Codebuch

Qualitative Inhaltsanalyse (nach Mayring) von 7 Leitfaden-gestützten Expertinnen-Interviews

07.04.2020

Stand: 07.04.2020

Inhaltsverzeichnis

A. Codesystem 5
1 nicht-kognitive Kompetenzen 9
1.1 Haltung & Einstellung 9
1.1.1 Begeisterung & Spaß am Problemlösen/Informatik 9
1.1.2 Durchhaltevermögen trotz Frust 9
1.1.3 Lernbereitschaft, Offenheit & Flexibilität 9
1.1.4 freiwillige Mitarbeit 9
1.1.5 aktive, regelmäßige Mitarbeit 10
1.1.6 Freude und Stolz über eigene Ergebnisse 10
1.2 Selbstkompetenz 10
1.2.1 Vorkenntnisse durch Schule, Ausbildung, etc. 10
1.2.2 Angemessenes Selbstvertrauen in eigene Fähigkeiten 10
1.2.3 Kreativität 10
1.3 Programmiererfahrung sammeln durch zeitintensives Üben 10
1.4 Sozial-kommunikative Kompetenzen 11
1.4.1 Kooperieren und Kollaborieren 11
1.4.2 Kommunizieren 11
2 Wissensdimensionen 12
2.1 Faktenwissen 12
2.2 Konzeptionelles Wissen 12
2.3 Prozedurales Wissen 12
2.4 Meta-kognitives Wissen 13
3 kognitive Prozessdimensionen 14
3.1 Erinnern 14
3.1.1 Elementare Programmiersprachliche Konstrukte kennen 14
3.1.2 Objektorientierte Programmierung kennen 14
3.1.3 Technologische Grundlagen kennen 14
3.2 Verstehen 14
3.2.1 Konzepte der Programmierung in eigenen Worten erklären 14
3.3 Anwenden 14
3.3.1 Werkzeuge zur Software-Entwicklung nutzen 14
3.3.2 Qualitätskriterien auf Quellcode anwenden 15
3.4 Analysieren 15
3.4.1 gegebene Problemstellungen in einzelne Teile zerlegen 15
3.4.2 Compiler & Interpreter-Meldungen lesen können 15
3.4.3 eigenen Code erklären können 15
3.4.4 fremden Code lesen können 15
3.4.5 Ausgabe von fremdem Code bestimmen 15
3.5 Bewerten 15

3.5.1 Suchen von Fehlern in Programmen ... 15
3.5.2 Angemessenes Vorgehen zur Problemlösung auswählen ... 16
3.5.3 Angemessenheit programmiersprachlicher Lösungen beurteilen ... 16
3.5.4 Testen von Programmen ... 16
3.5.5 Verantwortung übernehmen für Lernerfolg ... 16
3.5.6 Externe Ressourcen zum Lernen nutzen ... 16
3.5.7 Selbstreflexion ... 16
3.6 Erzeugen ... 16
3.6.1 Programmiersprachliche, lauffähige Lösungen für Probleme schreiben ... 16
3.6.2 Logische Ausdrücke und Operatoren zur Problemlösung benutzen ... 17
3.6.3 Software(-projekte) strukturiert entwerfen ... 17
3.6.4 Selbstständig Methoden schreiben ... 17
3.6.5 Einzelne Klassen schreiben ... 17
3.6.6 Modellieren von Problemen und Prozessen ... 17
3.6.7 Vorgegebenen Programmcode erweitern ... 17
3.6.8 Programm mit mehreren Klassen schreiben ... 17
3.6.9 In vorgegebene Methodensignatur Code schreiben ... 17
3.6.10 Selbstständig Code schreiben ... 17
3.6.11 (Programmier-)Sprachkonstrukte korrekt verwenden ... 17
3.6.12 Schleifen schreiben ... 18
3.6.13 Algorithmen mit Hilfe von Entwurfsmustern problemadäquat entwerfen ... 18
3.6.14 Selbstständig Klasse mit Methode schreiben ... 18
3.6.15 Systematik beim Problemlösen entwickeln ... 18
3.6.16 selbstständige Organisation des Lernprozesses ... 18
3.6.17 Abstraktion von Handlungsvorschriften (Rekursion) ... 18
3.6.18 Transfer ... 19
4 Herausforderungen beim Lernen und Lehren von Programmieren ... 20
4.1 Herausforderungen Lehrende ... 20
4.1.1 (Prüfungs-)Aufgaben konzipieren und Bewerten ... 20
4.1.2 Lernatmosphäre in Präsenzveranstaltungen schaffen ... 20
4.1.3 Begrenzte Kapazitäten der Lehrenden für Feedback ... 20
4.1.4 Verhindern von Trittbrettfahrern & Plagiaten ... 21
4.1.5 begrenzte Kompetenzen von Tutoren ... 21
4.1.6 Programmiersprachen-unabhängige Konstrukte identifizieren ... 21
4.1.7 Theorierelevanz in der Programmierung aufzeigen ... 21
4.2 Herausforderungen Lernende ... 21
4.2.1 Informatik-spezifische Strategien erlernen ... 21
4.2.2 Aufeinander aufbauende Inhalte und Fähigkeiten ... 21
4.2.3 Programmieren lernen in endlicher Zeit ... 21
4.2.4 Abschreckende Nerd-Klischees ... 22
4.2.5 Zeitliche Einschränkungen durch Nebenjob ... 22
4.2.6 Angst vor Mathematik und formalen Ausdrücken ... 22

4.2.7 Nichterscheinen in Präsenzveranstaltungen 22
4.2.8 Fach-Englisch für Informatiker 22
4.2.9 kognitive Kompetenzen 22
4.2.10 nicht-kognitive Kompetenzen 23

A. Codesystem

1 nicht-kognitive Kompetenzen	(187)[1]
1.1 Haltung & Einstellung	(89)
1.1.1 Begeisterung & Spaß am Problemlösen/Informatik	24
1.1.2 Durchhaltevermögen trotz Frust	23
1.1.3 Lernbereitschaft, Offenheit & Flexibilität	20
1.1.4 freiwillige Mitarbeit	10
1.1.5 aktive, regelmäßige Mitarbeit	9
1.1.6 Freude und Stolz über eigene Ergebnisse	3
1.2 Selbstkompetenz	(43)
1.2.1 Vorkenntnisse durch Schule, Ausbildung, etc.	23
1.2.2 Angemessenes Selbstvertrauen in eigene Fähigkeiten	14
1.2.3 Kreativität	6
1.3 Programmiererfahrung sammeln durch zeitintensives Üben	38
1.4 Sozial-kommunikative Kompetenzen	(17)
1.4.1 Kooperieren und Kollaborieren	10
1.4.2 Kommunizieren	7
2 Wissensdimensionen	(185)
2.1 Faktenwissen	3
2.2 Konzeptionelles Wissen	2
2.3 Prozedurales Wissen	98
2.4 Meta-kognitives Wissen	82
3 kognitive Prozessdimensionen	(185)
3.1 Erinnern	(5)
3.1.1 Elementare Programmiersprachliche Konstrukte kennen	3
3.1.2 Objektorientierte Programmierung kennen	1
3.1.3 Technologische Grundlagen kennen	1
3.2 Verstehen	(2)
3.2.1 Konzepte der Programmierung in eigenen Worten erklären	2
3.3 Anwenden	(8)
3.3.1 Werkzeuge zur Software-Entwicklung nutzen	5
3.3.2 Qualitätskriterien auf Quellcode anwenden	3
3.4 Analysieren	(22)
3.4.1 gegebene Problemstellungen in einzelne Teile zerlegen	11

[1] Werden die Häufigkeiten in runden Klammern angegeben, wird lediglich die Summe der Codes aller Subkategorien angezeigt.

3.4.2 Compiler & Interpreter-Meldungen lesen können	5
3.4.3 eigenen Code erklären können	4
3.4.4 fremden Code lesen können	1
3.4.5 Ausgabe von fremdem Code bestimmen	1
3.5 Bewerten	(34)
3.5.1 Suchen von Fehlern in Programmen	4
3.5.2 Angemessenes Vorgehen zur Problemlösung auswählen	2
3.5.3 Angemessenheit programmiersprachlicher Lösungen beurteilen	2
3.5.4 Testen von Programmen	1
3.5.5 Verantwortung übernehmen für Lernerfolg	10
3.5.6 Externe Ressourcen zum Lernen nutzen	9
3.5.7 Selbstreflexion	6
3.6 Erzeugen	(114)
3.6.1 Programmiersprachliche, lauffähige Lösungen für Probleme schreiben	17
3.6.2 Logische Ausdrücke und Operatoren zur Problemlösung benutzen	9
3.6.3 Software(-projekte) strukturiert entwerfen	7
3.6.4 Selbstständig Methoden schreiben	4
3.6.5 Einzelne Klassen schreiben	3
3.6.6 Modellieren von Problemen und Prozessen	3
3.6.7 Vorgegebenen Programmcode erweitern	3
3.6.8 Programm mit mehreren Klassen schreiben	2
3.6.9 In vorgegebene Methodensignatur Code schreiben	2
3.6.10 Selbstständig Code schreiben	2
3.6.11 (Programmier-)Sprachkonstrukte korrekt verwenden	2
3.6.12 Schleifen schreiben	1
3.6.13 Algorithmen mit Hilfe von Entwurfsmustern problemadäquat entwerfen	1
3.6.14 Selbstständig Klasse mit Methode schreiben	1
3.6.15 Systematik beim Problemlösen entwickeln	26
3.6.16 selbstständige Organisation des Lernprozesses	18
3.6.17 Abstraktion von Handlungsvorschriften (Rekursion)	10
3.6.18 Transfer	3
4 Herausforderungen beim Lernen und Lehren von Programmieren	(79)
4.1 Herausforderungen Lehrende	(47)
4.1.1 (Prüfungs-)Aufgaben konzipieren und Bewerten	(26)
4.1.1.1 Menschliche Hilfe in Lernsystem nachbilden	1
4.1.1.2 Angemessene Prüfungsformate auswählen	6

4.1.1.3 Entwicklung angemessener, bewertbarer Aufgaben	19
4.1.2 Lernatmosphäre in Präsenzveranstaltungen schaffen	5
4.1.3 Begrenzte Kapazitäten der Lehrenden für Feedback	4
4.1.4 Verhindern von Trittbrettfahrern & Plagiaten	3
4.1.5 begrenzte Kompetenzen von Tutoren	3
4.1.6 Programmiersprachen-unabhängige Konstrukte identifizieren	3
4.1.7 Theorierelevanz in der Programmierung aufzeigen	3
4.2 Herausforderungen Lernende	(32)
4.2.1 Informatik-spezifische Strategien erlernen	10
4.2.2 Aufeinander aufbauende Inhalte und Fähigkeiten	8
4.2.3 Programmieren lernen in endlicher Zeit	6
4.2.4 Abschreckende Nerd-Klischees	3
4.2.5 Zeitliche Einschränkungen durch Nebenjob	2
4.2.6 Angst vor Mathematik und formalen Ausdrücken	1
4.2.7 Nichterscheinen in Präsenzveranstaltungen	1
4.2.8 Fach-Englisch für Informatiker	1
4.2.9 kognitive Kompetenzen	0[2]
4.2.9.1 Systematik beim Problemlösen entwickeln	0
4.2.9.2 Abstraktion von Handlungsvorschriften	0
4.2.9.3 Transfer	0
4.2.9.4 Logische Ausdrücke korrekt benutzen	0
4.2.9.5 Werkzeuge zur Software-Entwicklung nutzen	0
4.2.9.6 Programme schreiben zur Problemlösung	0
4.2.9.7 Externe Ressourcen zum Lernen nutzen	0
4.2.9.8 Selbständige Organisation des Lernprozesses	0
4.2.9.9 Konzepte der Programmierung in eigenen Worten erklären	0
4.2.10 nicht-kognitive Kompetenzen	0
4.2.10.1 Sozial-kommunikative Fähigkeiten	0
4.2.10.1.1 mangelnde Kooperation & Kollaboration	0
4.2.10.1.2 mangelnde Kommunikation	0
4.2.10.2 Haltung & Einstellung	0
4.2.10.2.1 fehlendes Durchhaltevermögen bei Frust	0
4.2.10.2.2 fehlende Begeisterung & Spaß am Problemlösen/Informatik	0
4.2.10.2.3 fehlende Lernbereitschaft/Offenheit	0

[2] Um eine Dreifachkodierung im Material zu vermeiden wurden lediglich Verweise auf die bereits definierten Kategorien angelegt. Daher werden kein zweites Mal Häufigkeiten angeführt.

4.2.10.2.4 fehlende aktive, regelmäßige Mitarbeit	0
4.2.10.2.5 fehlende freiwillige Mitarbeit	0
4.2.10.3 Programmiererfahrung fehlt durch zu wenig zeitintensives Üben	0
4.2.10.4 Selbstkompetenz	0
4.2.10.4.1 fehlendes Selbstvertrauen in eigene Fähigkeiten	0
4.2.10.4.2 fehlende Kreativität	0
4.2.10.4.3 fehlende Vorkenntnisse	0

1 nicht-kognitive Kompetenzen

Unter dieser Kategorie werden nicht-kognitive Kompetenzen zusammengefasst, die sich auf Haltung und Einstellung der Studierenden, ihre Selbstkompetenzen, sozial-kommunikativen Kompetenzen sowie das Sammeln von Programmiererfahrung beziehen.

1.1 Haltung & Einstellung

Unter dieser übergeordneten Kategorie werden verschiedene nicht-kognitive Kompetenzen zusammengefasst, die sich auf die Haltung und Einstellung der Studierenden beziehen. Darunter werden Interessen, Bereitschaft zu Lernhandlungen sowie volitionale und motivationale Aspekte zusammengefasst.

1.1.1 Begeisterung & Spaß am Problemlösen/Informatik

Die Studierenden haben Interesse an Informatik-Themen, Mathematik, Sci-Fi, oder anderen „Nerd"-Aktivitäten und Hobbies. Sie haben Spaß am Programmieren, am Ausprobieren und Knobeln (oder auch nicht, was sich ungünstig auswirken kann). Nichtsdestotrotz führt die Freude und das Interesse sowie geduldiges Probieren allein nicht automatisch zum Studienerfolg. (Die Lernmotivation allein kann zwar geringere kognitive Fähigkeiten bei einfacheren Aufgaben kompensieren, bei schwierigeren Aufgaben jedoch besteht diese Kopplung von kognitiven Kompetenzen und motivationalen Einstellungen nicht mehr. Dementsprechend führt eine hohe Lernmotivation nicht automatisch zu Lernerfolgen und der Bewältigung kognitiv anspruchsvoller Aufgabe (vgl. Hasselhorn & Gold 2009, S. 102)[3].

1.1.2 Durchhaltevermögen trotz Frust

Die Studierenden sind dazu in der Lage, bei schwierigen Aufgaben weiter an Problemlösungen zu arbeiten, auch wenn diese nicht sofort gelöst werden können. Trotz zeitintensivem Anspruch, Frustration beim Problemlösen durch kontinuierliches negatives Feedback durch Compiler und Interpreter und fehlende Erfolgserlebnisse arbeiten Studierende kontinuierlich weiter an ihren Programmen, bis sich erste Erfolge einstellen. Da sich diese erst über einen längeren Zeitraum einstellen, benötigen Studierende Durchhaltevermögen und Ehrgeiz, um diesen Zeitpunkt erreichen zu können, sodass sie (den Programmierkurs, respektive das Studium) nicht abbrechen.

1.1.3 Lernbereitschaft, Offenheit & Flexibilität

Die Studierenden sind dazu bereit, neue Konzepte wahrzunehmen und kognitiv zu verarbeiten, auch wenn diese schwierig erscheinen. Sie sind offen, Neues zu lernen und auszuprobieren. Auch die Bereitschaft, neue bzw. individuelle Strategien auszuprobieren wird hierunter eingeordnet, da beim Programmieren Flexibilität in der Herangehensweise an Probleme erforderlich ist, um adäquate Entscheidungen treffen zu können. (Der Subcode wird auch dann vergeben, wenn diese Bereitschaft nicht vorhanden ist, wenn Studierende zum Beispiel introvertiert sind, oder sich aus Angst keinen neuen und fremd aussehenden Themen annähern wollen.)

1.1.4 freiwillige Mitarbeit

Die Studierenden sind freiwillig dazu bereit, mitzuarbeiten (zu Hause und in Präsenzphasen) und gegebene Aufgabenstellungen zu bearbeiten, um bestmögliche Lernfortschritte erzielen zu können.

[3] Hasselhorn, M., & Gold, A. (2009). Pädagogische Psychologie: Erfolgreiches Lernen und Lehren (2. Auflage). Stuttgart: Kohlhammer.

1.1.5 aktive, regelmäßige Mitarbeit

Die Studierenden sind dazu in der Lage, aktiv und regelmäßig mitzuarbeiten. Die regelmäßige Mitarbeit bezieht sich unter anderem darauf, dass kontinuierlich, während des gesamten Semesterverlaufs mitgearbeitet wird, da die Grundlagen der Programmierung nicht innerhalb von zwei Wochen vor einer Prüfung erlernt werden können. (Der Subcode wird auch dann vergeben, wenn diese aktive Mitarbeit fehlt, wenn beispielsweise typische Vermeidungsstrategien in der Übung an den Tag gelegt werden, z. Bsp. Studierende erscheinen in Gruppen, oder zu spät, gehen früher oder möchten nicht über ihren Code sprechen.)

1.1.6 Freude und Stolz über eigene Ergebnisse

Die Studierenden erleben ihre eigene Kompetenz indem sie funktionierende Programme entwickelnd. Dieses Erfolgserlebnis äußert sich anhand von Freude und Stolz über das produzierte Ergebnis und fördert Motivation sowie Durchhaltevermögen im weiteren Lernprozess der Programmierausbildung.

1.2 Selbstkompetenz

In dieser Kategorie werden verschiedene nicht-kognitive Kompetenzen zusammengefasst, die sich auf Selbstkompetenzen beziehen, sprich individuelle Voraussetzungen des Selbsts berücksichtigen.

1.2.1 Vorkenntnisse durch Schule, Ausbildung, etc.

Die Studierenden sind dazu in der Lage, unabhängig von ihren Vorkenntnissen Programmieren lernen. Das unterschiedliche Vorwissen der Studierenden und ganzer Kohorten in allen Varianten wird in dieser Kategorie subsummiert. Durch die unterschiedlichen Vorkenntnisse aus Schule, Ausbildung, oder (Praxis-)Erfahrungen sind die Leistungsunterschiede sehr groß. Manche Studierende kommen gänzlich ohne Vorkenntnisse nach dem Abitur zum Studium. Nichtsdestotrotz kann das fehlende Vorwissen durch Erfahrung, die im Studium gesammelt wird, kompensiert werden. In dieser Kategorie werden die verschiedenen Wissens- und Erfahrungsstände zusammengefasst.

1.2.2 Angemessenes Selbstvertrauen in eigene Fähigkeiten

Studierende sind sich ihrer eigenen (akademischen und programmiertechnischen) Fähigkeiten bewusst und vertrauen darauf, neue Aufgabenstellungen lösen zu können, auch wenn sie anfangs noch nicht alle Kenntnisse und Fähigkeiten entwickelt haben.

1.2.3 Kreativität

Die Studierende sind dazu in der Lage, Kreativität ergänzend zu Intuition, ablauforientiertem Denken und Systematik (sprich ergänzend zu kognitiven und metakognitiven Kompetenzen) in die Entwicklung von Problemlösungen einzubringen, um flexibel neue, individuelle Lösungswege entwickeln zu können.

1.3 Programmiererfahrung sammeln durch zeitintensives Üben

Die Studierenden sammeln darüber hinaus Programmiererfahrung durch zeitintensives Üben. Mit dieser Kategorie wird die Anwendung von erlerntem Wissen und Fähigkeiten (sprich kognitiver Kompetenzen) beschrieben. Erst durch zeitintensives Üben lernt man viele verschiedene Probleme und deren Lösungen kennen. Sind all diese Probleme hinreichend oft durchgearbeitet worden, können bei neuen Problemen Rückschlüsse auf vorherige Problemlösungen gezogen werden und diese Erfahrungen für den Lösungsansatz des aktuell vorliegenden Problems nutzen. Erfahrung generiert damit neues Wissen auf kognitiver Ebene und unterstützt die Sensitivität für die Möglichkeiten kognitiver Aktivitäten (vgl. Hasselhorn & Gold 2009, S. 95) was mitunter zum Erwerb von meta-kognitiven Kompetenzen beiträgt.

1.4 Sozial-kommunikative Kompetenzen

Unter dieser übergeordneten Kategorie werden *Kommunizieren sowie Kooperieren und kollaborieren* zusammengefasst.

1.4.1 Kooperieren und Kollaborieren

Die Studierenden sind dazu in der Lage, im Team gemeinsam an (Software-)Projekten und Problemen zu arbeiten, um gemeinsam Lösungen zu erzielen. Das gemeinsame Arbeiten steht hierbei im Vordergrund.

1.4.2 Kommunizieren

Die Studierenden sind dazu in der Lage, Diskussionen in der Gruppe zu führen, sich gruppenweise auszutauschen, in Gruppen Präsentationen und zu erstellen und diese vorzutragen. Die erfolgreiche, angemessene und respektvolle Kommunikation (mit Mitstudierenden, Lehrenden und Tutoren) und Verbalisierung eigens entwickelter Ergebnisse steht im Vordergrund.

2 Wissensdimensionen[4]

Die Wissensdimensionen nach der Taxonomie von Anderson und Krathwohl (2001) unterscheiden zwischen den folgenden vier Dimensionen: *Faktenwissen*, *Konzeptionelles Wissen*, *Prozedurales Wissen* und *Meta-kognitives Wissen*. Die im Sample vorgefunden kognitiven Lehr- und Lernziele bezogen auf die für das Programmieren erforderlichen Kenntnisse und Fähigkeiten werden in diese Dimensionen eingeordnet. Damit wird gleichzeitig die AKT-Matrix für die Informatik, speziell die Programmierung, neu interpretiert. Im Folgenden werden die Definitionen der Kategorien mitsamt Subtypen nach Anderson und Krathwohl zusammengefasst.

2.1 Faktenwissen

Faktenwissen (WAS): Inhalte und deren Details in der Tiefe kennen, ganze Zusammenhänge, Aussagen und Inhalte sollen von Lernenden „erzeugt" im Sinne von „rezipiert" werden können. Mit dieser Ebene wird vor allem die Reproduktion von Fakten, oder das Wissen um isolierte Begrifflichkeiten beschrieben. Das Abstraktionsniveau ist sehr gering, wie die beiden Subkategorien mit Beispielen in Klammern zeigen:

a. Wissen um Terminologie (Vokabular, Wortschatz, Begriffe, Symbole, Zeichen wiedergeben)
b. Wissen um spezifische Details und Elemente (genaue Zahlen, Daten, Orte, Ressourcen benennen) (vgl. Anderson & Krathwohl 2001, S. 39-48).

2.2 Konzeptionelles Wissen

Konzeptionelles Wissen (WAS): Wissen zu übergeordneten Ideen und Konzepten steht im Fokus. Zusammenhänge zwischen Inhalten und deren Verbindung zu allgemeineren Konzepten, Prinzipien, Modellen und Theorien sollen klar werden, indem mentale Modelle oder Schemata gebildet werden. Die Kategorie beschreibt damit komplexeres Wissen, das in irgendeiner Form systematisiert und organisiert vorliegt, und mit anderen Disziplinen in Verbindung gebracht werden kann. Der Transfer auf andere Kontexte und/oder Probleme wird durch konzeptionelles Wissen möglich, da das Wissen tiefer durchdrungen und organisiert wird. Dabei werden drei Subkategorien unterschieden:

a. Wissen um Klassifikationen und Kategorien (Typologien, Epochen, Wortarten)
b. Wissen um Prinzipien und Verallgemeinerungen (Gesetzmäßigkeiten der Natur, Rechenprinzipien, Rahmenverträge und deren Auswirkungen)
c. Wissen um Theorien, Modelle und Strukturen (Wissen über Zusammenhänge zw. chem. Prinzipien, Gesamtstrukturen und Institutionen, Evolutionstheorie, Plattentektonik, DNA, usw.) (vgl. Anderson & Krathwohl 2001, S. 39-52).

2.3 Prozedurales Wissen

Prozedurales Wissen (WIE): Inhalte werden lediglich als Ausgangspunkt genutzt, um Strategien, Herangehensweisen, Prozesse, Abläufe und Methoden zu vermitteln. Damit sind Verfahrenskenntnisse gemeint, die beispielsweise für Vergleiche genutzt werden können. Das Wissen um das Wie steht im Mittelpunkt, wie die folgenden Subkategorien mit Beispielen zeigen:

a. Wissen um fachspezifische Strategien und Algorithmen (Problemlösestrategien für Rechenaufgaben, Divisionsregeln, Abläufe und Techniken im Sport, Zeichentechniken)
b. Wissen um fachspezifische Techniken und Methoden (Forschungsmethoden, Interviewtechniken, Verfahren mit offeneren, weniger streng bestimmten Ergebnissen)

[4] Anderson, L. W., & Krathwohl, D. (2001). A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. New York: Addison Wesley Longman.

c. Wissen um Kriterien zur Bestimmung der Eignung von Verfahren (Kriterien zur Anwendung von Gesetzen, Rechenoperationen, Textgenres, Zeichnungen) (vgl. Anderson & Krathwohl 2001, S. 40-55).

2.4 Meta-kognitives Wissen

Meta-kognitives Wissen (Wissen ÜBER Wissen): Lernende sollen Inhalte als Ausgangspunkt nutzen, um Methoden, Verfahren und Strategien zum Verstehen, Analysieren und Erkennen zu entwickeln. Dabei sollen diese Strategien nicht stupide angewendet, sondern reflektiert und metakognitiv betrachtet werden. Dazu gehört das infrage stellen der Verfahren, das Lernen aus Fehlern, das Planen und Regulieren von Lehr-/Lernhandlungen. Weiterhin ist das Bewusstsein bzw. Wissen über Kognition im Allgemeinen sowie der gegenüber der eigenen Erkenntnis zentrales Element. Nicht zuletzt soll die metakognitive Kategorie Selbstreflexion, Kontroll- und Selbststeuerungsprozesse abbilden, um die Beziehung zwischen Kognition, dem Individuum und verschiedenen Fachdisziplinen zu verdeutlichen:

a. Strategisches Wissen (Wissen um Strategien zum Lernen, z. Bsp. Mnemotechniken, Mindmaps, Diagramme oder das Schreiben von Textzusammenfassungen)
b. Wissen über kognitive Aufgaben (Wissen um die Bedeutung und den kognitiven Anspruch verschiedener Strategien zum Lernen, z.Bsp. ist das Fachbuch schwerer zu verstehen als das Sachbuch für Einsteiger).
c. Selbstkenntnis (Wissen um eigene Interessen, Performanz, Ziele, Stärken und Schwächen) (vgl. Anderson & Krathwohl 2001, S. 40-60).

3 kognitive Prozessdimensionen

Die Bloom'sche Taxonomie beschreibt eine Klassifikation von kognitiven Lehr- und Lernzielen und untergliedert die Tiefe der Wissensdurchdringung dabei in sechs Stufen. Eine aktualisierte, im Original englische Fassung der Taxonomie von Anderson und Krathwohl (2001) wird hier angewendet und besteht aus den revidierten kognitiven Prozessdimensionen *Remembering*, *Understanding*, *Applying*, *Analysing*, *Evaluating* und *Creating* (vgl. Anderson & Krathwohl 2001).

3.1 Erinnern

Die im englischen Original *Remembering* (dt. *Erinnern*) genannte Dimension beschreibt das Erinnern, Wiederholen, sprich Verbalisieren oder Aufsagen von zuvor gesehenen oder gehörten Informationen, oder von Wissen aus dem Gedächtnis.

3.1.1 Elementare Programmiersprachliche Konstrukte kennen

Die Studierenden kennen programmiersprachliche Konstrukte (z. Bsp. Lexikon einer Programmiersprache, einfache Sprachelemente, Operatoren, Ausdrücke, Literale). Studierende kennen außerdem die Ausnahmen in einzelnen Programmiersprachen, bei denen z. Bsp. Abweichungen von einheitlichen Ausdrücken genutzt werden.

3.1.2 Objektorientierte Programmierung kennen

Die Studierenden kennen Objektorientierte Programmierung, deren Techniken, Klassen, und die dahinterliegende Philosophie.

3.1.3 Technologische Grundlagen kennen

Die Studierenden erinnern sich an technologische Grundlagen im Sinne von Methoden und Strategien aus den Themenbereichen Datenbanken, Netzwerke, Kommunikation, Visualisierung, Internet, User-Interfaces, HCI, etc.

3.2 Verstehen

Understanding bzw. *Verstehen* als kognitive Prozessdimension beschreibt das Erklären von Sachverhalten und deren Bedeutung in eigenen Worten anhand von Interpretationen, Beispiele, Klassifizierung, Zusammenfassen, Vergleiche, oder ähnliche Methoden.

3.2.1 Konzepte der Programmierung in eigenen Worten erklären

Studierende sind dazu in der Lage, theoretische Konzepte der Programmierung bzw. Bestandteile ihrer eigenen Programme verbal und in ihren eigenen Worten korrekt auszudrücken (sprich von einem Lehrbuchtext in eigene Worte übersetzen).

3.3 Anwenden

Die Kategorie *Applying* (dt. *Anwenden*) beschreibt die Ausführung, Anwendung oder Implementierung von Prozessen, Vorgängen oder Strategien.

3.3.1 Werkzeuge zur Software-Entwicklung nutzen

Die Studierende können bestehende Werkzeuge zur Softwareentwicklung anwenden und deren Funktionen sinnvoll nutzen. Dazu gehören zum Beispiel Entwicklungsumgebungen, Compiler, etc.

3.3.2 Qualitätskriterien auf Quellcode anwenden

Die Studierenden sind dazu in der Lage, lesbaren, wartbaren, wiederverwendbaren Quellcode zu schreiben. Darüber hinaus halten Sie sich an Programmierkonventionen und Konventionen zur Dateiorganisation. Qualitätskriterien für Quellcode umfassen weiterhin Portabilität, Nachhaltigkeit, Wartbarkeit von Code bzw. Programmen. Benutzerfreundlichkeit und Effizient werden außerdem berücksichtigt.

3.4 Analysieren

In der kognitiven Prozessdimension *Analyzing* (dt. *Analysieren*) wird die Zerlegung in Teile und deren logische Erklärung durch Klarlegung von Beziehungen, deren Bedeutung im Gesamtkontext, eine (Re-)Organisation oder Charakterisierung eingeordnet.

3.4.1 gegebene Problemstellungen in einzelne Teile zerlegen

Die Studierenden können komplexe Probleme dahingehend analysieren, dass Sie diese als lösbare, respektive unlösbare Aufgabenstelllungen einordnen (und daraufhin lösen) können. Probleme werden dazu in (ggf.) lösbare Teilprobleme untergliedert.

3.4.2 Compiler & Interpreter-Meldungen lesen können

Die Studierenden können die Fehlerausgabe und -meldungen von Compiler und Interpreter analysieren, und dabei wichtige von unwichtigen Informationen unterscheiden können, um korrekte Handlungsweisen daraus ableiten zu können. Die Studierenden erkennen dabei die Struktur und Zusammenhänge zwischen den Meldungen und des eigenen Programmcodes. (Lesen setzt hierbei setzt voraus, dass Nutzerinnen die jeweiligen Sprachkonstrukte wie auch Fehlermeldungen sehr gut kennen und daraus Bedeutung ableiten können.)

3.4.3 eigenen Code erklären können

Die Studierenden sind dazu in der Lage, die inneren Zusammenhänge einzelner Handlungsvorschriften in eigenem Programmcode beschreiben und erklären zu können, sodass klar wird, welche Anweisungen die einzelnen Zeilen ausführen und was sie Gesamtkontext des Programms bewirken.

3.4.4 fremden Code lesen können

Die Studierenden sind dazu in der Lage, die inneren Zusammenhänge einzelner Handlungsvorschriften in fremdem Programmcode beschreiben und erklären zu können, sodass klar wird, welche Anweisungen die einzelnen Zeilen ausführen und was sie Gesamtkontext des Programms bewirken.

3.4.5 Ausgabe von fremdem Code bestimmen

Die Studierenden sind dazu in der Lage, die inneren Zusammenhänge einzelner Handlungsvorschriften in fremdem Programmcode dahingehend nachzuvollziehen, dass sie dessen Ausgabe bestimmen können.

3.5 Bewerten

Der Dimension *Evaluating* (dt. *Bewerten*) wird die begründete Auswahl aus mehreren Alternativen durch Überprüfung von Kriterien und Standards zugeordnet.

3.5.1 Suchen von Fehlern in Programmen

Die Studierenden sind dazu in der Lage, Programme systematisch nach Fehlern zu durchsuchen und dementsprechend zu testen.

3.5.2 Angemessenes Vorgehen zur Problemlösung auswählen

Die Studierenden sind dazu in der Lage, über ein angemessenes Vorgehen zur Problemlösung zu entscheiden, um schrittweise eine geeignete Problemlösung zu entwickeln.

3.5.3 Angemessenheit programmiersprachlicher Lösungen beurteilen

Die Studierenden sind dazu in der Lage, angemessene programmiersprachliche Lösungen zu entwickeln und die Angemessenheit ihrer eigenen Lösungen für ein Problem kritisch beurteilen zu können. Studierende können dementsprechend eine problemangemessene Programmiersprache und ein angemessenes Paradigma auswählen.

3.5.4 Testen von Programmen

Die Studierenden sind dazu in der Lage, Algorithmen und Programme auf Fehler hin zu testen. Dabei werden ggf. Werkzeuge und Methoden angewendet, deren Ergebnisse und jeweilige Tragweite im letzten Schritt durch ein Urteil bewertet bzw. validiert werden muss.

3.5.5 Verantwortung übernehmen für Lernerfolg

Die Studierenden sind dazu in der Lage, Verantwortung für Ihre eigenen Lernprozesse zu übernehmen, indem Sie zum Beispiel selbst aktiv werden im Selbststudium. Dieses lernstrategische Verhalten wird sichtbar durch die selbstständige Erarbeitung von Lösungen, das Stellen vieler Fragen am Anfang oder bei Problemen, oder durch das eigenständige Handeln und Nutzen von etwa zusätzlichen Ressourcen, Literatur, Fachbüchern. etc. (Durch das Überwachen, Beurteilen und Korrigieren von Lernstrategien während des Selbststudiums wird die Übernahme der Verantwortung über die eigenen Lernprozesse operationalisiert.)

3.5.6 Externe Ressourcen zum Lernen nutzen

Die Studierenden sind dazu in der Lage, adäquate Lernstrategien wie etwa die Nutzung externer Ressourcen für Ihre Lernprozesse auszuwählen und anzuwenden. Diese metakognitive Lernhandlung resultiert aus der reflektierten Einschätzung und Beurteilung des eigenen kognitiven Zustands und der daraus abgeleiteten Bedarfe an weiteren Ressourcen zum Lernen.

3.5.7 Selbstreflexion

Studierende sind dazu in der Lage, ihre eigenen Kenntnisse und Fähigkeiten, Stärken und Schwächen sowie Grenzen realistisch wahrzunehmen und einzuschätzen, um daraus im nächsten Schritt die nötigen Lernhandlungen und -strategien abzuleiten.

3.6 Erzeugen

Die kognitive Prozessdimension *Create* (dt. *Erschaffen*) beschreibt die Zusammenstellung von Elementen oder Einzelteilen zu einem neuen, funktionierenden Ganzen durch Planung, Konzeption oder Produktion.

3.6.1 Programmiersprachliche, lauffähige Lösungen für Probleme schreiben

Die Studierenden sind dazu in der Lage, Probleme zu lösen, indem sie geeignete, lauffähige programmiersprachliche Lösungen schreiben (=Programme schreiben).
(Diese Kategorie wird vor allem dann vergeben, wenn eine konkrete Programmiersprache benannt wird, bzw. wenn von der programmiersprachlichen Umsetzung (in einem bestimmten Paradigma) die Rede ist, d.h. aus Algorithmen Programme zu schreiben gemeint ist.)

3.6.2 Logische Ausdrücke und Operatoren zur Problemlösung benutzen

Die Studierenden sind dazu in der Lage, logische Ausdrücke, Operatoren, Kontrollstrukturen, if-Abfragen, Wahrheitswerten, etc. einzusetzen um logische Zusammenhänge in programmiersprachlichen Ausdrücken zur Problemlösung einzusetzen.
(Die umgangssprachlich verwendeten Beschreibungen des logischen oder analytischen Denkens werden in dieser Kategorie zusammengefasst.)

3.6.3 Software(-projekte) strukturiert entwerfen

Die Studierenden sind dazu in der Lage, Software oder mittelgroße Programme strukturiert zu entwerfen, indem Sie ihr Softwareprojekt entsprechend untergliedern.

3.6.4 Selbstständig Methoden schreiben

Die Studierenden sind dazu in der Lage, selbstständig Methoden programmiersprachlich zu schreiben (mit korrekter Signatur, Parametern und Rückgabewerten).

3.6.5 Einzelne Klassen schreiben

Die Studierenden sind dazu in der Lage, selbstständig einzelne Klassen programmiersprachlich zu schreiben.

3.6.6 Modellieren von Problemen und Prozessen

Die Studierenden sind dazu in der Lage, Probleme und Programme in beispielsweise der Standardnotation UML in Modellen abzubilden. In den Modellen wird jeweils eine Organisation bzw. Struktur in formalisierter Art und Weise abgebildet, sodass Studierende ein eigenständiges, neues Produkt konstruieren.

3.6.7 Vorgegebenen Programmcode erweitern

Die Studierenden können Ihnen vorgegebenen programmiersprachlichen Code gemäß gemachter Vorgaben erweitern oder anpassen (sprich re-strukturieren, neu planen und konstruieren), sodass festgelegte Zielvorgaben erreicht werden.

3.6.8 Programm mit mehreren Klassen schreiben

Die Studierenden sind dazu in der Lage, ein kleines Projekt mit mehreren Klassen zu planen und programmiersprachlich zu produzieren, sodass mehrere Algorithmen zusammenwirken.

3.6.9 In vorgegebene Methodensignatur Code schreiben

Die Studierenden sind dazu in der Lage, zu einer vorgegebenen Methode sinnvollen programmiersprachlichen Code zu ergänzen um zu einer Problemlösung zu gelangen.

3.6.10 Selbstständig Code schreiben

Die Studierenden sind dazu in der Lage, eigenständig und kreativ programmiersprachlichen Code zu planen und zu schreiben, ohne detaillierte Vorgaben zum Vorgehen und Entwurf zu erhalten.

3.6.11 (Programmier-)Sprachkonstrukte korrekt verwenden

Die Studierenden sind dazu in der Lage, Sprachkonstrukte von Programmiersprachen korrekt und aktiv beim Schreiben von Programmen anzuwenden und dadurch Probleme zu lösen. Die Programmiersprache und ihre einzelnen Sprachkonstrukte selbst sind hierbei als Methode zu begreifen, die zur Planung und Konstruktion von Lösungen beitragen.

3.6.12 Schleifen schreiben

Die Studierenden sind dazu in der Lage, Schleifen in einer konkreten Programmiersprache zu schreiben um damit dadurch Problemlösung zu konstruieren.

3.6.13 Algorithmen mit Hilfe von Entwurfsmustern problemadäquat entwerfen

Die Studierenden sind dazu in der Lage, bekannte Entwurfsmuster für Algorithmen (Greedy, Divide-and-Conquer, sowie objektorientierte Entwurfskonzepte) problemadäquat anzuwenden, um algorithmische Lösungen planen und konstruieren zu können.

3.6.14 Selbstständig Klasse mit Methode schreiben

Die Studierenden sind dazu in der Lage, selbstständig eine Klasse mit dazugehöriger Methode zu schreiben (um ein Problem zu lösen).

3.6.15 Systematik beim Problemlösen entwickeln

Die Studierenden sind dazu in der Lage, Probleme systematisch zu adressieren, indem sie eine Strategie zum Problemlösen planen und entwickeln. Die Strategie ist den Lernenden bewusst, mindestens aber intuitiv klar. Daher wird die konstruierte, eigene Systematik der meta-kognitiven Wissensdimension zugeordnet.
(Diese „ablauforientierte Herangehensweise" kann unter Umständen seit der frühen Kindheit spielerisch erlernt worden sein. In der Programmierausbildung sollte die Ablauf-Orientierung in der Systematik bei der Problemlösung münden und dahingehend weiterentwickelt werden, zur Programmierexpertin zu werden. Dazu müssen Studierende über Wissen über das eigene kognitive System verfügen und Lernprozesse selbstständig planen sowie in einem nächsten Schritt überwachen und regulieren.)

3.6.16 selbstständige Organisation des Lernprozesses

Die Studierenden sind dazu in der Lage, ihre eigenen Lernprozesse selbstständig zu planen, zu überwachen und zu steuern und damit ihren Lernprozess zu organisieren. Sie nutzen dazu ihr Wissen über ihre kognitiven Funktionen und das Bewusstsein über die eigenen kognitiven Fähigkeiten, Empfindungen und Zustände, um Lernprozesse dementsprechend erfolgreich zu organisieren. Die Studierenden nutzen ihr Wissen über das eigene kognitive System und erweitern es (im Sinne von „Jedes Tun ist Erkennen, und jedes Erkennen ist Tun" - Maturana & Varela 1987, S. 31)[5].

3.6.17 Abstraktion von Handlungsvorschriften (Rekursion)

Die Studierenden sind dazu in der Lage, das Wissen über das eigene kognitive System zu nutzen, um ihnen vorgelegte Aufgaben oder Sachverhalte zu abstrahieren und daraus neues Wissen und Strategien zur Problemlösung zu entwickeln. Durch die bewusste oder zumindest intuitive Analyse und (Re-)Strukturierung von unbekannten Probleme werden Muster erkannt. Diese Vorschriften können in abstrahierter (mathematischer) Form konstruiert und aufgeschrieben werden. Dadurch entsteht neues Wissen über Kognition im Allgemeinen und die individuelle Kognition.
(Bei Rekursion beispielsweise erfolgt die Übersetzung von Handlungsvorschriften in natürlicher Sprache in Variablen und andere mathematische Operationen. Dazu ist die Klassifizierung der vorhandenen Informationen in Basisfall, rekursiven Aufruf und Abbruchbedingung erforderlich. Dazu müssen Regeln und Handlungsvorschriften abstrahiert werden in die Sprache der Mathematik. Die metakognitiven Strategien hierfür müssen Studierende erst erlernen, wobei Intuition respektive Erfahrungswissen den Kompetenzzugewinn in aller Regel unterstützt.)

[5] Maturana, H. R., & Varela, F. (1987). Der Baum der Erkenntnis. Die biologischen Wurzeln des menschlichen Erkennens. Bern: Scherz Verlag.

3.6.18 Transfer

Die Studierenden sind dazu in der Lage, das Wissen über das eigene kognitive System, das eigene (inhaltliche) Wissen, dessen Grenzen und Einsatzmöglichkeiten zumindest intuitiv oder bewusst zu nutzen, um in anderen Kontexten (z. Bsp. in neuen, unbekannten Aufgabenstellungen) Problemlösungen erzielen zu können.

Durch das Überdenken von Inhalten, deren Grenzen bzw. Übertragbarkeit können bestehende kognitive Fertigkeiten in anderen Kontexten angewendet werden, um neue Lösungen zu erschaffen und die eigenen metakognitiven Fähigkeiten erweitern zu können. Daraus resultiert die Einordnung in die meta-kognitive Prozessdimension. Erfahrung wirkt auf die Entwicklung dieser Kompetenz ein und unterstützt die Ausprägung derselben.

4 Herausforderungen beim Lernen und Lehren von Programmieren

4.1 Herausforderungen Lehrende

4.1.1 (Prüfungs-)Aufgaben konzipieren und Bewerten

Lehrende erleben im Zusammenhang mit der Gestaltung von Übungs- und Prüfungsaufgaben eine Reihe von Herausforderungen, wie etwa bei den Versuchen, menschliche Hilfe während Präsenzübungen in digitalisierten Übungstools nachzubilden, bei der Auswahl geeigneter Prüfungsformate und bei der Entwicklung von angemessenen und (leicht) bewertbaren Aufgabenstellungen.

4.1.1.1 Menschliche Hilfe in Lernsystem nachbilden

Lehrende stehen vor der Herausforderung, menschliche Hilfe wie die etwa während Präsenzübungen, in digitalen Systemen zum Üben nachzubilden, sodass Studierende individuelle und hilfreiche Hinweise erhalten.

4.1.1.2 Angemessene Prüfungsformate auswählen

Lehrende stehen vor der Herausforderung, besonders für Massenveranstaltungen wie etwa Programmieren I und II geeignete Formate zu definieren, die die festgelegten Lehr- und Lernziele geeignet abprüfen und dem Lehrbetrieb während des Semesters gerecht werden (Programmieren auf Papier vs. Programmieren am Computer). Für ein dem Constructive Alignment gerecht werdenden Prüfungsformat müssen rechtliche Rahmenbedingungen für E-Klausuren der jeweiligen Bildungseinrichtung berücksichtigt werden.

4.1.1.3 Entwicklung angemessener, bewertbarer Aufgaben

Lehrende stehen vor der Herausforderung, bei der Entwicklung von Aufgaben für digitale (Online)-Übungstools den eingegebenen Code der Studierenden in angemessener Art und Weise zu überprüfen und (automatisiert) auszuwerten.
Weitere Herausforderungen bestehen darin, Teilpunkte vergeben zu können, den Lösungsweg zu bepunkten und nicht allein das Ergebnis zu testen, sondern ob der Lösungsweg den Anforderungen gerecht wird, etc. Constructive Alignment nach Biggs erfordert außerdem die Übereinstimmung zwischen Lehr und Lernzielen, Übungsaufgaben und der abschließenden Prüfung.
(Die vielen möglichen Problemlösungen überfordern Studierende, insbesondere zu Beginn des Studiums. Es gibt selten die eine richtige Lösung. Davon werden Studierende leicht überfordert und fühlen sich erschlagen. Gleichzeitig wird dadurch die Unterstützung durch Feedback erschwert, sowohl durch Tutoren, als auch durch digitale Übungsprogramme und/oder weitere Tools.)

4.1.2 Lernatmosphäre in Präsenzveranstaltungen schaffen

Lehrende stehen vor der Herausforderung, eine geeignete und lernförderliche Lernatmosphäre zu schaffen, zum Beispiel durch die Wahl der Raumsituation, Kommunikation mit und unter Studierenden, Gruppenarbeiten und deren Organisation und Betreuung.

4.1.3 Begrenzte Kapazitäten der Lehrenden für Feedback

Lehrende stehen vor der Herausforderung, den Individuen einer Übungsgruppe während der Präsenzlehre gerecht zu werden, sodass individuellen Fragen und Anliegen berücksichtigt werden und Studierende Feedback zu ihren Lösungsansätzen erhalten.

4.1.4 Verhindern von Trittbrettfahrern & Plagiaten

Lehrende stehen vor der Herausforderung, Trittbrettfahrer in Gruppen oder Studierende, die ihre Lösung nicht selbst entwickelt haben, zu identifizieren und im Idealfall von vornherein zu vermeiden.

4.1.5 begrenzte Kompetenzen von Tutoren

Lehrende stehen vor der Herausforderung, geeignete Tutoren für die Unterstützung von Präsenzübungen auszuwählen. Nicht alle Tutoren verfügen über die notwendigen Eigenschaften und Qualifikationen, um andere Lernende gut betreuen zu können. Dementsprechend können z. Bsp. zwischenmenschliche Konfliktsituationen entstehen. Weiterhin können die Tutoren zu Randthemen oder anderen spezialisierten Inhalten nicht immer die notwendige Unterstützung anbieten, da sie selbst noch nicht vollständig ausgebildet sind.

4.1.6 Programmiersprachen-unabhängige Konstrukte identifizieren

Lehrende stehen vor der Herausforderung, Programmiersprachen-unabhängige Konstrukte zu identifizieren, die als Basis der Programmierausbildung gelehrt werden sollten. Die programmiersprachlichen Besonderheiten resultieren in verschiedenen Schwierigkeiten der verschiedenen Programmiersprachen und –paradigmen, insbesondere für Anfängerinnen.

4.1.7 Theorierelevanz in der Programmierung aufzeigen

Lehrende stehen vor der Herausforderung, Studierenden die Relevanz theoretischer Konstrukte aufzuzeigen wie etwa Konzepte und Methoden aus der Mathematik, welche die Programmierung theoretisch unterfüttern und deren Basis darstellen.

4.2 Herausforderungen Lernende

Lernende erleben im Zusammenhang mit dem Programmieren lernen einige Schwierigkeiten. Dazu zählen speziell für die Informatik notwendige Herangehensweisen an Probleme, die stark aufeinander aufbauenden Inhalte und Fähigkeiten, die begrenzte Zeit zum Programmieren lernen, abschreckende Klischees, Zeitprobleme im Studium, Angst vor formalen Ausdrücken und Mathematik sowie Motivationsprobleme und fehlende Englisch-Kenntnisse.

4.2.1 Informatik-spezifische Strategien erlernen

Lernende stehen vor der Herausforderung, angemessene Strategien zum Programmieren zu identifizieren und zu nutzen. Eine angemessene Strategie in der Informatik bewegt sich in dem Kontinuum zwischen dem Verstehen von Konzepten und dem Ausprobieren durch die Anwendung von Versuch und Irrtum. Diesen Mittelweg zu finden, stellt Studierende vor eine Herausforderung, da bei dem Schreiben eines jeden Programms zahlreiche Entscheidungen getroffen werden müssen. Hierfür in jedem Einzelfall selbstständig korrekte, problemadäquate Entscheidungen direkt zu Beginn zu treffen, stellt eine Herausforderung dar.

4.2.2 Aufeinander aufbauende Inhalte und Fähigkeiten

Lernende stehen vor der Herausforderung, dass die zu Beginn der Programmierausbildung vermittelten Inhalte und Fähigkeiten sehr stark aufeinander aufbauen. Daher ist eine konsequente und kontinuierliche Mitarbeit und ggf. Nacharbeitung unerlässlich, um nicht abgehängt zu werden und keine Defizite entstehen zu lassen.

4.2.3 Programmieren lernen in endlicher Zeit

Lernende stehen vor der Herausforderung, Programmieren in endlicher Zeit zu lernen und die Mindestanforderungen einer Übungsstunde, Prüfung oder Lehrveranstaltung in der jeweilig

vorgegebenen Zeit zu erfüllen. (Programmieren lernen kann grundsätzlich jedes Individuum, in formalen Bildungskontexten steht hierfür jedoch nur eine begrenzte Zeitspanne zur Verfügung).

4.2.4 Abschreckende Nerd-Klischees

Lernende stehen vor der Herausforderung, sich vor Aufnahme des Studiums nicht von „Nerd-Klischees" abschrecken zu lassen, und sich stattdessen mit dem Studienfach zu identifizieren (was einen Faktor für den Studienerfolg darstellt).

4.2.5 Zeitliche Einschränkungen durch Nebenjob

Lernende stehen vor der Herausforderung, ihre zeitlichen Ressourcen zum Lernen einzuteilen und gegenüber anderen Faktoren, wie dem Nebenjob, abzuwägen. Dadurch besteht die Gefahr, nicht ausreichend Zeit in die notwendige Übung zu investieren.

4.2.6 Angst vor Mathematik und formalen Ausdrücken

Lernende stehen vor der Herausforderung, sich auf mathematische Operationen einzulassen und dahingehende Ängste zu überwinden.

4.2.7 Nichterscheinen in Präsenzveranstaltungen

Lernende stehen vor der Herausforderung, freiwillige Angebote zur Unterstützung des Lernprozesses wahrzunehmen (wie etwa Übungsstunden oder Tutorien), um Hilfe bei Fragen und Problemen wahrnehmen zu können. Dazu muss die Bereitschaft bestehen, Hilfe und Feedback anzunehmen.

4.2.8 Fach-Englisch für Informatiker

Lernende stehen vor der Herausforderung, fachspezifische Ausdrücke in der englischen Sprache zu erlernen, diese zu verstehen und anwenden zu können.

4.2.9 kognitive Kompetenzen[6]

4.2.9.1 Systematik beim Problemlösen entwickeln

4.2.9.2 Abstraktion von Handlungsvorschriften

4.2.9.3 Transfer

4.2.9.4 Logische Ausdrücke und Operatoren zur Problemlösung benutzen

4.2.9.5 Werkzeuge zur Software-Entwicklung nutzen

4.2.9.6 Programmiersprachliche, lauffähige Lösungen für Probleme schreiben

4.2.9.7 Externe Ressourcen zum Lernen nutzen

4.2.9.8 Selbständige Organisation des Lernprozesses

4.2.9.9 Konzepte der Programmierung in eigenen Worten erklären

[6] An dieser Stelle erfolgen lediglich Verweise auf die bereits definierten kognitiven Kategorien, die sich als Schwierigkeit erweisen können. Die Definitionen können in den

4.2.10 nicht-kognitive Kompetenzen[7]

4.2.10.1 Sozial-kommunikative Fähigkeiten

4.2.10.1.1 mangelnde Kooperation & Kollaboration
4.2.10.1.2 mangelnde Kommunikation

4.2.10.2 Haltung & Einstellung

4.2.10.2.1 fehlendes Durchhaltevermögen bei Frust
4.2.10.2.2 fehlende Begeisterung & Spaß am Problemlösen/Informatik
4.2.10.2.3 fehlende Lernbereitschaft/Offenheit
4.2.10.2.4 fehlende aktive, regelmäßige Mitarbeit
4.2.10.2.5 fehlende freiwillige Mitarbeit

4.2.10.3 fehlende Programmiererfahrung durch zu wenig zeitintensives Üben

4.2.10.4 Selbstkompetenzen

4.2.10.4.1 fehlendes Selbstvertrauen in eigene Fähigkeiten
4.2.10.4.2 fehlende Kreativität
4.2.10.4.3 fehlende Vorkenntnisse

[7] An dieser Stelle erfolgen lediglich Verweise auf die bereits definierten nicht-kognitiven Kategorien, die sich als Schwierigkeit erweisen können.