Raphael Zender et al. (Hrsg.): Die 18. Fachtagung Bildungstechnologien (DELFI),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020 187

Kompetenzmodellierung für die grundlegende Programmierausbildung

Eine kritische Diskussion zu Vorzügen und Anwendbarkeit der Anderson Krathwohl Taxonomie im Vergleich zum Kompetenzmodell der GI

Natalie Kiesler[1]

Abstract: Dieser Forschungsbeitrag untersucht Kompetenzanforderungen an Informatik-Studierende in der grundlegenden Programmierausbildung. Basierend auf einer qualitativen Inhaltsanalyse aktueller Lehr- und Lernziele deutscher Hochschulen und der anhand von Interviews erhobenen Perspektive von Hochschullehrenden werden angestrebte Programmierkompetenzen der Basisausbildung identifiziert. Da das Kompetenzmodell der Gesellschaft für Informatik diverse Defizite aufweist, werden die entwickelten inhaltlichen Kategorien in die Anderson Krathwohl Taxonomie (AKT) kognitiver Bildungsziele eingeordnet. Daraufhin erfolgt eine Überarbeitung der Dimensionen und Subtypen der AKT hin zu einem für die Informatik spezifischen Modell mit dem Ziel, Programmierkompetenzen gemäß ihrer kognitiven Komplexität klassifizieren zu können. Die präsentierte Handreichung ist auf den Ebenen Lehrveranstaltungskonzeption sowie Entwicklung und Bewertung von Assessments zur Messung von Programmierkompetenzen anwendbar.

Keywords: Programmierkompetenz, Wissensdimensionen, kognitive Prozessdimensionen

1 Einleitung

Der Entwicklung kontextspezifischer Kompetenzmodelle [Kl04, We01] wurde in der Informatik bisher wenig Beachtung geschenkt. Zum einen ist die Informatik eine sich stets weiterentwickelnde Fachdisziplin, zum anderen benötigen valide Instrumente zur Kompetenzmessung empirische Daten als Basis [Ko08]. Aktuell werden Kompetenzmodelle für die universitäre Informatik-Ausbildung nur in vereinzelten Forschungsarbeiten berücksichtigt [Li13, KTB16, Sc12]. Durch die Gesellschaft für Informatik (GI) wurde ein Kompetenzmodell als Empfehlung für die Entwicklung von Kompetenzen im Informatik-Studium vorgestellt [Ge16]. Obwohl deren Modell auf der Anderson Krathwohl Taxonomie (AKT) [AK01] basiert, werden einige Wissensdimensionen sowie kognitive Prozessdimensionen nicht berücksichtigt und bezüglich Kontextspezifika der Informatik fehlinterpretiert. Die vorliegende Forschungsarbeit untersucht daher die Anwendbarkeit der AKT [AK01] als anerkannten Rahmen zur Kategorisierung kognitiver Kompetenzen in der grundlegenden Programmierausbildung - dem Kern eines jeden Informatik-Studiums.

[1] Goethe-Universität Frankfurt, Akademie für Bildungsforschung und Lehrerbildung, Juridicum, Senckenberganlage 31-33, 60325 Frankfurt am Main, kiesler@em.uni-frankfurt.de

2 Kompetenzmodelle in der Informatik

Ziel des von der GI [Ge16] entwickelten, an die AKT angelehnten Modells ist es, die in Informatik-Studiengängen angestrebten kognitiven Kompetenzen zu beschreiben um Hilfestellung zur Gestaltung, Weiterentwicklung und Beurteilung derselben zu ermöglichen [Ge16]. Das Modell differenziert in der ersten von zwei Zeilen vier kognitive Prozessdimensionen: *Verstehen*, *Anwenden*, *Analysieren* und *Erzeugen*. Durch die beiden Zeilen soll zwischen *Typen des Wissenschaftlichen Arbeitens* mit den Subtypen T1 bis T6 unterschieden werden, die durch *Stufen der Kontextualisierung von Anwendungen* (K1 bis K5) und *Wissensdimensionen* (W1 bis W4) charakterisiert werden. Erläuterungen zur Bedeutung der Wissensdimensionen für die Informatik und zur reduzierten Darstellung in Form von zwei Zeilen fehlen jedoch. Implizierte Annotationen zu den jeweiligen Typen T, K und W in den Beispielen der 17 Inhaltsbereiche bleiben aus. Definierte Kategorien (K5) bleiben ungenutzt und die Entwicklung metakognitiver Kompetenzen scheint ausschließlich der Promotion vorbehalten zu sein. Indem die Klassifizierungen in den Beispielen von den Autoren selbst nicht präzise und nachvollziehbar angewendet werden können, offenbart sich eine der größten Schwächen des Modells. Lediglich kognitive Prozessdimensionen werden in den Beispielen erkennbar (wenn auch nicht nachvollziehbar) zugeordnet.

Die zweite Zeile des Modells beinhaltet weitere Stufen kognitiver Prozessdimensionen: 2a *Übertragen* und 3a *Bewerten*. Die aus der AKT bekannte Dimension *Erinnern* wird wegen vermeintlicher Redundanz nicht berücksichtigt. Die Felder in der zweiten Zeile unter Stufe 1 und 4 bleiben leer, weil „sich die Prozessdimension ‚Verstehen' eher auf grundlegende Zusammenhänge beziehen soll und Stufe 4 ‚Erzeugen' im Bachelor-Studiengang kaum grundlegende wissenschaftliche Innovation in komplexeren Anwendungszusammenhängen erwarten lässt" [Ge16, S. 10]. Demnach wird die Dimension *Erzeugen* in Bachelor-Studiengängen nicht angestrebt und bleibt ungenutzt. Spätestens dadurch wird die Fehleinschätzung zum kognitiven Anspruch allein der Programmierung deutlich.

Gegenläufig zum GI-Modell liefert die AKT bereits ein komplexes Kontinuum kognitiver Kompetenzen und Wissensdimensionen. Eine weitere, explizite Differenzierung nach Kontextualisierung von Inhalten durch die GI ist nicht notwendig, da die Wissensdimensionen kontextspezifisches Wissen anhand der Subtypen adressieren. Darüber hinaus fasst das GI-Modell an anderer Stelle Dimensionen zusammen, die zu differenzierende kognitive Prozesse abbilden, wie zum Beispiel *Erinnern* und *Verstehen*. In der Informatik und speziell der Programmierung müssen Studierende stets grundlegende Fakten und Informationen aus dem Langzeitgedächtnis abrufen, so z. B. Schlüsselworte, Operatoren, Wertebereiche, etc. Das Erinnern an derartige Details ist ohne jegliches Verständnis möglich, sodass die Position der GI weiter konterkariert wird. Besonders scharfe Kritik bedarf die Position zur Kategorie des Erzeugens. Während die GI damit einhergehende kognitive Prozesse Bachelor- und Master-Arbeiten vorbehält, vertritt die Autorin die Position, dass bereits das Schreiben kleiner Algorithmen und Programme planerische, gestalterische und konstruktive Leistungen zur Produktion neuer Lösungen erforderlich macht. Diese kognitiven Prozesse sind entsprechend der AKT in die Dimension des *Erzeugens* einzuordnen.

Zusätzlich werden metakognitive Kompetenzen im Bachelor-Studium benötigt, um Lernprozesse zu organisieren, systematisch Probleme zu lösen und Selbsterkenntnis zu erreichen. Laut GI werden diese Kompetenzen erst ab der Promotion ausgebildet [Ge16].

Da die Einordnungen der GI entlang der verbleibenden kognitiven Prozessdimensionen nicht den Typen, Subtypen und Kategorien kognitiver Prozesse entsprechen, bedarf es einer Handreichung für die Klassifizierung von Kompetenzen entlang der AKT-Matrix in der Informatik. Besonders die Programmierung kann von einem komplexeren Modell in Anlehnung an die AKT profitieren, um die Gestaltung von Lehrangeboten für NovizInnen zu optimieren, da relevante kognitive Prozesse und Wissensdimensionen darin abgebildet werden. So kann zum einen das Bewusstsein von Lehrenden für den kognitiven Anspruch der Programmierung geschärft, zum anderen können Aufgaben entlang der verschiedenen Dimensionen entwickelt und damit lernförderlicher gestaltet werden. Zukünftige (Self-) Assessments könnten Programmierkompetenzen damit differenzierter abbilden.

3 Forschungsdesign und Ergebnisse

Die Anderson Krathwohl Taxonomie zur Klassifikation von Bildungszielen (AKT) bildet kognitive Prozesse und Wissensdimensionen in ausreichender Komplexität unter Berücksichtigung von Fachspezifika ab. Eine Reduktion in Kombination mit neuen Dimensionen zur Kontextualisierung, wie im GI-Modell gezeigt, erscheint paradox. Aufgrund der Fehlinterpretationen der GI wird die AKT vor dem Hintergrund der kognitiven Anforderungen in der Programmierung als Kern jedes Informatik-Studiums neu bewertet. Folgende Forschungsfragen werden untersucht: *(1) Inwieweit kann die Anderson Krathwohl Taxonomie in der Grundlagenausbildung der Programmierung Anwendung finden, um Programmierkompetenzen zu klassifizieren? (2) Wie fachspezifisch ist die AKT zu begreifen?*

Zunächst wurde eine qualitative Inhaltsanalyse aktueller Kompetenzziele deutscher Hochschulen im Bereich der grundlegenden Programmierausbildung durchgeführt. Ergänzend wurde anhand von Experten-Interviews die Perspektive von Hochschullehrenden erfasst. Die zusammengefassten und abstrahierten inhaltlichen Kategorien von Kompetenzen wurden in die Dimensionen AKT eingeordnet. Daraufhin erfolgte eine Überarbeitung der Dimensionen und Subtypen der AKT hin zu einem für die Informatik spezifischen Modell.[2]

Als Ergebnis zeigt Abb. 1 die neu interpretierten Wissensdimensionen der AKT mitsamt Subtypen für die Basisausbildung der Programmierung. Darüber hinaus weist Abb. 2 die entsprechende Bedeutung der kognitiven Prozessdimensionen aus, indem Definitionen und Beispiele präsentiert werden. Die beiden Tabellen sind als kontextspezifische Adaption der AKT für die grundlegende Programmierung zu verstehen und können als Handreichung zur Klassifikation von Kompetenzzielen dieser Domäne genutzt werden.

[2] Die Klassifizierung aller kognitiven Kompetenzen sowie die vollständige Übersicht nicht-kognitiver Kompetenzen wurde bereits veröffentlicht [Ki20]. Die Analyseergebnisse sind anhand des Codebuchs bzw. der Übersicht codierter Segmente online verfügbar (https://github.com/nkiesler-cs/delfi2020.git).

Typen und Subtypen	Definition und Beispiele für die Programmierung
A. Faktenwissen – Grundlegende Inhalte und Details, die Studierende für die Einarbeitung in ein Thema wissen müssen	
AA. Wissen um Terminologie	Fachvokabular (z. B. Sprachelement wie Schlüsselwörter, Escape-Sequenzen, Literale, binäres Zahlensystem, Operatoren, elementare Datentypen, Ausnahmen einer Programmiersprache)
AB. Wissen um spezifische Details und Elemente	Fachspezifische Ressourcen und Quellen (z. B. Geschichte der Informatik, des Computers, der Programmiersprachen, Phasen der Softwareentwicklung; Kenntnis zuverlässiger Lehrbücher und Werkzeuge zum Erlernen des Programmierens
B. Konzeptionelles Wissen – Übergeordnete Zusammenhänge zw. Elementen als Teile einer übergeordneten Struktur	
BA. Wissen um Klassifikationen und Kategorien	Wissen um Programmierparadigmen, Klassen der Zeitkomplexität und algorithmischen Effizienz, Eigenschaften von Algorithmen, Generationen von Programmiersprachen, numerische Datentypen, Chomsky-Hierarchie als Klassifikation für formale Sprachen
BB. Wissen um Prinzipien und Verallgemeinerungen	Kenntnisse der Prinzipien von Programmierparadigmen und Programmiersprachen (z. B. Typsysteme, Namensgebung, Funktionen, Parameterübergabe, Speicherverwaltung)
BC. Wissen um Theorien, Modelle und Strukturen	Kenntnis der Komplexitätstheorie (Effizienz, Laufzeitkomplexität), Modelle von Computerarchitekturen, Kenntnis von Speichermodellen und Rechnermodellen (z. B. Turingmaschine)
C. Prozedurales Wissen – Methoden und Regeln für die Anwendung von Algorithmen, Techniken, Strategien	
CA. Wissen um fachspezifische Strategien und Algorithmen	Syntax und Semantik von Programmiersprachen, Algorithmen für typische Probleme kennen, Code schreiben, Umwandeln von Dezimalzahlen in Fließkommazahlen (oder anderes Zahlensystem)
CB. Wissen um fachspezifische Techniken und Methoden	Konventionen für guten Programmierstil, Methodik zur Analyse von Problemen und Algorithmen, Werkzeuge zur Softwareentwicklung kennen und nutzen, Modellierung und Entwurf von Algorithmen, Algorithmen testen und debuggen
CC. Wissen um Kriterien zur Bestimmung der Eignung von Verfahren	Angemessene Verwendung eines Entwurfsmusters oder Algorithmus zur Lösung eines unbekannten Problems (z. B. Auswahl eines problemadäquaten Entwurfsmusters oder Sortierverfahrens)
D. Meta-kognitives Wissen – Allgemeines Wissen über Kognition sowie Bewusstsein und Kenntnis der eigenen Kognition	
DA. Strategisches Wissen	Wissen um Organisation von Lernprozessen und wann welche Lernstrategien für welchen Zweck sinnvoll eingesetzt werden sollten
DB. Wissen über kognitive Aufgaben	Wissen über Bedingungen kognitiver Aufgaben (z. B. nicht-lösbare Probleme, wann zusätzliche Ressourcen zur Problemlösung nötig werden, wann Möglichkeiten zum Transfer von Wissen auf neue Aufgaben bestehen, Bewusstsein über systematische Ansätze zur Problemanalyse und -lösung)
DC. Selbsterkenntnis	Reflexion persönlicher Stärken und Schwächen, Bewusstsein über verwendete Strategien, Angemessenheit des Selbstvertrauens und der Selbstwirksamkeit, Bewusstsein über Ziele, Motivation und persönliche Interessen, Übernahme von Verantwortung für Lernprozesse

Abb. 1: Wissensdimensionen in der Programmierausbildung

4 Diskussion

Die Einordnung der Programmierkompetenzen in die AKT macht den hohen kognitiven Anspruch an StudienanfängerInnen der Informatik im Rahmen der Programmierausbildung deutlich. In kaum einem anderen Studienfach (mit Ausnahme weniger anderer Natur- bzw. Ingenieurswissenschaften) werden von Novizen in derartigem Umfang konstruktive Leistungen erwartet. In der Programmierung stellt das Schreiben eigenen Codes als Methode zur Problemlösung eine konstruktive Leistung dar. Paradoxerweise wird in den analysierten Daten nur in geringem Ausmaß Faktenwissen und Konzeptionelles Wissen auf der Ebene des Erinnerns und Verstehens ausgezeichnet. Gleichzeitig häufen sich Kompetenzziele in der Dimension *Erzeugen* von *Prozeduralem Wissen*. Die explizite Ausformulierung niedrigerer kognitiver Prozesse dieser Wissensdimension wurde häufig ausgespart. Wenngleich diese Aussparungen anteilig aufgrund des inkludierenden Charakters der höheren Dimensionen zu erwarten waren, sollten Ziele in niedrigeren kognitiven Prozessdimensionen explizit formuliert werden. Die fehlende Sichtbarmachung der weniger

Kognitive Prozesse und Alternativen	Definition und Beispiele für die Programmierung
1. Erinnern – Aufrufen von bekannten Informationen aus dem Langzeitgedächtnis	
1.1 Erkennen Identifizieren	Zugreifen auf Wissen des Langzeitgedächtnisses, das mit vorgelegtem Material konsistent ist (z. B. Daten wichtiger Ereignisse kennen: wann der Computer erfunden wurde und von wem)
1.2 Erinnern Abrufen	Abrufen von relevantem Wissen aus dem Langzeitgedächtnis (z. B. Erinnern an Daten wichtiger Ereignisse in der Geschichte der Informatik)
2. Verstehen – Ableiten von Bedeutung aus Instruktion und Kommunikation	
2.1 Interpretieren Erklären, Umschreiben, Darstellen, Übersetzen	Wechseln von einer Darstellungsform (numerisch, schriftlich) zu einer anderen (verbal) (z. B. paraphrasieren von Literatur, Dokumentationen, Handbüchern; schriftliche Aufgabenstellung mündlich beschreiben)
2.2 Verdeutlichen Illustrieren, Instanziieren	Aufzeigen eines spezifischen Beispiels bzw. Illustration eines Konzepts oder Prinzips (z B. Stack als Beispiel eines abstrakten Datentyps erläutern)
2.3 Klassifizieren Kategorisieren, Subsumieren	Zugehörigkeit zu einer Kategorie oder Klasse bestimmen (z. B. zu einer Klasse von Datentypen; Integer als primitiven Datentyp klassifizieren)
2.4 Zusammenfassen Abstrahieren, Generalisieren	Zusammenfassen eines allgemeinen Themengebiets oder Motivs, bzw. eines oder mehrerer wichtiger Punkte (z. B. Kernkonzepte der jeweiligen Programmierparadigmen zusammenfassen)
2.5 Entnehmen Schlussfolgern, Extra-/Interpolieren, Vorhersagen	Aus präsentierten Informationen logische Schlussfolgerungen ziehen (z. B. Zustände von zwei Booleschen Variablen logisch verknüpfen und Wahrheitswert ableiten)
2.6 Vergleichen Gegenüberstellen, Abgleichen, Zuordnen	Erkennen von Übereinstimmungen zwischen zwei Ideen, Objekten, Konzepten (z. B. Gegenüberstellen von Datentypen, Datenstrukturen oder Algorithmen; Von-Neumann und Harvard-Architektur vergleichen)
2.7 Erklären Konstruieren von Modellen	Konstruktion eines Modells zu Ursache/Wirkung eines Systems (z. B. unterschiedlichen Speicherplatzverbrauch und Laufzeit von Iteration vs. Rekursion erklären; Von-Neumann-Architektur erklären)
3. Anwenden – situative Ausführung, Anwendung oder Implementierung von Prozessen, Vorgängen oder Strategien	
3.1 Ausführen Durchführen	Anwenden eines Verfahrens auf eine bekannte Aufgabe (z. B. Dezimalzahlen in Dualzahlen, Oktalzahlen, Hexadezimalzahlen umkodieren)
3.2 Umsetzen Verwenden	Anwenden eines Verfahrens auf eine unbekannte Aufgabe (z. B. Qualitätskriterien und Programmierkonventionen auf eigenen Quellcode anwenden)
4. Analysieren – Zerlegung von Material in Teile und logische Erklärung durch Klarlegung derer Beziehung, Bedeutung im Gesamtkontext, Organisation oder Charakterisierung	
4.1 Differenzieren Unterscheiden, Orientieren, Auswählen	Unterscheiden zwischen relevanten und irrelevanten bzw. wichtigen und unwichtigen Teilen eines präsentierten Materials (z. B. eine Problemstellung zerlegen und zur Problemlösung relevante Aspekte auswählen)
4.2 Organisieren Zusammenhänge finden, integrieren, skizzieren, zerlegen, strukturieren	Bestimmen, wie Elemente innerhalb einer Struktur zusammenpassen oder funktionieren (z. B. Bestandteile fremden Codes zerlegen und dadurch dessen Funktionsweise skizzieren).
5. Bewerten – begründete Auswahl aus mehreren Alternativen, Beurteilung durch Überprüfung von Kriterien und Standards	
5.1 Prüfen Koordinieren, Entdecken, Überwachen, Testen	Erkennen von Inkonsistenzen in Prozessen oder Produkten; Erkennen der Wirksamkeit eines Verfahrens während der Umsetzung (z. B. Testen von Algorithmen und Programmen auf Korrektheit und Eigenschaften, Fehlersuche in Programmen, Verantwortung übernehmen für Lernerfolg)
5.2 Kritisieren Beurteilen	Erkennen von Inkonsistenzen zwischen einem Produkt und extern formulierten Kriterien oder Standards; Erkennen positiver und negativer Eigenschaften eines Produkts; Erkennen der Angemessenheit eines Verfahrens für ein Problem (z. B. Beurteilen, welcher Algorithmus bzw. welches Programm das Angemessenste ist, um ein bestimmtes Problem zu lösen; Komplexität von Algorithmen beurteilen)
6. Erzeugen – Zusammenstellung von Elementen und Einzelteilen zu einem neuen, funktionierenden Ganzen, (Einzigartigkeit, Originalität) durch Planung, mentale (Re-)Strukturierung, Konzeption, Produktion	
6.1 Generieren Hypothesen bilden	Probleme auf neue Art darstellen und dadurch alternative Hypothesen und Möglichkeiten zur Problemlösung aufstellen; Grenzen von Vorwissen und Theorien werden überschritten (z. B. Modellieren von Problemen; Verschiedene neue Algorithmen zur Lösung eines Problems zusammentragen (Iteration vs. Rekursion); Transfer von Erkenntnissen auf neue Probleme und deren Lösung)
6.2 Planen Designen	Bewusste oder unbewusste Ausarbeitung eines geeigneten Verfahrens zur Erfüllung einer Aufgabe, ggf. Festlegen von Teilzielen und Arbeitsschritten (z. B. Modellieren eines Programms; Algorithmenentwurf; Datenstrukturen entwerfen; Schnittstellen entwerfen)
6.3 Produzieren Konstruieren	Probleme auf Arbeitsplan zur Problemlösung verfolgen und damit ein Produkt erfinden oder entwickeln, das den gestellten Anforderungen gerecht wird (z. B. Programmiersprachliche, lauffähige Lösungen für Probleme schreiben; GUIs programmieren; Systematik zur Problemlösung entwickeln)

Abb. 2: Kognitive Prozessdimensionen in der Programmierausbildung

komplexen kognitiven Kompetenzen könnte eine Ursache für zu hohe Erwartungen an ProgrammieranfängerInnen sein, die von McCracken et al. [Mc01] identifiziert wurden.

Weiterhin wird durch den hohen Abstraktionsgrad der AKT eine Gültigkeit für alle Fachgebiete erreicht. Jedoch können leicht missverständliche Interpretationen erfolgen, wie das angelehnte GI-Modell zeigt. Die erschwerte Zuordnung der einzelnen Dimensionen ist vor allem der Ambiguität der Schlüsselworte geschuldet. Einige vermeintliche Beispiele, wie etwa „Konzept“, können sogar in die Irre führen, da sie eben nicht zwangsläufig auf *Konzeptionelles Wissen* verweisen. Daher ist eine fachspezifische Variation der AKT, insbesondere für die Informatik unabdingbar. Wie das adaptierte Beispiel in Tabelle 1 und 2 zeigt, kann die AKT in der Grundlagenausbildung der Programmierung durchaus Anwendung finden, um Programmierkompetenzen zu klassifizieren. Die allgemeinen und stark abstrahierten Dimensionen bedürfen lediglich einer fachspezifischen Adaption um auch in der Informatik Anwendung zu finden, z. Bsp. bei der Überarbeitung von Curricula oder der Entwicklung von (adaptiven) Assessments.

Literaturverzeichnis

[AK01] Anderson, L. W.; Krathwohl, D. R.: A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom’s Taxonomy of Educational Objectives. 2001.

[Ge16] Gesellschaft für Informatik e.V.: Empfehlungen für Bachelor und Masterprogramme im Studienfach Informatik an Hochschulen. 2016.

[Ki20] Kiesler, N.: Towards a Competence Model for the Novice Programmer Using Bloom’s Revised Taxonomy. In: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. S. 459-465, 2020.

[Kl04] Klieme, E.: Was sind Kompetenzen und wie lassen sie sich messen? Pädagogik, 56(6):10-13, 2004.

[Ko08] Koeppen, K. et al.: Current issues in competence modeling and assessment. Zeitschrift für Psychologie/Journal of Psychology, 216(2):61-73, 2008.

[KTB16] Kramer, M.; Tobinski, D. A.; Brinda, T.: Modelling Competency in the Field of OOP: From Investigating Computer Science Curricula to Developing Test Items. In: Stakeholders and Information Technology in Education. S. 1-8, 2016.

[Li13] Linck, B. et al.: Competence model for informatics modelling and system comprehension. In: IEEE Global Engineering Education Conference. S. 85–93, March 2013.

[Mc01] McCracken, M. et al.: A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In: Working Group Reports from ITiCSE. S. 125-180, 2001.

[Sc12] Schäfer, A. et al.: The Empirically Refined Competence Structure Model for Embedded Micro- and Nanosystems. In: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education. S. 57-62, 2012.

[We01] Weinert, F.: Concept of competence: A conceptual clarification. In: Defining and selecting key competencies. 2001.