Codebuch

Qualitative Inhaltsanalyse (nach Mayring) von 129 Modulbeschreibungen 35 deutscher Universitäten und (Fach-)Hochschulen

07.04.2020

Stand: 07.04.2020

Inhaltsverzeichnis

A. Codesystem mit Häufigkeiten

1 nicht operationalisiert	245
2 Modulnamen[1]	(129)[2]
2.1 Algorithmen	1
2.2 Algorithmen und Datenstrukturen	14
2.3 Algorithmen und Datenstrukturen (Praktikum)	1
2.4 Algorithmen und Datenstrukturen (Vorlesung und Praktikum)	1
2.5 Algorithmen und Datenstrukturen (Vorlesung und Übung)	1
2.6 Algorithmen und Datenstrukturen (Vorlesung)	1
2.7 Algorithmen und Datenstrukturen I	1
2.8 Algorithmen und Datenstrukturen II	1
2.9 Algorithmen und Datenstrukturen/ logische Programmierung	1
2.10 Algorithmen und Programmierung für IN und II	1
2.11 Algorithmen, Datenstrukturen und Datenabstraktionen	1
2.12 Algorithmen-Design	1
2.13 Datenstrukturen	1
2.14 Datenstrukturen und Algorithmen	2
2.15 Datenstrukturen und Algorithmen	1
2.16 Deklarative Programmierung	1
2.17 Einführung in die Informatik	1
2.18 Einführung in die Programmierung	3
2.19 Einführung in die Programmierung (Java)	1
2.20 Einführung in die Programmierung (Vorlesung, Übung und Praktikum)	1
2.21 Einführung in die Programmierung für Informatiker	1
2.22 Einführung in die Softwareentwicklung (Vorlesung und Übung)	1
2.23 Formale Grundlagen der Informatik I	1
2.24 Funktionale Programmierung	1
2.25 Funktionale und objektorientierte Programmierkonzepte	1
2.26 Graphentheoretische Konzepte und Algorithmen (Vorlesung und Praktikum)	1
2.27 Grundkonzepte der Programmierung	1
2.28 Grundlagen der Informatik	1
2.29 Grundlagen der Informatik 1	1
2.30 Grundlagen der Informatik 2	1
2.31 Grundlagen der Informatik I	1

[1] Einige wenige Modulnamen wurden mehrfach in den 129 Modulen wiedergefunden, wie die Häufigkeiten in der rechten Spalte zeigen. Daher werden lediglich 93 verschiedene Kategorien aufgeführt.
[2] Werden die Häufigkeiten in runden Klammern angegeben, wird lediglich die Summe der Codes aller Subkategorien angezeigt.

2.32 Grundlagen der Informatik II	1
2.33 Grundlagen der Programmierung	1
2.34 Grundlagen der Programmierung 1	1
2.35 Grundlagen der Programmierung 2	1
2.36 Grundzüge von Algorithmen und Datenstrukturen	1
2.37 Hardwarenahe Programmierung	1
2.38 Imperative und funktionale Programmierung	1
2.39 Informatik I	1
2.40 Informatik II	1
2.41 Informatikgrundlagen	1
2.42 Modellierung	1
2.43 Objektorientierte Modellierung und Programmierung	1
2.44 Objektorientierte Programmierung	4
2.45 Praktikum Systemprogrammierung	1
2.46 Praktikum: Grundlagen der Programmierung	1
2.47 Praktische Informatik 1: Imperative Programmierung und Objektorientierung	1
2.48 Praktische Informatik 2: Algorithmen und Datenstrukturen	1
2.49 Praktische Informatik 3: Funktionale Programmierung	1
2.50 Professionelle Softwareentwicklung (Programmierpraktikum I)	1
2.51 Programm- und Datenstrukturen I	1
2.52 Programm- und Datenstrukturen II	1
2.53 Programmierbeleg	1
2.54 Programmieren	1
2.55 Programmieren 1	1
2.56 Programmieren 2	1
2.57 Programmieren 3	1
2.58 Programmieren I	2
2.59 Programmieren II	2
2.60 Programmieren III	1
2.61 Programmiermethoden und -techniken (Praktikum)	1
2.62 Programmiermethoden und -techniken (Vorlesung)	1
2.63 Programmiermethodik	2
2.64 Programmiermethodik II (Vorlesung und Praktikum)	1
2.65 Programmierparadigmen	3
2.66 Programmierprojekt	1
2.67 Programmierprojekt	1
2.68 Programmiersprachen (Vorlesung und Übung)	1
2.69 Programmiersprachen und Typsysteme	1

2.70 Programmiertechnik (Vorlesung und Praktikum)	1
2.71 Programmierung	2
2.72 Programmierung 1	5
2.73 Programmierung 2	5
2.74 Programmierung 3	1
2.75 Programmierung I	2
2.76 Programmierung II	2
2.77 Programmierung III	1
2.78 Programmierung JAVA 1	1
2.79 Programmierung Teil 1 und 2	1
2.80 Programmierung und Modellierung	1
2.81 Programmierung und Software-Entwicklung	1
2.82 Programmierung, Datenstrukturen und Algorithmen	1
2.83 Softwareentwicklung I	1
2.84 Softwareentwicklung II	1
2.85 Softwareentwicklung im Team (Programmierpraktikum II)	1
2.86 Softwareentwicklungspraktikum	1
2.87 Softwareentwurf und -test	1
2.88 Softwareprojekt	1
2.89 Softwaretechnik	2
2.90 Softwaretechnik 1	1
2.91 Systemnahe und Parallele Programmierung	1
2.92 Systempraktikum	1
2.93 Weiterführende Themen der Programmierung	1
3 nicht-kognitive Kompetenzen	(48)
3.1 Sozial-kommunikative Fähigkeiten	(37)
3.1.1 Kooperieren und kollaborieren	22
3.1.2 Kommunizieren	13
3.1.3 Organisation von Teamarbeit und Projekten (Projektmanagement)	2
3.2 Programmiererfahrung sammeln	10
3.3 Sensibilisierung für IT-Sicherheit (affektiv)	1
4 Wissensdimensionen	(707)
4.1 Faktenwissen	1
4.2 Konzeptionelles Wissen	36
4.3 Prozedurales Wissen	647
4.4 Meta-kognitives Wissen	23
5 kognitive Prozessdimensionen	(707)
5.1 Erinnern	(162)

5.1.1 Literale zu Datentypen zuordnen	1
5.1.2 Begriffe & Kategorien zu Komplexität & Effizienz von Algorithmen kennen	7
5.1.3 Merkmale von Algorithmen kennen	3
5.1.4 Funktionsbibliotheken für Algorithmen kennen	3
5.1.5 Elemente von GUIs kennen	1
5.1.6 Grundprinzipien und Methoden von Programmierparadigmen kennen	50
5.1.7 Kenntnis grundlegender Algorithmen & Datenstrukturen	39
5.1.8 Methoden der Softwareentwicklung kennen	12
5.1.9 Konzepte zur Modellierung von Algorithmen und Prozessen kennen	9
5.1.10 Entwurfsmöglichkeiten von Algorithmen & Datenstrukturen kennen	8
5.1.11 Aufbau und Funktionsweise des Computers kennen	5
5.1.12 Implementierungsmöglichkeiten von Datentypen & -strukturen kennen	4
5.1.13 Prozesskenntnis der Laufzeitanalyse	4
5.1.14 Werkzeuge zur Softwareentwicklung, deren Funktionen und Anwendungszwecke kennen (IDEs, Debugger, Profiler)	3
5.1.15 Qualitätskriterien für Quellcode nennen	3
5.1.16 Mathematische Grundlagen von Algorithmen kennen	2
5.1.17 Konzepte zur Datenverwaltung kennen	2
5.1.18 Begriff der Kontrollstrukturen kennen	2
5.1.19 Aufbau und Funktionsweise von Netzen kennen	1
5.1.20 Funktionsweise des Compilers kennen	1
5.1.21 Verteilte Systeme & paralleles Programmieren kennen	1
5.1.22 Techniken zur formalen Definition von Programmiersprachen kennen	1
5.2 Verstehen	(29)
5.2.1 Konzepte von Programmierparadigmen und -sprachen beschreiben	11
5.2.2 Algorithmische Probleme und Aufgabenstellungen beschreiben	3
5.2.3 Begriffe formaler Verifikationstechniken erläutern	2
5.2.4 Algorithmen, Datenstrukturen und Datentypen charakterisieren	9
5.2.5 Softwarearchitektur & deren Entwicklung beschreiben	2
5.2.6 Notwendigkeit von Werkzeugen zur Softwareentwicklung begründen	2
5.3 Anwenden	(51)
5.3.1 Werkzeuge zur Software-Entwicklung nutzen	30
5.3.2 Mathematische Berechnungen und Codierungen durchführen	5
5.3.3 Bestehende Bibliotheken einsetzen	5
5.3.4 Programme professionell dokumentieren	4
5.3.5 Qualitätskriterien und Programmierkonventionen auf Quellcode anwenden	4
5.3.6 Computer bedienen	1
5.3.7 LaTex benutzen	1

5.3.8 UNIX-Systeme bedienen mit Hilfe der Kommandozeile	1
5.4 Analysieren	(75)
5.4.1 Programmiersprachen & -paradigmen differenzieren	6
5.4.2 Komplexität von Algorithmen analysieren	24
5.4.3 Problemstellung zerlegen	20
5.4.4 Algorithmen auf Eigenschaften und Eignung hin analysieren	9
5.4.5 (Fremde) Programme beschreiben können	8
5.4.6 Datenstrukturen auf Eigenschaften und Eignung hin analysieren	6
5.4.7 Formale Sprachen durch Chomsky-Hierarchie differenzieren	1
5.4.8 Grenzen der Berechnungskraft von Rechnermodellen aufzeigen	1
5.5 Bewerten	(85)
5.5.1 Angemessenheit von Algorithmen, Datenstrukturen und Datentypen beurteilen	34
5.5.2 Testen von Algorithmen und Programmen auf Fehler	19
5.5.3 Komplexität von Algorithmen beurteilen	14
5.5.4 Eigenschaften von Algorithmen und Programmen beweisen	10
5.5.5 Angemessenheit programmiersprachlicher Lösungen beurteilen	5
5.5.6 Beurteilen von Programmierwerkzeugen & Templates	2
5.5.7 Angemessenheit des Selbstkonzepts bewerten	1
5.6 Erzeugen	(305)
5.6.1 Programmiersprachliche, lauffähige Lösungen für Probleme schreiben	80
5.6.2 Algorithmen mit Hilfe von Entwurfsmustern problemadäquat entwerfen	41
5.6.3 Algorithmen implementieren	33
5.6.4 Modellieren von Problemen und Programmen	26
5.6.5 kleinere Programme strukturiert entwerfen	17
5.6.6 Standard-Algorithmen anwenden	16
5.6.7 Standard-Datenstrukturen anwenden	14
5.6.8 Datenstrukturen entwerfen	12
5.6.9 Datenstrukturen anpassen	8
5.6.10 Algorithmen anpassen	7
5.6.11 (Abstrakte) Datentypen konzipieren & einsetzen	5
5.6.12 Gegebene Programm(-komponenten) anpassen & verändern	5
5.6.13 Formale Syntaxbeschreibungen entwerfen	4
5.6.14 Programm-Spezifikation entwerfen	3
5.6.15 größere Anwendungen/Applikationen entwickeln	2
5.6.16 Formale Werkzeuge & Testfälle entwickeln	2
5.6.17 GUIs programmieren	2
5.6.18 nebenläufige und parallele Prozesse entwerfen	2
5.6.19 Bibliotheken entwickeln	1

5.6.20 Methoden aufrufen	1
5.6.21 Objekte erzeugen	1
5.6.22 Schnittstellen entwerfen	1
5.6.23 Transfer von Wissen & Erfahrung auf neue Aufgaben und Programmiersprachen	13
5.6.24 Selbstständige Organisation des Lernprozesses (unter Anwendung externer Ressourcen)	7
5.6.25 Abstraktion von Problemen	2

1 nicht operationalisiert

Nicht alle Lehr- und Lernziele können nach der Anderson und Krathwohl-Taxonomie (2001)[3] klassifiziert werden. Ursache hierfür ist das Fehlen operationalisierter Verben oder Verben insgesamt. Häufig werden Worte wie „erlernen", „lernen", „beherrschen", „schulen", „erfassen", „erlangen", „erkennen", „begegnen", „wissen" und „verstehen" verwendet, die sich weder beobachten, noch messen lassen. Auch wenn Studierende einen „Überblick" oder „Einblick" „erhalten", „erwerben" oder „bekommen", kann kein kompetenzorientiertes Ziel sichtbar werden.

Unvollständige Sätze oder einzelne Satzbausteine, aus denen sich kein Sinn bzw. keine eindeutige Zuordnung zur Taxonomie erschließt, werden außerdem hier als nicht operationalisiert kategorisiert. Nicht zuletzt werden Listen von Inhalten dementsprechend als nicht operationalisiert eingeordnet.

[3] 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.

2 Modulnamen

Die Modulnamen werden jeweils als gleichnamige Unterkategorie mitcodiert. Daher folgt an dieser Stelle lediglich eine Auflistung aller Modulnamen, ohne weitere Beschreibungen.

2.1 Algorithmen

2.2 Algorithmen und Datenstrukturen

2.3 Algorithmen und Datenstrukturen (Praktikum)

2.4 Algorithmen und Datenstrukturen (Vorlesung und Praktikum)

2.5 Algorithmen und Datenstrukturen (Vorlesung und Übung)

2.6 Algorithmen und Datenstrukturen (Vorlesung)

2.7 Algorithmen und Datenstrukturen I

2.8 Algorithmen und Datenstrukturen II

2.9 Algorithmen und Datenstrukturen/ logische Programmierung

2.10 Algorithmen und Programmierung für IN und II

2.11 Algorithmen, Datenstrukturen und Datenabstraktionen

2.12 Algorithmen-Design

2.13 Datenstrukturen

2.14 Datenstrukturen und Algorithmen

2.15 Datenstrukturen und Algorithmen

2.16 Deklarative Programmierung

2.17 Einführung in die Informatik

2.18 Einführung in die Programmierung

2.19 Einführung in die Programmierung (Java)

2.20 Einführung in die Programmierung (Vorlesung, Übung und Praktikum)

2.21 Einführung in die Programmierung für Informatiker

2.22 Einführung in die Softwareentwicklung (Vorlesung und Übung)

2.23 Formale Grundlagen der Informatik I

2.24 Funktionale Programmierung

2.25 Funktionale und objektorientierte Programmierkonzepte

2.26 Graphentheoretische Konzepte und Algorithmen (Vorlesung und Praktikum)

2.27 Graphentheorie

2.28 Grundkonzepte der Programmierung

2.29 Grundlagen der Informatik

2.30 Grundlagen der Informatik 1

2.31 Grundlagen der Informatik 2

2.32 Grundlagen der Informatik I

2.33 Grundlagen der Informatik II

2.34 Grundlagen der Programmierung

2.35 Grundlagen der Programmierung 1

2.36 Grundlagen der Programmierung 2

2.37 Grundzüge von Algorithmen und Datenstrukturen

2.38 Hardwarenahe Programmierung

2.39 Imperative und funktionale Programmierung

2.40 Informatik I

2.41 Informatik II

2.42 Informatikgrundlagen

2.43 Modellierung

2.44 Objektorientierte Modellierung und Programmierung

2.45 Objektorientierte Programmierung

2.46 Praktikum Systemprogrammierung

2.47 Praktikum: Grundlagen der Programmierung

2.48 Praktische Informatik 1: Imperative Programmierung und Objektorientierung

2.49 Praktische Informatik 2: Algorithmen und Datenstrukturen

2.50 Praktische Informatik 3: Funktionale Programmierung

2.51 Professionelle Softwareentwicklung (Programmierpraktikum I)

2.52 Programm- und Datenstrukturen I

2.53 Programm- und Datenstrukturen II

2.54 Programmierbeleg

2.55 Programmieren

2.56 Programmieren 1

2.57 Programmieren 2

2.58 Programmieren 3

2.59 Programmieren I

2.60 Programmieren II

2.61 Programmieren III

2.62 Programmiermethoden und -techniken (Praktikum)

2.63 Programmiermethoden und –techniken (Vorlesung)

2.64 Programmiermethodik

2.65 Programmiermethodik II (Vorlesung und Praktikum)

2.66 Programmierparadigmen

2.67 Programmierprojekt

2.68 Programmierprojekt

2.69 Programmiersprachen (Vorlesung und Übung)

2.70 Programmiersprachen und Typsysteme

2.71 Programmiertechnik (Vorlesung und Praktikum)

2.72 Programmierung

2.73 Programmierung 1

2.74 Programmierung 2

2.75 Programmierung 3

2.76 Programmierung I

2.77 Programmierung II

2.78 Programmierung III

2.79 Programmierung JAVA 1

2.80 Programmierung Teil 1 und 2

2.81 Programmierung und Modellierung

2.82 Programmierung und Software-Entwicklung

2.83 Programmierung, Datenstrukturen und Algorithmen

2.84 Softwareentwicklung I

2.85 Softwareentwicklung II

2.86 Softwareentwicklung im Team (Programmierpraktikum II)

2.87 Softwareentwicklungspraktikum

2.88 Softwareentwurf und –test

2.89 Softwareprojekt

2.90 Softwaretechnik

2.91 Softwaretechnik 1

2.92 Systemnahe und Parallele Programmierung

2.93 Systempraktikum

2.94 Weiterführende Themen der Programmierung

3 nicht-kognitive Kompetenzen

Unter dieser Kategorie werden nicht-kognitive Kompetenzen zusammengefasst, die sich auf sozial-kommunikative Kompetenzen, das Sammeln von Programmiererfahrung sowie die Entwicklung einer Werthaltung gegenüber IT-Sicherheit beziehen.

3.1 Sozial-kommunikative Fähigkeiten

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

3.1.1 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.

3.1.2 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 Kommunikation (alleine und mit den Mitstudierenden) und Verbalisierung eigens entwickelter Ergebnisse steht im Vordergrund.

3.1.3 Organisation von Teamarbeit und Projekten (Projektmanagement)

Die Studierenden sind dazu in der Lage, das Arbeiten in einem Team abzustimmen, gemeinsame Arbeitsschritte zu planen, durchzuführen, zu überwachen und ggf. zu steuern, sodass Projekte/Aufgabenstellungen erfolgreich abgeschlossen werden können.

3.2 Programmiererfahrung sammeln

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)[4] was mitunter zum Erwerb von meta-kognitiven Kompetenzen beiträgt.

3.3 Sensibilisierung für IT-Sicherheit (affektiv)

Die Studierenden entwickeln eine sensible (Wert-)Haltung gegenüber Sicherheitsproblemen in der Informatik.

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

4 Wissensdimensionen[5]

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.

4.1 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).

4.2 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).

4.3 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)

[5] 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).

4.4 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).

5 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).

5.1 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.

5.1.1 Literale zu Datentypen zuordnen

Die Studierenden können Literale einer Programmiersprache den entsprechenden Datentypen zuordnen.

5.1.2 Begriffe & Kategorien zu Komplexität & Effizienz von Algorithmen kennen

Die Studierenden kennen Effizienzmaße für Algorithmen, sowie Komplexitätsklassen für Laufzeit und Speicherbedarf (P, NP, NPC). Sie kennen außerdem Grundlagen (Churchsche These) und typische Probleme bezüglich der Effizienz von algorithmischen Lösungen.

5.1.3 Merkmale von Algorithmen kennen

Die Studierenden sind dazu in der Lage, den Algorithmus-Begriff zu definieren. Sie kennen entsprechende Grundbegriffe zu Merkmalen und Eigenschaften von Algorithmen sowie deren Grenzen.

5.1.4 Funktionsbibliotheken für Algorithmen kennen

Die Studierenden kennen typische Funktionsbibliotheken für Algorithmen in Programmiersprachen (z. Bsp. in Java oder C) und deren Umfang.

5.1.5 Elemente von GUIs kennen

Die Studierenden kennen grafische Benutzerschnittstellen (GUIs) und Elemente, aus denen sie bestehen.

5.1.6 Grundprinzipien und Methoden von Programmiersprachen und -paradigmen kennen

Die Studierenden kennen ganz konkrete Konzepte, Methoden und Strategien sowie Anwendungskontexte einer Programmiersprache (Methoden als prozedurales Wissen).

Die Studierenden kennen darüber hinaus Konzepte von Programmierparadigmen, beispielsweise prozedurale Programmierung und Objektorientierung. Die Studierenden kennen nicht nur verschiedene Programmierparadigmen, sondern können auch deren grundlegende Gemeinsamkeiten und Unterschiede wiedergeben.

5.1.7 Kenntnis grundlegender Algorithmen & Datenstrukturen

Die Studierenden kennen Eigenschaften, Vor- und Nachteile, Einsatzszenarien und Leistungsparameter von grundlegenden Algorithmen und Datenstrukturen (und Datentypen).

Die Studierenden kennen dementsprechend elementare Datenstrukturen und Algorithmen für immer wiederkehrende Probleme, wie etwa Such- und Sortierverfahren.

5.1.8 Methoden der Softwareentwicklung kennen

Die Studierenden kennen grundlegende Methoden der Softwareentwicklung, wie z. Bsp. des Konfigurationsmanagements. Sie kennen Werkezuge für die Entwicklung professionellen Codes, automatisierte Tests, Standardarchitekturmuster, und Methoden zur Programmverifikation.

Sie kennen außerdem den Ablauf des Erstellungsprozesses von Software sowie Probleme bei der Durchführung von Softwareprojekten, sodass diese später vermieden werden können.

5.1.9 Konzepte zur Modellierung von Algorithmen und Prozessen kennen

Die Studierenden kennen Konzepte zur Modellierung von Algorithmen und Prozessen (z. Bsp. durch UML). Die Studierenden lernen, wie sie solche Modelle konzipieren, und welche Eigenschaften gute Modelle ausmachen. Dazu wird auch die Kenntnis spezieller objektorientierte Modelle und das Kennen von UML als Modellierungsnotation gezählt.

5.1.10 Entwurfsmöglichkeiten von Algorithmen & Datenstrukturen kennen

Die Studierenden kennen verschiedene Möglichkeiten, um Datenstrukturen und Algorithmen zur Problemlösung zu entwerfen. Dazu gehören z. Bsp.: Divide-and-conquer, dynamische Programmierung, Iteration, Rekursion, Backtracking, Greedy-Algorithmen, Dezimierung, Hierarchisierung, Randomisierung als Entwurfsmuster.

5.1.11 Aufbau und Funktionsweise des Computers kennen

Die Studierenden kennen den Aufbau und die Funktionsweise eines elektronischen Computers und können aufsagen, wie diese Algorithmen verarbeiten.

5.1.12 Implementierungsmöglichkeiten von Datentypen & -strukturen kennen

Die Studierenden kennen die wichtigsten Implementierungsmöglichkeiten für Datentypen und Datenstrukturen (in verschiedenen Programmierparadigmen).

5.1.13 Prozesskenntnis der Laufzeitanalyse

Die Studierenden kennen Arbeitsschritte und Modelle zur Laufzeitanalyse einfacher Algorithmen (z. Bsp. anhand der O-Notation).

5.1.14 Werkzeuge zur Softwareentwicklung, deren Funktionen und Anwendungszwecke kennen (IDEs, Debugger, Profiler)

Die Studierenden kennen Werkzeuge zur Programmierung und Softwareentwicklung sowie deren Funktionen und Anwendungszwecke (Profiler, Debugger, Entwicklungsumgebungen, etc.).

5.1.15 Qualitätskriterien für Quellcode kennen

Die Studierenden kennen die in der Informatik-Community geltenden Konventionen für einen guten Programmierstil, um später lesbaren, wartbaren und wiederverwendbaren Code erzeugen zu können.

5.1.16 Mathematische Grundlagen von Algorithmen kennen

Studierende kennen Zusammenhänge zwischen Mengenlehre, Logik, Relationen, Wahrscheinlichkeitsrechnung etc. als mathematische Grundlagen der Informatik (sprich von Algorithmen und Programmen).

5.1.17 Konzepte zur Datenverwaltung kennen

Die Studierenden kennen Konzepte zur Modellierung, Verwaltung und Nutzung von Datenbeständen (einschließlich der Ein- und Ausgabe).

5.1.18 Begriff der Kontrollstrukturen kennen

Die Studierenden kennen grundlegende Kontrollstrukturen, und können diese z. Bsp. in Form von Struktogrammen oder durch Syntax darstellen.

5.1.19 Aufbau und Funktionsweise von Netzen kennen

Die Studierenden kennen den Aufbau und die Funktionsweise von Netzen (wie z. Bsp. dem Internet).

5.1.20 Funktionsweise des Compilers kennen

Die Studierenden kennen Konzepte des Übersetzens und Compilerentwurfs.

5.1.21 Verteilte Systeme & paralleles Programmieren kennen

Die Studierenden kennen verteilte Systeme und die Möglichkeit der parallelen Programmierung.

5.1.22 Techniken zur formalen Definition von Programmiersprachen kennen

Die Studierenden kennen Techniken und Notationen zur formalen Definition von (kontextfreien) Programmiersprachen.

5.2 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.

5.2.1 Konzepte von Programmierparadigmen (und –sprachen) beschreiben

Die Studierenden sind dazu in der Lage, Konzepte sowohl der imperativen, als auch der objekt-orientierten Programmierung zu erklären. Dabei werden u.a. deren Eigenschaften charakterisiert, Anwendungsgebiete, sowie Vor- und Nachteile der verschiedenen Paradigmen beschreiben.
(Konzepte der Programmierparadigmen meint allgemeine, abstrakte, schematische Aspekte, keine konkreten Methoden, Prozeduren, etc. Die eindeutige Einordnung in das Kontinuum zwischen konzeptionellem und prozeduralem Wissens ist bei dieser Kategorie nicht vollständig gegeben.)

5.2.2 Algorithmische Probleme und Aufgabenstellungen beschreiben

Die Studierenden können Ihnen gestellte Aufgabenstellungen in eigenen Worten formulieren und das zugrundeliegende Problem beschreiben.

5.2.3 Begriffe formaler Verifikationstechniken erläutern

Die Studierenden können Begriffe formaler Verifikationstechniken wie z. Bsp. Laufzeit und Korrektheit erläutern.

5.2.4 Algorithmen, Datenstrukturen und Datentypen charakterisieren

Die Studierenden sind dazu in der Lage, Datentypen, Datenstrukturen und Algorithmen zu charakterisieren. Sie können neben der Leistung und Anwendbarkeit von exemplarischen Algorithmen (wie etwa Suchen oder Sortieren) auch die dahinterliegenden mathematischen Konzepte (z. Bsp. Iteration oder Rekursion) sowie die rechnerinterne Darstellung von Daten und Zahlen erläutern.

5.2.5 Softwarearchitektur & deren Entwicklung beschreiben

Die Studierenden können grundlegende Software-Architekturen und deren Entwicklung beschreiben. (Das Verständnis der zugrundeliegenden Techniken und Methoden geht über konzeptionelles Wissen hinaus, da Entwicklungsprozesse erklärt werden können.)

5.2.6 Notwendigkeit von Werkzeugen zur Softwareentwicklung begründen

Die Studierenden können die Notwendigkeit von Werkzeugen zur Softwareentwicklung, wie zum Beispiel Unit-Tests oder Versionskontrollsystemen, erläutern.

5.3 Anwenden

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

5.3.1 Werkzeuge zur Software-Entwicklung nutzen

Die Studierenden sind dazu in der Lage, Spezifikationstools und Werkzeuge zur Software-Entwicklung mitsamt ihrer Funktionen zu benutzen. Dazu gehören z. Bsp. Werkzeuge zur Build-Automatisierung in größeren Projekten, Entwicklungsumgebungen wie IDEs, Debugger und Profiler. Interpreter und

Compiler sowie Tools zur Versionsverwaltung und Bug-Tracker können angewendet werden, genauso wie Rechnersimulationsprogramme.

Weiterhin können Studierende zusätzliche Tools und Methoden (Grundlagen der Berechnung, Übersetzung von Programmkonstruktoren, Programmtransformationen, Verifikation von Programmeigenschaften) zur Entwicklung von Software anwenden.

5.3.2 Mathematische Berechnungen und Codierungen durchführen

Die Studierenden sind dazu in der Lage, einfache mathematische Berechnungen (z. Bsp. aus der Kombinatorik und Wahrscheinlichkeitsrechnung) sowie Codierungen durchzuführen. Die Studierenden können Ganzzahlen, Gleitkommazahlen und andere Daten in Binärdarstellung codieren und decodieren.

5.3.3 Bestehende Bibliotheken einsetzen

Die Studierenden können bestehende, typische Funktionsbibliotheken für Algorithmen in Programmiersprachen einsetzen. (Die Datenstrom-Ein-und Ausgabe wird hierunter zusammengefasst, da sie in C++ zum Beispiel in Form der IOStream Library Teil der Standardbibliothek ist.)

5.3.4 Programme professionell dokumentieren

Die Studierenden sind dazu in der Lage, Programme (unter Zuhilfenahme von Werkzeugen) professionell und aussagekräftig zu dokumentieren.

5.3.5 Qualitätskriterien und Programmierkonventionen 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.

5.3.6 Computer bedienen

Die Studierenden sind dazu in der Lage, einen Rechner sicher zu bedienen.

5.3.7 LaTex benutzen

Die Studierenden können die Software LaTex zur Erstellung einfacher Dokumente nutzen.

5.3.8 UNIX-Systeme bedienen mit Hilfe der Kommandozeile

Die Studieren können UNIX-Systeme anhand der Kommandozeile bedienen.

5.4 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.

5.4.1 Programmiersprachen & -paradigmen differenzieren

Die Studierenden sind dazu in der Lage, verschiedene programmiersprachliche Konstrukte einzuordnen. Sie können den Aufbau und die Prinzipien von Programmiersprachen analysieren, sie dadurch formal beschreiben und gemäß der verschiedenen Programmierparadigmen differenzieren und unterscheiden, bzw. Zusammenhänge finden.

5.4.2 Komplexität von Algorithmen analysieren

Die Studierenden können die Komplexitätsklasse von Algorithmen durch die Abschätzung des Laufzeitaufwands und des Speicherplatzbedarfs bestimmen, und damit ihre Leistungsfähigkeit analysieren. Dazu gehört zum Beispiel die erste Aufwandsabschätzung zur Laufzeit mit Hilfe der O-Notation (bzw. anderer Notationen). Die Studierenden können außerdem Vergleiche zwischen der Komplexität (oder ausschließlich der Laufzeit) von Algorithmen durchführen.

5.4.3 Problemstellung 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.

5.4.4 Algorithmen auf Eigenschaften und Eignung hin analysieren

Die Studierenden sind dazu in der Lage, Algorithmen auf Eigenschaften und Eignung zur Problemlösung hin zu analysieren. Sie erkennen durch ihre Analyse z. Bsp., ob es sich um eine Funktion höherer Ordnung handelt, oder wie es um die Wartbarkeit eines Algorithmus bestimmt ist.

5.4.5 (Fremde) Programme beschreiben können

Die Studierenden sind dazu in der Lage, Programme als Problemlösungen zu analysieren. Sie können fremden Code untersuchen, sprich diesen „lesen" und durch analysieren auch verstehen. (Zum Verstehen ist zwingend die vorherige Analyse notwendig.)

5.4.6 Datenstrukturen auf Eigenschaften und Eignung hin analysieren

Die Studierenden können für vorgegebene Anwendungsfälle geeignete Datenstrukturen auswählen, indem sie diese auf ihre Eigenschaften und Leistungsparameter hin analysieren.

5.4.7 Formale Sprachen durch Chomsky-Hierarchie differenzieren

Die Studierenden sind dazu in der Lage, formale Sprachen mit Hilfe der Chomsky-Hierarchie zu klassifizieren und dadurch zu differenzieren.

5.4.8 Grenzen der Berechnungskraft von Rechnermodellen aufzeigen

Die Studierenden sind dazu in der Lage, die Grenzen der Berechnungskraft von Rechnermodellen aufzuzeigen, indem sie logische Argumente nutzen.

5.5 Bewerten

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

5.5.1 Angemessenheit von Algorithmen, Datenstrukturen und Datentypen beurteilen

Die Studierenden können die Angemessenheit von Datentypen, Datenstrukturen und Algorithmen gemessen an den Anforderungen einer Aufgabe oder eines Problems bewerten, indem Sie ein eigenes Urteil dazu entwickeln und abgeben.

Die Bewertung erfolgt, indem Studierende geeignete Algorithmen, Datentypen und Datenstrukturen für ein bestimmtes Problem identifizieren und begründet auswählen können. Weiterhin können Studierende die jeweiligen Datentypen und Datenstrukturen auf ihre Eigenschaften und deren Passung zum Problem hin beurteilen.

5.5.2 Testen von Algorithmen und Programmen auf Fehler

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 müssen. Mit eingeschlossen in diese Kategorie wird die Durchführung von Unit-Tests, durch die Studierende auf korrekte Funktionalität testen und abschließend darüber urteilen.

5.5.3 Komplexität von Algorithmen beurteilen

Die Studierenden sind dazu in der Lage, Leistungsparameter von Algorithmen wie Laufzeit und Speicherplatzbedarf (Zeit- und Platzkomplexität) zu beurteilen. Damit können sie die jeweilige Eignung für konkrete Aufgaben und Problemstelllungen bewerten.

5.5.4 Eigenschaften von Algorithmen und Programmen beweisen

Die Studierenden sind dazu in der Lage, Eigenschaften von Algorithmen und Programmen, sowie deren Korrektheit formal zu beweisen, indem Sie Ihnen bekannte Beweistechniken anwenden und daraufhin zu einem Urteil gelangen.

5.5.5 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 Paradigma auswählen.

5.5.6 Beurteilen von Programmierwerkzeugen & Templates

Die Studierenden sind dazu in der Lage, den sinnvollen Einsatz von Programmierwerkzeugen (wie z. Bsp. Compiler, Editor, Templates und Standardbibliotheken) sowie deren Grenzen einschätzen zu können.

5.5.7 Angemessenheit des Selbstkonzepts bewerten

Studierende können ihr Selbstkonzept (Stärken, Schwächen, Selbstwirksamkeit, akademisches Selbstbewusstsein, Motivation, Interessen) realistisch einschätzen.

5.6 Erzeugen

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

5.6.1 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.)

5.6.2 Algorithmen mit Hilfe von Entwurfsmustern problemadäquat entwerfen

Die Studierenden sind dabei in der Lage, bekannte Entwurfsmuster für Algorithmen (Greedy, Divide-and-conquer, sowie objektorientierte Entwurfskonzepte) problemadäquat anzuwenden, um algorithmische Lösungen zu entwickeln. Dabei berücksichtigen die Studierenden u.a. die Performanz der zu entwickelnden Problemlösung.

5.6.3 Algorithmen implementieren

Die Studierenden sind dazu in der Lage, Algorithmen, Datenstrukturen und kleinere Programme (bis zu einigen hundert Zeile) in einer Programmiersprache zu formulieren. Das Programm muss dabei noch nicht vollständig oder lauffähig sein. Darunter wird auch die Implementierung (Einbau/Umbau) von Software in ein größeres System zusammengefasst.

5.6.4 Modellieren von Problemen und Programmen

Die Studierenden sind dazu in der Lage, Probleme und Programme in beispielsweise der Standardnotation UML, in Struktogrammen (Nassi-Schneidermann & Co.), Klassendiagrammen als Modell der Objektorientierung, oder gemäß Graphen-theoretischer Modellierungsparadigmen 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.

5.6.5 kleinere Programme strukturiert entwerfen

Die Studierenden sind dazu in der Lage, kleine bis mittelgroße Problemlösungen zu entwerfen, indem sie einen einfachen Programm-Entwurf strukturiert konstruieren. Dabei berücksichtigen sie ggf. eine

vorgegebene Entwurfsidee oder ein vorgegebenes Programmierparadigma (imperativ, funktional, objektorientiert, etc.). Die Implementierung oder Umsetzung in ein lauffähiges Programm wird noch nicht in diese Kategorie eingeschlossen.

5.6.6 Standard-Algorithmen anwenden

Die Studierenden sind dazu in der Lage, existierende Standard-Algorithmen auf neue Probleme anzuwenden und diese Probleme damit zu lösen. Dazu sind Änderungen und Anpassungen an das Problem nötig, weshalb ein neuer, angepasster Algorithmus durch die Studierenden geschaffen werden muss.

5.6.7 Standard-Datenstrukturen anwenden

Die Studierenden sind dazu in der Lage, Standard-Datenstrukturen (Liste, Array, Bäume, Heap, Stack, Queue, etc.) in eigenen Programmen zu verwenden und damit ein neues Programmkonstrukt zu erzeugen.

5.6.8 Datenstrukturen entwerfen

Die Studierenden sind dazu in der Lage, eigene und problemadäquate Datenstrukturen neu zu entwerfen. Hierbei wird für ein unbekanntes, neues Problem eine neue Datenstruktur entworfen.

5.6.9 Datenstrukturen anpassen

Die Studierenden sind dazu in der Lage, bekannte Datenstrukturen an neue Anforderungen und Problemstellungen anzupassen. Dabei spielt es keine Rolle, ob es sich um bereits vorhandene oder in Teilen bekannte Standard-Datenstrukturen handelt. Die Studierenden sind außerdem dazu in der Lage, zum Zwecke der Optimierung Datenstrukturen anzupassen.

5.6.10 Algorithmen anpassen

Die Studierenden können Standard-Algorithmen an neue Aufgabenstellung anpassen und damit speziellen Problemanforderungen (z. Bsp. bezüglich Performanz) gerecht werden, indem sie Verbesserungen planen, entwerfen und konstruieren.

5.6.11 (Abstrakte) Datentypen konzipieren & einsetzen

Die Studierenden sind in der Lage, (abstrakte) Datentypen zu entwickeln, spezifizieren und anzuwenden (im Sinne von programmieren).

5.6.12 Gegebene Programm(-komponenten) anpassen & verändern

Die Studierenden können gegebene Programme an veränderte Anforderungen anpassen, Teile davon benutzen und in andere Programme integrieren, oder gegebene Lösungen optimieren, indem sie neue Konstrukte planen, entwerfen und umsetzen.

5.6.13 Formale Syntaxbeschreibungen entwerfen

Die Studierenden können formale Syntaxbeschreibungen für einfache Sprachen oder kontextfreie Programmiersprachen entwickeln. Zugrunde liegende Regeln werden dabei abstrahiert und in syntaktische und semantische Beschreibungen von Programmiersprachen formalisiert.

5.6.14 Programm-Spezifikation entwerfen

Die Studierenden sind dazu in der Lage, Anforderungen an Programme zu spezifizieren.

5.6.15 größere Anwendungen/Applikationen entwickeln

Die Studierenden können größere Anwendungen oder Applikationen entwickeln.

5.6.16 Formale Werkzeuge & Testfälle entwickeln

Die Studierenden sind dazu in der Lage, formale Werkzeuge (z. Bsp. Grundlagen der Berechnung, Übersetzung von Programmkonstruktoren, Programmtransformationen, Verifikation von Programmeigenschaften) oder Testfälle zu entwickeln.

5.6.17 GUIs programmieren

Die Studierenden sind dazu in der Lage, grafische Benutzerschnittstellen zu programmieren.

5.6.18 nebenläufige und parallele Prozesse entwerfen

Die Studierenden sind dazu in der Lage, nebenläufige sowie parallele Prozesse zu entwerfen.

5.6.19 Bibliotheken entwickeln

Die Studierenden sind dazu in der Lage, eigene Bibliotheken zu entwickeln.

5.6.20 Methoden aufrufen

Studierende sind dazu in der Lage, Methoden aufzurufen.

5.6.21 Objekte erzeugen

Studierende sind dazu in der Lage, Funktionalitäten in Klassen zu kapseln und Objekte zu erzeugen

5.6.22 Schnittstellen entwerfen

Studierende sind dazu in der Lage, gemäß der Aufgabenstellung eine passende Schnittstelle zu entwerfen.

5.6.23 Transfer von Wissen und Erfahrung auf neue Aufgaben und Programmiersprachen

Die Studierenden sind dazu in der Lage, neues kognitives Wissen zu schaffen, indem Wissen über andere kognitive Aufgaben in einem neuen Kontext angewendet wird. (Die Beziehung zwischen Individuum, Fachdisziplin und Kognition wird hieran deutlich. Zum Transfer wird das Wissen über das eigene kognitive System infrage gestellt und reflektiert, bevor eine Anwendung in einem anderen Kontext oder in der Berufspraxis erfolgen kann. Das Ziel dabei ist es, Wissen über das neue Wissen zu generieren, bzw. bestehendes Wissen zur Generierung neuen Wissens zu nutzen.)

5.6.24 Selbstständige Organisation des Lernprozesses (unter Anwendung externer Ressourcen)

Die Studierenden sind dazu in der Lage, ihren Lernprozess selbstständig zu organisieren, indem sie zum Beispiel externe Ressourcen (Literatur, Dokumentationen, Handbücher, etc.) zur Informationsgewinnung nutzen (und damit ihr Wissen über das eigene kognitive System nutzen und erweitern („Jedes Tun ist Erkennen, und jedes Erkennen ist Tun" - Maturana & Varela 1987, S. 31)[6].

5.6.25 Abstraktion von Problemen

Die Studierenden sind dazu in der Lage, Ihnen unbekannte Probleme zu analysieren und zu restrukturieren, sodass sie bekannte Muster wiedererkennen. Aufgrund dem dazu benötigten Wissen über Kognition und dem dadurch entstehenden neuen Wissen über Kognition (sprich über Muster und immer wiederkehrende Schemata) erfolgt die Einordnung in die metakognitive Wissensdimension.

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