.35: Transkript zur Bearbeitung 516 von Testperson 19 (Programmiererfahrung 5)

A.5.12 Testperson 20

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	27.57	27.10	8.66	-0.84	-11.15	-6.54
Anzahl	1.00	2.00	-2.00	0.00	-2.00	-1.00

Tabelle A.36: Übersicht über die Differenzen der Merkphasen von Testperson 20

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.73			

2	S	17.43	-48	2	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣int`

3	P	3.96			
		17.43 (3.96)	-48	2	
4	M	9.48			

5	S	6.90	-10	3	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣Strintg␣farbe,`

6	P	2.01			

7	S	18.67	-49	3	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe,`
					
					`public␣void␣streiche(String␣farbe{␣`
					`↪ this.farbe␣=␣farbe`

8	P	3.50			

9	S	2.03	-2	0	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe,`
					
					`public␣void␣streiche(String␣farbe{␣`
					`↪ this.farbe␣=␣farbe␣}}`

10	P	7.78			

11	S	0.00	-1	1	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe,`
					
					`public␣void␣streiche(String␣farbe{␣`
					`↪ this.farbe␣=␣farbe␣}}`

12	P	4.32			

	Phase	Zeit in s	LS	G	Inhalt

13	S	0.00	1	0	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe,`
					
					`public␣void␣streiche(String␣farbe{␣`
					`↪ this.farbe␣=␣farbe␣}}`

		27.60 (17.62)	-61	7	

Tabelle A.37: Transkript zur Bearbeitung 446 von Testperson 20 (Programmiererfahrung 40)

	Phase	Zeit in s	LS	G	Inhalt
1	M	14.68			

2	S	17.21	-50	1	`boolean␣istVielfaches(␣int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(`

3	P	5.09			

4	S	17.71	-48	1	`boolean␣istVielfaches(␣int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0)␣return␣true`
					`↪ ;`
					`else␣return␣false;`

		34.93 (5.09)	-98	2	

Tabelle A.38: Transkript zur Bearbeitung 451 von Testperson 20 (Programmiererfahrung 40)

A.5.13 Testperson 22

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-4.77	-11.17	-1.54	-22.34	-21.04	-26.50
Anzahl	0.00	0.00	-1.00	-2.00	1.00	0.00

Tabelle A.39: Übersicht über die Differenzen der Merkphasen von Testperson 22

	Phase	Zeit in s	LS	G	Inhalt
1	M	6.92			
2	S	8.01	-18	0	`public␣class␣Haus␣{`
3	P	2.55			

4	S	18.39	-61	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(`

5	P	2.71			
		26.40 (5.26)	-79	0	
6	M	7.85			

7	S	16.61	-31	1	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(String␣farbe)`
					`{␣this.farbe␣=␣farbe;`

8	P	2.80			

9	S	0.36	-2	0	`public␣class␣Haus␣{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche␣(String␣farbe)`
					`{␣this.farbe␣=␣farbe;}}`

		16.97 (2.80)	-33	1	

Tabelle A.40: Transkript zur Bearbeitung 405 von Testperson 22 (Programmiererfahrung 14)

	Phase	Zeit in s	LS	G	Inhalt
1	M	13.57			

2	S	16.77	-47	1	`boolean␣istVielfacher␣(int␣zahl,␣int␣`
					`↪ vielfaches)`
					`{␣`

3	P	3.18			

	Phase	Zeit in s	LS	G	Inhalt
4	S	1.42	-3	0	`boolean␣istVielfacher␣(int␣zahl,␣int␣` ↪ `vielfaches)` `{␣if␣(`
5	P	3.45			
		18.19 (6.64)	-50	1	
6	M	7.60			
7	S	27.35	-51	0	`boolean␣istVielfacher␣(int␣zahl,␣int␣` ↪ `vielfaches)` `{␣if␣(vielfacher␣%␣zahl␣==0)` `{return␣true;}␣else{return␣false;}}5`
		27.35 (0.00)	-51	0	

Tabelle A.41: Transkript zur Bearbeitung 407 von Testperson 22 (Programmiererfahrung 14)

A.5.14 Testperson 26

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-7.64	1.40	-7.03	-17.55	-7.93	-27.58
Anzahl	-3.00	1.00	-5.00	-3.00	-1.00	-4.00

Tabelle A.42: Übersicht über die Differenzen der Merkphasen von Testperson 26

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.00			
2	S	10.19	-32	0	`public␣class␣Haus␣{␣public␣int␣Number;`
3	P	2.24			
		10.19 (2.24)	-32	0	
4	M	3.09			
5	S	2.12	-1	0	`public␣class␣Haus␣{␣public␣int␣umm` ↪ `Number;`
6	P	4.16			
7	S	11.27	-19	12	`public␣class␣Haus␣{␣public␣int␣numm` ↪ `Number;␣private␣string␣farbe;`
		13.39 (4.16)	-20	12	
8	M	3.13			
9	S	10.90	-34	0	`public␣class␣Haus␣{␣public␣int␣nummer;` ↪ `␣private␣string␣farbe;␣public␣` ↪ `void␣streiche(String␣farbe){`
		10.90 (0.00)	-34	0	
10	M	1.48			
11	P	4.20			
		0.00 (4.20)	0	0	
12	M	1.21			
13	S	5.92	-19	0	`public␣class␣Haus␣{␣public␣int␣nummer;` ↪ `␣private␣string␣farbe;␣public␣` ↪ `void␣streiche(String␣farbe){this` ↪ `.farbe=farbe;}}`
14	P	2.66			
		5.92 (2.66)	-19	0	
15	M	2.24			
16	P	5.11			
17	S	0.23	-1	1	`public␣class␣Haus␣{␣public␣int␣nummer;` ↪ `␣private␣sString␣farbe;␣public␣` ↪ `void␣streiche(String␣farbe){this` ↪ `.farbe=farbe;}}`
18	P	2.15			

	Phase	Zeit in s	LS	G	Inhalt
		0.23 (7.25)	-1	1	
19	M	5.92			
20	S	3.51	-5	5	`public␣class␣Haus␣{␣publricvate␣int␣nummer;␣private␣String␣farbe;␣public␣void␣streiche(String␣farbe){this.farbe=farbe;}}`
		3.51 (0.00)	-5	5	

Tabelle A.43: Transkript zur Bearbeitung 447 von Testperson 26 (Programmiererfahrung 37)

	Phase	Zeit in s	LS	G	Inhalt
1	M	2.74			
2	S	9.67	-29	0	`boolean␣istVielfaches(int␣Zahl`
		9.67 (0.00)	-29	0	
3	M	2.21			
4	P	2.20			
5	S	5.49	-18	2	`boolean␣istVielfaches(int␣Zzahl,␣int␣vielfaches)␣{`
6	P	2.22			
		5.49 (4.42)	-18	2	
7	M	5.06			
8	S	22.79	-56	3	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{␣if(vielfaches%zahl==0){return␣true;}else{return␣false;}}`
		22.79 (0.00)	-56	3	

Tabelle A.44: Transkript zur Bearbeitung 449 von Testperson 26 (Programmiererfahrung 37)

A.5.15 Testperson 28

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-9.31	-5.12	5.29	-12.82	1.92	-12.48
Anzahl	-4.00	0.00	3.00	-5.00	0.00	-4.00

Tabelle A.45: Übersicht über die Differenzen der Merkphasen von Testperson 28

	Phase	Zeit in s	LS	G	Inhalt
1	M	6.51			
2	P	2.74			
		0.00 (2.74)	0	0	
3	M	2.24			
4	S	12.11	-20	2	`public␣class␣Haus{`
		12.11 (0.00)	-20	2	
5	M	0.16			
6	P	2.38			
		0.00 (2.38)	0	0	
7	M	0.97			

8	S	8.66	-28	0	`public␣class␣Haus{`
					`private␣String␣farbe;`
					`private␣int␣`

9	P	2.07			
		8.66 (2.07)	-28	0	
10	M	0.61			
11	P	1.17			
		0.00 (1.17)	0	0	
12	M	0.94			

13	S	24.52	-32	3	`public␣class␣Haus{`
					`private␣String␣farbe;`
					`private␣int␣nummer;`
					
					`public␣void␣Streiche(){`
					`this.farbe␣=␣farbe;`
					`}`

14	P	2.21			

15	S	1.65	-11	0	`public␣class␣Haus{`
					`private␣String␣farbe;`
					`private␣int␣nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`

	Phase	Zeit in s	LS	G	Inhalt
16	P	2.66			
		26.18 (4.87)	-43	3	
17	M	7.85			
18	P	3.56			

19	S	1.05	-2	0	`public␣class␣Haus{`
					`private␣String␣farbe;`
					`private␣int␣nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

20	P	6.00			

21	S	0.00	-1	0	`public␣class␣Haus{`
					`private␣String␣farbe;`
					`c␣private␣int␣nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

22	P	5.22			

23	S	4.68	2	1	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`c␣private␣int␣nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

24	P	2.82			

25	S	0.00	-1	1	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`␣␣␣c␣private␣int␣nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

26	P	4.70			

	Phase	Zeit in s	LS	G	Inhalt

27	S	2.27	-18	23	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;␣␣␣c␣private␣int␣`
					`↪ nummer;`
					
					`public␣void␣Streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

		8.00 (22.29)	-20	25	

Tabelle A.46: Transkript zur Bearbeitung 595 von Testperson 28 (Programmiererfahrung 35)

	Phase	Zeit in s	LS	G	Inhalt
1	M	11.90			
2	S	12.14	-30	1	`boolean␣istVielfaches(int␣zahl`
		12.14 (0.00)	-30	1	
3	M	1.45			
4	S	5.25	-17	0	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ vielfaches){`
5	P	2.56			
		5.25 (2.56)	-17	0	
6	M	1.74			

7	S	9.35	-19	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`␣␣␣␣if(vielfaches␣%␣zahl␣`

8	P	2.57			

9	S	6.33	-5	1	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣=␣=␣0){`

10	P	2.07			

11	S	0.00	0	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣=␣=␣0){`
					`␣␣␣␣`

12	P	2.78			

13	S	12.42	-31	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣=␣=␣0){`
					`return␣true;`
					`}␣else␣{`
					`return␣false;`
					`}`

Phase	Zeit in s	LS	G	Inhalt
	28.10 (7.43)	-55	1	

Tabelle A.47: Transkript zur Bearbeitung 597 von Testperson 28 (Programmiererfahrung 35)

A.5.16 Testperson 29

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-4.82	-5.13	-0.46	-9.46	-6.38	-9.14
Anzahl	0.00	-1.00	-1.00	-5.00	-4.00	-5.00

Tabelle A.48: Übersicht über die Differenzen der Merkphasen von Testperson 29

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.00			
2	S	7.43	-17	2	`public␣class␣haut{`
		7.43 (0.00)	-17	2	
3	M	3.18			
4	S	13.47	-34	7	`public␣class␣haut{private␣String␣` `↪ nummer;␣private␣String␣farbe`
5	P	2.36			
6	S	2.26	-2	2	`public␣class␣haut{private␣int␣String␣` `↪ nummer;␣private␣String␣farbe`
7	P	2.52			
8	S	2.39	-1	0	`public␣class␣haut{private␣int␣String␣` `↪ nummer;␣private␣String␣farbe;`
		18.12 (4.88)	-37	9	
9	M	3.55			
10	S	13.44	-30	3	`public␣class␣haut{private␣int␣String␣` `↪ nummer;␣private␣String␣farbe;` `public␣void␣streiche(String␣farbe){`
		13.44 (0.00)	-30	3	
11	M	3.92			
12	S	5.13	-17	0	`public␣class␣haut{private␣int␣String␣` `↪ nummer;␣private␣String␣farbe;` `public␣void␣streiche(String␣farbe){` `↪ this.farbe=farbe;`
13	P	2.89			
		5.13 (2.89)	-17	0	
14	M	1.63			
15	S	1.72	-2	0	`public␣class␣haut{private␣int␣String␣` `↪ nummer;␣private␣String␣farbe;` `public␣void␣streiche(String␣farbe){` `↪ this.farbe=farbe;}}`
16	P	3.54			

	Phase	Zeit in s	LS	G	Inhalt
17	S	0.21	-1	1	`public␣class␣hauts{private␣int␣String␣nummer;␣private␣String␣farbe; public␣void␣streiche(String␣farbe){this.farbe=farbe;}}`
18	P	6.01			
19	S	1.50	-8	7	`public␣class␣haus{private␣int␣String␣nummer;␣private␣String␣farbe; public␣void␣streiche(String␣farbe){this.farbe=farbe;}}`
		3.43 (9.55)	-11	8	

Tabelle A.49: Transkript zur Bearbeitung 466 von Testperson 29 (Programmiererfahrung 34)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.93			
2	P	0.05			
		0.00 (0.05)	0	0	
3	M	0.68			
4	S	18.95	-45	8	`boolean␣istVielfachen(int␣zahl,␣int␣vielfaches)`
		18.95 (0.00)	-45	8	
5	M	5.51			
6	S	10.13	-22	2	`boolean␣istVielfachen(int␣zahl,␣int␣vielfaches){if(vielfachen%zahl==0)`
7	P	3.65			
8	S	2.11	-1	1	`boolean␣istVielfachen(int␣zahl,␣int␣vielfaches){if(vielfachens%zahl==0)`
		12.24 (3.65)	-23	3	
9	M	2.62			
10	S	3.77	-13	0	`boolean␣istVielfachen(int␣zahl,␣int␣vielfaches){if(vielfaches%zahl==0){return␣true;`
11	P	2.64			
12	S	1.43	-1	0	`boolean␣istVielfachen(int␣zahl,␣int␣vielfaches){if(vielfaches%zahl==0){return␣true;}`
		5.20 (2.64)	-14	0	
13	M	2.12			

	Phase	Zeit in s	LS	G	Inhalt

14	S	3.55	-6	2	`boolean␣istVielfachen(int␣zahl,␣int␣`
					`↪ vielfaches){if(vielfaches%zahl`
					`↪ ==0){return␣true;}`
					`else␣`

		3.55 (0.00)	-6	2	
15	M	0.73			

16	S	4.28	-14	0	`boolean␣istVielfachen(int␣zahl,␣int␣`
					`↪ vielfaches){if(vielfaches%zahl`
					`↪ ==0){return␣true;}`
					`else␣{return␣false;}}`

17	P	2.69			

18	S	0.71	0	1	`boolean␣istVielfachen(int␣zahl,␣int␣`
					`↪ vielfaches){if(vielfaches%zahl`
					`↪ ==0){return␣true;}`
					`else␣{return␣false;}}`

		4.99 (2.69)	-14	1	

Tabelle A.50: Transkript zur Bearbeitung 470 von Testperson 29 (Programmiererfahrung 34)

A.5.17 Testperson 30

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	2.10	1.42	3.72	-3.00	6.74	1.31
Anzahl	-1.00	1.00	1.00	1.00	1.00	1.00

Tabelle A.51: Übersicht über die Differenzen der Merkphasen von Testperson 30

	Phase	Zeit in s	LS	G	Inhalt
1	M	1.42			
2	S	6.57	-18	0	`public␣class␣Haus␣{`
		6.57 (0.00)	-18	0	
3	M	1.95			

4	S	9.15	-39	0	`public␣class␣Haus␣{`
					`private␣int␣Nummer;`
					`private␣String␣farbe;`

		9.15 (0.00)	-39	0	
5	M	3.65			

6	S	10.99	-35	1	`public␣class␣Haus␣{`
					`private␣int␣Nummer;`
					`private␣String␣farbe;`
					`public␣void␣streiche(String␣farbe)␣{`

		10.99 (0.00)	-35	1	
7	M	2.08			

8	S	6.66	-19	0	`public␣class␣Haus␣{`
					`private␣int␣Nummer;`
					`private␣String␣farbe;`
					`public␣void␣streiche(String␣farbe)␣{`
					`this.farbe␣=␣farbe;␣}}`

		6.66 (0.00)	-19	0	

Tabelle A.52: Transkript zur Bearbeitung 336 von Testperson 30 (Programmiererfahrung 34)

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.27			
2	S	11.50	-47	0	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ vielfaches)␣{`
		11.50 (0.00)	-47	0	

	Phase	Zeit in s	LS	G	Inhalt
3	M	5.51			
4	S	17.13	-52	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches)␣{ if(vielfache␣%␣zahl␣==␣)␣{ return␣true␣} else␣{ return␣false␣}}`
		17.13 (0.00)	-52	0	

Tabelle A.53: Transkript zur Bearbeitung 339 von Testperson 30 (Programmiererfahrung 34)

A.5.18 Testperson 31

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-2.78	0.92	6.92	-9.49	-6.28	-4.93
Anzahl	0.00	0.00	-1.00	-3.00	-2.00	-4.00

Tabelle A.54: Übersicht über die Differenzen der Merkphasen von Testperson 31

	Phase	Zeit in s	LS	G	Inhalt
1	M	6.83			

2	S	21.75	-55	4	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣int␣farbe;`

		21.75 (0.00)	-55	4	
3	M	0.17			
4	P	1.75			
		0.00 (1.75)	0	0	
5	M	6.37			

6	S	18.17	-52	1	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣int␣farbe;`
					
					`public␣void␣streiche(String␣farbe)`
					`{`
					`this.farbe=farbe;`
					`}`

		18.17 (0.00)	-52	1	

Tabelle A.55: Transkript zur Bearbeitung 376 von Testperson 31 (Programmiererfahrung 33)

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.26			
2	S	16.14	-47	2	`boolean␣istVielfaches(int␣zahl,␣int␣` `↪ vielfaches){`
3	P	2.24			
		16.14 (2.24)	-47	2	
4	M	3.58			

	Phase	Zeit in s	LS	G	Inhalt

5	S	11.19	-23	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`

6	P	2.17			
		11.19 (2.17)	-23	0	
7	M	1.83			

8	S	12.79	-29	1	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`return␣true;`
					`}`
					`else{`
					`returen␣false`

		12.79 (0.00)	-29	1	

Tabelle A.56: Transkript zur Bearbeitung 380 von Testperson 31 (Programmiererfahrung 33)

A.5.19 Testperson 33

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-0.47	4.30	6.90	-1.64	0.14	-19.13
Anzahl	1.00	0.00	-3.00	-6.00	-1.00	-7.00

Tabelle A.57: Übersicht über die Differenzen der Merkphasen von Testperson 33

	Phase	Zeit in s	LS	G	Inhalt
1	M	10.70			
2	S	10.61	-30	4	`public␣class␣Haus{` `private␣int␣`
3	P	2.38			
		10.61 (2.38)	-30	4	
4	M	1.58			

5	S	17.02	-52	1	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣Haus(String␣farbe){`

6	P	2.27			
		17.02 (2.27)	-52	1	
7	M	1.29			
8	P	1.44			
		0.00 (1.44)	0	0	
9	M	1.45			
10	P	2.90			

11	S	5.13	-11	8	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣Hauvoid␣streiche(String␣farbe){`

12	P	2.02			

13	S	9.29	-19	5	`public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

		14.42 (4.92)	-30	13	

Tabelle A.58: Transkript zur Bearbeitung 246 von Testperson 33 (Programmiererfahrung 33)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.91			
2	S	7.85	-31	0	`boolean␣istVielfaches(int␣zahl1␣`
		7.85 (0.00)	-31	0	
3	M	1.34			
4	S	6.14	-17	2	`boolean␣istVielfaches(int␣zahl1,␣int␣` ↪ `vielfaches){`
		6.14 (0.00)	-17	2	
5	M	1.97			
6	S	4.80	-18	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl)`
7	P	2.10			
		4.80 (2.10)	-18	0	
8	M	1.80			
9	S	3.06	-3	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl␣==␣0)`
10	P	4.84			
11	S	4.08	-13	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl␣==␣0){` `return␣true;`
		7.14 (4.84)	-16	0	
12	M	1.23			
13	S	6.81	-18	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl␣==␣0){` `return␣true;` `}␣else␣{` `return␣false`
14	P	2.09			
15	S	1.67	-2	0	`boolean␣istVielfaches(int␣zahl,␣int␣` ↪ `vielfaches){` `if(vielfaches␣%␣zahl␣==␣0){` `return␣true;` `}␣else␣{` `return␣false` `}` `}`
		8.47 (2.09)	-20	0	

Tabelle A.59: Transkript zur Bearbeitung 250 von Testperson 33 (Programmiererfahrung 33)

A.5.20 Testperson 36

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-4.09	-6.35	6.82	-4.23	-19.70	-43.78
Anzahl	-2.00	-1.00	4.00	-1.00	-5.00	-7.00

Tabelle A.60: Übersicht über die Differenzen der Merkphasen von Testperson 36

	Phase	Zeit in s	LS	G	Inhalt
1	M	4.24			

2	S	7.96	-18	0	`Public␣class␣Haus{`
					
					`}`

3	P	4.18			
		7.96 (4.18)	-18	0	
4	M	1.70			

5	S	4.92	-20	0	`Public␣class␣Haus{`
					`private␣int␣number;`
					`}`

6	P	2.11			

7	S	5.54	-17	1	`Public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣Haus;`
					`}`

8	P	2.25			
		10.46 (4.36)	-37	1	
9	M	2.64			
10	P	4.03			

11	S	0.87	-4	0	`Public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe␣Haus;`
					`}`

12	P	6.71			

13	S	0.83	1	5	`Public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe␣Haus;`
					`}`

14	P	2.21			

15	S	0.11	0	1	`Public␣class␣Haus{`
					`private␣int␣numbmer;`
					`private␣String␣farbe;`
					`}`

16	P	2.70			
		1.80 (15.64)	-3	6	
17	M	6.46			

	Phase	Zeit in s	LS	G	Inhalt

18	S	15.98	-35	7	`Public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣Streiche(String␣farbe){`
					`}`
					`}`

19	P	2.14			

20	S	6.75	-17	0	`Public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣Streiche(String␣farbe){`
					`␣␣␣␣this.farbe␣=␣farbe;`
					`}`
					`}`

21	P	3.78			
		22.73 (5.92)	-52	7	
22	M	4.30			
23	P	2.66			

24	S	0.16	-1	1	`Public␣class␣Haus{`
					`private␣int␣nummer;`
					`private␣String␣farbe;`
					`public␣void␣Sstreiche(String␣farbe){`
					`this.farbe␣=␣farbe;`
					`}`
					`}`

		0.16 (2.66)	-1	1	

Tabelle A.61: Transkript zur Bearbeitung 395 von Testperson 36 (Programmiererfahrung 20)

	Phase	Zeit in s	LS	G	Inhalt
1	M	3.28			
2	S	12.40	-28	3	`boolean␣istVielfaches(int){` `}`
3	P	5.38			
		12.40 (5.38)	-28	3	
4	M	1.57			

5	S	4.57	-19	0	`boolean␣istVielfaches(int␣zahl,␣int␣`
					`↪ Vielfaches){`
					`}`

6	P	2.11			
		4.57 (2.11)	-19	0	
7	M	8.06			
8	P	2.79			

	Phase	Zeit in s	LS	G	Inhalt

9	S	6.57	-12	0	`boolean␣istVielfaches(int␣zahl,␣int␣Vielfaches){`
					`if(){`
					`}else{`
					`}`
					`}`

10	P	2.68			
		6.57 (5.47)	-12	0	
11	M	8.70			
12	P	2.38			

13	S	9.71	-18	4	`boolean␣istVielfaches(int␣zahl,␣int␣Vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`}else{`
					`}`
					`}`

14	P	3.98			

15	S	5.06	-13	2	`boolean␣istVielfaches(int␣zahl,␣int␣Vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`return␣true;`
					`}else{`
					`}`
					`}`

16	P	2.58			

17	S	3.90	-12	4	`boolean␣istVielfaches(int␣zahl,␣int␣Vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`return␣true;`
					`}else{`
					`return␣false;`
					`}`
					`}`

18	P	4.45			

19	S	0.07	-1	1	`boolean␣istVielfaches(int␣zahl,␣int␣Vielfaches){`
					`if(vielfaches␣%␣zahl␣==␣0){`
					`return␣true;`
					`}else{`
					`return␣false;`
					`}`
					`}`

		18.74 (13.38)	-44	11	

Tabelle A.62: Transkript zur Bearbeitung 397 von Testperson 36 (Programmiererfahrung 20)

A.5.21 Testperson 37

	Klasse1	Algo1	Klasse2	Algo2	Klasse3	Algo3
Länge	-21.52	-1.62	22.80	3.31	1.72	5.47
Anzahl	-9.00	-1.00	5.00	-4.00	-5.00	-2.00

Tabelle A.63: Übersicht über die Differenzen der Merkphasen von Testperson 37

	Phase	Zeit in s	LS	G	Inhalt
1	M	11.20			
2	S	3.95	-13	0	`public␣class␣`
3	P	2.82			
		3.95 (2.82)	-13	0	
4	M	1.48			
5	S	2.67	-5	0	`public␣class␣Haus{`
6	P	3.66			
		2.67 (3.66)	-5	0	
7	M	2.19			

8	S	13.10	-39	0	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`

		13.10 (0.00)	-39	0	
9	M	4.01			

10	S	13.22	-31	2	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(int␣farbe){`

		13.22 (0.00)	-31	2	
11	M	2.81			

12	S	4.56	-17	0	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(int␣farbe){`
					`this.farbe=farbe;`

13	P	4.72			

	Phase	Zeit in s	LS	G	Inhalt

14	S	1.58	-1	0	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(int␣farbe){`
					`this.farbe=farbe;`
					`}`

15	P	2.10			
		6.14 (6.82)	-18	0	
16	M	0.93			

17	S	2.50	-1	0	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(int␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

18	P	2.06			
		2.50 (2.06)	-1	0	
19	M	1.98			
20	P	1.92			
		0.00 (1.92)	0	0	
21	M	1.90			
22	P	2.00			
		0.00 (2.00)	0	0	
23	M	1.57			
24	P	1.81			
		0.00 (1.81)	0	0	
25	M	2.11			
26	P	2.74			

27	S	1.90	-4	3	`public␣class␣Haus{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(Strintg␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

28	P	3.87			
		1.90 (6.61)	-4	3	
29	M	1.62			

	Phase	Zeit in s	LS	G	Inhalt
30	P	6.57			
		0.00 (6.57)	0	0	
31	M	1.32			

32	S	0.90	-1	0	`public␣class␣Haus␣{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

33	P	2.25			
		0.90 (2.25)	-1	0	
34	M	2.58			
35	P	1.44			
		0.00 (1.44)	0	0	
36	M	3.59			
37	P	3.95			

38	S	1.81	1	1	`public␣class␣Haus␣{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

39	P	3.48			
		1.81 (7.43)	1	1	
40	M	1.19			

41	S	0.00	0	0	`public␣class␣Haus␣{`
					`private␣int␣number;`
					`private␣String␣farbe;`
					
					`public␣void␣streiche(String␣farbe){`
					`this.farbe=farbe;`
					`}`
					`}`

		0.00 (0.00)	0	0	

Tabelle A.64: Transkript zur Bearbeitung 318 von Testperson 37 (Programmiererfahrung 20)

	Phase	Zeit in s	LS	G	Inhalt
1	M	6.54			
2	S	7.75	-27	0	`boolean␣istVielfaches(int␣)`

	Phase	Zeit in s	LS	G	Inhalt
3	P	2.47			
		7.75 (2.47)	-27	0	
4	M	2.18			
5	S	2.63	-9	0	`boolean istVielfaches(int int zahl,)`
6	P	2.78			
7	S	7.97	-11	4	`boolean istVielfaches(int zahl, int z` `↪ vielfachl,es){`
		10.59 (2.78)	-20	4	
8	M	4.68			

9	S	7.24	-17	0	`boolean istVielfaches(int zahl, int`
					`↪ vielfaches){`
					`if(vielfaches%0=)`

10	P	2.59			
		7.24 (2.59)	-17	0	
11	M	1.65			

12	S	2.41	-3	1	`boolean istVielfaches(int zahl, int`
					`↪ vielfaches){`
					`if(vielfaches%0zahl=)`

13	P	2.63			

14	S	3.99	-3	1	`boolean istVielfaches(int zahl, int`
					`↪ vielfaches){`
					`if(vielfaches%zahl==0){`

		6.40 (2.63)	-6	2	
15	M	1.36			

16	S	7.90	-17	0	`boolean istVielfaches(int zahl, int`
					`↪ vielfaches){`
					`if(vielfaches%zahl==0){`
					`return true;`
					`else{`

		7.90 (0.00)	-17	0	
17	M	3.12			
18	P	6.59			

19	S	7.74	-14	1	`boolean istVielfaches(int zahl, int`
					`↪ vielfaches){`
					`if(vielfaches%zahl==0){`
					`return true;`
					`}else{`
					`retunrn false;`
					`}`

20	P	3.18			

	Phase	Zeit in s	LS	G	Inhalt

21	S	0.73	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches){`
					`if(vielfaches%zahl==0){`
					`return␣true;`
					`}else{`
					`retunrn␣false;`
					`}`
					`}`

22	P	5.24			
		8.48 (15.01)	-15	1	
23	M	1.03			
24	P	2.48			

25	S	0.00	-1	0	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches){`
					`if(vielfaches%zahl==0){`
					`return␣true;`
					`}else{`
					`retunrn␣false;`
					`␣␣␣␣}`
					`}`

26	P	4.58			

27	S	0.00	0	1	`boolean␣istVielfaches(int␣zahl,␣int␣vielfaches){`
					`if(vielfaches%zahl==0){`
					`return␣true;`
					`}else{`
					`retunrn␣false;`
					`}`
					`}`

		0.00 (7.06)	-1	1	

Tabelle A.65: Transkript zur Bearbeitung 323 von Testperson 37 (Programmiererfahrung 20)

Inhaltsverzeichnis der CD

masterarbeit-mike-barkmin.pdf
Digitale Version dieser Arbeit

datensatz.csv
Die erhobenen Daten

praesentation.pdf
Die verwendete Präsentation zur Begleitung der Durchführung

transkriptverzeichnis.pdf
Transkripte aller Bearbeitungen der Items Klasse1 und Algo1

commoop.pdf
Beschreibung der Implementierung der Webanwendung COMMOOP

questionnaire.pdf
Beschreibung der Implementierung des Fragebogen Aufgabentyps für die Webanwendung COMMOOP

corsi-blocks.pdf
Beschreibung der Implementierung des Corsi-Block-Tapping-Tests für die Webanwendung COMMOOP

typing-speed.pdf
Beschreibung der Implementierung des Aufgabentyps zur Ermittlung der Tippgeschwindigkeit für die Webanwendung COMMOOP

commoop.zip
Quelltext der Webanwendung COMMOOP. Hinweise zur Installation befinden sich in dieser Datei.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AG	Arbeitsgedächtnis
A	Aufmerksamkeit
BCS	Basic Controll Structures
C.alpha	Cronbachs Alpha
CB-SEM	Covariance-based Structure Equation Modelling
D2-GZ	Gesamtzahl aller bearbeiteten Zeichen
D2-GZ-F	Gesamtzahl aller bearbeiteten Zeichen abzüglich Fehlerrohwert
D2-F	Fehlerrohwert
D2-KL	Konzentrationsleitungswert
D2-SB	Schwankungsbreite
DEV	durschnittlich erfasste Varianz
DG.rho	Dillon-Golsteins rho
DOM	Document Object Model
FRE_{DE}	Flesch-Reading-Ease-DE
F	Fähigkeit
G	Anzahl gelöschter Zeichen
HTML	Hypertext Markup Language
IST-Fig	I-S-T 2000-R figurale Merkfähigkeit
IST-Verb	I-S-T 2000-R verbale Merkfähigkeit
JSON	JavaScript Object Notation
LS	Levenshtein-Distanz
MC	Method Complexity
OOP	Objektorientierte Programmierung
PE	Programmiererfahrung
PE-Beruf	Programmiererfahrung im Vergleich zu einem Berufstätigen

PE-Java	Programmiererfahrung mit der Programmiersprache Java
PE-Kom	Programmiererfahrung im Verlgeich zu den Kommilitonen
PE-OOP	Programmiererfahrung in der objektorientierten Programmierung
PE-Selbst	Selbsteingeschätzte Programmiererfahrung
PLS-SEM	Partial Least Square Structure Equation Modelling
TWCC	Total Weighted Class Complextiy
WCC	Weighted Class Complexity
WS	erste neue Wiener-Sachtextformel

Literaturverzeichnis

Adelson, B. (1981). „Problem solving and the development of abstract categories in programming languages“. In: *Memory & Cognition* 9.4, S. 422–433. ISSN: 1532-5946. DOI: 10.3758/BF03197568.

Amstad, T. (1978). *Wie verständlich sind unsere Zeitungen?* Abhandlung: Philosophische Fakultät I. Zürich. 1977. Studenten-Schreib-Service.

Amthauer, R. u. a. (2006). *Intelligenz-Struktur-Test 2000 R*. Bd. 2. Göttingen: Hogrefe.

Anderson, J. R. (1983). *Cognitive science series. The architecture of cognition*. Cambridge.

– **(2015)**. *Cognitive psychology and its implications*. 8. Aufl. New York: Worth Publishers.

Baddeley, A. (2000). „The episodic buffer: a new component of working memory?“ In: *Trends in Cognitive Sciences* 4.11, S. 417–423. ISSN: 1364-6613.

– **(2003)**. „Working memory: looking back and looking forward“. In: *Nature reviews neuroscience* 4.10, S. 829–839.

– **(2012)**. „Working Memory: Theories, Models, and Controversies“. In: *Annual Review of Psychology* 63.1. PMID: 21961947, S. 1–29. DOI: 10.1146/annurev-psych-120710-100422.

Baddeley, A. und G. Hitch (1974). „Working Memory“. In: Hrsg. von Gordon H. Bower. Bd. 8. Psychology of Learning and Motivation. Academic Press, S. 47–89.

Bamberger, R. (1984). *Lesen - verstehen - lernen - schreiben : die Schwierigkeitsstufen von Texten in deutscher Sprache*. ger. Wien: Jugend und Volk [u.a.] ISBN: 3224152508.

Barfield, W. (1986). „Expert-novice differences for software: implications for problem-solving and knowledge acquisition“. In: *Behaviour & Information Technology* 5.1, S. 15–29. DOI: 10.1080/01449298608914495.

Barnes, D. J. und M. Kölling (2016). *Objects First with Java: A Practical Introduction Using BlueJ*. Pearson Education.

Bateson, A. G., R. A. Alexander und M. D. Murphy (1987). „Cognitive processing differences between novice and expert computer programmers“. In: *International Journal of Man-Machine Studies* 26.6, S. 649–660. ISSN: 0020-7373.

Bortz, J. und N. Döring (2006). *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. 4. Springer Medizin Verlag. ISBN: 3540333053 9783540333050.

Brickenkamp, R. (1967). *Test d2: Aufmerksamkeits-Belastungs-Test*. Bd. 2. Göttingen: Hogrefe.

Brickenkamp, R. (1994). *Test d2: Aufmerksamkeits-Belastungs-Test*. Bd. 8. Göttingen: Hogrefe.

Brunetti, R., C. Del Gatto und F. Delogu (2014). „eCorsi: Implementation and testing of the Corsi block-tapping task for digital tablets“. In: *Frontiers in Psychology* 5. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2014.00939.

Chase, W. G. und H. A. Simon (1973). „Perception in chess“. In: *Cognitive Psychology* 4.1, S. 55–81. ISSN: 0010-0285.

Chin, W. (2010). „How to Write Up and Report PLS Analyses“. In: *Handbook of Partial Least Squares: Concepts, Methods and Applications*. Hrsg. von Vincenzo Esposito Vinzi u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 655–690. ISBN: 978-3-540-32827-8. DOI: 10.1007/978-3-540-32827-8_29.

Claus, V. und A. Schwill (2001). „Duden Informatik“. In: *Fachlexikon für Studium und Praxis. Bibliographisches Institut. Mannheim, Leipzig, Wien, Zürich* 4.

Corsi, P. M. (1972). *Human Memory and the Medial Temporal Region of the Brain*. McGill theses.

De Groot, A. D. (1965). *Thought and Choice in Chess*. Den Haag: Mouton.

Doberkat, E. E. und S. Dißmann (2002). *Einführung in die objektorientierte Programmierung mit Java*. De Gruyter. ISBN: 9783486593822.

Echtle, K. und M. Goedicke (2000). *Lehrbuch der Programmierung mit Java*. dpunkt-Verlag.

Fitts, P. M. und M. Posner (1967). *Human performance*. Brooks/Cole.

Flesch, Rudolph (1948). „A new readability yardstick.“ In: *Journal of Applied Psychology* 32.3, S. 221–233.

Gobet, F. und P. C. R. Lane (2012). „Chunking“. In: *Encyclopedia of the Sciences of Learning*. Hrsg. von N. M. Seel. Boston, MA: Springer US, S. 541–541. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_3398.

Gobet, F. und H. A. Simon (1996a). „Recall of random and distorted chess positions: Implications for the theory of expertise“. In: *Memory & Cognition* 24.4. cited By 109, S. 493–503. DOI: 10.3758/BF03200937.

– **(1996b)**. „Recall of rapidly presented random chess positions is a function of skill“. In: *Psychonomic Bulletin and Review* 3.2. cited By 98, S. 159–163.

– **(1996c)**. „Templates in chess memory: A mechanism for recalling several boards“. In: *Cognitive Psychology* 31.1. cited By 281, S. 1–40. DOI: 10.1006/cogp.1996.0011.

Guerin, B. und A. Matthews (1990). „The Effects of Semantic Complexity on Expert and Novice Computer Program Recall and Comprehension“. In: *The Journal of General Psychology* 117.4. PMID: 28142341, S. 379–389. DOI: 10.1080/00221309.1990.9921144.

Guida, A. u. a. (2012). „How chunks, long-term working memory and templates offer a cognitive explanation for neuroimaging data on expertise acquisition: A two-stage framework“. In: *Brain and Cognition* 79.3, S. 221–244. ISSN: 0278-2626.

Hair, J. F. (2013). *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. SAGE Publications. ISBN: 9781452217444.

Hayes, J. und N. A. Chenoweth (2006). „Is Working Memory Involved in the Transcribing and Editing of Texts?“ In: *Written Communication* 23.2, S. 135–149. DOI: 10.1177/0741088306286283.

Hayes, J. und L. Flowers (1980). „Identifying the organization of writing processes“. In: Hrsg. von L. Gregg und E. Steinberg. Cognitive Processes in Writing. Hillsdale, NJ: Erlbaum, S. 3–30.

Kellogg, R. T. (1996). „A model of working memory in writing.“ In: Torrance, Mark und Gaynor C Jeffery. *The Cognitive demands of writing : processing capacity and working memory in text production*. Amsterdam University Press, S. 57–71.

Kessels, R. u. a. (2000). „The Corsi Block-Tapping Task: Standardization and Normative Data“. In: *Applied Neuropsychology* 7.4, S. 252–258. DOI: 10.1207/S15324826AN0704_8.

Klieme, E. u. a. (2003). *Expertise zur Entwicklung nationaler Bildungsstandards*. URL: https://www.bmbf.de/pub/Bildungsforschung_Band_1.pdf (besucht am 20. 04. 2017).

Kramer, M., P. Hubwieser und T. Brinda (2016). „A Competency Structure Model of Object-Oriented Programming“. In: *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, S. 1–8. DOI: 10.1109/LaTiCE.2016.24.

Kramer, M., D. A. Tobinski und T. Brinda (2016). „On the Way to a Test Instrument for Object-oriented Programming Competencies“. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling '16. Koli, Finland: ACM, S. 145–149. ISBN: 978-1-4503-4770-9. DOI: 10.1145/2999541.2999544.

Levenshtein, Vl (1966). „Binary Codes Capable of Correcting Deletions, Insertions and Reversals“. In: *Soviet Physics Doklady* 10, S. 707.

Magliaro, S. und J. K. Burtin (1987). „Adolescents' Chunking of Computer Programs“. In: *Comput. Sch.* 4.3-4, S. 129–138. ISSN: 0738-0569. DOI: 10.1300/J025v04n03_14.

Miller, G. A. (1956). „The magical number seven, plus or minus two: Some limits on our capacity for processing information.“ In: *Psychological review* 63.2, S. 81.

Misra, S. und A. Adewumi (2015). „Object-Oriented Cognitive Complexity Measures“. In: *Handbook of Research on Innovations in Systems and Software Engineering*. IGI Global, S. 150–170. DOI: 10.4018/978-1-4666-6359-6.ch006.

Misra, S. und K. I. Akman (2008). „Weighted Class Complexity: A Measure of Complexity for Object Oriented System“. In: *Journal of Information Science and Engineering*.

Pickering, S. J. (2001). „Cognitive approaches to the fractionation of visuo-spatial working memory“. In: *Cortex* 37.4, S. 457–473.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Revelle, W. (2016). *psych: Procedures for Psychological, Psychometric, and Personality Research*. R package version 1.6.12. Northwestern University. Evanston, Illinois. URL: https://CRAN.R-project.org/package=psych.

Sala, G. und F. Gobet (2017). „Experts' memory superiority for domain-specific random material generalizes across fields of expertise: A meta-analysis". In: *Memory and Cognition* 45.2. cited By 0, S. 183–193. DOI: 10.3758/s13421-016-0663-2.

Schmidt, A. L. (1986). „Effects of experience and comprehension on reading time and memory for computer programs". In: *International Journal of Man-Machine Studies* 25.4, S. 399–409. ISSN: 0020-7373.

Shao, J. und Y. Wang (2003). „A new measure of software complexity based on cognitive weights". In: *Canadian Journal of Electrical and Computer Engineering* 28.2, S. 69–74. ISSN: 0840-8688. DOI: 10.1109/CJECE.2003.1532511.

Shneiderman, B. (1976). „Exploratory experiments in programmer behavior". In: *International Journal of Computer & Information Sciences* 5.2, S. 123–143. ISSN: 1573-7640. DOI: 10.1007/BF00975629.

Siegmund, J. u. a. (2014). „Measuring and Modeling Programming Experience". In: *Empirical Softw. Engg.* 19.5, S. 1299–1334. ISSN: 1382-3256. DOI: 10.1007/s10664-013-9286-4.

Spencer, L. und D. Snape (2003). „The foundations of qualitative research". In: *Qualitative research practice. A guide for social science students and researchers*. Sage, London, S. 1–25.

Tenenhaus, M. u. a. (2005). „PLS path modeling". In: *Computational Statistics & Data Analysis* 48.1. Partial Least Squares, S. 159–205. ISSN: 0167-9473.

Weinert, F. E. (2001). „Vergleichende Leistungsmessung in Schulen - eine umstrittene Selbstverständlichkeit". In: *Leistungsmessungen in Schulen*. Beltz, S. 17–31.

Wentura, D. und C. Frings (2012). *Kognitive Psychologie*. Basiswissen Psychologie. Springer Fachmedien Wiesbaden. ISBN: 9783531931258.

Selbstständigkeitserklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen und Formulierungen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle als Entlehnung kenntlich gemacht.

_____, den 29. Mai 2017 _____

(Ort) (Unterschrift)