Steffen Büchner

Ermittlung zentraler Konzepte der Entwicklung eingebetteter Systeme anhand eines fachdidaktisch begründeten Kriterienkataloges

universi
UNIVERSITÄTSVERLAG SIEGEN

Ermittlung zentraler Konzepte der Systemarchitektur eingebetteter Systeme anhand eines fachdidaktisch begründeten Kriterienkataloges

Genehmigte
DISSERTATION
zur Erlangung des Grades eines Doktors
der Naturwissenschaften

vorgelegt von
Dipl.-Inform. Steffen Büchner

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2014

Universitätsverlag Siegen - universi
Adolf-Reichwein-Str. 2
57076 Siegen
info@universi.uni-siegen.de

ISBN: 978-3-936533-58-3

Zugl.: Siegen, Univ., Diss., 2015
Erste Gutachterin: Prof. Dr. Sigrid Schubert
Zweiter Gutachter: Prof. Dr. Rainer Brück
Vorsitzende: Prof. Dr. Sigrid Schubert
Tag der mündlichen Prüfung: 20. Februar 2015.

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Kurzfassung

Die vorliegende Arbeit stellt ein Konzept zur Ermittlung zentraler Methoden und Sichtweisen für die Entwicklung eingebetteter Systeme vor. Ein solcher Beitrag ist notwendig, da eingebettete Systeme ein technologisch schnelllebiges Anwendungsgebiet darstellen, für das die Auswahl langfristig relevanter Lerninhalte schwierig ist.

Die Basis der Forschungsmethodik bildet zum einen das DFG-Projekt „Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen“ (KOMINA), in dem ein Kompetenzstrukturmodell für den Anwendungsbereich erstellt wurde, und zum anderen die von Schwill vorgestellte Methodik der „fundamentalen Ideen der Informatik“. Kern des Vorgehens ist es, nicht mehr eine spezifische Technik, sondern die mit ihr verbundene Idee an die Studierenden zu vermitteln. Die Fundamentalität jeder dieser Ideen kann durch fachdidaktisch begründete Kriterien geprüft werden. Alle bisherigen Arbeiten zur Ermittlung fundamentaler Ideen entstanden mit Fokus auf ein schulisches Bildungsniveau, weswegen der Kriterienkatalog in der vorliegenden Arbeit auf die Eignung im Hochschulkontext überprüft, angepasst und erweitert wurde. Daraus resultiert eine Sammlung von fünf Kriterien, mit denen sich prüfen lässt, ob eine Idee fundamental für die Entwicklung eingebetteter Systeme ist. Das Horizontalkriterium sichert die breite Anwendbarkeit der Idee und ihren Bezug zu den Ebenen des Entwicklungsvorgehens. Im Fortbildungskriterium wird analysiert, inwiefern die Vermittlung der Idee auf Bachelorebene möglich ist und welche Grundlage sie für andere Ideen bereitstellt. Das Zeitkriterium überprüft, wie lange eine Idee im Anwendungsbereich auch in der Vergangenheit beobachtbar war und welche Tendenz sich daraus für die kommenden Jahre ableiten lässt. Da Studierende nach ihrer Fachausbildung entweder im industriellen oder im wissenschaftlichen Bereich arbeiten werden, analysiert das Sinnkriterium die Relevanz der Idee für diese beiden Berufswege. Das Varianzkriterium greift als neues Kriterium die Bezüge zwischen anderen Ideen auf und analysiert, inwiefern eine Idee bereits in anderen Konzepten vorhanden ist.

Anstatt einer subjektive Zusammenfassung des Anwendungsgebietes als Ausgangslage für die Ideenanalyse wurde in dieser Arbeit eine Sammlung wissenschaftlicher Fragestellungen und Forschungsgebiete normativ ausgewertet. Hierzu wurden die ACM „Transactions on Embedded Computing“ über den Zeitraum der letzten fünf Jahre analysiert und insgesamt zehn fundamentale Ideen ermittelt. Neben formalisierten Beurteilungsrichtlinien für jedes Kriterium wurde die Anwendbarkeit der fundamentalen Ideen durch die Ableitung einer Veranstaltungsstruktur verdeutlicht. Dabei geben die Kriterien eine logische Ordnung für die inhaltliche Organisation der Veranstaltungsstruktur vor, die direkt aus der Analyse eines Themengebietes ermittelt werden kann. Erstmalig ist das Konzept somit nicht nur

auf Curriculaebene, sondern auch auf Veranstaltungsniveau theoretisch fundiert. Exemplarisch wurde eine Veranstaltung nach diesem Vorbild für die fundamentale Idee der rekonfigurierbaren Architekturen beschrieben.

Abstract

This thesis proposes a research concept to determine essential methods and points of views for the development of embedded systems. This contribution is necessary because embedded system development is affected by a fast-paced technology change which makes the selection of long lasting learning content difficult.

The DFG-funded project "competence development with embedded micro- and nanosystems" (KOMINA) in which a competence structure model has been proposed, as well as Schwill's research on fundamental ideas of informatics provide the basis of this thesis. The latter focuses on the ideas behind a technology rather than the ideas' specific application in order to determine learning topics which are in particular long lasting. A set of criteria is needed to distinguish fundamental from non-fundamental ideas. All previous research approaches which used the concept of fundamental ideas focused on primary and secondary education instead of higher education. Thus, every criterion had to be reviewed and adapted for this research project resulting in five criteria which describe fundamental characteristics of embedded system development. The horizontal criterion checks the ideas' broad applicability and its relation to the different steps in the development process. In order to satisfy the advanced training criterion, an idea has to be teachable to undergraduates and it has to be a foundation for further learning topics. The criterion of time requires that an idea has been of importance for embedded system development in the past and thus, will arguably be so in the future. Furthermore, the ideas are analyzed in terms of practical and theoretical relevance for embedded system development by the criterion of sense because two typical career paths for students include working in a research facility or in a company. The criterion of variance highlights the relationship between ideas and verifies that they have certain aspects which distinguish them from each other.

Instead of building on a subjective summary of the field of application, the approach used in this thesis centers around a normative analysis of scientific research areas. The ACM "Transactions on Embedded Computing" have therefore been analyzed over the last five years, resulting in ten fundamental ideas for the development of embedded systems.

Besides objective review guidelines for every criterion, this thesis also enhances the applicability of the fundamental ideas by proposing a course structure which can be directly derived from the analysis process. For the first time, the methodology of the fundamental ideas is therefore not only targeted at curricula creation but at the creation of lectures, too. This thesis gives an example on how such a course concept can be conceptualized for the idea of reconfigurable architectures.

Vorwort

Die vorliegende Dissertationsschrift ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl „Didaktik der Informatik und E-Learning“ der Universität Siegen und dem DFG-geförderten Forschungsprojekt „Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen“ entstanden.

Mein besonderer Dank gilt Prof. Dr. Sigrid Schubert, die mir in unseren Diskussionen hilfreiche Rückmeldungen zu Forschungsansätzen gegeben hat und die Arbeit hauptsächlich begleitete. Ebenso möchte ich Prof. Dr. Rainer Brück danken, der als Zweitbetreuer für Rückfragen zur Verfügung stand und durch kritische Anmerkungen den Forschungsprozess fokussiert und geschärft hat.

Ebenfalls möchte ich mich bei Andrea Baule, Deborah Amazu und Gerd Müller für ihre Arbeit am Lehrstuhl und für die moralische Unterstützung bedanken. Für die Möglichkeit zur Fachdiskussion möchte ich meinen Kollegen Steffen Jaschke und Andrè Schäfer danken, die durch ihre Argumente verschiedene wissenschaftliche Blickwinkel offengelegt und so die Schnittstellen zu verwandten Forschungsprojekten geprägt haben. Mein Dank gilt des Weiteren Philipp Reh als unermüdlicher Korrekturleser und kritischer Diskussionspartner sowie Armin Grünewald als Wissenschaftler, der sich insbesondere für die Beantwortung von Fachfragen viel Zeit genommen hat.

Danken möchte ich auch Jens Aßmann, der seit Beginn der Promotion fachlich wie methodisch kritische Nachfragen stellte und als geduldiger Zuhörer immer ein offenes Ohr für das Forschungsvorhaben hatte.

Ohne den familiären Rückhalt und die beständige Förderung durch meine Eltern und meine Schwester wäre dieses wissenschaftliche Projekt undenkbar gewesen. Daher möchte ich meiner Familie von ganzem Herzen danken.

Außerordentlicher Dank gilt meiner Verlobten Nicole für die ausdauernde Unterstützung auch über das Forschungsprojekt hinaus und ihren beständigen Zuspruch in den schwierigen Etappen. Danke!

Siegen, im April 2015

Steffen Büchner

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ACM	Association for Computing Machinery
ADC	Analog/Digital-Converter
API	Application Specific Interface
ASIC	Application Specific Integrated Circuit
BMBF	Bundesministerium für Bildung und Forschung
CAD	Computer-aided Design
CCS	Computing Classification System
CLB	Combined Logic Block
CPLD	Complex Programmable Logic Device
DAC	Digital/Analog-Converter
DFG	Deutsche Forschungsgemeinschaft
DSE	Design-Space-Exploration
DSL	Domain Specific Language
DSP	Digital Signal Processor
EAP	Entwurfs- und Anwendungspraktikum für eingebettete Mikrosysteme
EKSM	Empirisch verfeinertes KSM
ELVE	Explorative Lern- und Visualisierungsumgebung
EMNS	Eingebettetes Mikro- und Nanosystem
FPGA	Field Programmable Gate Array
GI	Gesellschaft für Informatik
GSM	Global System for Mobile Communication
HaPra	Hardwarepraktikum
HDL	Hardware Description Language
IDE	Integrated Development Environment
IDE	Integrierte Entwicklungsumgebung
IEEE	Institute of Electrical and Electronics Engineers

IP	Intellectual Property
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
KNM	Kompetenzniveaumodell
KOMINA	Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen
KSM	Kompetenzstrukturmodell
LUT	Lookup-Table
MBD	Modell-based Design
MoC	Model of Computation
MOOC	Massive Open Online Course
MPSoC	Multi-processor SoC
NKSM	Normatives KSM
NoC	Network-on-Chip
PCB	Printed Circuit Board
PWM	Puls-Weiten-Modulation
QDH	Qualifikationsrahmen für deutsche Hochschulabschlüsse
SoC	System-on-Chip
TECs	Transactions on Embedded Computing
UML	Unified Modelling Language
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VM	Virtual Machine
XML	Extensible Markup Language

1. Einleitung

Neben klassischen Anwendungsgebieten der Informatik wie Datenbanksysteme oder Netzwerktechnik hat sich in den vergangenen Jahren ein neues Anwendungsfeld an den Grenzen zu anderen Disziplinen wie der Elektrotechnik, der Physik oder der Regelungstechnik gebildet: eingebettete Systeme. Anders als bei herkömmlichen PCs dienen sie nicht der Datenverarbeitung, sondern der Überwachung, Regelung und Steuerung von Prozessen, wobei diese oft physikalischer oder mechanischer Natur sind. Eingebettete Systeme stellen keine Nischenanwendung dar, sondern finden sich in einer Vielzahl von Produkten und Systemen wieder, die einen breiten Anwendungsbezug aufweisen. Klassische Beispiele für eingebettete Systeme umfassen Technologien aus der Automobilbranche wie Airbags und Consumer-Produkte wie Router oder Steuerungseinheiten für industrielle Fertigungsanlagen. Die wirtschaftliche Relevanz eingebetteter Systeme lässt sich neben den weitläufigen Einsatzmöglichkeiten auch über nationale Förderungssummen bestimmen. Mit einem Marktvolumen von 19 Mrd. € (2010) und einer stetigen Wachstumsrate von neun Prozent bezeichnen manche Experten die damit verbundene Forschung und Entwicklung als die treibende Kraft deutscher Informations- und Kommunikationstechnologien [Bil+07], [Bak+10].

Durch neue technische Möglichkeiten einerseits, wie bspw. der schrumpfenden Transistorgröße oder der Synthetisierbarkeit abstrakter Sprachen hin zu maschinennahem Code, als auch durch die gesellschaftliche Bedeutung eingebetteter Systeme hat sich ein Forschungsfeld etabliert, das auf eigene Prozesse und Methoden zur Entwicklung eingebetteter Systeme angewiesen ist. Dessen rasche Entwicklung erschwert jedoch eine fundierte Ausbildung zukünftiger Experten und Fachkräfte. Die Begründung hierfür ist zum einen die kurze Zeitspanne, in der Methoden und Konzepte aktuell sind, und zum anderen die interdisziplinäre Natur des Forschungsfeldes. Ersteres ist problematisch, da ein Lerninhalt nach dem Studienende schon wieder überholt sein kann und damit keine praktische Relevanz mehr besitzt. Der zweite geschilderte Aspekt erschwert zuzüglich die Identifikation von Themenbereichen, die die Basis für ein Hochschulstudium bilden sollen, das nicht nur eine Sammlung an Lerninhalten anderer Disziplinen darstellt, sondern diese um eigenständige Fachbezüge erweitert. Umso wichtiger ist die Erforschung geeigneter Lehr-/Lernansätze und Technologien, um die Kompetenzen späterer Entwickler frühzeitig zu fördern. Bestehende didaktische Modelle der Informatik, Elektrotechnik, Mathematik und Physik sind unter diesen Aspekten kritisch zu überdenken und ggf. mit neuen Ansätzen zu ergänzen.

Der Bedarf neuer didaktischer Lehr-/Lernmethoden sowie die Zusammenführung und Neuaufstellung wichtiger Kompetenzen für Entwickler eingebetteter Systeme

zeigt sich ebenso deutlich in wissenschaftlichen Beiträgen beteiligter Fachdisziplinen. Die *Nationale Roadmap Embedded Systems* stellt in ihren abschließenden Empfehlungen fest:

> „Eine weitere Herausforderung stellt die Sicherung eines ausreichenden Angebotes qualifizierter Fachkräfte dar. Es gibt nur wenige Studiengänge, die die Bandbreite der für den Bereich Eingebettete Systeme notwendigen Wissensdomänen integrieren. Darüber hinaus ist der Bereich Systems-Engineering in der klassischen Ausbildung zu wenig verankert. Hier sind verstärkt koordinierte Anstrengungen zur Sicherstellung entsprechender Ausbildungsangebote auf allen Ausbildungsebenen einschließlich der beruflichen Weiterbildung erforderlich.“ [Dam+09, S. 126]

Weitere Forderungen nach einer didaktischen Fundierung und Erschließung der Disziplin finden sich ebenso in [Raj+10], [Est+02], [Cas+05] und [JC05].

Aufbauend auf wichtigen fachdidaktischen Vorarbeiten wie dem DFG-Forschungsprojekt „Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen“ (KOMINA) leistet diese Arbeit einen Beitrag, um die genannte Problemstellung zu lösen.

1.1. Forschungsziele und Forschungsmethodik

Das Ziel der vorliegenden Arbeit ist die Erarbeitung zentraler Entwicklungskonzepte, die ergänzend zu den Ergebnissen des KOMINA-Projektes für die Ausbildung zukünftiger Experten der Entwicklung eingebetteter Systeme hilfreich sind. Grundlage für alle damit verbundenen Forschungsschritte ist das empirisch verfeinerte Kompetenzstrukturmodell (EKSM), das im Rahmen des KOMINA-Projektes entwickelt wurde. Die mit dem Hauptforschungsziel verbundenen Teilziele sind:

I. Beobachtung des Hardwarepraktikums und Entwicklung fachdidaktisch begründeter Hilfsmittel.

   Das Kompetenzstrukturmodell bietet eine Strukturierung kompetenzorientiert formulierter Lerninhalte. Zur praktischen Validierung dieser Beschreibungen wurde eine an der Universität Siegen etablierte Laborveranstaltung für Studierende der Informatik und Elektrotechnik neu konzeptioniert. Um die Evaluation der Laborveranstaltung zu ermöglichen, ist eine Studierenden-Beobachtung zu entwerfen und durchzuführen. Erfahrungen daraus können genutzt werden, um lernförderliche Software und Hardware zu entwickeln, die auch außerhalb der Universität Siegen nutzbar sind und ein ganzheitliches Lernsystem zur Verfügung stellen. Marwedel betont die Schwierigkeit der Lehrveranstaltungskonzeption für den Bereich „eingebettete Systeme“ mit dem Verweis auf den erschwerten Zugang zu Hardwarebauteilen [Mar11]. Vorbereitungen für zukünftige Laborveranstaltungen können nur mit thematisch reduziertem Umfang durchgeführt werden, wenn die Studierenden keine Möglichkeit haben, auf die hochspezialisierte und meist teure Hardware im

Labor zuzugreifen. Ebenso problematisch ist die Nutzung ausgereifter CAD-Tools, deren Benutzungsschnittstelle und Bedienung viele der Studienanfänger überfordern. Dieses Teilziel schließt deswegen die Entwicklung fachdidaktisch begründeter Lehr-/Lernhilfen ein.

II. Erforschen einer fachdidaktischen Methode zur Bewältigung der Schnelllebigkeit technischer Konzepte und Sichtweisen für die Entwicklung eingebetteter Systeme.

    Für die Strukturierung der Fachdisziplin stellen Curricula einen begründeten themenzentrierten Leitfaden dar, der durch die Schnelllebigkeit der Disziplin jedoch ständig angepasst werden muss. Das *Computer Engineering Curriculum* der ACM/IEEE [ACM04] widmet diesem Umstand einen eigenen Diskussionspunkt, verfehlt das Ziel der Aktualität inzwischen aber selbst. Es besteht die Gefahr, dass Techniken und Methoden an Studierende vermittelt werden, die nicht mehr berufsrelevant sind, wenn die Studierenden das Studium beenden (vgl. [Cas+05]). Durch die jeweiligen Unternehmensausrichtungen werden Absolventen nach ihrem Studium mit einer Vielzahl verschiedener, unbekannter Werkzeuge und Entwicklungsvorgehen konfrontiert. Die Vermittlung aller möglichen Kombinationen kann im Studium allein aus Zeitgründen jedoch nicht vollzogen werden.

    Die bereits angesprochenen Herausforderungen für Projekte im eingebetteten Systembereich erfordern eine Verlagerung von technikspezifischen zu methodikspezifischen Inhaltselementen. Dazu ist es notwendig, aktuelle Technologien mit Hilfe eines auf die Zielgruppe angepassten didaktischen Werkzeugs zu analysieren. Hierdurch soll die konkrete Technik (bspw. die Programmiersprache Java) gegen deren Bezüge zu relevanten Methoden (bspw. Objektorientierung, Kapselung, etc.) abgegrenzt werden. Die so festgestellten Wissenselemente können in weiterführenden Arbeiten durch die Verknüpfung von Prozesswissen und nicht-kognitiven Elementen zu einer detaillierten Orientierungshilfe für Kompetenzzielstellungen im Bildungsbereich erweitert werden, ohne dabei auf bestimmte technologiespezifische Werkzeuge oder Methoden festgelegt zu sein.

III. Vorstellung von Lehr-/Lernempfehlungen auf Basis der erarbeiteten fachdidaktischen Methode.

    Die für die Erreichung des Teilziels II erforderliche fachdidaktische Methode ist im Kontext der Vorarbeit von KOMINA und den Gegebenheiten an der Universität Siegen konzeptionell umzusetzen, um ein Beispiel zur Vorlage ähnlicher Vorhaben zu geben. Die eingangs erwähnten Problemstellungen sind nicht nur auf Ebene der ausgewählten Lerninhalte, sondern auch hinsichtlich der Veranstaltungsstruktur zu berücksichtigen und durch Empfehlungen aus der Informatikdidaktik zu ergänzen.

Die Teilziele I und III erfordern eine Umsetzung, die sich an den Gegebenheiten der Universität Siegen orientiert. Teilziel II ist abstrakter formuliert, um für andere Disziplinen und Anwendungsgebiete als konzeptuelle Vorlage zu dienen.

Forschungsmethodik

Das empirisch verfeinerte Kompetenzstrukturmodell wurde in einem gemischt normativ-empirischen Prozess erforscht. Eine praktische Umsetzung der wichtigen Kompetenzbeschreibungen wurde im Rahmen der Restrukturierung des Hardwarepraktikums an der Universität Siegen erreicht.

Phase 1: Die erste Forschungsphase besteht aus der Konzeption und Durchführung einer Laborbeobachtung. Eng verknüpft mit dieser Phase des Forschungsprozesses ist die Entwicklung lehr-/lernförderlicher Hilfsmittel. Diese werden, fokussiert auf Zielgruppe und Anwendungsaufgabe, für das neu konzipierte Labor begründet, umgesetzt und anschließend hinsichtlich der Studierendenleistungen evaluiert. Nach dieser Phase ist Teilziel I des Forschungsprozesses erreicht.

Phase 2: Ausgehend von der Neukonzeption des Hardwarepraktikums musste in Phase zwei eine Neubewertung der Kompetenzbeschreibungen des empirisch verfeinerten Kompetenzstrukturmodells vorgenommen werden. Durch die unterschiedlichen Literaturquellen zeigt sich in den Kompetenzbeschreibungen des EKSM eine große Heterogenität, die die Umsetzung der Resultate in eine Lehrveranstaltung erschweren (siehe Kapitel 3). Dieser Problematik muss mit einer fachdidaktisch erprobten Methodik begegnet werden. Da für den Anwendungsbereich, wie auch für die Technische Informatik im Allgemeinen, wenig fachdidaktische Vorarbeit vorhanden ist, ist davon auszugehen, dass Methoden aus der Informatikdidaktik thematisch am ehesten für eine Adaption auf die in dieser Arbeit avisierte Zielgruppe geeignet sind.

Phase 3: Zusammen mit den bereits in KOMINA veröffentlichten Kompetenzbeschreibungen sind durch die Anwendung der in Phase zwei erforschten fachdidaktischen Methodik Empfehlungen für die intendierte Zielgruppe und das Anwendungsgebiet eingebettete Systementwicklung auf Hochschulebene zu formulieren. Durch die Verknüpfung dieser Empfehlungen untereinander wird ein wesentlicher Beitrag geleistet, um die praktische Umsetzbarkeit zentraler Lehrinhalte zu ermöglichen. Zusammen mit den Forschungsarbeiten aus Phase zwei ist damit das Teilziel II erfüllt.

Phase 4: Der letzte Abschnitt des Forschungsprozesses führt zu einer exemplarischen Lehrveranstaltungskonzeption auf Basis der in Phase drei vorgestellten Lehr-/Lernempfehlungen. Hierbei werden die Rahmenbedingungen und Vorwissensstände der Studierenden an der Universität Siegen berücksichtigt, ohne die Allgemeingültigkeit des Forschungskonzeptes für deutsche Hochschulen und Universitäten zu verwerfen. Teilziel III ist mit der Formulierung der praktischen Lehr-/Lernempfehlungen erfüllt.

Durch die Einbettung der Arbeit im Rahmen des KOMINA-Projektes ist der Beginn der Forschungsmethodik partiell vorgegeben. Der Forschungsprozess lässt sich deswegen grob in einen praktisch-empirischen (parallel zu KOMINA) und einen theoriefokussierten, analytischen Teil (nach KOMINA) gliedern. Parallel zu KOMINA sind all diejenigen Forschungsarbeiten einzuordnen, die sich mit der

Umsetzung des Hardwarepraktikums und der damit verbundenen Beobachtung beschäftigen. Die Entwicklung der Lehr-/Lernhilfsmittel gehört ebenso zur Projektphase von KOMINA. Diese Arbeiten wurden, wenn nicht anders angegeben, in Zusammenarbeit mit Steffen Jaschke und André Schäfer durchgeführt. Die Arbeitsschwerpunkte in diesen Abschnitten sind in den dazugehörigen Sektionen der Dissertation deutlich herausgestellt.

Nach KOMINA sind alle Forschungsarbeiten bezüglich der Erstellung von Lehr-/Lernempfehlungen wie die Bewertung des Hardwarepraktikums und des empirisch verfeinerten Kompetenzstrukturmodells, die Erforschung und spätere Adaption einer Explorationsstrategie auf den Kontext Hochschule, die praktische Anwendung der fachdidaktischen Methode und die daraus resultierende Sammlung an Empfehlungen sowie die Umsetzung exemplarisch gewählter Beispiele in einem Vorlesungskonzept anzusiedeln. Diese wurden vom Autor alleine erforscht, konzipiert und umgesetzt.

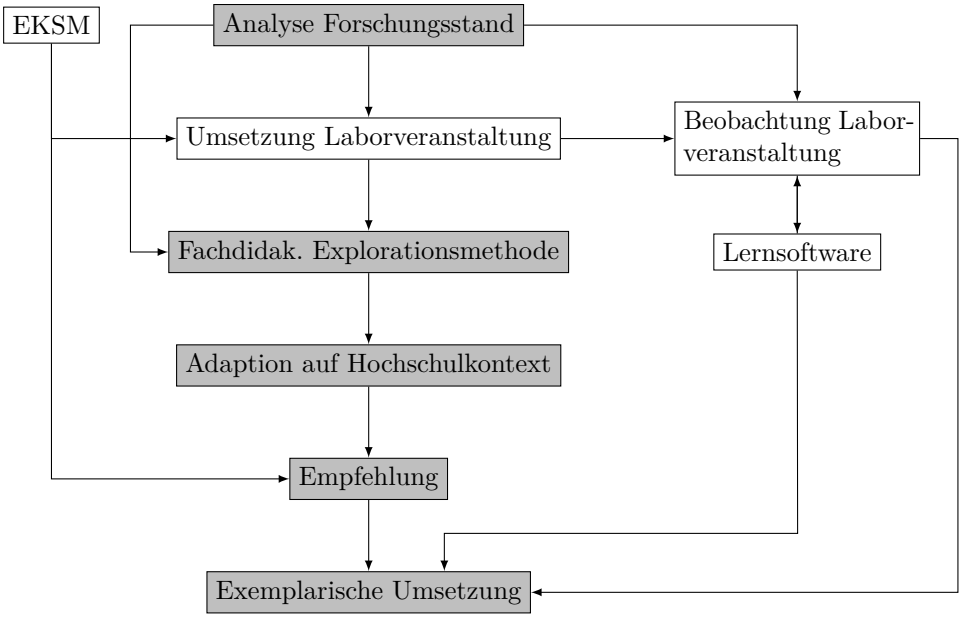

Abbildung 1.1.: Forschungsphasen und Forschungsmethodik

Die Abbildung 1.1 erläutert die Zusammenhänge zwischen den einzelnen Projektphasen grafisch. Die grauen Einträge stellen Arbeitsschritte dar, die der Autor außerhalb des KOMINA-Projektes allein durchgeführt hat. Einen Teil der Lernsoftware hat der Autor ebenso eigenständig entwickelt (siehe Abschnitt 3.4).

Das EKSM als Ausgangspunkt des Forschungsprozesses ist in jeden Teilabschnitt direkt oder indirekt eingebunden. Die Analyse des Forschungsstandes ist im Anwendungsbereich mit Hinsicht auf Vorarbeiten aus dem Bereich der Didaktik für Informatik wie auch für Technische Informatik durchzuführen. Die maßgebliche Zielstellung dieser Forschungsetappe stellt die Identifizierung und Eignungsprüfung

einer Methodik dar, die die in Teilziel II aufgezeigte Problematik löst. Durch die Beobachtung des Hardwarelabors werden Hinweise auf besonders schwierig zu vermittelnde Kompetenzen aufgedeckt. Die Entwicklung lehr-/lernunterstützender Systeme steht in wechselseitigem Bezug zur Leistung der Studierenden in der ersten und zweiten Labordurchführung und ist insbesondere auf die als problematisch erkannten Situationen hin konzipiert. Die Umsetzung der Empfehlungen erhält weitere empirische Zuarbeit aus der Beobachtung der Laborveranstaltung, wobei die bereits für das Hardwarelabor entwickelte Sammlung an Hilfssystemen wiederverwendet werden kann.

Die abschließende Bewertung des Forschungsziels ist nur eingeschränkt nach Abschluss dieser Arbeit möglich, da das Teilziel II und damit verbunden auch Teilziel III nur über einen Zeitraum mehrerer Jahre zu überprüfen sind. Möglichkeiten zur Kompetenzvalidierung sind in den entsprechenden Abschnitten jedoch beschrieben.

1.2. Rahmenbedingungen des Forschungsprozesses

Der nachfolgende Abschnitt erläutert das in dieser Arbeit angesetzte Verständnis über das Anwendungsgebiet und die zugehörige Zielgruppe.

Der interdisziplinäre Charakter eingebetteter Systementwicklung erschwert die Auswahl an Bildungsempfehlungen, da in der deutschen Hochschulbildung nur ein Zeitfenster von etwa drei Jahren (Bachelor-Regelstudienzeit) zu deren Vermittlung zur Verfügung steht, das ursprünglich für die Förderung *einer* Fachwissenschaft vorgesehen ist. Schaumont unterstreicht, dass es zeitlich nicht möglich ist, Studierenden jede mögliche technische Implementierungstechnik für eingebettete Systeme zu vermitteln [Sch08]. Auch die ACM/IEEE sprechen in ihrem neusten Curriculumsvorschlag die Problematik an, eine grundlegende Auswahl an Lerninhalten zu finden, deren zeitlicher Rahmen nicht über den aktuellen hinausgeht und somit die Studierenden nicht zusätzlich an die Universität bindet [SR13].

Jeder Entwicklungsprozess lässt sich in verschiedene Phasen gliedern und enthält somit implizit mögliche Schnittstellen zwischen thematisch oder zeitlich getrennten Entwicklungsaktivitäten (siehe Abschnitt 2.1). Eine Fokussierung auf einen kleinen Bereich eines solchen Entwicklungsprozesses ist notwendig, um keinen Überhang an Lehrmaterial für einen zu knapp bemessenen Zeitraum zu schaffen. Aus fachdidaktischer und fachlicher Sicht ist der Aspekt zu wählen, der die meisten Bezüge zu anderen Entwicklungsstadien aufweist, um Anknüpfungspunkte für weitere Forschungsprojekte zu ermöglichen.

Nach Meinung des Autors sind Systemarchitekten und die mit ihrer Arbeit verbundenen Entwicklungsschritte prädestiniert, um eine großflächige Abdeckung von Entwicklungskompetenzen zu fördern. Durch die Vermischung von Top-Down- und Bottom-Up-Techniken [WM00], [Sch+12c] und der damit verbundenen Betrachtung von Software, Hardware und Umgebungsparametern werden auf Ebene der Systemarchitektur alle zentralen Gebiete der Entwicklung thematisch berücksichtigt (vgl. [Cas+05]). Eine aktuelle Studie über die Arbeitsaufteilung in der eingebetteten

Systementwicklung bestätigt, dass mehr als die Hälfte aller Befragten in den Prozess der Architekturauswahl und -spezifikation mit einbezogen sind [Tec13]. Darüber hinaus ist der Entwurf einer der Entwicklungsschritte, in dem am meisten Zeit verbracht wird [Tec13]. Die Nationale Roadmap Embedded Systems sieht Architekturprinzipien als ein Hauptforschungsziel, welches in der deutschen Industrie bis 2019 verstärkt gefördert wird [Dam+09]. Die Begründung, Kompetenzen und Lehr-/Lerninhalte im Kontext von Design und Systemarchitektur didaktisch zu erforschen, ist somit gegeben. Die Auswahl des Themenbereiches lässt sich auch organisatorisch durch das KOMINA-Projekt stützen. Ein Großteil besonders gut bewerteter Kompetenzen findet sich im Bereich der Analyse und des Entwurfs von Systemen (siehe Kapitel 3.2 und [Sch+12c]).

Die Gesellschaft für Informatik hat in Ihren Curriculaempfehlungen der Technischen Informatik [Hoc+11] eine Zielgruppen-Klassifikation genutzt, die auch für die vorliegende Arbeit verwendet werden kann. Studierende werden demzufolge prinzipiell in drei Gruppen eingeteilt, die sich in ihrem Verhältnis von Haupt- zu Nebenfach unterscheiden.

> „Typ 1: Studiengänge Informatik
> Typ 2: Informatik-Studiengänge mit einem speziellen Anwendungsbereich
> Typ 3: Interdisziplinäre Studiengänge mit einem Informatikanteil, der mit dem Anteil der anderen beteiligten Fachdisziplinen gleichgewichtig ist.“ [Inf05, S.11]

Die Zielgruppe dieser Arbeit ist vornehmlich im Typ 2 des zitierten Ausschnittes zu sehen. Diese Gruppe belegt im Verlauf ihres Studiums zwischen 40% und 50% informatischer und 20-30% anwendungsfeldspezifischer Module [Inf05]. Fachübergreifende Schlüsselkompetenzen sowie mathematisch-naturwissenschaftliche Grundlagen beanspruchen je 10-20% aller Lerninhalte.

Die Ausbildung an einer Hochschule ist nur ein erster Schritt der Lernerfahrung zukünftiger Experten. Spätestens im Berufsalltag, mit seinen vielfältigen Aufgabenstellung, ist die Anpassung und der Aufbau vergangener Kompetenzfelder erforderlich, um praktischen Anforderungen gerecht zu werden. Die Hochschulausbildung ist wie jede institutionelle Bildungseinrichtung nur begrenzt in der Lage, zukünftige Aufgabenbereiche und Rahmenbedingungen der beruflichen Praxis vorauszusagen und somit zu fördern. Eine fachdidaktische Methodik kann zur Lösung dieser Aufgabe beitragen, wenn sie selbst als methodische Kompetenz aufgegriffen wird und bei der Analyse zukünftiger technischer Paradigmenwechsel als Leitfaden Anwendung findet.

Wie aus den vorherigen Erläuterungen ersichtlich, ist die Zielgruppe dieser Arbeit Studierende der Informatik mit Themenschwerpunkt oder Nebenfach eingebettete Systementwicklung. Um Empfehlungen für zukünftige Experten im Bereich der eingebetteten Systementwicklung aussprechen zu können, ist jedoch auch die Betrachtung über das Studium hinaus notwendig. Ziel ist somit neben der didaktischen Unterstützung in Bachelor- und Masterstudiengängen auch die darüber hinausreichende Vermittlung von Entwicklerkompetenzen für viele Berufsjahre.

1.3. Gliederung der Arbeit

Die folgende Gliederung beschreibt die Struktur des Forschungsprozesses und den damit verbundenen Ergebnissen.

Kapitel 2: Grundlagen zur Entwicklung eingebetteter Systeme im didaktischen Kontext

Kapitel 2 erläutert das in dieser Arbeit verwendete Verständnis über eingebettete Systeme und deren Entwicklung. Die Erläuterungen werden innerhalb der Arbeit insbesondere in den Abschnitten zu den entwickelten Lehr-/Lernhilfen und der Beobachtung des Entwurfs- und Anwendungspraktikums für eingebettete Mikrosysteme (EAP) aufgegriffen. Im zweiten Teil des Kapitels wird auf fachdidaktisch verwandte Arbeiten eingegangen und die wesentlichen Unterschiede zur vorliegenden Forschungsarbeit erläutert.

Kapitel 3: Projektarbeit KOMINA

In Kapitel 3 wird das DFG-geförderte Projekt *Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen* (KOMINA) vorgestellt. Ausgehend von der Zielstellung des Forschungsvorhabens werden die Forschungsergebnisse zweier Kompetenzstrukturmodelle genannt, die im Rahmen dieser Arbeit relevant sind. Eine besondere Gewichtung haben die Abschnitte über die Beobachtung des EAPs und den daraus abgeleiteten Lernhypothesen. Zusammen mit den Entwicklungen von Lehr-/Lernhilfsmitteln bilden sie den Schwerpunkt der Arbeit des Autors im KOMINA-Projekt und die Basis für die weitere Forschung ab Kapitel 4.

Kapitel 4: Kriterien für fundamentale Lehr-/Lerninhalte eingebetteter Systeme

Kapitel 4 bildet den theoretischen Kern der Dissertation. Ausgehend von existierenden fachdidaktischen Methoden zur Bestimmung besonders wichtiger Lehr-/Lerninhalte wird analysiert, welche Adaptionen hinsichtlich Zielgruppe und Anwendungsgebiet notwendig sind, um ähnliche Aussagen für den Bereich der Entwicklung eingebetteter Systeme zu treffen. Alle Kriterien werden im Kontext der Hochschullehre untersucht und aufeinander abgestimmt. Neben einer Erklärung, wie methodischen Nachteilen begegnet wird, umfasst der abschließende Abschnitt des Kapitels Beispiele und Gegenbeispiele für alle vorher definierten Kriterien, um deren Wirkprinzipien zu verdeutlichen.

Kapitel 5: Anwendung des Kriterienkataloges auf fachspezifische Methoden und Sichtweisen

Die in Kapitel 4 erarbeiteten Kriterien zur Bestimmung zentraler Lehr-/Lerninhalte werden in Kapitel 5 auf eine Sammlung fachspezifischer Methoden und Sichtweisen angewendet. Im ersten Abschnitt des Kapitels werden verschiedene Sammlungen

von Themengebieten analysiert und bewertet, damit ein möglichst breites und kontextreiches Netz an Entwicklertätigkeiten für das Anwendungsgebiet zur Verfügung steht. Anschließend werden die Bewertungsrichtlinien für die Analyse erläutert und die geclusterte Begriffssammlung vorgestellt. Eine exemplarische Durchführung der Analyse für ein Themengebiet aus dieser Sammlung verdeutlicht die Anwendung der Methode. Das Kapitel enthält die Aufstellung aller Kriterienbewertungen für jede untersuchte Idee und benennt darauf aufbauend die als zentral ermittelten Lehr-/Lerninhalte. Abschließend werden die Kriterien selbst auf ihre Aussagekraft im kompletten Analyseprozess evaluiert und mögliche Anpassungen für Forschungsarbeiten mit vergleichbaren fachdidaktischen Vorgehen erläutert.

Kapitel 6: Ableitung eines Lehrkonzeptes aus den Kriteriendefinitionen

Neben der Aufstellung besonders relevanter Themengebiete kann aus den Erläuterungen und Definitionen zur methodischen Vorgehensweise auch ein Rahmenkonzept für eine Lehrveranstaltung abgeleitet werden. Das in Kapitel 6 gezeigte Konzept orientiert sich somit inhaltlich wie strukturell am Forschungsprozess. Beispielhaft werden die verschiedenen Veranstaltungselemente und Einzeltermine erläutert und mit aktuellen Fachdiskussionen gestützt.

Kapitel 7: Zusammenfassung, Fazit und Ausblick

Das siebte Kapitel analysiert den in dieser Arbeit beschriebenen Forschungsprozess hinsichtlich der genannten Zielstellungen und gibt in diesem Kontext eine Zusammenfassung aller Forschungsetappen und ihrer Ergebnisse. Offene Fragen für die weitere Forschung werden am Ende des Kapitels erläutert.

Forschungstätigkeiten im Verlauf des Forschungsprojektes

Die Auflistung 1.1 gibt einen Überblick über die im Forschungsprozess erreichten Meilensteine und den mit diesen verknüpften Veröffentlichungen, an denen der Autor beteiligt war.

In der gesamten Arbeit wird aus Gründen der besseren Lesbarkeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Sämtliche Personenbezeichnungen gelten gleichwohl für beiderlei Geschlecht.

Aktivitäten und Meilensteine	Veröffentlichungen
Entwurf eines Kompetenzmodells für Entwickler eingebetteter Mikro- und Nanosysteme	Schäfer et al., [Sch+12b]
Erforschung fachdidaktischer Grundlagen zur Vermittlung von Nano-Technologie	Kleinert et al., [Kle+12]
Vorstellung eines empirisch verfeinerten Kompetenzstrukturmodells	Schäfer et al., [Sch+12c]
Entwurf des EAPs und dessen Beobachtung	Jaschke et al., [Jas+12]
Vorstellung einer virtuellen Toolchain zur Nutzung in Laborveranstaltungen	Büchner und Jaschke, [BJ13]
Methodik zur Vermittlung komplexer Anforderungen eingebetteter Systeme	Büchner, [Büc13]
Explorative Lernumgebung für die Lehre mit eingebetteten Mikrosystemen	Jaschke, Büchner und Schubert, [JBS13]
Verbesserung der Vergleichbarkeit von Praxisberichten mittels einer vierstufigen Taxonomie	Büchner, Jaschke und Schubert, [BJS13]
Ermittlung fundamentaler Themengebiete eingebetteter Systeme durch die Verknüpfung emprischer und normativer Ansätze	Büchner, [Büc14]
Kriterienbasierte Bestimmung zentraler Themengebiete der Entwicklung eingebetteter Systeme für die Hochschule	Büchner und Schubert, [BS14]

Tabelle 1.1.: Forschungsverlauf und Veröffentlichungen

2. Grundlagen zur Entwicklung eingebetteter Systeme im fachdidaktischen Kontext

Das vorliegende Kapitel zeigt zu dieser Arbeit verwandte wissenschaftliche Beiträge auf. Der Großteil der dargestellten Quellen ist mit Hinblick auf die Konzeption von Lehr-/Lernempfehlungen zu sehen, die einen direkten Bezug zu Teilziel II und III der Forschungsmethodik besitzen.

Um zu erfassen, welche fachdidaktischen Beiträge sich mit dem Anwendungsgebiet befassen, gibt der erste Abschnitt des Kapitels eine Definition der in dieser Arbeit verwendeten Begriffe „eingebettete Systeme“ und „Cyber-physical Systems“ (siehe Abschnitt 2.1). Aufgrund der notwendigen Betrachtung mehrerer Disziplinen und der in diesen erarbeiteten didaktischen Konzepten zur Förderung der Studierenden wird in Abschnitt 2.2 auf die prinzipielle Vergleichbarkeit von wissenschaftlichen Projekten zum Themenbereich eingegangen. Büchner und Jaschke haben hierzu eine Taxonomie vorgestellt, die beim Vergleich verschiedener fachdidaktischer Ansätze hilfreich ist [BJS13].

Ausgehend von der im vorherigen Kapitel dargestellten Zielgruppendefinition werden in Hinsicht auf das Anwendungsfeld folgend die Publikationen der wissenschaftlichen Gemeinschaft als verwandt aufgeführt, die sich mit der didaktischen Erforschung eingebetteter Systeme auf Ebene der Architektur oder des Designs beschäftigen (siehe Abschnitt 2.3). Beiträge, die sich mit der Konzeption von Lehr-/Lernempfehlungen für den Bereich eingebetteter Systeme im Allgemeinen befassen und zumindest teilweise den in der Zielgruppendefinition dargestellten Anwendungsbereich umfassen, werden in Abschnitt 2.4 aufgeführt und diskutiert. Arbeiten, die ausschließlich einen Fachbezug, jedoch keine didaktischen Elemente beinhalten, sind aus Gründen des Umfangs nicht aufgeführt.

2.1. Eingebettete Systeme

Der erste Abschnitt der folgenden Diskussion zum Themenbereich eingebetteter Systeme zeigt die Relevanz für informatiknahe Disziplinen und natürlich die Informatik selbst auf. Um verwandte Arbeiten besser vergleichen zu können, erläutert der darauffolgende Abschnitt, welches Verständnis des Begriffes „eingebettetes System“ im Rahmen der Arbeit angesetzt wird.

Auswirkungen und Bedeutungen eingebetteter Systeme für die Informatik

Eingebettete Systeme verbinden traditionelle Fachbezüge wie die der Informatik, Physik und Steuerungstechnik [Cas+05]. Wie bereits von Caspi et al. dargelegt, ist eine einheitliche Definition des Begriffes „eingebettetes System“ deswegen nicht gesichert und besitzt in Industriebranchen wie der Luft- und Raumfahrt oder der Automobiltechnik verschiedene Deutungsweisen [Cas+05]. Hieraus folgt, dass eine fundierte Abhandlung im Themenbereich nicht nur die informatische Sicht berücksichtigen darf, sondern interdisziplinär zu erforschen ist.

Anders als bei herkömmlichen Computersystemen verfügen eingebettete Systeme oft über keine Schnittstelle zur Interaktion mit einem Benutzer. Im Gegenteil sind diese Systeme oft darauf ausgelegt, gar nicht mehr vom Benutzer wahrgenommen zu werden. Während der traditionelle Heimcomputer als Kommunikationspartner, Handlungsraum oder mediales System begriffen werden kann, zeichnen sich eingebettete Systeme durch ihre Allgegenwärtigkeit und Unsichtbarkeit aus [Her07]. Die mit dieser Auffassung verbundenen Diskussionen finden sich unter Begriffen wie *„Ubiqitious Computing“* und *„Persuasiv Computing“* wieder [Wei91], [Han+01].

Für Nutzer ist deswegen nicht direkt ersichtlich, mit wie vielen eingebetteten Systemen sie interagieren. Bereits im Jahr 2000 konnte davon ausgegangen werden, dass der durchschnittliche amerikanische Bürger mit ungefähr 100 Mikroprozessoren pro Tag in Kontakt kommt [WM00]. Auch aus einem ökonomischen Blickwinkel betrachtet zeigt sich die weitreichende Relevanz des Themengebietes in dem Umstand, dass die deutliche Mehrzahl der weltweit hergestellten Prozessoren (98%) nicht in traditionellen Computersystemen eingesetzt werden, sondern in Geräten, die der Gruppe der eingebetteten Systeme zugeordnet werden [JC05].

Das Bundesministerium für Bildung und Forschung (BMBF) bezeichnet eingebettete Systeme nicht nur als Querschnittstechnologie, sondern auch als die Stärke der deutschen Informations- und Kommunikationstechnologien [Bil+07]. Der referenzierte Bericht des BMBFs fasst sechs Forschungsfelder der Disziplin als zentralen Bestandteil zukünftiger industrieller Interessen zusammen. Diese sind *autonome Systeme*, *Seamless Interaction*, *Virtual Engineering*, *verteilte Echtzeit-Situationserfassung und Lösungsfindung*, *sichere Systeme* und *Architekturprinzipien*. Bis 2019 wird die deutsche Industrie mindestens 2,5 Mrd. € zur Erforschung der Forschungsfelder einsetzen [Dam+09]. Ein wissenschaftliches, industrielles und gesellschaftliches Interesse an eingebetteten Systemen ist also gegeben.

Definition der begrifflichen Ebene

Die Hauptaufgabe vieler eingebetteter Systeme besteht in der Überwachung oder Regelung von Prozessen der Umgebung, in die sie eingebettet sind. Gegenüber traditionellen Computersystemen hat der Umstand, dass physikalische Prozesse zentraler Bestandteil des Systems sind, ebenso den Begriff der *„Cyber-physical Systems“* hervorgebracht [Lee08]. Eine weitere Verfeinerung des Begriffs kann durch den Einfluss von Nanotechnik eingeführt werden. Aktuelle eingebettete Systeme

werden mittels Strukturen und Komponenten im Mikrometerbereich hergestellt. Die Zusammensetzung solcher Systeme mit Nanosystemen wird als eingebettete Mikro- und Nanosysteme (EMNS) bezeichnet.

Abbildung 2.1 stellt die Komponenten und Schnittstellen eines eingebetteten Systems schematisch dar. Grundlegend lassen sich die Bestandteile eines solchen Systems in die drei Kategorien *Software*, *Hardware* und *Umgebung* strukturieren. Letztere wird in der Regel durch mathematische bzw. formale Modelle beschrieben. Damit Signale aus der Umgebung geregelt werden können, ist die Wandlung in ein digitales (Analog-Digital-Converter, abgekürzt: *ADC*) oder analoges Signal (Digital-Analog-Converter, abgekürzt: *DAC*) notwendig. Die hardwareseitige Recheneinheit wird heutzutage entweder als Mikroprozessor, Field Programmable Gate-Array (FPGA) oder als anwendungsspezifischer integrierter Schaltkreis (ASIC) realisiert. Namensgebend für ersteren ist die computerähnliche Abarbeitung von Instruktionen, mit der sich auch Schnittstellen zur Hardwareinteraktion (bspw. *„Interrupts“*) ansprechen lassen. Ein FPGA wird nicht mit einer traditionellen Programmiersprache, sondern mit einer Hardwarebeschreibungssprache entwickelt, die angibt, wie Logikkomponenten und dedizierte Subsysteme miteinander verknüpft und parametrisiert werden. ASICs sind fest verdrahtete Komponenten, die sich in keiner Weise programmieren lassen. Üblicherweise zeichnen sie sich durch hervorragende Leistung und eine günstige Massenanfertigung aus. Der Entwurf und die Erstellung einer Maske für den Herstellungsprozess ist hingegen sehr teuer, weswegen ASICs meist erst bei hohen Auflagenzahlen verwendet werden [Zah03].

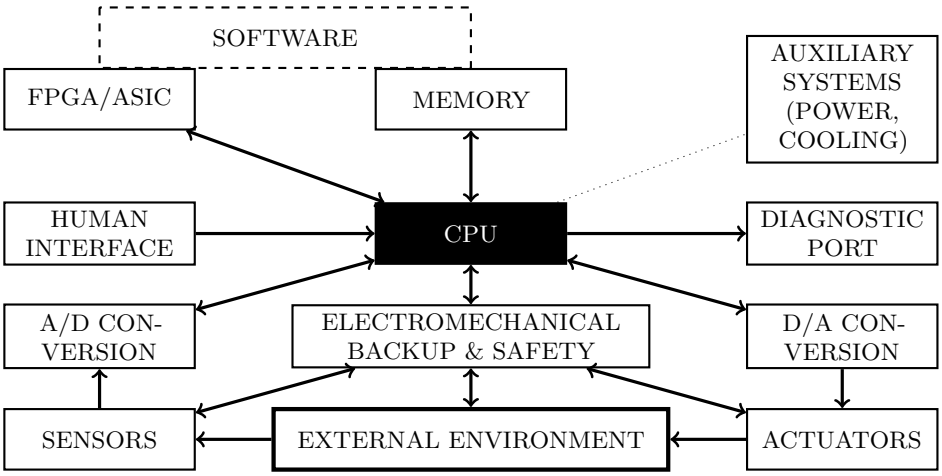

Abbildung 2.1.: Struktureller Aufbau eines eingebetteten Systems nach Koopman [Koo96]

Zusammen mit den Sensoren, Aktoren, ADCs, DACs und der Berechnungseinheit bildet der Speicher die logische Einheit „eingebettetes System“. Erweitert wird dieses um eine optionale Schnittstelle zur Benutzerinteraktion, einer ebenfalls optionalen Programmier- oder Debugschnittstelle sowie der Stromversorgung. Während die

Hardware die unterste Ebene eines solchen Systems darstellt, kontrollieren ein optionales Betriebssystem und die Systemsoftware die Steuerung des Gesamtsystems. Mikrocontrollergestützte eingebettete Systeme lassen sich mit traditionellen Programmiersprachen wie C, C++ oder Java programmieren. Die automatische Erstellung der Implementierung aus endlichen Automaten und ähnlichen grafischen Modellierungswerkzeugen ist eingeschränkt möglich. Anstrengungen zur Reduktion der Entwicklungszeit und Komplexität heutiger eingebetteter Systeme werden durch die Verwendung hochsprachiger Konzepte jedoch vermehrt erprobt [Bag+10], [Bri+05], [BGP07]. FPGAs werden mit Hardware-Beschreibungssprachen wie VHDL oder Verilog beschrieben.

Für eingebettete Systeme wird die folgende Definition von Marwedel genutzt:

> „Embedded systems are information processing systems embedded into enclosing products." [Mar11, S. xiii]

Die in der Definition aufgegriffenen „Produkte" können andere technische Systeme, aber auch physikalische Prozesse aus der Umwelt sein. Für den letzteren Fall hat sich der Begriff der „Cyber-physical Systems" etabliert, der in dieser Arbeit immer dann verwendet wird, wenn die Bezüge zu Prozessen der Umwelt besonders hervorgehoben werden sollen. Die Vielzahl an Einsatzgebieten und Zielgruppen erschweren eine einheitliche Definition des Begriffes „eingebettetes System", die alle Aspekte in genügendem Umfang berücksichtigt. Aus diesem Grund existieren verschiedene Vorschläge (bspw. [Mar11], [HS07] und [LS11b]). Bestandteil dieser Definitionen sind neben dem Aufbau und der Zielstellung eingebetteter Systeme oft auch deren Einsatzgebiet.

Das Entwicklungsvorgehen eingebetteter Systeme unterscheidet sich deutlich zum Vorgehen der klassischen Softwareentwicklung. Während bei letzterer zügig Prototypen erstellt werden können, ist die Bindung an eine noch zu entwerfende Hardwareplattform in der Entwicklung eingebetteter Systeme problematisch. Für die späteren Analysearbeiten in Abschnitt 5.5 und im Anhang (Abschnitt A) ist die Nutzung eines fundierten Entwicklungsprozessmodells wichtig. Für die vorliegende Arbeit wurde der ISO-Standard ISO-15288 gewählt, der den Prozess einer kompletten Systementwicklung (Hardware und Software) auf mehreren Ebenen beschreibt [ISO08]. Der Standard wurde gewählt, da er die Systementwicklung auf generischer Ebene beschreibt und somit besonders für das Anwendungsgebiet eingebetteter Systeme mit einem hohen Grad an Interdisziplinarität geeignet ist.

Der in ISO-15288 vorgeschlagene Prozess zur Gestaltung eines Systemlebenszyklus ist in die vier Prozessgruppen „Enterprise Processes", „Agreement Processes", „Project Processes" und „Technical Processes" unterteilt [ISO08]. Für die später durchgeführten Analysen ist die letzte Prozessgruppe von Bedeutung, da in ihr der technische Entwicklungsprozess strukturiert wird. Demnach können die folgenden Entwicklungsphasen unterschieden werden:

1. „Stakeholder Requirements and Definition Process"
2. „Requirements Analysis Process"
3. „Architectural Design Process"

4. „Implementation Process“
5. „Integration Process“
6. „Verification Process“
7. „Transition Process“
8. „Validation Process“
9. „Operation Process“
10. „Maintenance Process“
11. „Disposal Process“

Die Entwicklungsphasen gleichen im Wesentlichen auch denen des V-Modells, das in der klassischen Softwareentwicklung, aber auch in der Entwicklung eingebetteter Systeme verwendet werden kann. Der wesentliche Unterschied zu ISO-15288 ist, dass dieser den Entwicklungsprozess nicht so feingranular wie das V-Modell strukturiert. Eine feinere Struktur ist für den Forschungsprozess in dieser Arbeit nicht notwendig und würde das Verständnis der in Abschnitt 5.5 und Abschnitt A durchgeführten Analysen unnötig erschweren.

2.2. Vergleichbarkeit fachdidaktischer Publikationen

Die vorhandenen Publikationen zum Themenbereich der eingebetteten Systeme zeichnen sich nicht nur auf technischer, sondern auch auf didaktischer Ebene durch große Heterogenität aus. Diese Situation führt dazu, dass die Beiträge oft schlecht miteinander vergleichbar sind und vorgeschlagene Konzepte und Lösungsansätze nicht übernommen werden können. Büchner und Jaschke haben 2013 eine Taxonomie veröffentlicht, um wissenschaftliche Arbeiten im Anwendungsbereich vergleichbarer zu gestalten [BJS13].

Die Autoren unterscheiden dabei die folgenden Formatbeiträge: Curriculaempfehlungen, Fachbücher, Kursmaterialien und wissenschaftliche Artikel. Ein Großteil der wissenschaftlichen Artikel enthält Praxisberichte, die sich im Vergleich zu den anderen genannten Formaten schon aufgrund ihres oft kurzen Umfangs schwierig in einen didaktischen Kontext einordnen lassen. Einige der dabei wichtigen Inhaltsaspekte sind:

- Der institutionelle Rahmen des Praxisberichtes.
- Die Zielgruppe, die angesprochen wird.
- Eine Auswahl von die Lehre unterstützenden Werkzeugen wie zum Beispiel Hardware und Software.
- Der zeitliche Umfang und die zeitliche Strukturierung der Lehr-/Lerneinheit.
- Die von den Studierenden zu lösende Aufgabe.

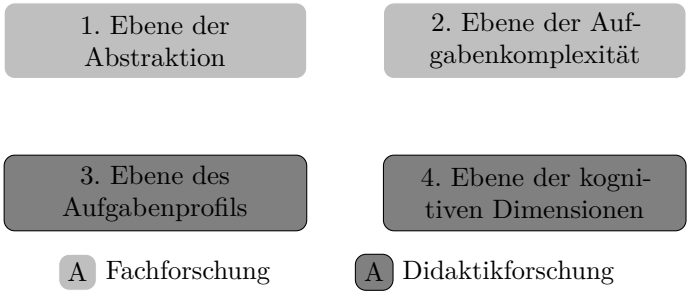

Abbildung 2.2.: Ebenen der Taxonomie nach Büchner und Jaschke [BJS13]

Ein weiterer Aspekt ist die Evaluation der Praxiserprobung, die nach unterschiedlichen Kriterien durchgeführt werden kann und somit nicht für jeden Forscher die gleiche Aussagekraft besitzt. Deswegen ist ein Werkzeug notwendig, das Lehrpersonen genug Informationen bietet, um zu entscheiden, ob die Grobkonzeption einer vorgestellten Lehrveranstaltung mit den eigenen Möglichkeiten und Zielstellungen übereinstimmt. Die von Büchner und Jaschke diskutierte Taxonomie bündelt vier grundlegende Dimensionen, die zur besseren Vergleichbarkeit notwendig sind, um dieses Ziel zu unterstützen. Die Dimensionen lassen sich strukturell in die zwei Kategorien *Didaktik* und *Fach* gliedern (siehe Abbildung 2.2).

Jede Dimension lässt sich auf einer Skala darstellen, bei der mehrere Einträge möglich sind (siehe Vorlage in Abbildung 2.3). Es ist darauf hinzuweisen, dass sich die Einteilung der Skalen an didaktisch orientierten Lehrveranstaltungskonzepten und nicht an Industrie- oder Wissenschaftsprojekten orientieren. Bei diesen sind die zeitlichen Rahmenbedingungen wie auch die Komplexität der zu lösenden Aufgaben grundverschieden. Die einzelnen Dimensionen werden im Folgenden erläutert.

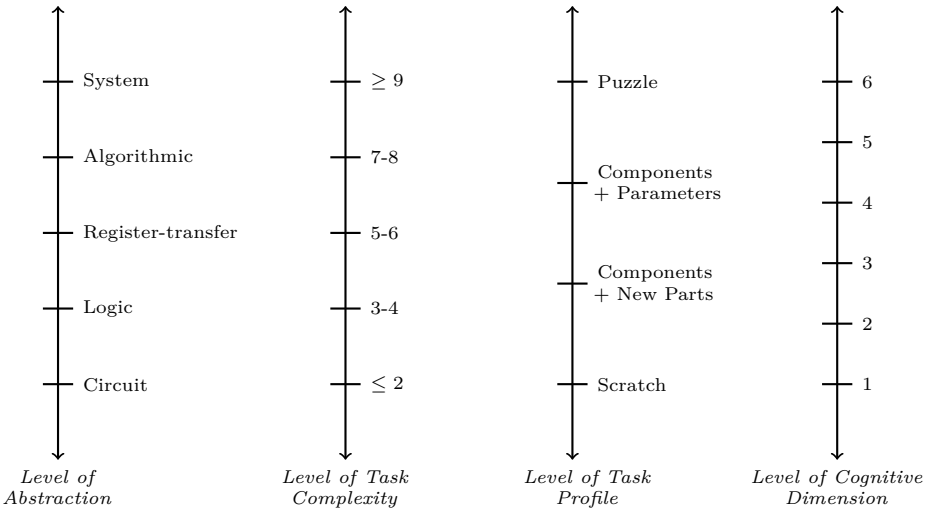

Abbildung 2.3.: Vorlage für die Anwendung der vorgestellten Taxonomie [BJS13]

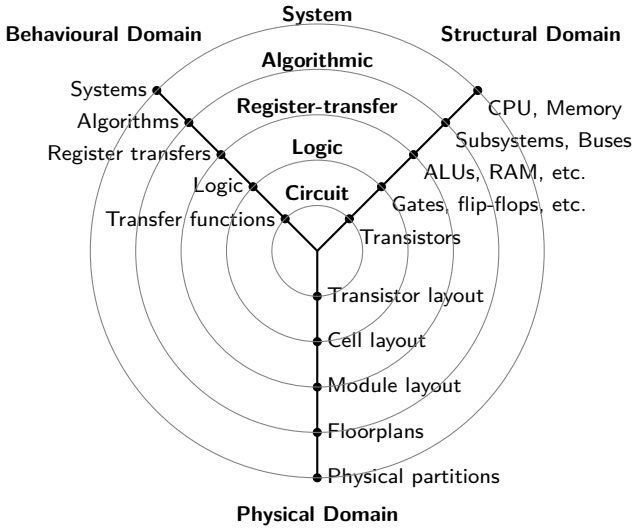

Abbildung 2.4.: Y-Diagramm nach Gajski und Kuhn [GK83] aus [BJS13]

Abstraktionsniveau

Die erste Dimension wird durch das Abstraktionsniveau beschrieben, auf dem ein eingebettetes System oder eine Komponente zu entwickeln ist. Um die Ebenen, auf denen dies geschehen kann, zu bestimmen, haben sich die Autoren an dem bekannten Hardware-Entwicklungsdiagramm von Gajski und Kuhn orientiert [GK83]. Dieses 1983 vorgestellte Schaubild beschreibt verschiedene Sichten auf den Prozess der Hardwareentwicklung. Zusätzlich wird bei allen Sichten ein fünfstufiges Abstraktionsniveau unterschieden, das für die diskutierte Taxonomie von besonderer Bedeutung ist, da es die Gliederung der Skala bildet. Somit können System-, Algorithmus-, Register-Transfer-, Logik- und Schaltkreisebene als Abstraktionsniveaus in einem Lehrkonzept unterschieden werden (siehe Abbildung 2.4).

Aufgabenkomplexität

Im Anwendungsgebiet eingebetteter Systeme kann der Begriff der Komplexität unter anderem durch die Art und die Anzahl der Verbindungen zwischen unterschiedlichen Systemanforderungen eingeschätzt werden [Wei11]. Systemanforderungen werden üblicherweise in funktionale und nicht-funktionale Anforderungen aufgeteilt. Beispiele für funktionale und nicht-funktionale Anforderungen können in der entsprechenden Analyse im Anhang (Abschnitt A.22) nachgelesen werden. Für die Bestimmung der Aufgabenkomplexität wurde ein Ansatz gewählt, der die Definition des Begriffes „Komplexität“ vermeidet und stattdessen mit Anforderungen und Eigenschaften arbeitet, die üblicherweise in komplexen Systemen vorhanden sind. Die Einordnung auf der entsprechenden Skala zur Aufgabenkomplexität wird durch das Zusammenzählen der zutreffenden Aufgaben- und Systemcharakteristika

erreicht. Eine beispielhafte Liste, die in [BJS13] vorgestellt wurde, enthält folgende Elemente:

- Regelung von physikalischen Prozessen und Werten,
- sehr großer logischer Zustandsraum oder eine hohe Komponentenanzahl,
- hohe Ansprüche an Leistung und Optimierung,
- die Notwendigkeit von Echtzeitanforderungen,
- Nutzung paralleler oder nebenläufiger Prozesse,
- Integration heterogener Komponenten,
- Erfordernis, einen Standard zu erfüllen,
- Bedarf an Kommunikation mit anderen Systemen oder Komponenten,
- nicht-funktionale Anforderungen wie maximale Größe, maximaler Energieverbrauch, Sicherheit oder Verlässlichkeit.

Die vorgestellte Liste kann durch das angesprochene Verfahren einfach erweitert werden, ohne bereits bestehende Klassifizierungen zu beeinflussen, da für Lehrveranstaltungen besonders die unteren Komplexitätsniveaus interessant sind. Wird beispielsweise der Wert sieben bis acht auf der Skala erreicht, ist dieser auch noch gültig, wenn der Katalog typische Charakteristika komplexer Systeme erweitert wird. Komplexere Aufgaben sind nur im zeitlichen Rahmen von Bachelor-, Master- oder Diplomarbeiten oder Projektgruppen durchführbar, ohne die Studierenden zu überfordern.

Aufgabenprofil

Das Aufgabenprofil ist die erste der beiden didaktischen Dimensionen und beschreibt das Verhältnis zwischen den Vorgaben einer Aufgabenstellung durch Betreuer oder Lehrpersonal und dem von Studierenden zu leistenden Eigenanteil. Als illustrierendes Beispiel kann die Entwicklung eines Spurfolgeroboters dienen. Im ersten Szenario wird den Studierenden ein Quelltextrahmen sowie die für die Umsetzung benötigten Programmbibliotheken vorgegeben. Die von den Studierenden zu erwartenden Leistungen beschränken sich in diesem Fall lediglich auf die korrekt parametrisierte Abfrage des Helligkeitssensors. In einem gegensätzlichen Szenario erhalten die Veranstaltungsteilnehmer lediglich das für die Entwicklung erforderliche Programmierwerkzeug in Form einer Entwicklungsumgebung. Die Strukturierung und Umsetzung des zur Spurensuche notwendigen Algorithmus ist Aufgabe der Studierenden. Implementierungsdetails wie beispielsweise die richtige Kalibrierung des Helligkeitssensors sind von den Teilnehmern ebenso eigenständig durchzuführen.

Obwohl die Aufgabenkomplexität (Dimension zwei) und das Abstraktionsniveau (Dimension eins) identisch sind, ist das Aufgabenprofil der beiden Szenarien verschieden. Büchner und Jaschke schlagen folgende Niveaustufung der Dimension vor:

1. Entwurf eines Systems von Grund auf. Es existiert keine Vorarbeit in Form von Modulen oder Rahmenwerken.
2. Entwurf eines neuen Systems auf Basis vorgefertigter Module oder Rahmenwerke. Die Teilnehmer nutzen bestehende Komponenten, um eine eigene, vollständig neue Komponente zu entwerfen.
3. Entwurf eines Systems ohne die Erstellung neuer Module, sondern mit der richtigen Verknüpfung und Parametrisierung der vorhandenen Komponenten.
4. Entwurf eines Systems ohne die Parametrisierung oder Erstellung neuer oder vorhandener Module. Nur die Verbindung zwischen diesen Modulen wird hergestellt, damit das System funktionsfähig ist. Diese Niveaustufe kann mit einem Puzzle verglichen werden.

Bower hat bereits 2008 eine Unterscheidung von Aufgabentypen in der Informatik vorgenommen und diese in einer eigenen zehnstufigen Taxonomie veröffentlicht [Bow08]. Die Autoren haben sich gegen die Nutzung seiner Arbeit entschieden, da Bower die kognitive Dimension mit der Aufgabendimension fest verbindet. Es ist klar zu erkennen, dass zwischen den beiden didaktischen Dimensionen ein Zusammenhang besteht. Dieser muss allerdings flexibel und nicht statisch interpretiert werden. Das Design eines Systems kann beispielsweise durch starke Unterstützung von Lehrpersonen auf Niveauebene vier „bausteinartig“ geschehen, ohne dass hiermit zwangsläufig die sonst für Designtätigkeiten übliche Kompetenzebene des „Erstellens“ (Niveauebene sechs) angesprochen wird. Im Kontext von Lehrveranstaltungen ist von einem gewissen Anteil an didaktischer Reduktion auszugehen, sodass die unterste Aufgabenstufe, in der Studierende keinerlei Hilfe bekommen, selten vorhanden sein wird. Dies würde im Umkehrschluss bedeuten, dass Lernende die höchste Kompetenzstufe im Modell von Bower nie erreichen.

Kognitive Dimension

Wie aus den vorherigen Erläuterungen zu erkennen ist, ist die Ebene des Kompetenzniveaus als kognitive Dimension differenziert zu betrachten. Büchner und Jaschke haben sich dafür entschieden, die Lernzieltaxonomie von Anderson und Krathwohl als Strukturierung der Skala zu nutzen [AK01]. Dementsprechend können die Niveaus *„Erinnern“*, *„Verstehen“*, *„Anwenden“*, *„Analysieren“*, *„Evaluieren“* und *„Erstellen“* unterschieden werden.

Die Autoren haben bei der Taxonomieauswahl auch die für die Informatik besonders relevante Veröffentlichung von Fuller et al. von 2007 berücksichtigt [Ful+07]. Die Autoren nehmen eine Umstrukturierung der Taxonomie von Anderson und Krathwohl vor und ordnen die erwähnten sechs Stufen in einer Matrix an (Beispiel in Abschnitt 3.2, Abbildung 3.3). Deswegen lässt sich die Taxonomie jedoch schlecht in einem neuen Klassifizierungskonzept darstellen. Die Vorarbeit von Anderson und Krathwohl ist zudem wesentlich bekannter und sollte deswegen für den Großteil des Lehrpersonals einfacher auf ein eigenes Kurskonzept überführbar sein. Es ist jedoch hervorzuheben, dass die Zuordnung der informatikspezifischen Begriffe auf die Taxonomie möglich und sinnvoll ist. Die Tätigkeit des Programmierens

würde demnach nicht auf der Ebene des Erstellens, sondern der des Anwendens einzuordnen sein.

Büchner und Jaschke haben in [BJS13] eine beispielhafte Kategorisierung von drei unterschiedlichen Lehrveranstaltungen zum Themenbereich des Hardware-/Software Co-Designs vorgenommen, um die Anwendung der Taxonomie zu erläutern. Die Forschungsarbeit schafft somit nicht nur die Basis für eine gesteigerte Vergleichbarkeit von Lehrveranstaltungen, sondern ist ebenso ein Werkzeug, mit dem die Intention der eigenen Veranstaltungskonzeption differenziert aufgegliedert werden kann. Die Taxonomie wird auch auf die Lehrveranstaltungsplanung in Kapitel 6 angewandt, um deren intendierten Anwendungskontext zu spezifizieren.

2.3. Eng verwandte Arbeiten

Die Dissertation von Rumpler, in der die Zusammenhänge zwischen Kognitionswissenschaft und dem Design eingebetteter Systeme untersucht werden, stellt die thematisch ähnlichste Arbeit dar [Rum08]. Rumpler fokussiert dabei jedoch nicht die Erforschung und Weiterentwicklung fachdidaktischer Ansätze im Themengebiet der eingebetteten Systeme, sondern verknüpft Forschungsergebnisse der Kognitionswissenschaften mit fachspezifischen Entwicklungsmethoden, um die Komplexität der Komponentenentwicklung für eingebettete Systeme zu reduzieren. Sein Ziel ist der Entwurf einer Richtlinie für die Komponentenschnittstellen, deren reduzierte Komplexität sich in den jeweils angeschlossenen Komponenten fortsetzt. Rumpler analysiert zunächst die Zusammenhänge der Fachforschung mit der kognitionswissenschaftlichen Auffassung von Komplexität und zeigt Gründe auf, die einen interdisziplinären Forschungsansatz für die Entwicklung moderner eingebetteter Systeme rechtfertigen. Aus Entwicklerperspektive orientiert er sich ebenso wie die vorliegende Arbeit an Design auf Architekturebene, wobei er speziell Echtzeitsysteme untersucht, die nicht den fachlichen Kern dieser Arbeit beschreiben. Den wesentlichen Beitrag der Kognitionswissenschaften beschreibt Rumpler durch die Grundlagen zum menschlichen Gedächtnis und zur Wahrnehmung. In diesem Zusammenhang geht er auf die relationale Komplexität und mentale Modelle zur Komplexitätsbewältigung ein, die nach Meinung des Autors auch eine fachdidaktische Untersuchung rechtfertigen. Eine praktische Anwendung der theoretisch gewonnenen Erkenntnisse zeigt Rumpler an einem Framework für die Entwicklung verteilter Echtzeitanwendungen. Während seine Arbeit einen deutlichen Schwerpunkt auf die Methoden der Fachforschung zum Themenbereich Echtzeitsysteme setzt, sind seine Erkenntnisse hinsichtlich der Komplexitätsbewältigung in der Entwicklung eingebetteter Systeme auch von fachdidaktischer Relevanz. Die Zielgruppe für eine didaktische Umsetzung seiner Arbeit entspricht am ehesten Master- und Diplomstudierenden, die aufgrund der in Abschnitt 4.2.2 geschilderten Anforderungen nicht den einzigen Zielgruppenbereich der vorliegenden Arbeit ausmachen und deswegen nur eingeschränkt berückstichtigt werden konnte.

Weitere eng verwandte Arbeiten entstanden im Rahmen des KOMINA-Projektes, in dem der Autor der Arbeit mitgearbeitet hat. Besondere Aufmerksamkeit ist der Konzeption und dem Entwurf eines Kompetenzstrukturmodells zu widmen,

das die Fachdisziplin erstmalig kompetenzorientiert gliedert und somit für einen Großteil der fachdidaktischen Arbeiten im Anwendungsgebiet Relevanz besitzt. Da der wesentliche Teil der vom Autor mitentwickelten Lehr-/Lernhilfen auf dem Kompetenzstrukturmodell aufbaut, sind die Verknüpfungspunkte der beiden Forschungsarbeiten in einem eigenen Abschnitt beschrieben (siehe Kapitel 3).

2.4. Verwandte Arbeiten

Die im Folgenden erläuterten Arbeiten zeichnen sich ebenso wie die eng verwandten Arbeiten durch eine starke fachdidaktische Komponente aus, beziehen sich aber nicht auf die Ebene der Architektur und des Designs eingebetteter Systeme. Manche der vorgestellten Forschungsarbeiten betrachten eingebettete Systeme darüber hinaus nur als ein kleines Teilgebiet der Technischen Informatik.

Lehr-/Lernempfehlungen wurden insbesondere durch Expertenkomitees wie der Gesellschaft für Informatik (GI), der ACM/IEEE oder dem ARTIST-Netzwerk veröffentlicht [Hoc+11], [SR13], [ACM04], [Cas+05]. In seltenen Fällen enthalten Publikationen einzelner Wissenschaftler oder kleinerer Forschergruppen allgemeine Sammlungen von Lehr-/Lernempfehlungen (bspw. [HBH06], [RJS08]), die den vorher genannten insbesondere hinsichtlich ihres Umfangs deutlich unterlegen sind.

Der Curriculavorschlag der GI stellen die aktuellste der genannten Empfehlungen für den Bereich der Technischen Informatik dar und beinhalten gleichzeitig die Vorlage der Zielgruppendefinition für diese Arbeit (siehe Abschnitt 1.2). Dabei orientiert sich die Expertengruppe der GI hauptsächlich am Bachelorstudiengang und gibt lediglich eine Auflistung weiterführender thematischer Hinweise für Masterstudierende. Die Aufstellung der für diese Arbeit relevanten Themengebiete ist recht kurz gehalten und an einigen Stellen nicht detailliert genug für eine praktische Umsetzung beschrieben. So ist es beispielsweise schwierig, fachdidaktische und fachliche Zielstellungen für den Themenschwerpunkt „Kommunikationssysteme - Netzwerkanbindungen“ im Bereich der Technischen Informatik auszumachen. Die Autoren weisen jedoch darauf hin, dass genau dieser niedrige Detailgrad angedacht ist, um die Umsetzbarkeit der Empfehlungen für alle Universitäten zu ermöglichen, unabhängig von deren Forschungsschwerpunkt.

Die Abbildung 2.5 zeigt den von der GI für Bachelorstudierende vorgeschlagenen Fächerkatalog auf konzeptueller Ebene. Inhalte aus der Digitaltechnik (DT), der Rechnerorganisation (RO), der Betriebssysteme (BS) und der Rechnernetze (RN) sind verpflichtend. Eingebettete Systeme (ES) und Rechnerarchitekturen (RA) gehören in den Wahlpflichtbereich, der vorzugsweise in späteren Semestern zu hören ist [Hoc+11]. Das vorgestellte Curriculum ist nicht spezifisch auf eingebettete Systeme, sondern auf Technische Informatik im Allgemeinen ausgerichtet.

Die Kompetenzbeschreibungen, die in den GI-Empfehlungen vorgeschlagen werden, sind nur in wenigen Fällen überprüfbar. Die Formulierung „Sie besitzen Verständnis über ...“ kann bspw. durch einen Beobachter nicht eindeutig wider- oder belegt werden. Hier zeigt sich die abstrakte Natur der Empfehlungen. Eine direkte Übernahme

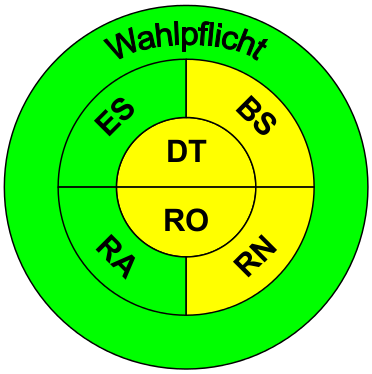

Abbildung 2.5.: Curriculums-Aufbau für den Bachelorstudiengang [Hoc+11]

der Empfehlungen in ein Lehrkonzept ist deswegen schwierig. Dies trifft ebenso auf die transparente Prüfungsgestaltung und -bewertung zu, die sich am intendierten Kompetenzgewinn der Studierenden ausrichten sollte (siehe *Constructive Alignment* in Abschnitt 6.1).

Obwohl die Empfehlungen des ARTIST-Gremiums älter als die der GI sind, beschäftigen sie sich bereits mit dem Studienschwerpunkt der Entwicklung eingebetteter Systeme als einen Aspekt der Technischen Informatik. Die Autoren haben die Problematik schnelllebiger Paradigmenwechsel erkannt und ihre Curriculaempfehlung entsprechend ausgerichtet. Der Lösungsansatz der Problemstellung lässt sich in der folgenden Aussage zusammenfassen:

> „*Emphasize the Role of Basic Knowledge and Computer Science.* We believe that a solid foundation in the fundamentals of engineering and computer science will allow students to benefit from the continuous training they will be exposed to throughout their professional life“ [S. 593 Cas+05, Hervorhebung im Original].

Die im Curriculum vermittelten Fundamente dienen also zum einen der Kompetenzaneignung im Studium, aber zum anderen auch explizit der längerfristig angelegten Expertenbildung in der Praxis. Im Unterschied zu den GI-Empfehlungen orientiert sich die ARTIST-Gemeinschaft an Lehr-/Lerninhalten auf Master- und nicht auf Bachelorniveau. Fachliche Schwerpunkte werden auf Signalverarbeitung, Theoretische Informatik, Echtzeitsysteme, verteilte Systeme, nicht-funktionale Anforderungen und Design auf Ebene der Systemarchitektur gesetzt. Die Autoren sind der Meinung, dass das spätere Arbeitsumfeld nicht der geeignete Ort ist, um diese Themenauswahl gründlich genug zu vermitteln oder sich selbst anzueignen. Mit einer ähnlichen Begründung empfehlen sie eine möglichst hohe Diversität an methodischen Ansätzen, um die Unterschiede zwischen diesen an die Studierenden weiterzugeben. Im industriellen Umfeld ist die verfügbare Zeit zu knapp, um dies nachzuholen. Die Suche nach Fundamenten einer Disziplin ist ein wesentliches Merkmal der Empfehlungen, dessen Ansatz sich auch in der Entwicklung des in dieser Arbeit vorgestellten fachdidaktischen Konzeptes widerspiegelt (siehe Kapitel 4).

In Zusammenarbeit mit der IEEE hat die ACM im Dezember 2004 eine Curriculaempfehlung für den Bereich *Computer-Engineering* veröffentlicht, der fachlich gesehen zwischen der Elektrotechnik und der Informatik liegt [ACM04]. Somit ist es die Disziplin, die am ehesten dem deutschen Verständnis von *Technischer Informatik* entspricht. Ausführlich schildern die Experten, wie der Prozess zur Erstellung der Empfehlung strukturiert wurde. Neben einer Einordnung der Disziplin zwischen Informatik und Elektrotechnik spezifizieren die Autoren die mit dem Lehrplan verknüpften, zu erwartenden Eigenschaften eines Absolventen im Fachbereich. Diese sind folgendermaßen zusammengefasst:

Professionalität Studierende sollen ein Verständnis dafür haben, dass die von ihnen entwickelten Systeme die Gesellschaft direkt und indirekt beeinflussen. Die damit verbundene gesellschaftliche und ethische Verantwortung soll sich in ihrer Arbeit widerspiegeln.

Fähigkeit zum Designen Sie sollen Theorien und Konzepte der Wissenschaften nutzen, um ihre Produkte zu entwerfen. Zentral ist dabei die Analyse und Auswertung von Entwurfs- bzw. Implementierungsmöglichkeiten und Trade-Offs.

Breite des Wissens Die Autoren heben hervor, dass ein Studienplan für die Technische Informatik prinzipiell zwischen zwei Polen - einer breiten, aber tendenziell eher oberflächlichen und einer engen, aber sehr detaillierten Inhaltsauswahl - liegt.

Konzeptuell unternehmen die Autoren bereits an dieser Stelle einen ähnlichen Versuch wie die vorliegende Arbeit. Da der Raum möglicher Umsetzung eines Studiums der Technischen Informatik unüberschaubar groß ist, müssen Konzepte gefunden werden, die ein Mindestmaß an Gemeinsamkeiten gewährleisten und die Disziplin grundlegend strukturieren. Eine interessante Erkenntnis der Experten ist dabei die Notwendigkeit, das Curriculum immer wieder zu aktualisieren, da die Technische Informatik im Besonderen eine schnelllebige Disziplin ist und sich deswegen die technischen Grundlagen ständig ändern [ACM04]. Hier unterscheidet sich das Anwendungsgebiet von den klassischen Naturwissenschaften wie der Mathematik, der Biologie und der Chemie, die ein breites Spektrum an beständigen Grundlagen aufweisen können.

Das Konsortium empfiehlt 18 Knowledge-Areas (KAs), die die Lerninhalte thematisch gliedern und durch Angaben wie Pflichtstunden und intendierten Kompetenzzuwachs beschreiben. Da die Empfehlungen der ACM und der IEEE nicht spezifisch auf eingebettete Systeme ausgerichtet sind, fokussiert nur eine KA Kompetenzen im Anwendungsgebiet. In der KA für eingebettete Systeme sollen die Studierenden insgesamt 20 Kernstunden belegen, die in sechs obligatorische Gebiete strukturiert sind. Darunter sind auch die zwei designspezifischen Lehr-/Lerneinheiten *Reliable-System-Design* und *Design-Methodologies*. Neben der Tatsache, dass der Bereich des Entwurfes eingebetteter Systeme durch diese zwei Module nur ausschnittsweise dargestellt ist, muss auch kritisiert werden, dass die Lernziele nicht durchgängig kompetenzorientiert definiert sind (vgl. [Kra02], [Ful+07]).

Die drei analysierten Curriculaempfehlungen von der GI, der ARTIST-Gemeinschaft und der ACM/IEEE betonen die Suche nach Fundamenten der Disziplin. Die fachdidaktische Forschung von Schwill [Sch93] zum Themengebiet der *„Fundamentalen Ideen der Informatik“* ist aus diesem Grund eine wichtige wissenschaftliche Vorarbeit, die wegen des besonderen Stellenwertes für die Arbeit in einem eigenen Kapitel erläutert und erweitert wird (siehe Abschnitt 4).

3. Projektarbeit KOMINA

Im Rahmen des Projektes „Kompetenzentwicklung mit Eingebetteten Mikro- und Nanosystemen“ (KOMINA), in dem der Autor mit geforscht hat, wurden bereits einige entscheidende Schritte hin zu einer Didaktik für Entwickler eingebetteter Systeme erzielt. Dieser Abschnitt stellt das Forschungsvorgehen chronologisch dar. Abschließender Gegenstand des Kapitels sind die entwickelten Lehr-/Lernunterstützungen, die auf den Vorarbeiten von KOMINA aufbauen (siehe Abschnitt 3.4) und den Arbeitsschwerpunkt des Autors im Projekt bildeten.

3.1. Notwendigkeit und Zielstellung

Die Lehrstühle *Rechnerarchitektur der Universität Erlangen-Nürnberg*, *Mikrosystemtechnik der Universität Siegen* und die *Didaktik der Informatik und E-Learning der Universität Siegen* führten von Juli 2010 bis Juli 2014 das DFG-geförderte KOMINA-Projekt durch. Inhaltlicher Schwerpunkt war ein Beitrag zur Technischen Informatik und ihrer Didaktik mit folgenden Forschungszielen:

> „1) Ein theoretisch fundierter Beitrag zur systemorientierten Didaktik der Technischen Informatik einschließlich lernförderlicher Experimente wird erbracht.
>
> 2) Ein Kompetenzmodell (Dimensionen, Niveaustufen) wird eingeführt, um das Entwerfen von EMNS mit gemischt digital-analogen Funktionsprinzipien zu unterstützen. Dazu ist ein Verstehen grundlegender Hardwarefunktionalität durch Beschreibung physikalischer Phänomene auf einer qualitativen Ebene erforderlich.
>
> 3) Der Paradigmenwechsel hin zu Bottom-Up-Techniken und Redundanz beim Entwurf von EMNS wird nachvollzogen. Hier ist es bedeutsam zu lernen, durch Redundanz fehlerbehaftete Systeme nutzbar zu machen und den Weg von den klassischen zentral gesteuerten Systemen hin zu parallelen, verteilten Systemen in Kombination mit einer Strukturbildung durch Selbstorganisation zu finden.“ [SBF10, S. 1]

Das systemorientierte didaktische Vorgehen, das in Ziel eins vorgestellt wurde, bezieht sich auf die Erforschung lernförderlicher Wissensstrukturen, Aufgabenklassen und Explorationsmodule, wie es auch in [BS02] angewandt wurde. Grundvoraussetzung für alle genannten Forschungsziele ist die Etablierung eines

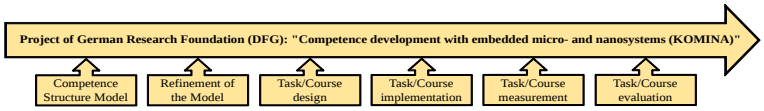

Abbildung 3.1.: Forschungsvorgehen im KOMINA-Projekt [Sch+12a]

Kompetenzstruktur- und im weiteren Verlauf auch eines Kompetenzniveaumodells (KSM bzw. KNM), um sich über die Inhalte und den mit ihnen verknüpften Prozessen klar zu werden. Diese Aufgabe findet sich in der zweiten Zielstellung des Projektes wieder.

Wie Marwedel hervorgehoben hat, ist die didaktische Situation in der Lehre eingebetteter Systeme dadurch beeinträchtigt, dass Hardwarekomponenten und Laborinstrumente nur schlecht verfügbar sind [Mar11]. In Teilziel eins inbegriffen ist deswegen die Konzeption, Gestaltung und Evaluation fachdidaktisch begründeter Lehr-/Lernapplikationen, um die genannte Problemsituation zu verbessern.

Das KSM orientiert sich nicht an der Universität Siegen, sondern wird in einem zweigeteilt normativ-empirischen Verfahren aus international anerkannten Empfehlungen erarbeitet. Anders als bei einem KNM gliedert ein KSM das Anwendungsgebiet durch die Erfassung verschiedener Kompetenzbeschreibungen. Diese Struktur ist wichtig, um praktische Lehr-/Lernszenarien hinsichtlich des intendierten Kompetenzgewinns einzuordnen. Die Forscher planten die Erweiterung des KSM um eine Kompetenzniveaustufung, welche die Abbildung von Lerninhalten auf Studiumsfortschritt und Vorerfahrungen unterstützt (das KNM).

Das dritte und letzte Schwerpunktziel der Forschergruppe ist der Paradigmenwechsel im Systementwurf eingebetteter Mikro- und Nanosysteme. Durch die Möglichkeit immer kleinerer Fertigungsgrößen im Entwurf eingebetteter Systeme müssen zukünftige Experten den Einfluss physikalischer Gesetze auf Ebenen der Mikro- und Nanotechnik verstehen. Dadurch ergeben sich Anforderungen und Rahmenbedingungen, die auf den Ebenen des Designs, der Implementierung und des Tests nicht vernachlässigt werden dürfen. Die Förderung der damit verknüpften Kompetenzen aus dem KSM sollten innerhalb eines virtuellen Praktikums „Nano-Arch-Online“ erfolgen. Eine Validierung der Ergebnisse war durch die Aufstellung einer Kontrollgruppe mit traditionellen Lerninhalten gegeben.

Die in KOMINA avisierte Zielgruppe sind Studierende mit Schwerpunkt Informatik oder Elektrotechnik. Die Forschungsergebnisse sollten dabei weder auf ein spezifisches Nebenfach, noch auf den Hochschulabschluss (Bachelor oder Master) eingegrenzt werden.

3.2. Forschungsprozess und -ergebnisse

Durch die Analyse von Modulbeschreibungen verschiedener deutscher Universitäten mit besonders gutem Ruf (Exzellenzuniversitäten) und der Untersuchung internationaler Curricula von IEEE/ACM [Cas+08] sowie den GI-Bildungsstandards [Hoc+11]

C	Kompetenzdimension	Kompetenzklasse
C1	**Vorwissen**	
C1.1	Mathematik	A
C1.2	Physik	B
C1.3	Informatik	B
C1.4	Elektrotechnik	C
C1.5	Englisch	B
C1.6	Wissenschaftliches Arbeiten	C
C1.7	Lernprozesse	B
C2	**Entwicklungskompetenzen**	
C2.1	Organisation	A,B
C2.2	Anforderungsanalyse	A,B,C
C2.3	Systemdesign	A,B,C
C2.4	Implementierung	A,B
C2.5	Optimierung and Test	B,C
C3	**Multi-level Entwicklungskompetenzen**	
C3.1	Top-Down Design	A,B
C3.2	Bottom-Up Design	A,C,D
C3.2.1	Nano-Effekte	~
C3.3	Jojo-Design	B
C4	**Nicht-kognitive Kompetenzen**	
C4.1	Einstellungen	B
C4.2	Soziale- und Kommunikationsfähigkeiten	A
C4.3	Motivationale & Volitionale Fähigkeiten	A

Tabelle 3.1.: Überblick über die Kompetenzdimensionen [Jas+12]

wurde eine normative Sammlung an Kompetenzbeschreibungen in einem „normativen Kompetenzstrukturmodell“ (NKSM) erarbeitet. Innerhalb des Forschungsprozesses wurden vier Kompetenzdimensionen ermittelt, die die Disziplin der eingebetteten Systementwicklung thematisch strukturieren (siehe [Sch+12a]).

Die erste Dimension „C1: Kompetenzen als Voraussetzung“ erfasst Kompetenzen von der Informatik nahestehenden Bereichen wie der Physik, Mathematik und der Elektrotechnik. Neben der Informatik selbst sind hier auch die Kategorien Lernorganisation und wissenschaftliches Arbeiten zugeordnet. Da es sich bei den in der ersten Dimension aufgelisteten Einträgen hauptsächlich um ganze Disziplinen handelt, haben sich die Projektmitglieder dazu entschieden, keine weiter differenzierte Angabe zu den jeweils beinhalteten Kompetenzbeschreibungen anzugeben. Für detailliertere Kompetenzbeschreibungen müsste ein für jede genannte Disziplin anerkanntes Kompetenzstrukturmodell vorliegen.

Für die Informatik wurden differenzierte Kompetenzbeschreibungen in Dimension 1 anfangs noch angegeben, aufgrund der damit verbundenen Heterogenität innerhalb der Dimension aber wieder verworfen. Der gleiche themenzentrierte Strukturierungsansatz wurde für die zweite Dimension „C2: Entwicklungskompetenzen“

gewählt. Die Ausrichtung der Subdimensionen folgt dem in der eingebetteten Systementwicklung üblichen Entwicklungsvorgehen (siehe Abschnitt 2.1).

Die Kompetenzdimension „C3: Kompetenzen für die Multilevel-Entwicklung“ ist nach Ansatz- und nicht nach Themenschwerpunkten strukturiert. Sie besteht aus dem Top-Down-, Bottom-Up-, Meet-in-the-Middle- und Jojo-Ansatz [Sch+12a]. Die Auftrennung von Entwicklungskompetenzen in C2 und C3 soll die Bedeutung neuartiger Entwicklungsvorgehen berücksichtigen, die Top-Down- und Bottom-Up-Techniken auch für die Hardwareentwicklung miteinander verbinden.

Neben den fachlich-kognitiven Kompetenzen wurden in „C4: Nicht-kognitive Kompetenzen“ Subdimensionen für Kompetenzen in Zusammenhang mit Einstellung, sozial-kommunikativen Verhalten sowie Motivation und Volition eingeführt (d.h. Orientierung am Kompetenzbegriff nach Weinert und Klieme [Kli+04]). Das so entstandene Kompetenzstrukturmodell war das erste seiner Art für die Technische Informatik und stellt eine international gültige, thematisch orientierte Kompetenzsammlung dar. Um zu überprüfen, ob die Strukturierung valide ist und alle wichtigen Kompetenzbeschreibungen erfasst wurden, bedurfte es einer empirischen Verfeinerung. Diese wurde zweigeteilt durchgeführt. Wenn im Folgenden vom empirisch verfeinerten Kompetenzstrukturmodell gesprochen wird, ist damit die Zusammenfassung beider Umfragen gemeint.

In der ersten Umfrage wurden 96 Professoren der Informatik an deutschen Universitäten um ihre Einschätzung zu allen Kompetenzbeschreibungen im NKSM gebeten (38 Rückmeldungen, 96 versendet). Die Experten konnten die Kompetenzbeschreibungen als *sehr wichtig*, *eher wichtig*, *eher unwichtig* und *unwichtig* bewerten und somit die Validität des Modells einschätzen. Nach dieser Umfrage konnten große Teile des NKSMs als gültig bestätigt werden. Für die empirische Verfeinerung befragte das Projektteam 75 Fachhochschulprofessoren in gleicher Art und Weise (21 Rückmeldungen, 75 versendet). Die Kategorisierung der Dimensionen und die Einschätzung der Experten ist überblicksartig in Tabelle 3.1 dargestellt.

Die Umfragebögen für die Universitäts- und Fachhochschulprofessoren waren nicht identisch. Zum einen enthielt die Subdimension C1.3 Informatik, wie bereits erläutert, Kompetenzbeschreibungen, die in späteren Versionen nicht übernommen wurden. Zusätzlich enthielt der Fragebogen für die Fachhochschulprofessoren mehrere englische Kompetenzbeschreibungen, die nicht für den ersten Fragebogen verwendet worden sind. Diese sind *„Discuss the concept of parallel processing and the relationship between parallelism and performance“* und *„Understand how performance can be increased by incorporating multiple processors on a single chip“*. Insgesamt konnten die Experten im ersten Fragebogen (Universität) 48 und im zweiten (Fachhochschule) 47 Kompetenzbeschreibungen bewerten.

Die Abbildung 3.2 zeigt das Ergebnis der Expertenbefragung für jede Kompetenzbeschreibung. Aus Platzgründen sind nur die Nummern der Kompetenzbeschreibungen, nicht aber deren Formulierung in der Grafik eingetragen. Wie in der genannten Abbildung zu sehen, sind nach Meinung der Teilnehmer die nicht-kognitiven Kompetenzen (C4) und die ersten Schritte im Entwicklungsvorgehen (C2.1 und C2.2) besonders wichtig. Erstaunlicherweise wurden die Kompetenzen,

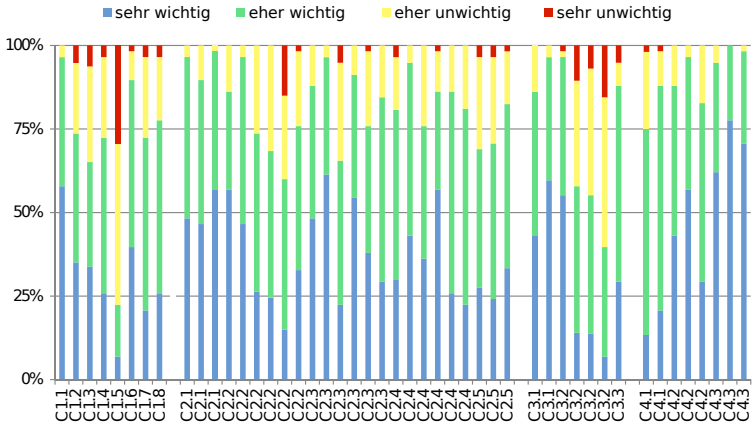

Abbildung 3.2.: Ergebnisse beider Expertenumfragen [Sch+12c]

die in Zusammenhang zu einem Bottom-Up-geprägten Entwicklungsvorgehen stehen, schlecht bewertet. Bei genauerer Betrachtung betrifft dies jedoch lediglich die mit Nanotechnik verbundenen Aspekte. Bottom-Up-Vorgehensweisen an sich sehen über 90% der Experten als wichtig an. Hieraus konnte geschlossen werden, dass Kompetenzen der Nanotechnologie für Entwickler eingebetteter Systeme eine vergleichsweise geringe Rolle spielen. Ein Grund kann sein, dass der Themenbereich eher der Fertigungstechnik zugeordnet wird. Eine Begründung für die auffallend vom Mittelwert aller Items abweichende Bewertung gab keiner der Experten an. Die schlechteste Expertenbewertung erhielt die Subdimension „C1.5 Materialwissenschaften“. Weniger als 25% der befragten Universitäts- und Hochschulprofessoren halten Kompetenzen in diesem Bereich für wichtig. Diese zwei Besonderheiten führten zu einem gegenüber dem NKSM abgeänderten EKSM. Die Nanotechnik-Kompetenzen wurden aus dem Bereich „Bottom-Up“ herausgenommen und in eine darunter liegende Subdimension eingeordnet. Die unterschiedliche Gewichtung von Bottom-Up und Nanotechnik ist so deutlicher zu erkennen, ohne die thematische Verwandtschaft der Begriffe aufgelöst zu haben.

Aufgrund der mehrheitlich schlechten Bewertung der Materialwissenschaften wurden diese aus der endgültigen Version des EKSM entfernt. Das Experten-Rating hat belegt, dass Kompetenzen in diesem Bereich unwichtig für das Verständnis und die Entwicklung eingebetteter Systeme sind. Weitere Erläuterungen zur Analyse der EKSM-Ergebnisse können in Schäfer et al. nachgelesen werden [Sch+12a].

Die Forschungsergebnisse des KOMINA-Teams wurden während der ganzen Projektphase mehrmals der wissenschaftlichen Gemeinschaft vorgestellt und diskutiert. Ein erster Ansatz zur Niveaustufung konnte in [Jas+11] aufgezeigt werden. Im Artikel wurde die Taxonomie von Anderson und Krathwohl für die Kompetenzbeschreibungen verwendet [AK01]. Rückblickend wäre die Taxonomie von Fuller et al. besser geeignet gewesen, da sie typische informatische Tätigkeiten (bspw. Modellieren und Debuggen) in die Taxonomie von Anderson und Krathwohl übersetzt und letztere deutlich erweitert [Ful+07]. Die Taxonomie von Anderson und Krathwohl

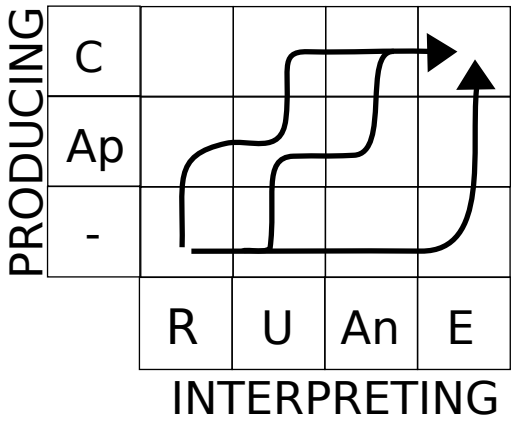

Abbildung 3.3.: Beispielhafte Lernpfade in der Taxonomie von Fuller et al. [Ful+07]

ist nicht an ein bestimmtes Fachgebiet oder Anwendungsfeld gebunden und gibt eine Skala mit sechs Stufen vor, die durch stellvertretend gewählte Substantive verschiedenen Niveaustufen zugeordnet werden können. Die Basis der Forschungsarbeiten von Anderson und Krathwohl ist die Bloomsche Taxonomie [BCE56]. Fuller et al. trennen die eindimensionale Taxonomie von Anderson und Krathwohl auf und ordnen sie in einer 4x3-Matrix neu an (siehe Abb. 3.3). Somit ergeben sich die zwei Achsen interpretierender und erstellender kognitiver Fähigkeiten. Die Ebenen *Erstellen (C)* und *Anwenden (Ap)* sind den Zeilen der Matrix zugeordnet. Zusätzlich gibt es noch eine Zeile für nicht-produzierende Fähigkeiten (-). Die Spalten sind mit den interpretierenden Fähigkeiten *Erinnern (R)*, *Verstehen (U)*, *Analysieren (An)* und *Evaluieren (E)* beschriftet. Durch diese neue Anordnung können Fähigkeiten differenzierter als bisher beschrieben werden. Ein Studierender kann beispielsweise einen Lerninhalt wiedergeben (Erinnern), aber nicht anwenden oder damit etwas erstellen. Entsprechend würde die Fähigkeit des Studierenden im linken, unteren Bereich der Matrix (-,R) liegen. Dadurch kann die Taxonomie verschiedene *Lernpfade* von Studierenden beschreiben. Einige Studierende arbeiten sofort praktisch, eignen sich das notwendige fachliche Wissen und das damit verbundene tiefere Verständnis aber erst später an, während andere Studierende den Vorgang erst von der theoretischen und später von der praktischen Position aus beginnen (siehe Abbildung 3.3).

Die Veröffentlichung [Jas+11] beinhaltete ebenso die Herausforderungen und die Konzeption von FPGA-Online. Dies ist ein Online-Praktikum an der Universität Erlangen-Nürnberg, in dem die Studierenden erste Erfahrungen mit rekonfigurierbarer Hardware machen können. Um eine flexible Lernsituation zu ermöglichen, sind alle Lerninhalte auch außerhalb der Lehrveranstaltung abrufbar. Dies schließt die Konfiguration der FPGAs mit ein.

Zusätzlich hat das KOMINA-Team den Prozess der Expertenumfragen hin zum empirisch verfeinerten Kompetenzstrukturmodell (EKSM) beschrieben. Erste Ent-

würfe für ein Nanotechnik-Praktikum, wie im Projektablauf geplant, stellte Kleinert et al. in [Kle+12] vor. Die Auswertung der Ergebnisse und damit die Endfassung des EKSM wurde in [Sch+12c] präsentiert. Um eine differenzierte Betrachtung der Kompetenzbewertungen zu ermöglichen, entschied sich die Forschergruppe für ein Klassifizierungssystem bestehend aus sechs Kategorien (A-F). Die Begründung hierfür liegt im überwiegend positiven Rating der Kompetenzbeschreibungen. So wurden 45 der 47 Kompetenzbeschreibungen der Fachhochschulumfrage als gut bewertet. Somit ist die Einteilung in gut bzw. schlecht nicht mehr aussagekräftig. Folgende Abstufung wurde vorgenommen:

A: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibung als *sehr wichtig*.

B: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibungen als *sehr wichtig* oder *eher wichtig* (mindestens eine Wertung). Die Anzahl der *sehr wichtig*-Bewertungen ist höher als die zusammengefassten Bewertungen für *eher unwichtig* und *sehr unwichtig*.

C: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibungen als *sehr wichtig* oder *eher wichtig*. Die Anzahl der *sehr wichtig*-Bewertungen ist kleiner oder gleich der zusammengefassten Bewertungen für *eher unwichtig* und *sehr unwichtig*.

D: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibungen als *sehr unwichtig* oder *eher unwichtig*. Die Anzahl der *sehr wichtig*- und *eher wichtig*-Bewertungen ist größer als die Anzahl der *sehr unwichtig*-Bewertung.

E: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibungen als *sehr unwichtig* oder *eher unwichtig* (mindestens eine Wertung). Die Anzahl der *sehr wichtig*- und *eher wichtig*-Bewertungen ist kleiner oder gleich der Anzahl der *sehr unwichtig*-Bewertung.

F: Mehr als die Hälfte der Experten bewerteten die Kompetenzbeschreibungen als *sehr unwichtig*.

Neben einer abgestuften Einordnung der jeweiligen Kompetenzgebiete ist auch die Zuordnung der Kompetenzdimensionen überprüfbar. Dazu wird neben der Bewertung aller Kompetenzbeschreibungen einer Kategorie auch der Durchschnittswert für diese gebildet (siehe Abbildung 3.4). Somit können „statistische Ausreißer“ gefunden werden. Ein Beispiel ist die Kompetenzsubdimension der „C3.2 Bottom-Up“ Techniken. Während der Durchschnittswert (in der Grafik schwarz geschrieben) in Klasse C liegt, finden sich ebenso noch Beschreibungen dieser Subdimension in Klasse D und A. Während erstere nur eine Ebene neben der Durchschnittsklasse liegt, ist ein Rating in Kategorie A deutlich von dieser entfernt. Vermutet wird eine unsaubere Vermischung zwei verschiedener Kompetenzgebiete. Wie bereits vorher erläutert, ist in diesem Fall die Einschätzung der Experten zu Kompetenzen im Bereich Bottom-Up und denen zur Nanotechnik grundverschieden.

Eine Kompetenzdimension, die nach der Meinung eines Experten nicht bedacht wurde, ist die der Echtzeitsysteme. Das finale EKSM wurde in [Sch+12c] präsentiert und ist zusammengefasst auf der rechten Seite der Abbildung 3.4 zu sehen.

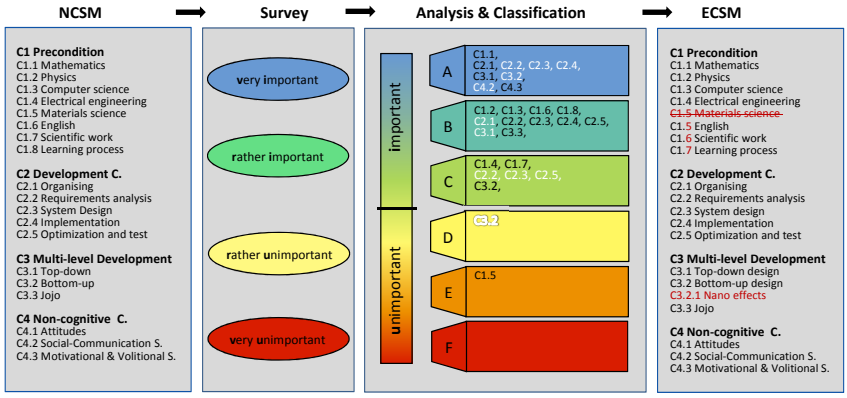

Abbildung 3.4.: Herleitung des empirisch verfeinerten Kompetenzstrukturmodells [Sch+12c]

Wie bereits erwähnt, sind die Kompetenzen aus Wertungsklasse A von besonderer Bedeutung für die Erstellung einer exemplarischen Laborveranstaltung, dem Entwurf und der Umsetzung lernförderlicher Informatiksysteme und der Auswahl von Themenschwerpunkten bei der Anwendung des für Teilziel drei zu erstellenden Lehrkonzeptes (siehe Kapitel 4). Das EKSM ist zugleich eine Grundlage für die später umgesetzten Lehr-/Lernhilfen (siehe Abschnitt 3.4.2 und 3.4.3).

3.3. Entwurfs- und Anwendungspraktikum (EAP)

Das bestehende Hardwarepraktikum (HaPra) der Universität Siegen konnte mit den Forschungsergebnissen des EKSM exemplarisch in ein „Entwurfs- und Anwendungspraktikum für eingebettete Mikrosysteme“ (EAP) umstrukturiert werden. Das Praktikum ist wesentlich für die vorliegende Arbeit, da in einer Beobachtung der Teilnehmer grundsätzliche Defizite erkannt wurden, die zur Entwicklung unterschiedlicher Lehr-/Lernunterstützungen sowie der Erforschung fundamentaler Themengebiete in der Entwicklung eingebetteter Systeme führte.

Die Resultate der bisherigen Projektarbeit des KOMINA-Teams fließen in Form der Kompetenzendimensionen A und B ein, die besonders im EAP gefördert werden. Das ursprüngliche HaPra enthielt elf Laborversuche, in denen die Studierenden schrittweise einen eigenen Prozessor auf einem FPGA entwickeln und umsetzen sollten. Dies ist insofern kritisch, da sich seit Entstehung des HaPras das Teilnehmerprofil deutlich geändert hat und so nicht davon ausgegangen werden kann, dass alle Teilnehmer ein vergleichbares Vorwissen hinsichtlich der gestellten Aufgaben besitzen. Durch den Bolognaprozess sind weniger Diplomstudierende und mehr Bachelorstudierende sowie vier verschiedene statt einem Studiengang vertreten. Während 2007 alle Teilnehmer Informatik mit Nebenfach Elektrotechnik belegten, waren es 2012 nur noch 45%. Zu etwa gleichen Teilen von 26% waren nun Studierende mit Nebenfach Mathematik und Medienwissenschaften unter den Teilnehmern.

Ein deutlich kleinerer Teil der Studierenden wählte Automotive System Engineering. Die damit einhergehenden Unterschiede im Vorwissen der Studierenden und die durch das Praktikum zu erlangenden Kompetenzen erforderten eine Umstrukturierung des Praktikums [Jas+12]. Das im EAP zu realisierende Projekt wurde in eine Hausautomation geändert. Die Teilnehmer erhielten pro Gruppe ein aus Holz gebautes Modellhaus, das mit verschiedenen Sensoren (Licht-, Temperatur-, Erschütterungssensor und Reed-Kontakt) und Aktuatoren (Heiz-/Kühlelement, Ventilator und LED) ausgestattet war. Eine dazugehörige Steuereinheit überwacht die Ansteuerung und Regelung dieser Peripherie und verfügt zusätzlich über eine Bluetooth-Schnittstelle, mit der die komplette Haussteuerung von einem Smartphone aus geregelt werden kann. Somit verfügt die Aufgabenstellung über einen Lebensweltbezug, der auch Studierende anderer Studiengänge motiviert. Durch die Diversität der Studiengänge liegt im EAP ein besonderer Fokus auf der nebenfachunspezifischen Vermittlung grundlegender Techniken und Methoden. Um den Vorwissensstand aller Teilnehmer einem Niveau anzunähern, ist das Praktikum neben der *Projektumsetzung* auch in einen *Einführungs*-Teil untergliedert.

Die als notwendig betrachteten Grundlagen der Elektrotechnik werden in den ersten vier Einführungsveranstaltungen praktisch erprobt. In der Implementierung nutzen die Studierenden die neu erworbenen Kompetenzen, um die genannte Hausautomation zu realisieren. Die Praktikumsversuche sind wie folgt strukturiert:

1. **Schaltkreise und Systemkomponenten:** Elektrotechnische Grundlagen wie die Brückenschaltung, das Messen von Strom- und Spannung sowie die Berechnung von Widerständen werden wiederholt und in kleinen Steckbrett-Aufbauten praktisch angewandt.

2. **Einführung Sensoren:** Studierende nutzen erstmalig Oszilloskop und Multimeter, um ohne Computerunterstützung Sensordaten wie die Raumtemperatur auszulesen.

3. **Einführung Aktuatoren:** Ähnlich zum vorherigen Praktikumsversuch verwenden die Teilnehmer einen Funktionsgenerator, Aktuatoren und Puls-Weiten-Modulation, um Umgebungsparameter in kleinem Maße zu beeinflussen.

4. **Grundlagen Transistoren:** Mittels eines Tief- und eines Hochpasses lernen die Studierenden mit dem Funktionsgenerator umzugehen, der für spätere Versuche wichtig ist.

5. **Programmierung:** Jedes Studierendenteam, bestehend aus zwei Personen, hat die Wahl, ob es die Hausautomation mit einem Mikrocontroller oder einem FPGA umsetzt. In dieser Veranstaltung werden die Grundprinzipien der jeweiligen Plattform erläutert und ein erstes Programm geschrieben.

6. **Logikgestützte Auswertung:** Die bereits in der ersten Hälfte des Praktikums verwendeten Sensoren sollen mit Hilfe der Recheneinheit (Mikrocontroller oder FPGA) ausgelesen und direkt in ein menschenlesbares Format umgewandelt werden.

7. **Logikgestützte Steuerung:** Durch Verwendung von Analog-Digital-Wandlern (ADCs) mit geeigneter Ansteuerung sind die Studierenden in der Lage, externe

Aktuatoren wie eine LED oder ein Peltier-Heizelement anzusprechen. Hierfür müssen sie sich gründlich mit der Dokumentation der jeweiligen Bauteile beschäftigen.

8. **Ganzheitlicher Steuerungszyklus:** Im letzten regulären Versuch verknüpfen die Teilnehmer die in den vorherigen Lehr-/Lerneinheiten gewonnenen Kompetenzen, um einen Regelungskreislauf zu gestalten. Teilaufgaben sind beispielsweise das Abschalten des Heizelementes, wenn sich die Haustür öffnet, oder die Temperaturregelung auf einen zuvor festgelegten Wert. Abhängig vom Praktikumsverlauf ist eine zusätzliche Aufgabe eingeplant, in der die Steuerung der Hausanlage über ein Bluetooth-fähiges Android-Smartphone umzusetzen ist.

Das bis hier beschriebene Kurskonzept ist als Testveranstaltung gedacht, deren Erkenntnisse auf zukünftige Lehrveranstaltungen angewandt werden können. Alle Aufgaben der geschilderten Veranstaltung sind eng an die Strukturierung und Gewichtung der Kompetenzbeschreibungen des EKSM gekoppelt [Jas+12]. Die Siegener Forscher haben zur besseren Organisation des EAP ein in die einzelnen Versuchsreihen strukturiertes Hilfsdokument erstellt, das auszugsweise an die Studierenden verteilt wurde. Neben einer Einführung in die Thematik und den damit verbundenen Grundlagen enthielt das Dokument auch ausgewählte Testaufgaben, mit denen die Teilnehmer selbstständig ihren Wissensstand verifizieren konnten. Durch die Darstellung der Kompetenzdimensionen zu einer Aufgabenstellung können Studierende nachvollziehen, welche Aspekte der Aufgabe von den Betreuern als zentral angesehen werden. Die in der Textbox „Kompetenz- und Aufgabenbeschreibungen“ dargestellten Kompetenz- und Aufgabenbeschreibungen sind der fünften Versuchsreihe entnommen, der ersten, in der mit FPGAs und Mikrocontrollern gearbeitet wird.

Kompetenz- und Aufgabenbeschreibungen:

Kompetenzen für den gesamten Versuch

- Die Studierenden können Schaltungen mittels einer Beschreibungssprache entwerfen.
- Die Studierenden verstehen den Umgang mit der Programmiersprache C oder der Hardwarebeschreibungssprache VHDL und deren Zusammenspiel mit der Hardware.

Aufgaben aus der Vorbereitung

- Mikrocontroller und FPGA arbeiten auf grundlegend verschiedene Weisen. Ihre Aufgabe ist es, sich über die Unterschiede ein Bild zu machen. Nutzen Sie hierfür die Materialien im Anhang dieser Versuchsreihe.
- Welche Software oder Hardware wird in der Regel benötigt, um einen Mikrocontroller zu programmieren?

Aufgaben aus der Durchführung

- Mikrocontrollergruppe: In der Vorbereitung haben Sie sich mit den Werkzeugen und Arbeitsschritten zur Übertragung von Programmen auf den Mikrocontroller auseinandergesetzt. Versuchen Sie nun den selbst erstellten Quellcode auf den Mikrocontroller zu laden.
- Bei Betätigung eines Tasters (wählen Sie einen beliebigen) auf dem Mikrocontroller soll für den Zeitraum des Tastendrucks eine LED leuchten. Wird der Taster losgelassen, soll auch die LED ausgehen. Im Datenblatt des Mikrocontrollers finden Sie die notwendigen Informationen bzgl. der PIN-Belegungen der Taster.

Aufgaben aus der Nachbereitung

- Mikrocontroller-Gruppen: Machen Sie sich mit den Möglichkeiten der ADCs und DACs auf dem Mikrocontroller vertraut. Wie werden die entsprechenden Ports angesprochen?
- FPGA-Gruppen: Beschreiben Sie den Ablauf einer „Programmerstellung“ für den FPGA. Welche Schritte werden wann durchlaufen? Welche Aufgabe und welches Endprodukt ist nach jeder Stufe zu erwarten? Gibt es Stufen, die übersprungen werden können?

Wie zu erkennen ist, sind alle Laborveranstaltungen in drei Teile gegliedert (*Vorbereitung, Durchführung* und *Nachbereitung*). Die Vorbereitung hat den Zweck, alle Teilnehmer auf ein Mindestniveau an technischen Kenntnissen zu stellen. Damit dieses Ziel erreicht wird, sind die bis zu 17 Aufgaben umfassenden Vorbereitungen vor der Durchführung an die Betreuer der Veranstaltung zu schicken, die prüfen, ob die eingereichten Lösungen korrekt sind. Dieser Prozess beinhaltet in der Regel auch die Bearbeitung einer Aufgabenstellung im Moodle-System. Moodle ist die E-Learning-Plattform der Universität Siegen, die neben Kursorganisation und Aufgabenbeschreibungen auch erweiterte Kommunikationsmechanismen (bspw. Chat und Forum) sowie ein umfangreiches Testmodul für alle Veranstaltungen der Universität enthält. Letzteres wird für die Überprüfung des Vorwissensstandes der Laborteilnehmer genutzt. Dabei steht nicht die Bewertung der Studierendenleistung im Vordergrund, sondern die Einschätzung darüber, ob die Studierenden den zu bearbeitenden Stoff der Vorbereitung wirklich verstanden oder nur angewandt haben. In Moodle sind dafür verschiedene Aufgabentypen umgesetzt, die typischerweise, Multiple-Choice und Freitextaufgaben umfassen. Es wird von jedem Teilnehmer bzw. jeder Teilnehmergruppe erwartet, dass der Test zu mindestens 50% richtig beantwortet wurde. Erst dann sind die Studierenden für die eigentliche Laborveranstaltung zugelassen. Wie beschrieben, gibt es keine Benotung der Ergebnisse. Der Moodle-Test ist so ausgelegt, dass er beliebig oft wiederholt werden kann, damit die Studierenden ihr Verständnis des Sachgebietes mit verschiedenen möglichen Szenarien abgleichen können. Um das Abschreiben von Antworten in diesen Tests zu erschweren, werden die im Fragenpool angelegten Aufgaben in zufälliger Reihenfolge ausgegeben.

Ebenso wurde in Moodle ein Peer-Reviewing durchgeführt. Dafür wurde die Moodle-Workshop-Funktion genutzt. Beim Peer-Reviewing bewerten die Teilnehmer die

Arbeit anderer Gruppen (siehe [RCT07]). Dazu laden alle Studierenden im Kurs bspw. Teile ihres Quellcodes auf den Moodle-Server. Bei der Nutzung der Funktion ergaben sich jedoch einige technische Probleme. So konnte nur der Studierende andere Gruppen bewerten, der die eigenen Ergebnisse hochgeladen hat. Die Studierenden brauchen des Weiteren für die Form und den Umfang ihrer Rückmeldung an Kommilitonen eine Vorlage, da sich zeigte, dass die Kommentare in Qualität und Form sehr heterogen waren. Der Umfang reichte von einem Satz bis hin zu einer kompletten Textseite.

Die Analyse der Durchführung findet sich aufgrund des zentralen Stellenwertes für die Arbeit in Abschnitt 3.4.1.

3.4. Entwicklung von Lehr-/Lernunterstützungen

Während die bisherigen Arbeiten gleichermaßen zwischen allen beteiligten KOMINA-Mitgliedern aufgeteilt waren, liegt in den im folgenden Abschnitt dargestellten Forschungsergebnissen der Projektschwerpunkt des Autors. Dies umfasst die Beobachtung des EAPs zur Feststellung, welche Laborversuche potentiell schwierig sind und die darauf aufbauende Entwicklung lehr-/lernunterstützender Hilfsmittel. Alle diese Arbeiten wurden zusammen mit Steffen Jaschke entwickelt, durchgeführt und vorgestellt (siehe [BJ13], [JBS13], [BJS13]).

3.4.1. Beobachtung des EAPs

Die Beobachtungsphase des EAPs diente der Analyse von Lernverhalten bezüglich dem konzipierten Praktikum und daraus abgeleiteten Lernhypothesen. Bei entsprechend großen Teilnehmerzahlen wird in der Forschung häufig auf die Analyse einer Kontroll- und einer Experimentalgruppe zurückgegriffen. Erstere absolviert die Veranstaltung nach der traditionellen Weise, die Experimentalgruppe hingegen nach dem vorgeschlagenen Konzept. Voraussetzung für ein solches Vorgehen sind gleichartige Gruppen von Studierenden und eine große Anzahl an Teilnehmern. Beide Anforderungen konnten für das EAP nicht erfüllt werden. Wie bereits erwähnt zeichnet sich das Praktikum nach der Studiengangsumstellung an der Universität Siegen durch sehr unterschiedliche Studierendenprofile aus. Insgesamt besuchten 2012 30 Studierende die Veranstaltung. Aus diesen Gründen musste die Evaluation des Praktikums ohne Experimentalgruppe vollzogen werden. Aus diesem Grund wurde sich auch für die Beobachtung der Studierenden während der Präsenzphase der Veranstaltung entschieden. Die Beobachtung diente der Aufzeichnung von Diskussionen und Herangehensweisen innerhalb der Studierendengruppen beim Erarbeiten einer Aufgabenlösung. Dazu gehören auch die vorhandenen Fehlvorstellungen und fachlichen Schwierigkeiten, die später zur Entwicklung der entsprechenden didaktischen Hilfsmittel geführt haben (siehe Abschnitt 3.4.2 und 3.4.3). Da die Laborbetreuer für jeden Versuch ein Zeitlimit vorgegeben haben, war die tatsächlich benötigte Zeit für die Beobachter interessant, um Rückschlüsse auf die Konzeptionsqualität der Veranstaltung zu schließen.

Für die Durchführung wurde eine offene, nicht teilnehmende Beobachtungsstrategie gewählt, um die Lernsituation der Studierenden nicht zu verfälschen. Eine verdeckte Vorgehensweise war aus organisatorischen Gründen nicht möglich, da weder die Kommunikation der Gruppenteilnehmer untereinander noch die Lösungswege auf dem Steckbrett bzw. die Konfiguration von Multimeter, Oszilloskop oder Funktionsgenerator erkennbar gewesen wären. Die von Cranach und Frenz beschriebene „kurzzeitig wirkende Variable“ der Leistung konnte wegen fehlenden Vergleichswerten nicht untersucht werden [CF75]. Alle Studierenden wurden vom Praktikumsleiter vor der ersten Versuchsreihe auf die Anwesenheit von Beobachtern und den dazugehörigen Rahmenbedingungen hingewiesen (bspw. keine fachlichen Fragen). Während der regulären Praktikumszeit von 14:00 Uhr bis 17:00 Uhr waren bei jedem Versuch drei Beobachter anwesend. Aufgrund der Studierendenanzahl war eine Aufteilung in je zwei bzw. drei Gruppen pro Beobachter unumgänglich (d.h. vier bis sechs Studierende pro Beobachter). In einigen Fällen dauerte die Bearbeitung der Aufgaben länger als die eigentliche Versuchszeit. Die Teilnehmer wurden in diesen Fällen nicht über die eigentliche Versuchszeit hinaus beobachtet, jedoch notiert, bis zu welcher Aufgabe die Studierenden gekommen sind. Oft zeigte sich, dass die veranschlagte Zeit für die Durchführung nicht ausreichend war (insbesondere in den späteren Versuchen).

Ein auf Basis eines Testdurchlaufes erstelltes Beobachtungsprotokoll diente als Grundlage für jede Beobachtung. Hierbei wurde mit Zeitstichproben gearbeitet, da die ereignisbasierte Protokollierung mangels ausreichender Vortests schwierig war. Die Beobachtung zielte auf Grund der Beobachteranzahl auf Gruppen- und nicht auf Einzelleistungen ab.

Für die Analyse der Beobachtungen wurden die von den Beobachtern notierten Schwierigkeiten in Kategorien zusammengefasst und quantitativ ausgewertet. Als problematisch erwies sich, dass eine solche Vorgehensweise nur in den ersten Laborversuchen möglich war (siehe [Jas+12]). Die Arbeit mit Mikrocontrollern und FPGAs machte die differenzierte Klassifizierung von Fehlern schwierig, weswegen hierfür keine aussagekräftige Statistik angegeben werden kann. Generelle Tendenzen konnten dennoch erfasst werden. So gab es im EAP keine Gruppe, die einen zielgerichteten, strukturierten Ansatz zur Informationsgewinnung aus Datenblättern für Bauteile nutzte. Dabei war nicht das technische Prozedere, die entsprechende Stelle im Datenblatt zu finden, problematisch, sondern die grundsätzliche Frage, in welcher Form die Daten dargestellt sind. Während die Konfiguration von ADCs und DACs oft über boolsche Werte in Registern erfolgt, sind Kennwerte von Sensoren und Aktuatoren meist grafisch dargestellt. Die Interpretation dieser Registerwerte und Funktionsverläufe ist ein großes Problem gewesen.

Damit ist die Schwierigkeit verbunden, die Bedeutung verschiedener Registerzuweisungen im Mikrocontroller zu verstehen. Bei ATmega-Prozessoren hängt die Interpretation eines Registerwertes unter Umständen von der Konfiguration eines anderen Registers ab. Auch der Vergleich von Sensorwerten mit Bedingungen im Programmcode war anfänglich problematisch, da die sonst übliche Notation aus dem Softwarekontext unter Umständen nicht mehr funktionierte (siehe Listing 1, Teil A). In Teil B des aufgeführten Programmcodes sind für die hardwarenahe Programmierung typische Konstrukte mit Bitoperationen zu sehen.

```
//---------Teil A---------//
if(DDRB == 0b00000001){...}
if(DDRB & 0b00000001){...}
if(DDRB && 0b00000001){...}

//---------Teil B---------//
DDRB |= (1<<PB5);
DDRB &= ~(1<<PB5);
ADCH = ADCL<<2;
REGA ^= 0b01010101;
```

Listing 1: Beispiele für Programmiersituationen mit hoher Fehlerrate [JBS13]

Diese waren Größtenteils unbekannt für die Praktikumsteilnehmer und stellten dementsprechend ein Hindernis bei der Lösung von Aufgaben dar. Das Fehlen einer vorkonfigurierten Debugschnittstelle für den Mikrocontroller erschwerte das Testen von Programmen zusätzlich. Hilfsmittel wie die Nutzung von LEDs, die je nachdem, ob eine Bedingung erfüllt wurde oder nicht, aus oder an sind, erfüllen Debug-Funktionalitäten nur bis zu einem gewissen Grad. Es zeigte sich, dass ein Abbruch der Versuche und eine anschließende Konsultation zu den generellen, gruppenübergreifenden Schwierigkeiten sehr konstruktiv ist, obwohl alle notwendigen Grundlagen als Vorarbeit gelernt und erfolgreich zum Beginn des Versuches geprüft wurden. Die Art der Vorbereitungsprüfung ist deswegen für zukünftige Durchführungen hinsichtlich didaktischer und inhaltlicher Schwerpunkte neu zu bestimmen.

Die Beobachtung zeigte, dass Studierende mit der Bedienung von Laborinstrumenten große Schwierigkeiten haben. Eine Auflistung der Problemhäufigkeiten in den ersten vier Versuchen, zusammengefasst nach Anwendungsgebiet, ist in Tabelle 3.2 zu sehen. Die in Spalte *Auftreten* dargestellten Werte beschreiben die absolute Häufigkeit von Problemen aller Gruppen in den ersten vier Versuchen. Kleinere Fehlkonfigurationen wurden in der Regel nicht als Problem eingestuft.

Auftreten	**Kategorie**
11	Oszilloskop: Interpretation und Bedienung
11	Bedienung des Funktionsgenerators und Verständnis der Puls-Weiten-Modulation
8	Multimeter-Einstellungen und Grundlagen der Elektrotechnik
5	Einfache Schaltungen
3	Netzteileinstellungen
3	Steckbrettaufbau

Tabelle 3.2.: Fehlerarten der ersten vier Versuche des HaPras (aus [JBS13])

Oszilloskop Die Studierenden hatten trotz Anleitungen zum im Labor verwendeten Oszilloskop Probleme, die richtige Verkabelung herzustellen. Bis auf wenige Außnahmen musste die Ausgabe des X-Y-Modus am Oszilloskop durch die Betreuer detailliert erklärt werden. Besonders interessant war dabei, dass sich kein Kompetenzgewinn abzeichnete und die Studierenden in leicht veränderten Situationen wieder ratlos vor der Konfiguration des Oszilloskops standen. In [Jas+12] kamen die Autoren zu dem Schluss, dass die textuelle Schulung der Studierenden für diesen Bereich nicht ausreicht (elf Textseiten). Eine E-Learning-Umgebung, in der Aufgaben zum Oszilloskop bearbeitet werden können und nach Niveaustufen gegliedert sind, ist daher empfehlenswert. Dabei ist der Lerninhalt so aufzubereiten, dass Studierende mit unterschiedlichen Geräten umgehen können, sich also nicht nur die Lage und die Bedeutung der Knöpfe und Regler eines spezifischen Oszilloskops schematisch merken.

Funktionsgenerator Die Einstellung des Funktionsgenerators auf einen zu untersuchenden Schaltkreis überforderte viele der Studierenden. Zusammen mit dem Oszilloskop vervielfachten sich die Möglichkeiten von Fehlkonfigurationen und Fehlinterpretationen. Der Funktionsgenerator wurde in den ersten Versuchen genutzt, um eine Puls-Weiten-Modulation (PWM) zu generieren und damit beispielsweise eine LED zu dimmen. Obwohl die Studierenden zum Gerät und dessen Funktionsweise wie auch zum Konzept der PWM Literatur und Übungsaufgaben zur Verfügung gestellt bekamen, konnten nur wenige dieses Wissen im Labor anwenden.

Multimeter und Grundlagen der Elektrotechnik Dieser Kategorie sind alle Fehler zugeordnet, welche grundlegende Verständnisprobleme zur Elektrotechnik oder die Bedienung des Multimeters betreffen. Häufig maßen die Studierenden die Stromstärke parallel und die Spannung in Reihe zum Schaltkreis. Auch die Interpretation der dargestellten Werte in Abhängigkeit zur gewählten Skala stellten Schwierigkeiten dar. Durch diese mangelnden Kompetenzen konnten die Studierenden häufig auch einfache Aufbauten nicht umsetzen, da ihnen die notwendigen fachlichen Grundlagen fehlten. Ähnlich zu den geschilderten Problemen beim Oszilloskop scheint eine ausschließlich textbasierte Vorbereitung entweder den Interessen der Studierenden nicht zu entsprechen oder zu kompliziert zu sein. Auch hierbei scheinen sich die Studierenden nur flüchtiges Wissen anzueignen der zu keinem Kompetenzzuwachs führt.

Weitere Fehler Beim Aufbau von grundlegenden Schaltungen wie bspw. dem Tiefpass hatten die Praktikumsteilnehmer Schwierigkeiten, die Verschaltung des Steckbrettes zu verstehen. Zusätzliche fachliche Verständnisprobleme machten die Hilfe der Betreuer notwendig. Auch die Bedienung des Netzteils war nicht immer fehlerfrei. Die gezeigte Beobachtung bis zu Versuch vier unterstützt deswegen vollständig die von Marwedel aufgestellte These (siehe [Mar11]), dass insbesondere der Zugriff auf Laborinstrumente und andere Hardware ein zentrales Problem bei der fachdidaktischen Unterstützung von Studierenden im Bereich der Entwicklung eingebetteter Systeme darstellt.

Programmierfehler Neben den genannten Verständnisproblemen und den Bedienungsschwierigkeiten der Laborausstattung gab es ab Versuch fünf (dem ersten Versuch mit Mikrocontroller und FPGA) viele Programmierfehler. Um die Studierenden für das Praktikum zu motivieren, wurden bereits im ersten Versuch Aufgaben zur Android-Programmierung mit der Programmiersprache Java gestellt. Zusammen mit C wurden beide Sprachen bereits in je einer Vorlesungreihe vor dem Praktikum vermittelt. Während fast alle Studierenden die Programmierung in Java gut beherrschten, waren einfachste Anwendungen in C kaum von den Praktikumsteilnehmern umsetzbar. Schwierigkeit traten bereits bei einfachen Bedingungen, wie dem Vergleich eines Registerwertes mit einer Bitmaske, auf (siehe Listing 1). Zentral war dabei, dass die Praktikumsteilnehmer sich *nur* über eine Trial-and-Error-Strategie an die korrekte Konfiguration heranarbeiteten, da die Debugging-Möglichkeiten bei dem im Praktikum verwendeten Mikrocontroller nicht gegeben waren. Ansonsten hätten die Studierenden Werte und Positionen im Programmablauf durch die Nutzung von Konsolenausgaben „erraten“ können. Die Betreuer empfahlen den Praktikumsteilnehmern LEDs als Ausgabe boolscher Werte zu nutzen. Bereits für die Ansteuerung einer LED sind jedoch mindestens zwei Register richtig zu konfigurieren, was einigen Gruppen Schwierigkeiten bereitete. Die Beobachter vermuten, dass es bei vielen Praktikumsteilnehmern kein Verständnis über und keine Erfahrung mit dem Konzept der Register gab und dies die Ursache der geschilderten Probleme darstellt. Während in der Softwareentwicklung üblicherweise Funktionen und Methoden Änderungen in Datensätzen und grafischen Oberflächen hervorrufen, sieht es für die Studierenden so aus, als wäre es in der hardwarenahen Programmierung, gerade umgekehrt, das Schreiben von Variablen und Konstanten (im weitesten Sinne). Ein Beispiel aus dem Praktikum illustriert die zuletzt aufgeführte Problemstellung. Wird eine LED mit einer PWM angesprochen, kann der Duty-Cycle so niedrig gewählt werden, dass die LED überhaupt nicht leuchtet, obwohl die Registerwerte für die PWM richtig definiert und auch der Duty-Cycle gesetzt ist. Die Studierenden hatte Schwierigkeiten zu erkennen, dass eine syntaktisch gültige Konfiguration der notwendigen Register nicht mit „der richtigen“ Konfiguration gleichzusetzen ist.

Durch die Zweiteilung der Gruppen ab Versuch fünf konnten auch Studierendengruppen beobachtet werden, die zur Umsetzung der Modellhausansteuerung einen FPGA verwendeten. Natürlicherweise stellte VHDL, eine bei den Praktikumsteilnehmern weitestgehend unbekannte Programmiersprache, die größte Schwierigkeit bei der Lösung der Aufgaben dar. Grundlegende Konzepte von VHDL wurden in einer gesonderten einmaligen Veranstaltung vor Versuch fünf erklärt. Als besonders problematisch empfanden die Studierenden die Umsetzung von sequentiellen und parallelen Funktionsblöcken bzw. Prozessen.

Diese Beobachtungen führten zur Entwicklung zwei lernunterstützender Werkzeuge: dem *Virtual Workspace* als virtueller Arbeitsplatz und Toolchain sowie der explorativen Lern- und Visualisierungsumgebung (*ELVE*) als Lernumgebung für fachliche Problemstellungen.

3.4.2. Virtual Workspace

Die Aufgaben, die die Studierenden im EAP als Vorbereitung lösen sollten, waren oft theoretischer Natur. Einige Bestandteile der Versuche eins bis vier enthielten bereits Programmieraufgaben. Zur Einführung des Praktikums wurden bspw. Android-Aufgaben gewählt, welche die Teilnehmer auf ihrem eigenen Smartphone installieren konnten. Insbesondere ab Versuch fünf ist die Konfiguration des FPGAs und die Programmierung des Mikrocontrollers Hauptbestandteil der Labortätigkeit. Damit einhergehend ändert sich auch der Schwerpunkt der Vorbereitung vom Verstehen theoretischer Konzepte hin zu ihrer Anwendung in einem konkreten Szenario. Wie von Marwedel erläutert, ist die Verfügbarkeit von Hardware ein weiteres Problem in der Unterrichtsplanung und -durchführung [Mar11]. Die Anzahl und Komplexität der notwendigen Werkzeuge für die Entwicklung eingebetteter Systeme erschwert die problemorientierte Vorgehensweise in Laborveranstaltungen. Zusätzlich ist die Verknüpfung verschiedener Werkzeuge (engl. „Toolchain“) meist nicht außerhalb der Universität verfügbar und häufig nur von erfahrenen Benutzern konfigurierbar.

Aus diesen Gründen konzipierten Büchner und Jaschke bereits für die erste Durchführung des EAPs den *Virtual Workspace* (vorgestellt in [BJ13]), bei dem der Autor der hauptverantwortliche Entwickler war. Der Virtual Workspace besteht aus einem USB-Stick (16 GB), auf dem ein Virtual-Machine-File der kostenlosen und frei verfügbaren Virtualisierungsumgebung „VirtualBox“ hinterlegt ist. Die virtuelle Festplatte ist in mehrere logische Dateien separiert, um die Dateigrößenbeschränkung des FAT32-formatierten USB-Sticks nicht zu verletzen. FAT32 wurde gewählt, damit die Studierenden den USB-Stick unabhängig von der Art ihres Betriebssystems (Windows, *nix) nutzen können. Das Betriebssystem der virtuellen Maschine ist ein Debian GNU/Linux, das für eine gesteigerte Leistung auf USB-Sticks angepasst wurde. Hierzu gehören die Anpassung des Schedulers, die Häufigkeit der Schreibzugriffe des Betriebssystems und die Verzögerung von Dateisystemüberprüfungen [BJ13]. Zusätzlich wurde eine minimale Desktopumgebung mit dem Fenstermanager *Openbox* eingerichtet. Dieser gilt als besonders ressourcensparsam und eignet sich deswegen für virtualisierte Computer. Der Erfolg dieser Maßnahmen wurde durch Laufzeittests im Vorfeld der Veranstaltung bewiesen. Alle in Tabelle 3.3 aufgelisteten Werte beziehen sich auf eine virtuelle Maschine auf dem USB-Stick mit einem zugewiesenen Prozessor (3,1 Ghz) und 2 GB Arbeitsspeicher. Die für die Studierenden zur Verfügung gestellten USB-Speichermedien besitzen einen USB-2.0-Anschluss. Eine Leistungssteigerung durch USB 3.0 ist deswegen zu erwarten.

Das bereits erwähnte Debian GNU/Linux wurde in einer 32-Bit-Version installiert, damit auch Studierende mit älteren PCs die Möglichkeit haben, die virtuelle Umgebung zu nutzen. Ein 64-Bit-Gastsystem lässt sich nur auf einem 64-Bit-PC nutzen und hätte einige Teilnehmer möglicherweise ausgeschlossen. Über die Funktion der geteilten Ordner (engl. *„Shared Folder“*) ist es möglich, Dateien zwischen Gast-Betriebssystem und Host-Betriebssystem auszutauschen. Der USB-Stick wurde so konfiguriert, dass 500 MB Speicherplatz neben der virtuellen Maschine zur Verfügung stehen, die für Dokumentationen, Quellcode, Projektdateien oder Binärkompilate genutzt werden können. Dadurch ist die Nutzung externer, nicht auf dem

Aufgabenbeschreibung	Benötigte Zeit (s)
1. Zeit vom Anschalten bis zur Benutzeranmeldung	17
2. Starten der Entwicklungsumgebung	11
3. Kompilieren eines C-Quellcodes (200 Zeilen) für den verwendeten ARM-Mikroprozessor	2
4. Start eines einfachen Java-Spiels mit (und ohne) dem Android-Emulator	55 (6)

Tabelle 3.3.: Leistungsmessungen des *Virtual Workspace* [BJ13]

Virtual Workspace enthaltener Werkzeuge möglich. Der Virtual Workspace stellt fünf Arbeitsumgebungen zur Verfügung, deren Teilbereiche miteinander verknüpft sind (siehe Abbildung 3.5).

Die Arbeitsumgebungen Simulation, Mikrocontroller, Smartphone und Browsersimulation bestehen ausschließlich aus freier, meist sogar OpenSource-Software. Für die FPGA-Entwicklung nutzen die Studierenden im EAP die Xilinx ISE. Diese ist kostenlos erhältlich, erfordert jedoch eine Registrierung auf der Betreiberwebsite. Die Entwicklungsumgebung Eclipse wird für die Mikrocontroller- und die Smartphone-Programmierung genutzt und stellt in diesem Zusammenhang eine Registerübersicht für den verwendeten Mikroprozessor und eine emulierte Smartphoneoberfläche für die Android-Entwicklung zur Verfügung. Bei der Konzeption des Virtual Workspace wurde darauf geachtet, universelle Entwicklungsumgebungen zu nutzen, um den Anpassungsaufwand an neue Werkzeuge für die Studierenden so gering wie möglich zu halten. Die Eclipse IDE ist für einen Großteil der gewählten Teilprojekte sinnvoll einsetzbar, kostenlos und auch im Berufsalltag von Softwareentwicklern oft genutzt.

Jede Gruppe, bestehend aus zwei Studierenden, erhält in der Veranstaltung einen USB-Stick. Der Virtual Workspace kann zu Hause wie auch im Labor genutzt werden und erlaubt die Programmierung des Mikrocontrollers durch die Abstraktionsschichten des Gast- und Hostbetriebssystems. Der erstellte Quellcode kann deswegen direkt aus der virtuellen Maschine auf den Mikrocontroller übertragen werden. Zum Zeitpunkt der Erstellung des USB-Sticks ließ sich die Konfiguration des FPGAs nicht auf diese Weise vornehmen, da der Parallel-Port erst in späteren VirtualBox-Versionen über die gleichen Abstraktionsebenen hinweg nutzbar geworden ist. Eine entsprechende Aktualisierung auf eine VirtualBox-Version nach 4.2 ist aus diesem Grund für weitere Arbeiten empfehlenswert. Zwar ist diese Funktionalität nur für Microsoft-Betriebssysteme offiziell verfügbar, jedoch ist auch die VirtualBox-Managementkonsole für Linux in der Lage, die notwendigen Konfigurationen an der virtuellen Maschine vorzunehmen.

Die Akzeptanz des Virtual Workspace seitens der Studierenden wurde in einer Befragung am Ende des EAPs ermittelt (2012, 2013 und 2014). Während im ersten Jahr nach den Problemen mit dem Werkzeug gefragt wurde, enthielten die weiteren Befragungen eine Teilung in allgemeinen Nutzen und Probleme (siehe Tabelle 3.4).

2012 - 31 Studierenden	
Probleme	**Anteil**
Keine Probleme	48,7%
Geschwindigkeitsprobleme	12,8%
Organisatorische Probleme	12,8%
Anwendungsspezifische Probleme	25,7%
2013 - 48 Studierende	
Probleme	**Anteil**
Probleme beim Austausch zwischen Gast und Host	46,1%
Allgemeine Probleme mit der Virtuellen Maschine	26,9%
Geschwindigkeitsprobleme	26,9%
Nutzen des Virtual Workspace	**Anteil**
Sinnvoll	56,2%
In Ordnung	20,5%
Gering	14,5%
Überflüssig	8,3%
2014 - 52 Studierende	
Probleme	**Anteil**
Keine Probleme	36,5%
Problem beim Betrieb eines 32-Bit-Hosts	7,6%
Defekt des USB-Sticks	46,1%
Geschwindigkeitsprobleme	19,2%
Nutzen des Virtual Workspace	**Anteil**
Sinnvoll	40,2%
In Ordnung	21,2%
Gering	15,4%
Überflüssig	23,1%

Tabelle 3.4.: Umfrage zu Problemen und Nutzen des Virtual Workspace

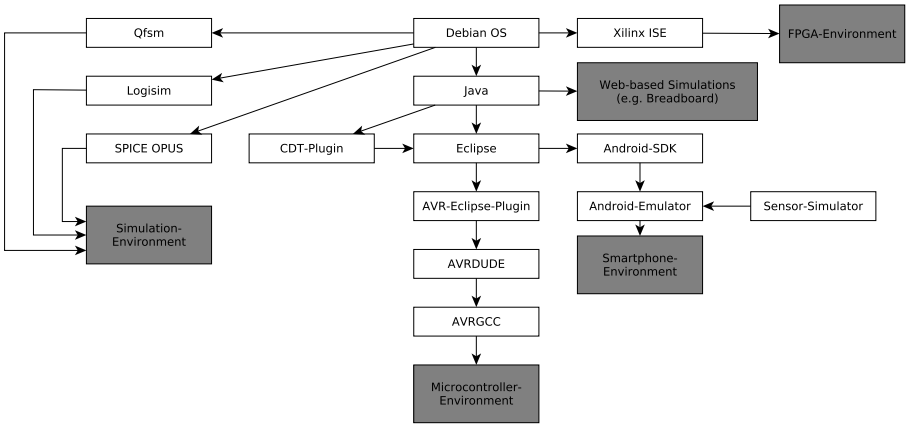

Abbildung 3.5.: Arbeitsumgebungen und Bestandteile des Virtual Workspace [BJ13]

Die Geschwindigkeitsprobleme der virtuellen Maschine kommen durch das Schreibverhalten des Betriebssystems in Kombination mit dem langsamen Speichermedium USB-Stick zustande. Obwohl entsprechende Optimierungen beim Gast-Betriebssystem vorgenommen wurden, hatten einige der USB-Sticks Geschwindigkeitsprobleme. Auch bei einer späteren Generation der Sticks konnte dieses Verhalten nicht verbessert werden.

Die organisatorischen Probleme wurden nur in der ersten Durchführung des EAPs ausgewertet, da sie konzeptioneller Art sind. Ein Kritikpunkt der Teilnehmer war die geringe Verfügbarkeit der USB-Sticks. Für zukünftige Praktika empfahlen die Studierenden die Nutzung von mehreren USB-Sticks pro Gruppe. Die ursprüngliche Überlegung, bei der Verwendung eines einzigen USB-Sticks pro Gruppe zu bleiben, enthielt die Vermutung, dass die Studierenden so auch außerhalb der Universität zu einer echten Gruppenarbeit geleitet werden.

Wie in der zweiten Umfrage ersichtlich, findet mehr als die Hälfte der Teilnehmer des EAP die Nutzung des USB-Sticks sinnvoll. Dies ist auch dann noch der Fall, wenn wie in Tabelle 3.4 aufgezeigt die Kommunikation zwischen Gast- und Host-Betriebssystem nicht funktioniert (*shared folder*), es allgemeine Probleme mit den Anwendungen in der virtuellen Maschine oder Geschwindigkeitsprobleme gibt. Letztere sind tendenziell die Gleichen wie im Vorjahr. Bei den Anwendungen bemängeln einige Studierende das Fehlen eines Texteditors außerhalb der Entwicklungsumgebungen sowie die eingeschränkten Möglichkeiten, PDFs zu betrachten.

Diese und weitere kleinere Änderungen wie die Vorbereitung für Labordrucker wurden für die Lehrveranstaltung 2014 implementiert. Während die vorherigen Probleme weitgehend behoben werden konnten, ist die Unzuverlässigkeit der USB-Sticks dramatisch gestiegen. Ein naheliegender Grund dafür war die Vermischung zweier USB-Stick-Generationen. Die Älteren wurden bereits 2012 im Hardwarepraktikum genutzt. Nahezu die Hälfte aller Teilnehmer musste einen neuen USB-Stick bekommen oder die virtuelle Maschine neu einrichten. Einige Geschwindigkeitsprobleme konnten dadurch erklärt werden, dass der virtuellen Maschine zu wenig RAM zugeordnet wurde. Obwohl der USB-Stick, wie beschrieben, auch für ältere 32-Bit-Betriebssysteme geeignet ist, hatten einige Studierende Schwierigkeiten, diesen bei ihrem privaten PC in Betrieb zu nehmen. Die Studierenden haben die Betreuer allerdings nicht auf diesen Umstand hingewiesen, sodass auch keine Hilfe geleistet werden konnte. Erst in der Evaluation der Lehrveranstaltung wurde die Kritik geäußert.

Zusammenfassend muss der Virtual Workspace von zwei Standpunkten aus betrachtet werden. Der Großteil der Studierenden findet die Hilfestellung sinnvoll. Die Form und die Art der Unterstützung scheint also gerechtfertigt zu sein. Die Mehrzahl der negativen Bewertungen des Konzeptes geht auf die Unzuverlässigkeit bzw. die Performanz des USB-Sticks zurück. Nach Meinung des Autors ist deswegen der Erfolg des didaktischen Konzeptes von der praktischen Umsetzung zu trennen. Auf die weitere Kritik aus den Jahren 2012 und 2013 wurde reagiert. Entsprechend fanden sich keine weiteren Programmwünsche mehr in der Evaluation des Jahrgangs 2014. Parallel zum technischen Fortschritt kann die Beschränkung von FAT 32 wahrscheinlich in den kommenden Jahren aufgehoben werden und für den USB-Stick ein moderneres und performanteres Dateisystem wie VFAT verwendet werden. Die Leistungsengpässe des Virtual Workspace sind verbunden mit der Leistung der Host-Hardware. Voraussichtlich wird sich auch diese in den kommenden Jahren um ein Vielfaches steigern und somit die Performanz des Gesamtsystems positiv beeinflussen.

3.4.3. Lernumgebung ELVE

In der Beobachtungsphase der Veranstaltung wurden Defizite bei den Studierenden bei Grundlagen der hardwarenahen Programmierung aufgedeckt (siehe Abschnitt 3.4.1). Die Lernumgebung ELVE ist ein standortunabhängiges Werkzeug, das im Kontext des EAP die Vorbereitung der Studierenden auf Laborveranstaltungen mit Mikrocontrollern sinnvoll unterstützen soll. Die problemorientierte Lehr-/Lernumgebung ELVE wurde von einer studentischen Projektgruppe umgesetzt und in [JBS13] erstmalig vorgestellt. Zwei Kompetenzen aus dem EKSM sollen durch den Gebrauch von ELVE in der Hochschullehre besonders umfassend vermittelt werden:

- „[Studierende] kennen die besonderen Randbedingungen des Entwurfs eingebetteter Systeme.“ (C2.2)[JBS13, S. 1, Übersetzt],
- „[Studierende] erlernen den Umgang mit der Programmiersprache C und deren Zusammenspiel mit der Hardware.“ (C3.1)[JBS13, S. 1, Übersetzt]

Die Ursache für die Probleme der Studierenden, die Register des Mikrocontrollers richtig zu konfigurieren, sahen die Forscher in der mangelnden Testbarkeit des Quellcodes begründet. Dies hat zwei Ursachen:

Simulation: Für viele Mikrocontroller existieren Simulationswerkzeuge. Naheliegenderweise wäre die Software von Atmel selbst (als Hersteller der Hardware) am ehesten dazu geeignet gewesen, Studierenden die Entwicklung von Programmteilen durch Simulation zu erläutern. Diese Möglichkeit konnte aus Kostengründen nicht umgesetzt werden, da weder auf den im Laborraum vorhandenen PCs noch im Virtual Workspace eine Installation des Windows-Betriebssystems möglich war. Zwar stehen der Universität vergünstigte Tarife zur Lizensierung zur Verfügung, die Konfiguration der Laborrechner wurde jedoch auf Linux ausgelegt und kostenpflichtige Programme in der entsprechenden Version gekauft. Eine Änderung des Betriebssystems hätte damit weitreichende Folgen für die anderen Veranstaltungen im Labor. Die Entwicklungsumgebung *Atmel-Studio* konnte nicht genutzt werden, da der Anbieter keine Linux-Version zur Verfügung stellt.

Eine vielversprechende Alternative bot die, auch unter Linux verfügbare, Anwendung *SimulAVR*. Da ein neuer, leistungsfähigeren Mikrocontroller für die Steuerung der Hausautomation eingesetzt wurde, war es wichtig, dass dieser bereits von der Simulationssoftware unterstützt wurde. Im Fall von SimulAVR war dies nicht gegeben. Die Applikation arbeitete mit den traditionellen ATmega- und nicht mit den neueren ATXmega-Prozessoren (ATXmega A128A1). Für Letztere muss eine andere Programmiersprache verwendet werden. Da es keine vorhandene Lösung gab, musste eine eigene Applikation entworfen werden. Folgende Kriterien waren dabei ausschlaggebend:

1. Didaktisch ausgerichtete Lernumgebung mit niedriger Einstiegshürde (gerade in Hinsicht auf die Diversität der Praktikumsteilnehmer).
2. Verwendung vertrauter Konzepte, ähnlich zu etablierten Entwicklungsumgebungen wie Eclipse.
3. Simulation oder Emulation der tatsächlich im Praktikum verwendeten Hardware.
4. Einfache Bedienung des Konfigurations-Backends für Betreuer.

Debugging Wie bereits in Abschnitt 3.4.1 erwähnt, konnte kein einsteigerfreundliches Debugging durchgeführt werden. Ein häufiges Mittel für eine Rückmeldung über den Applikationsstatus waren an der Hardware angeschlossene LEDs, die je nach Programmverzweigung leuchteten oder nicht, und damit zum Testen genutzt werden konnten. Für weitere Durchführungen wäre eine minimale Programmbibliothek zu empfehlen, durch die Informationen entweder über eine serielle Schnittstelle auf dem PC oder über ein angeschlossenes Display ausgegeben werden können. Dabei muss die Nutzung dieser Funktionalität so niederschwellig angesetzt werden, dass der Programmieraufwand den einer LED-Schaltung nicht übersteigt. Schon bei Letzterem zeigten sich nämlich Verständnisschwierigkeiten bei manchen Studierenden.

Umsetzung von ELVE

Studierende können sich ohne weitere Testmaßnahmen nicht strukturiert an den richtigen Code „herantesten“. ELVE bietet aus diesem Grund eine schrittweise Auswertung von C-Ausdrücken an. Das Prüfen einzelner Zeilen verspricht diesbezüglich ein schnelleres und vor allen Dingen gezielteres Fehler-Feedback als die sonst verfügbaren Compiler, die auf die syntaktische Korrektheit des gesamten Programmcodes angewiesen sind. Die Lernumgebung stellt keine Simulation im klassischen Sinne zur Verfügung, sondern eine im Vorhinein in XML spezifizierte textbasierte Verifikation von Quellcodefragmenten. Der Lehrende legt eine XML-Datei bestehend aus Quellcode und einer Menge an ELVE-spezifischen Anweisungen an. Eine dieser Anweisungen ist bspw., dass an einer festgelegten Stelle im vorgegebenen Quelltext eine Textlücke erscheinen soll. Der Lehrende fügt mit einer weiteren Anweisung die Liste an korrekten Eingaben für diese Textlücke und eventuell auszuführende Aktionen hinzu (siehe Quellcodeauszug).

```
<?xml version="1.0" ?>
<challengeXML xmlns:xsd="..." xsd:schemaLocation="...">
<filedescription>
        <headline>Sample headline</headline>
        <description>Sample description</description>
</filedescription>
<challengecode>
        <codefragment br="false" class="var" id="1">
                <option>ADC_CH_MUXPOS_PIN7_gc</option>
                <error>Sample error message!</error>
        </codefragment>
</challengecode>
<actionlist>
        <controllerelement lednr="4">
                <houseoption>1</houseoption>
        </controllerelement>
</actionlist>
</challengeXML>
```

Listing 2: Beispielhafter Aufbau einer ELVE-Aufgabenbeschreibung [JBS13]

Die Umsetzung von ELVE als Lückentextprogramm hat den Nachteil, dass nur das als richtig vom Programm erkannt wird, was auch in die Optionsliste von den Betreuern eingetragen wurde. Der große Vorteil dieser Methode ist, dass Inhalte geprüft werden können, die nicht nur die Korrektheit des Quellcodes, sondern auch dessen „Qualität“ und das Verständnis der Studierenden berücksichtigen. So kann beispielsweise geprüft werden, ob eine Funktion oder Anweisung gut programmiert ist oder nicht, indem nur solche Lösungsmöglichkeiten als korrekt hinterlegt werden. Für einen Compiler zählt diese Anforderung nicht, solange der Quellcode syntaktisch korrekt ist. Ebenso können in ELVE Lückentexte in Kommentaren eingefügt werden, bei denen Studierende das Verhalten des nachfolgenden Programmteils beschreiben

sollen.

Eingaben sind in ELVE immer textueller Art und geschehen an den vom Lehrenden definierten Stellen. Sollte der Nutzer die falsche Lösung eintragen, erscheint ein Hinweis, der weitere Informationen wie beispielsweise eine Textpassage eines Datenblattes enthält. Auch dieses Verhalten ist vom Lehrenden in der XML-Datei zu spezifizieren. Jede Aktion wird erst ausgeführt, wenn der Nutzer den Quellcode analysieren/übersetzen lässt. Zusätzlich ist vor der Überprüfung die Angabe externer Ereignisse wie eine Benutzerinteraktion oder die Änderung von Umgebungsdaten (bspw. Temperatur oder Helligkeit) möglich. ELVE kann somit die gleiche Sensor-Funktionalität wie das im EAP verwendete Hardware-Modell abdecken.

Die entworfene Lernumgebung kann verschiedene Szenarios zur Verfügung stellen, deren Bearbeitungsreihenfolge dem Nutzer des Systems überlassen wird. Um den Studierenden die Arbeit mit ELVE sinnvoll zu vermitteln, sind Laborinstrumente wie das Oszilloskop, eine interaktive Version des Mikrocontrollers (mit Tasten und LEDs) sowie eine schematische Darstellung des Modellhauses integriert. Aus fachdidaktischer Sicht empfahl sich ebenso die Entwicklung einer Registertabelle, welche die aktuelle Registerbelegung des Mikrocontrollers wiedergibt. Somit können Konfigurationen einfacher mit Datenblättern verglichen und die Wandlung verschiedener Zahlendarstellungen (bspw. aus dem Hexadezimalsystem) in eine binäre Codierung nachverfolgt werden.

Im Praktikum hilft die Lernumgebung, Präsenzphasen vorzubereiten und den dortigen Kompetenzgewinn zu fundieren. ELVE ist kein Simulator und sollte deswegen nicht mit Anwendungen wie *SimulAVR* oder *Atmel Studio* verwechselt werden. Diese sind weder didaktisch aufbereitet, um Anfänger der eingebetteten Systemprogrammierung zu unterstützen, noch speziell auf die im EAP entworfene Aufgabenstellung ausgelegt. Das Programm arbeitet browserbasiert, um Studierenden auch das Experimentieren außerhalb des Labors zu ermöglichen.

Die grafische Oberfläche von ELVE ist in die drei Abschnitte Quellcode, Instrumente und Visualisierung unterteilt (siehe Abbildung 3.6). Die Quellcode-Ansicht unterstützt typische Funktionsmerkmale einer Programmierumgebung wie Zeilennummerierung, interaktive Verknüpfung von Header- und Quellcode-Dateien sowie die farbliche Hervorhebung der Syntax. Durch den bekannten Aufbau der Oberfläche finden sich die Studierenden schnell zurecht und benötigen wenig zusätzliche Einarbeitungszeit für die Interaktion mit der Lernumgebung. Alle Eingaben werden von den Nutzern in die vom Lehrenden zu bestimmenden Lücken eingetragen. Nach der zeilenweisen Überprüfung des Quellcodes aktualisiert sich die Instrumenten- und die Visualisierungs-Sektion von ELVE. Aus fachdidaktischer Sicht kann ELVE als Lückentextprogramm betrachtet werden. Der große Vorteil gegenüber compilerbasierten Programmierumgebungen ist die zeilenweise Auswertung von Programmstatements, also ein interpreterähnliches Verhalten. Studierende sehen so nicht nur, welcher Teil ihrer Programmabschnitte, bspw. die in der Beobachtung angesprochenen Registerzuweisungen, richtig sind, sondern auch, welche Auswirkungen diese auf das System haben. Diese These wird durch fachdidaktische Arbeiten zum Lernen von Programmiersprachen unterstützt [Cha+00], [Gar05].

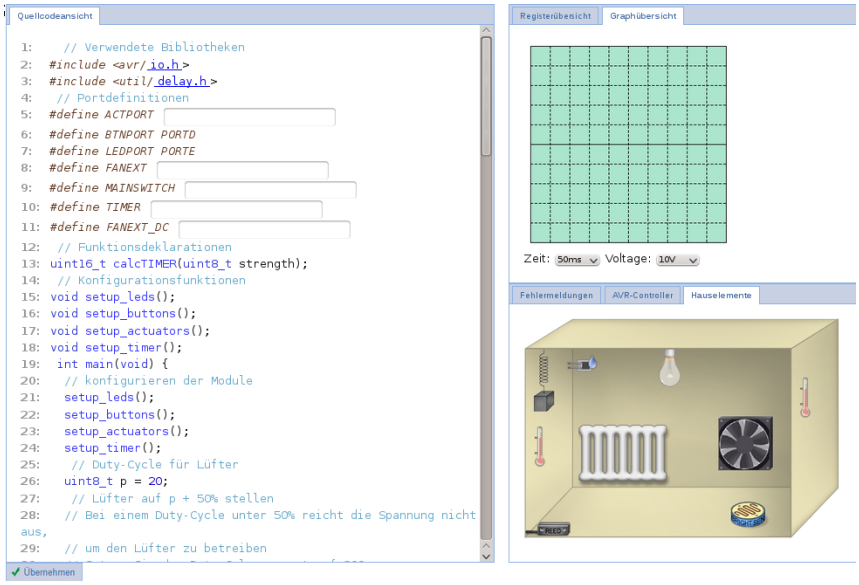

Abbildung 3.6.: Übersicht der ELVE-Nutzungsschnittstelle

Neben der Darstellung eines Oszilloskops für die Analyse eines PWM-Signals stellt dieser Abschnitt auch eine Registertabelle zur Verfügung (in der Grafik nicht zu sehen). Die Oszilloskop-Ansicht soll den Studierenden helfen, die programmierte Puls-Weiten-Modulation mit den gesuchten abzugleichen. Die Registertabelle wurde entwickelt, um den Praktikumsteilnehmern die Auswirkungen von Bitoperationen (*Shiften*, *AND* oder *XOR*) auf Registern zu visualisieren. Damit wird ein Stück der fehlenden Debuggingfunktionalität in einem didaktischen Kontext zur Verfügung gestellt.

Auch die Modellhausansicht kann gegen eine interaktive Version des Mikrocontrollers oder die Fehlerausgabe ausgetauscht werden. Alle Medienobjekte und Interaktionsmöglichkeiten in ELVE sind auf die konkrete Aufgabe des Entwurfs- und Anwendungspraktikums ausgelegt, um Studierenden den nicht notwendigen Kontextwechsel zwischen zwei unterschiedlichen Aufgaben-Settings zu ersparen. Die Fehlerausgabe stellt dem Anwender pro Lücke einen Hinweis auf die Art des Fehlers und eine Literaturangabe für weitere Informationen zur Verfügung. Beide Informationen können frei vom Betreuer eingestellt werden. Damit soll ELVE die in der Beobachtung als mangelhaft eingeschätzte Kompetenz zum Informationsgewinn aus Datenblättern fördern. Sind alle Angaben korrekt, werden die animierten Sensor- und Aktuatorenelemente in der Modellansicht aktiviert und mit zusätzlichen Parametern versehen. Damit geht ELVE über die Möglichkeiten des eigentlichen Praktikums hinaus. Für die Umsetzung einer Puls-Weiten-Modulation zur Lüftersteuerung müssen die Studierenden im Praktikum ein Oszilloskop verwenden, damit sie die tatsächliche prozentuale mit der intendierten Drehzahl vergleichen können. In ELVE kann zu einer korrekten PWM-Ansteuerung der Duty-Cylce angegeben werden.

Evaluation

In der Durchführung der Laborveranstaltung 2013 und 2014 fand keine direkte Evaluation von ELVE statt. Für 2013 stehen indirekte Daten der Betreuer zur Verfügung. Sie merkten an, dass die Nutzung von ELVE in der Vorbereitung geholfen hat, die mangelhafte Kompetenz der Literatur- und Datenblattrecherche zu verbessern. Allerdings konnte nur ein kurzfristiger Wissenserwerb, aber keine Kompetenzaneignung in den Durchführungen der Laborveranstaltungen beobachtet werden. Hieraus lässt sich schließen, dass ELVE als fachdidaktisches Werkzeug sinnvoll einsetzbar ist, eine ähnliche didaktische Begleitung aber auch in einer vollumfänglichen Entwicklungsumgebung wie Eclipse notwendig ist, um Kompetenzen zu festigen. In der Praktikumskonzeption für das Jahr 2014 wurde die duale Entwicklung mit Mikrocontroller und FPGA verworfen. Es wird nun nur noch der FPGA zur Steuerung der Hausautomation genutzt, weswegen der Bedarf von ELVE als Lernumgebung für Mikrocontrolleraufgaben nicht mehr vorhanden ist. Durch den modularen Aufbau und die fachgebietsunabhängige Konfiguration der Lernumgebung ist eine Verwendung in anderen Kontexten, wie bspw. für FPGAs, nach wie vor gewährleistet.

4. Kriterien für fundamentale Lehr-/Lerninhalte eingebetteter Systeme

Im vorherigen Kapitel wurde erläutert, welche praktischen und theoretischen Unterstützungen auf der Basis des KOMINA-Projekts entwickelt wurden, um die Strukturierung der Fachdidaktik im Bereich der Technischen Informatik zu fördern (EKSM und Lehr-/Lernunterstützungen). Dieses Kapitel widmet sich der theoriegestützten Weiterentwicklung der Kompetenzstrukturierung mit dem Ziel, darauf aufbauende Lehr-/Lernkonzepte konkreter für die Umsetzung in Lehrveranstaltungen zu beschreiben. Die heterogenen und teilweise vagen Kompetenzbeschreibungen des EKSMs werden durch einen normativen Ansatz inhaltlich erweitert bzw. konkretisiert, um diesen Prozess zu unterstützen. Der normative Ansatz stellt Themen und Inhalte mit besonderer Bedeutung für das Anwendungsgebiet der Entwicklung eingebetteter Systeme heraus, die als Verfeinerung des EKSM zu sehen sind und der Umsetzung von Kompetenzbeschreibungen in Lehrveranstaltungen dienen. Bei breit aufgestellten Disziplinen ist die Sicht auf diese wichtigen Lerninhalte stark von den subjektiven Standpunkten der Lehrenden abhängig. Um für die Lernenden eine möglichst objektive Auswahl relevanter Themengebiete zu erstellen, müssen Kriterien bestimmt und offengelegt werden, nach denen die Auswahl der Inhalte getroffen wird. In dieser Arbeit werden die *fundamentalen Ideen* bzw. der damit verbundene kriterienorientierte Ansatz verwendet. Bruner hat die Struktur des Ansatzes in seinem Buch zu „The Process of Education“ beschrieben, ohne eine Vorgehensbeschreibung zu geben, mit dem der Ansatz umgesetzt wird [Bru60]. Schwill hat die Arbeit aufgegriffen und eine Ablaufbeschreibung, wie auch eine Sammlung aus vier (später fünf) Kriterien hinzugefügt, die bei der Prüfung der Fundamentalität einer Idee genutzt werden können. Ebenso nutzte er Bruners allgemeindidaktische Erläuterungen für die Adaption auf den konkreten Anwendungsbereich der Schulinformatik. Auf der Vorarbeit dieser beiden Autoren aufbauend, werden die Kriterien für die Ermittlung fundamentaler Ideen im Bereich der Entwicklung eingebetteter Systeme angepasst. Die Berücksichtigung von hochschuldidaktischer Rahmenbedingungen hat darauf großen Einfluss (siehe Kapitel 5).

Bruner und Schwill heben hervor, dass der Ansatz besonders gut in einem Spiralcurriculum umgesetzt werden kann. Ein Spiralcurriculum beschreibt eine Lehrorganisation, in der jedem Lernenden in jedem Stadium seiner Entwicklung jeder Lehrgegenstand in einer „*intellektuell ehrlichen Form*“ beigebracht werden kann [Bru80, S. 61]. Die fundamentalen Ideen unterscheiden sich zu traditionellen Curricula in den folgenden drei Aspekten:

1. Ein Curriculum gibt eine einmalige Technik- und Methoden-Empfehlung auf Basis aktueller fachlicher und gesellschaftlicher Diskurse. Die fundamentalen Ideen als Konzept ermitteln eine Ideen-Sammlung, die technikunabhängig, aber dennoch fachrelevant ist. Deswegen werden Begriffe wie Methoden, Konzepte und Techniken auf die zugrundeliegende Idee abstrahiert.

2. Der Prozess der Analyse fundamentaler Ideen stellt, angepasst für die jeweilige Disziplin, nicht nur eine Sammlung von Lerninhalten, sondern eine eigene Kompetenzfacette dar. Studierende, die diese Kompetenz besitzen, können neue Techniken und Themengebiete mit dem vorgestellten Kriterienkatalog auf ihre Fundamentalität hin untersuchen. Darauf aufbauend können sie dann die Entscheidung treffen, ob das Lernen der Idee für sie relevant ist oder nicht.

3. Auf der Ebene von Lehrveranstaltungen kann das Konzept der fundamentalen Ideen mehrschichtige Bezüge zwischen verschiedenen Lerninhalten offenlegen. Die Dimensionen, auf denen diese Bezüge auftreten können, sind durch die Kriterien repräsentiert, die jeweils einen anderen Aspekt einer Idee fokussieren. Das Verständnis über die herausgearbeiteten Zusammenhänge hilft Lehrpersonen bei der Konzeption von Lehrveranstaltungen, indem geeignete Übergangspunkte zwischen Ideen erforscht werden. In Curriculaempfehlungen sind die Bezüge der Lerninhalte zwar enthalten, aber in der Regel nicht offengelegt. Die Nachvollziehbarkeit der Themenbezüge kann in Curriculaempfehlungen deswegen schwierig sein.

4.1. Fachdidaktische Herleitung fundamentaler Ideen

Die Diskussion um beständige, die Disziplin strukturierende Lerninhalte begann spätestens mit Bruners Buch „The Process of Education“ [Bru60]. Er greift den Gedanken fundamentaler Lehr-/Lerninhalte auf und verweist darauf, dass die Lehre von Fachstrukturen und nicht die Aneignung von Fakten und Techniken im Mittelpunkt aller Lehrbemühungen stehen sollten [Bru60]. Bruners Erörterungen zum Themengebiet der fundamentalen Ideen beinhalten nur implizit Kriterien, ohne diese zu definieren. Er sieht folgende vier Thesen als Vorteil der von ihm dargestellten Vorgehensweise an:

1. „Ein Lehrgegenstand wird faßlicher, wenn man seine Grundlagen versteht.“ [Bru80, S. 35]

2. „Eine gute Theorie ist nicht nur das Vehikel um ein Phänomen jetzt zu verstehen, sondern auch um es morgen in die Erinnerung zurückzurufen.“ [Bru80, S. 37]

3. „Das Verstehen grundlegender Prinzipien und Begriffe scheint, [...], der Hauptweg zu einem adäquaten ‘Übungstransfer’ zu sein.“ [Bru80, S. 37]

4. „Die vierte Behauptung zugunsten einer Betonung von Struktur und Prinzipien im Unterricht besagt, dass man dadurch, dass man den Unterrichtsstoff der Primar- und Sekundarschulen ständig auf seinen fundamentalen Charakter hin überprüft, den Abgrund zwischen 'fortgeschrittenem' und 'elementarem' Wissen verringern kann." [Bru80, S. 37]

Wie aus den zitierten Textstellen ersichtlich, ist Bruners Konzept allgemeiner Natur, ohne Einschlägigkeit hinsichtlich Zielgruppe und Anwendungsgebiet zu besitzen. Da Bruners Erläuterungen konzeptioneller Art sind, beinhalten sie ebenso keine Formalisierungen erforderlicher Kriterien noch eine Definition wichtiger Begriffe.

Schwill hat den Ansatz nachfolgend nicht nur übernommen, sondern um einen Kriterienkatalog ergänzt (siehe Abschnitt 4.1.2). Gleichzeitig nimmt er eine Fokussierung des Konzeptes auf die Schulinformatik vor, passt es also hinsichtlich einer Disziplin und einer Zielgruppe an. Schwills Kriterienkatalog bildet die Ausgangslage der Forschung zur Förderung von Kompetenzen im Bereich der Entwicklung eingebetteter Systeme. Das von ihm vorgestellte Konzept beeinflusste auch die Diskussion grundlegender Ideen in anderen naturwissenschaftlichen Disziplinen, wobei diese meist nicht durch Kriterien überprüft worden sind. Beispiele sind die Biologie [Nur03], die Mathematik [Deu12] und die Chemie [Gil97]. In der wissenschaftlichen Diskussion haben die Begriffe fundamentale Konzepte, Grundkonzepte, Leitideen, universale Ideen und zentrale Ideen häufig einen Bezug zum Konzept der fundamentalen Ideen [ZS06].

Die Diskussion um technikunabhängige, aber dennoch fundamentale Lerninhalte einer Disziplin ist stark an das Verständnis der Begriffe „Idee" und „Fundamental" gekoppelt. Nachfolgend werden die Auffassungen verschiedener Forscher beschrieben, um davon ausgehend die Kriterien für eingebettete Systementwickler abzuleiten.

4.1.1. Der Begriff der Idee

Die verschiedenen Positionen zum fachdidaktischen Verständnis des Ideenbegriffs sind stellvertretend durch Schwill, Vollrath und Schweiger illustriert. Vollrath beschreibt eine Idee als „[...] den entscheidenden Gedanken eines Themas, den wesentlichen Kern einer Überlegung, den fruchtbaren Einfall bei der Lösung eines Problems, die leitenden Fragestellungen einer Theorie, die zentrale Aussage eines Satzes, die einem Algorithmus zugrunde liegenden Zusammenhänge und die mit Begriffsbildungen verbundenen Vorstellungen." [Vol78, S. 29]. Schwill hingegen orientiert sich an der Informatik und fasst eine Idee als „Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema" auf, das ausgewählten Kriterien genügt [Sch93, p. 8]. In seiner mathematikdidaktischen Arbeit interpretiert Schweiger Ideen als „Ein Bündel von Handlungen, Strategien oder Techniken, sei es durch lose Analogie oder Transfer verbunden", die ebenfalls festgelegten Kriterien genügen müssen. Die zwei Definitionen von Schwill und Vollrath sind besonders interessant, da verschiedene Facetten des Ideenbegriffs ausformuliert werden und somit eine Richtlinie für Ideensammlungen vorgeben. Es können somit zwei begriffliche Ebenen unterschieden werden:

Methoden, Strategien, Schemata und Vorgehensweisen sind als Handlungsvorschriften, Anleitungen oder Heuristiken zur Problembewältigung zu sehen. Ein Beispiel ist das Component-based Design als Methode zur einfachen Wiederverwendung von Komponenten oder Extreme-Programming als Oberbegriff von Vorgehensweisen, die zuerst die Tests und darauf folgend die Implementierung entwickeln.

Sichtweisen, Auffassungen und Prinzipien sind oft modellhafte Repräsentationen der gedachten Wirklichkeit, die konkrete Auswirkungen auf Design und Implementierung besitzen (beispielsweise die Unterscheidung zwischen kontinuierlicher und diskreter Dynamik oder die Anerkennung des Umstandes, dass keine Programmiersprache existieren kann, die jede denkbare Funktion umsetzen kann (bspw. $f : \mathbb{N} \to \mathbb{B}$)). Ein Verständnis dieser Methoden, Strategien, Schemata und Vorgehensweisen hat Auswirkungen auf den Entwicklungsprozess, da sie oft entscheidende Aussagen über die praktische Umsetzbarkeit eines Lösungskonzeptes geben.

Die Begriffe einer Ebene sind nicht synonym, aber hinsichtlich ihres Anwendungsfeldes innerhalb der Disziplin artverwandt. Der Begriff der „Idee“ wird in der restlichen Arbeit deswegen stellvertretend für die zwei dargestellten begrifflichen Ebenen verwendet.

4.1.2. Der Begriff des Fundamentalen

Während der Ideenbegriff festlegt, *welche* Inhalte betrachtet werden, gibt die Definition des Begriffes „Fundamental“ die Art und Weise, also das *Wie*, vor. Bruners Forschungsarbeiten enthalten keine Definitionen zu erfüllender Kriterien, auch wenn er diese umschreibt [Bru60]. Schwill hat weitere Aspekte herausgearbeitet und in einem Katalog bestehend aus fünf Kriterien zusammengeführt, die als Basis der vom Autor erarbeiteten Kriteriendefinition genutzt werden. Ergänzt wird die Aufstellung durch das von Hartmann, Näf und Reichert vorgeschlagene Repräsentationskriterium.

> „Eine **fundamentale Idee** (bzgl. einer Wissenschaft) ist ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das
> (1) in verschiedenen Bereichen (der Wissenschaft) vielfältig anwendbar oder erkennbar ist (Horizontalkriterium),
> (2) auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden kann (Vertikalkriterium),
> (3) in der historischen Entwicklung (der Wissenschaft) deutlich wahrnehmbar ist und längerfristig relevant bleibt (Zeitkriterium),
> (4) einen Bezug zu Sprache und Denken des Alltags und der Lebenswelt besitzt (Sinnkriterium).“ [Sch93, S. 8](Hervorhebung im Original),
> „(5) zur Annäherung an eine gewisse idealisierte Zielvorstellung dient, die jedoch faktisch möglicherweise unerreichbar ist (Zielkriterium)“ [SS11, S. 65],
> „(6) sich auf verschiedenen kognitiven Repräsentationsstufen (enaktiv, ikonisch, symbolisch) darstellen lässt“ [HNR06, S. 32]

Der Ursprung und die meisten Weiterentwicklungen der fundamentalen Ideen stammen aus dem Bereich der Schulbildung. Da sich diese Arbeit jedoch auf die Expertenbildung an Fachhochschulen und Universitäten richtet und zudem einen anderen thematischen Schwerpunkt als vorherige Forschungsarbeiten besitzt, ist eine Anpassung der Kriterien erforderlich. Die konzeptuellen Unterschiede lassen sich wie folgt zusammenfassen:

Anwendungsgebiet Schwills Ziel ist, fundamentale Ideen für die komplette Informatik zu sammeln, ohne dabei den Anspruch zu haben, einen vollständigen Katalog zu erhalten. Ausgehend von einem typischen Softwareentwicklungsprozess (engl. *Software-Lifecycle*) zieht er Rückschlüsse auf die ganze Disziplin.

Im Gegensatz dazu ist das Anwendungsfeld der Entwicklung eingebetteter Systeme sehr viel spezifischer und durch starke Interdisziplinarität gekennzeichnet. Wann immer möglich, sollten in der Analyse auch Ideen der Informatik nahestehenden Fachdisziplinen berücksichtigt werden. Diese sind namentlich Mathematik, Physik, Elektrotechnik, Regelungs- und Steuerungstechnik sowie der Maschinenbau.

Zielgruppe Schwills Zielgruppe sind Schülerinnen und Schüler in Primar- und Sekundarstufe. Wie auch Bruner spricht er sich für ein Spiralcurriculum aus, in dem die gleiche Idee über verschiedene Schulstufen mit wachsendem Lernniveau und Detailgrad vermittelt werden soll. Das Spiralcurriculum hat viele Vorteile, die prinzipiell auch für die universitäre Lehre genutzt werden können (vgl. [HS99]). Hinderlich sind hierbei jedoch die möglichen Unterschiede im Kompetenzniveau der Studierenden. Die gleiche Situation besteht bei einer Gruppe von Studierenden mit unterschiedlichen Nebenfächern, Studienverlaufsplänen oder Studiumsschwerpunkten, die jedoch die gleichen Vorlesungen besuchen sollen. Die wissenschaftliche Auseinandersetzung mit dieser Problematik auf curricularer Ebene ist Gegenstand zukünftiger Forschung und kann aus zwei Gründen an dieser Stelle nicht gelöst werden. Zum einen zeigt die vorliegende Arbeit die Adaption der fundamentalen Ideen auf Ebene des Lehrkonzeptes und kann deswegen nicht beliebig auf die Niveaustufung über ein ganzes Curriculum Bezug nehmen. Zum anderen ist die Zusammenlegung artverwandter Studiengänge im Ermessen der jeweiligen Universität und in dieser Hinsicht auch von den dort angebotenen Vorlesungen und deren inhaltlicher Auslegung abhängig. Eine möglichst vielschichtige Analyse von Lerninhalten, wie durch die Kriterien der fundamentalen Ideen möglich, kann dem Ziel nur zuträglich sein.

Neben diesen Einschränkungen differiert auch das Bildungsziel der Schwillschen und der in dieser Arbeit intendierten Zielgruppe. Durch seinen Fokus auf die Schulbildung haben Schwills Kriterien oft einen Alltags- und keinen Arbeitsweltbezug. Letzterer ist für Bachelorstudierende wichtig, die kein Masterstudium beginnen wollen. Für zukünftige Masterstudierende ist hingegen der Bezug zu fachwissenschaftlichen Fragestellungen und Themen besonders wichtig. Diesen Einschränkungen kann durch die in Abschnitt 4.2 geschilderte Kriterienanpassung begegnet werden.

4.1.3. Abgrenzung verwandter didaktischer Diskussionen

Im folgenden Abschnitt sind alle fachlichen Diskurse aufgeführt, die einen Bezug zu fundamentalen Ideen haben, bzw. ein ähnliches Ziel verfolgen. Die vorgestellten Konzepte werden gegen die fundamentalen Ideen abgegrenzt.

Zendler, Spannagel und Klaudt versuchen in einem Konzept, ähnlich zu dem der fundamentalen Ideen, eine Prüfung von verschiedenen Prozessbereichen für die Eignung in der informatischen Bildung zu erlangen [ZSK07]. Hierbei führen sie eine empirische Analyse durch, in der ausgewählte Experten eine Sammlung von Begriffen (Ursprung der Begriffe aus einer Arbeit von Costa und Lieberman) hinsichtlich der bereits durch Schwill bekannten fünf Kriterien bewerten sollen. Allerdings hat die Arbeit von Zendler und Spannagel das Manko, dass der Kontext der Begriffe verloren gegangen ist. In ihrer Sammlung an Begriffen ist keine weitere Erklärung zu diesen zu finden. Somit ist der Interpretationsspielraum von Termini wie „Information“ oder „System“ seitens der Experten sehr groß und damit anzunehmen, dass mehrere Experten nicht das gleiche Verständnis der Begriffe vorausgesetzt haben. Der Sachverhalt einer „Information“ hat bspw. in der Anwendung kryptographischer Verfahren eine ganz andere Bedeutung als bei der Planung des Entwicklungsprozesses für ein Softwareprojekt. Ohne darauf hinzuweisen, könnten die Experten also ganz unterschiedliche Interpretationen des Begriffes haben, und diesen entsprechend unterschiedlich bewerten. In darauf aufbauenden Arbeiten, wie [ZSK11], setzt sich dieser Einfluss fort. Eine empirisch gestützte Einschätzung von fundamentalen Ideen ist nach Meinung des Autors erst dann sinnvoll, wenn der Kontext der Begriffe in einem normativen Verfahren erschlossen wurde. Dies sofort in einem empirischen Verfahren zu versuchen, scheint aufgrund der Menge an zu berücksichtigenden Quellen unpraktikabel und, wie geschildert, wenig aussagekräftig zu sein.

Große Ähnlichkeit haben fundamentale Ideen mit dem Ansatz der „Threshold-Concepts“ (auch „Threshold-Knowledge“) [ML03]. Dieser wird seit einiger Zeit auch für die Informatik diskutiert (bspw. [Eck+06] und [RR09]). Meyer und Land entwickelten das Konzept nach folgender Beobachtung: „[...] in certain disciplines there are 'conceptual gateways' or 'portals' that lead to a previously inaccessible, and initially perhaps 'troublesome', way of thinking about something.“ [ML05, S. 1]. Ähnlich zu den fundamentalen Ideen gibt es einen Kriterienkatalog, der erfordert, dass jedes untersuchte Konzept fünf ausgewählten Eigenschaften gerecht wird [ML03]. Die dargestellten Kriterien sind eher empirisch-pädagogischer als technischer Natur. Abgrenzungen zwischen den beiden Ansätzen sind bereits unter fachdidaktischen Aspekten diskutiert worden (siehe [Eck+06]). Ein Beispiel zur Abgrenzung fundamentaler Ideen gegen „Threshold-Concepts“ ist das Konzept der *Pointer* in C++. Das dazugehörige „Threshold-Concept“ ist die Anwendung und das Verständnis von Zeigern und deren Syntax. Das Konzept der fundamentalen Ideen bestimmt im Gegensatz dazu eher die Themen *Call by Value* und *Call by Reference* als wichtige Lerninhalte. In Rückblick auf die Zielstellung des Forschungsprozesses ist es die Unabhängigkeit von aktuellen technischen Trends, die für die fundamentalen Ideen sprechen. In einem späteren Forschungsschritt kann

es hingegen sinnvoll sein, die fundamentalen Ideen mittels „Threshold-Concepts“ zu verfeinern.

Pasternak und Vahrenhold schlagen das Konzept der „Roten Fäden“ vor [PV09]. Dieses soll helfen, die Unterrichtsinhalte nicht nur aus Sicht der Lehrenden zu strukturieren, sondern die Sichtweise der Adressaten einzubeziehen. Der Hintergrund des vorgestellten Ansatzes ist die Verknüpfung und Kontextualisierung ausgewählter Lehr-/Lerninhalte, um deren Wirklichkeitsrelevanz hervorzuheben (auch genannt: „Informatik im Kontext“). Pasternak und Vahrenhold definieren ebenso einen Anforderungskatalog, dem vier Kriterien zugeordnet sind. Dabei ist zu erkennen, dass die Autoren mit diesem Katalog den Fokus stark auf schulpraktische Problemstellungen richten und weniger auf die Fundamentalität von Gegenstandsbereichen. Das Konzept steht also den fundamentalen Ideen nicht konträr gegenüber, sondern stellt eine Art Verfeinerung und Erweiterung zu einmal festgestellten Ideen dar. Pasternak und Vahrenhold selbst beschreiben, dass ein „Roter Faden“ die Umsetzung einer fundamentalen Idee sein kann, aber nicht muss [PV09]. Ein weiterer Vorteil der fundamentalen Ideen ist, dass der Prozess der Analyse hinsichtlich der fünf Kriterien auch als eigenes Prozesswissen gesehen werden kann, das Studierenden dabei hilft, zukünftige Entscheidungen für oder gegen das Studium einer neuen Technik zu begründen. Beispielsweise kann die Aneignung einer neuen Programmiersprache mit Hilfe des Kriterienkatalogs analysiert werden, um herauszufinden, welche der in Frage kommenden Programmiersprachen einen fundamentalen Aspekt gegenüber den anderen enthält.

Denning erörtert die Grundlagen der Informatik anhand ihrer Prinzipien [Den03]. Er sucht nach Konzepten ähnlich zu solchen wie Photonen, Elektronen, Quarks und Wellenfunktionen aus der Physik oder wie Planeten, Sterne und Galaxien aus der Astronomie. Ein erster Schritt ist für ihn die Definition einer „Mechanik“, die in frühen Publikationen aus vier, später aus sieben Teilen besteht [Den07]. Im Gegensatz zu den fundamentalen Ideen stellt Dennings Arbeit ein theoretisches Rahmenwerk dar, das der Nachvollziehbarkeit von Prinzipien, aber nicht deren Ermittlung dient. Es gibt keinen Kriterienkatalog, mit dem Technologien oder Ansätze geprüft werden können. Dennings Ansatz verfolgt kein didaktisches Ziel, sondern versucht eine philosophisch gestützte Erklärung der Grundelemente der Informatik zu geben.

Alle vorgestellten Forschungsergebnisse beziehen sich nicht auf den Themenbereich der Entwicklung eingebetteter Systeme und sind zudem nicht auf Studierende, sondern Schülerinnen und Schüler ausgerichtet. Eine Adaption bisheriger Forschungsergebnisse ist deswegen, wie bei den fundamentalen Ideen auch, notwendig.

4.2. Festlegung eines Kriterienkataloges

Die Definition des Kriterienkataloges ist nicht nur für das Verständnis dieser Arbeit notwendig, sondern gleichermaßen ein möglicher Grundstein zukünftiger Beiträge

der wissenschaftlichen Community zum Themenkomplex. Die folgende Darstellung der Kriterien orientiert sich am Aufbau von Schwills Argumentationskette, beginnend mit dem Horizontalkriterium.

4.2.1. Analyse des Horizontalkriteriums

Die Bewertung des Horizontalkriteriums ist stark von der Sichtweise auf das Themengebiet der eingebetteten Systeme geprägt. Wäre lediglich die Betrachtung der Softwareseite eingebetteter Systeme Gegenstand der Arbeit, wäre die Zuordnung zur Informatik naheliegend. Da Vorgehensweisen und Designentscheidungen aber in der Regel projektumfassend sind, muss der interdisziplinäre Charakter eingebetteter Systeme berücksichtigt werden [Cas+05]. Dadurch entsteht ein Betrachtungsfeld, welches Aspekte der Elektrotechnik, der Mathematik und Physik sowie der Steuerungs- und Regelungstechnik mit einbindet. Das Horizontalkriterium muss diesem Umstand Rechnung tragen und die Verwendung potentiell fundamentaler Ideen angrenzender Disziplinen ermöglichen, ohne dabei den Fokus auf das Anwendungsfeld zu verlieren. Eine generelle Betrachtung der aufgezählten Disziplinen ist deswegen nicht sinnvoll. Die folgende Definition des Horizontalkriteriums bezieht aus diesem Grund die Verknüpfung anderer Disziplinen nur mit ein, wenn das Verständnis entsprechender Methoden und Sichtweisen für die Entwicklung eingebetteter Systeme essentiell ist.

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> „die im Bereich der Entwicklung eingebetteter Systeme breit anwendbar ist oder sich in mehreren Phasen des Entwicklungsvorgehens wiederfindet. Ideen angrenzender Fachdisziplinen erfüllen das Horizontalkriterium, falls sie notwendiger Betrachtungsgegenstand eingebetteter Systeme sind."

4.2.2. Analyse des Vertikalkriteriums

Das Vertikalkriterium zielt auf die Vermittelbarkeit der Idee auf verschiedenen Niveaustufen im Bildungsprozess ab. Der Hintergedanke bei diesem Kriterium ist die Auslegung der Ausbildung als Spiralcurriculum, der Form, die Bruner als geeignet für die Vermittlung von grundlegenden Ideen sieht. Die zu lernende Idee wird wiederkehrend, aber unterschiedlich anspruchsvoll behandelt. Im Idealfall lassen sich so recht früh Anknüpfpunkte für Ideen aufbauen, die mit Fortschreiten der Ausbildung sukzessive vertieft werden. In der Literatur wird das Vertikalkriterium neben dem Horizontalkriterium als maßgeblich für „Fundamentalität" dargestellt [SS11]. Die Fokussierung auf den universitären Bildungsbereich schmälert diesen Umstand nicht. Bei genauerer Betrachtung fällt auf, dass Schwill das Vertikalkriterium auch als vertikale Achse mit verschiedenen Niveaustufen sieht. Somit ist die Anpassung dieses Kriteriums auf eine spezifischeres Anwendungsgebiet nicht der Verzicht auf Fundamentalität, sondern die Betrachtung einer fundamentalen Idee auf einer

höheren Ebene des Vertikalkriteriums. Durch die Anforderungen an die kognitive Leistungsfähigkeit der Anwender variiert die Breite des Anwendungsbereiches und gleichzeitig der Anwendungsbezug - die Idee wird konkreter hinsichtlich des Sachgebietes. Die Grundzüge einer Idee können unter Umständen immer noch auf niedrigeren Niveaustufen vermittelt werden. Für den hier avisierten Anwendungs- und Zielgruppenbereich ist lediglich der *Einstieg* höher.

Die Stufung des Bildungsniveaus, wie sie das Vertikalkriterium beschreibt, findet sich nicht nur in der Grobstruktur des schulischen Bildungsweges, sondern auch in dessen Unterebenen. Die Grafik 4.1 stellt die Sachlage anschaulich dar.

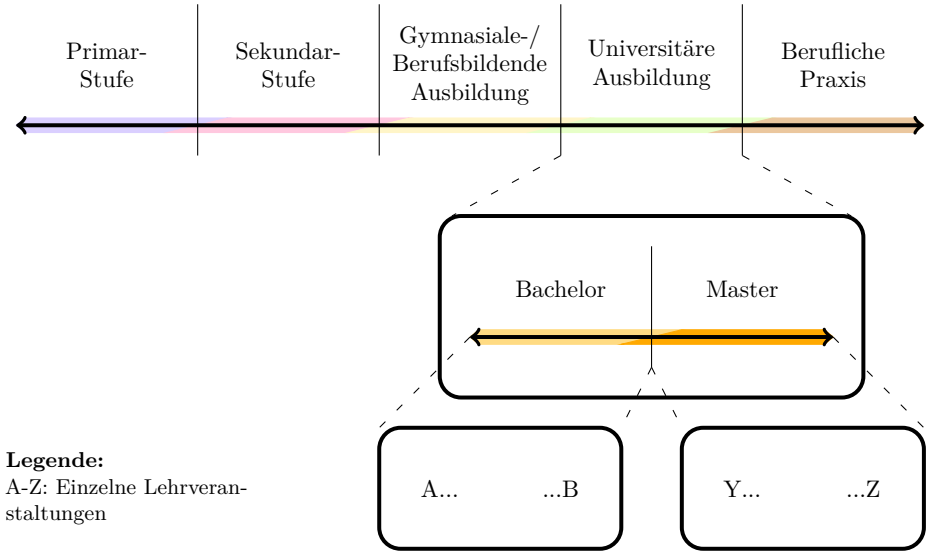

Abbildung 4.1.: Niveaustufenveranschaulichung des Vertikalkriteriums

Die von Schwill zur Darstellung des Kriteriums verwendete Achse lässt sich bis hin zu einzelnen Lehreinheiten unterteilen. Aus praktischen Gründen stellen die Lehreinheiten das kleinste Glied der Kette dar, da ein Leistungsfortschritt nur über einen gewissen Zeitraum erfolgen kann, der von Stufe zu Stufe kürzer wird. Für die Zielgruppe der Entwickler eingebetteter Systeme lässt sich dies konkretisieren. Demnach sind auf universitärer Seite zumindest die Ebenen Bachelor und Master relevant. Durch diese Anschauung kann die Formulierung des Kriteriums zielgruppen- und anwendungsgebietspezifischer erfolgen, indem für Bachelor- und Masterstudierende unterschiedliche Kompetenzstufungen angesetzt werden. An dieser Stelle ist also das Zusammenführen von fundamentalen Ideen und kompetenzorientierten Lehrkonzepten möglich. Damit verbunden sind die vielfältigen Vorteile einer „outcome-based“ Lehrmethode, wie sie in Fuller et al. genannt werden [Ful+07].

Für diese Art der Zusammenführung ist eine Taxonomie oder wissenschaftliche Rahmenstruktur gesucht, welche die Unterschiede zwischen Bachelor- und Masterstudiengängen mit Blick auf den erwarteten Kompetenzgewinn definiert. Naheliegenderweise eignet sich jede kompetenzorientierte Curriculaempfehlung, im

Fall der Informatik also bspw. die der ACM/IEEE oder der Gesellschaft für Informatik. Es gibt jedoch kein Dokument, das die Trennung von Kompetenz auf Bachelor- und Masterniveau technikunabhängig, also auf der Ebene von Methoden und Sichtweisen, beschreibt. Das ACM/IEEE-Curriculum für Informatik und Technische Informatik ist wie die GI-Empfehlungen für Technische Informatik auf Bachelor-Studierende, die Empfehlungen der ARTIST-Education-Group auf Masterstudierende ausgelegt. Eine Vermischung dieser Arbeiten nach den jeweiligen Schwerpunkten kann aufgrund der unterschiedlichen Detailgrade und Anwendungsgebiete sowie dem Verständnis über den Kompetenzbegriff nicht durchgeführt werden.

Eine Alternative für die Zusammenführung findet sich jedoch in den Europäischen und Deutschen Qualifikationsrahmen für Hochschulen, die im Prozess der Bologna-Reform entstanden sind. Im Bologna-Prozess wurde 1999 die Grundlage für einen europäischen Hochschulraum mit den Zielen einer vergleichbaren und transparenten Studiensituation in der Europäischen Union geschaffen. In Anlehnung an diesen Vorgang hat eine Gruppe aus Vertretern der Hochschulrektorenkonferenz, der Kultusministerkonferenz und dem Bundesministerium für Bildung und Forschung einen Qualifikationsrahmen für deutsche Hochschulabschlüsse vorgelegt (QDH) [HKB05]. In diesem Dokument findet sich die gesuchte Auftrennung der zwei Ausbildungsstufen an Universitäten. Der QDH erfüllt dabei nicht nur die Kompatibilität mit seinem europäischen Pendant, sondern ist auch hochschultypunabhängig beschrieben. Für die Adaption der Kriterien bedeutet dies, dass die vorgestellte Methode für Universitäten *und* Fachhochschulen Gültigkeit besitzt, da die Niveaustufungen von Bachelor- und Masterstudiengängen unabhängig von institutionellen Rahmenbedingungen definiert sind. Bei der folgenden Zusammenfassung sind die Beschreibungen auf ihre Wortwahl hin zu untersuchen, um ggf. auch eine Verbindung zu Lernzieltaxonomien von Anderson und Krathwohl oder Fuller et al. herstellen zu können [AK01], [Ful+07]. Bei der Analyse wurden die Ebenen *Wissen und Verstehen*, und *Können* des QDHs zusammengelegt, da beide Bestandteile der Weinertschen Kompetenzdefinition sind. Eine getrennte Betrachtung würde dem in dieser Arbeit verwendeten Kompetenzbegriff widersprechen.

Wie bereits erläutert, sind die Inhaltselemente des QDH nicht kompetenzorientiert definiert und lassen sich daher nicht ohne Abänderung für das Vertikalkriterium übertragen. Zusammenfassend unterscheiden sich Bachelor- und Masterstudierende vor allem durch die Tiefe des vorausgesetzten Wissens und dem Kontext der Wissensverknüpfung. Für Bachelorstudierende wird ein breites Wissen auf Basis etablierter Fachliteratur empfohlen. Masterstudierende hingegen sollen wissenschaftsrelevantes Wissen mit einem deutlich gestiegenen Anteil an Spezialisierung erlangen. Bei beiden Zielgruppen wird ein selbstgesteuerter Lernprozess erwartet, der bei Bachelorstudierenden hauptsächlich mit den Prozessdimensionen *Strukturieren, Erarbeiten* und *Anwenden*, bei Masterstudierenden mit den Prozessdimensionen *Integrieren, Problemlösen* und *Entwickeln* verknüpft ist. Diese Gliederung muss bei der Analyse möglicher Ideen mit Hilfe des Vertikalkriteriums berücksichtigt werden. Eine Idee, die sich nur in der Systemimplentierung wiederfindet oder eine Problemlösetechnik in einem eng umrahmten Spezialgebiet darstellt, kann nur auf Masterniveau erlernbar sein und damit dem Vertikalkriterium nicht genügen.

Aufgrund der weitreichenden Änderungen am Kriterium und der Niveaustufung durch den QDH wird das Vertikalkriterium als Fortbildungskriterium verstanden und diese Terminologie für den restlichen Teil der Arbeit verwendet. Das Fortbildungskriterium ist wie folgt definiert.

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> „die auf jedem für die Hochschullehre relevanten intellektuellem Niveau, also für Bachelor-, Master- und Diplomstudierende, erfolgreich vermittelt werden kann."

4.2.3. Analyse des Zeitkriteriums

Das Zeitkriterium ist weitestgehend zielgruppen- und anwendungsgebietunabhängig und deswegen ohne Abänderung zu übernehmen. Es stellt sicher, dass eine Idee in der Gegenwart und in der Vergangenheit von signifikanter Relevanz war. Die Annahme ist, dass eine solche Idee auch für zukünftige Generationen nützlich sein wird. Wie auch Modrow auffällt, kommt Schwill nur selten auf dieses Kriterium zurück [Mod06]. Nievergelt bestätigt ebenso, dass die Untersuchung vergangener erfolgreicher Konzepte in der Informatik schwierig ist [Nie90]. Allgemein ist es bei einer so jungen Disziplin wie der Informatik schwer, im gleichen zeitlichen Rahmen Techniken auszuwählen, wie dies bspw. bei der Mathematik, Biologie oder Physik der Fall ist.

Daraus ist zu schlussfolgern, dass die Relevanz und Wahrnehmbarkeit in einer anderen zeitlichen Größenordnung zu sehen sind, nicht aber ganz verworfen werden sollte. Der Autor schlägt einen Rahmen von ungefähr 20 Jahren vor, in denen das Thema Teil von wissenschaftlichen Diskussionen, gesellschaftlichen Fragestellungen oder von industriellen Aufgaben war. Dieser Wert leitet sich aus der Annahme ab, dass eingebettete Systeme nach heutigem Verständnis ab 1970 auftraten. Die Disziplin ist demnach ungefähr 45 Jahre alt. Die Hälfte dieser Zeit scheint für die historische Betrachtung fundamentaler Lerninhalte rechtfertigbar oder anders gesagt: Eine Idee, die eine Disziplin oder einen Anwendungsbereich über eine solche Zeit mitbestimmt hat, kann fundamentale Eigenschaften besitzen.

Ähnlich zum Horizontalkriterium ist bei der Analyse zu beachten, dass nicht nur die Historie der Informatik betrachtet wird. Caspi et al. weisen darauf hin, dass es viele Überschneidungen zwischen Disziplinen und Anwendungsbereichen gibt, die in der Diskussion um eingebettete Systeme beachtet werden müssen [Cas+05]. Das Zeitkriterium wird wie folgt definiert:

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> „die in der historischen Entwicklung eingebetteter Systeme über die letzten 20 Jahre deutlich wahrnehmbar ist und längerfristig relevant bleibt."

Wie sich in der später erfolgten Anwendung des Zeitkriteriums zeigte (siehe Kapitel 5.7), ist auch eine kürzere Zeitspanne argumentierbar. Die Begründung hierfür findet sich in der Schnelllebigkeit der technischen Innovationen; gerade in Hinblick auf Moores Law. Demzufolge ist die Leistungsfähigkeit der Techniken nicht durch eine lineare Funktion über die Zeit abzubilden. Die Festlegung des Zeitschwellwertes hat deswegen immer disziplinspezifisch zu erfolgen.

4.2.4. Analyse des Zielkriteriums

Das Zielkriterium ist nicht in den ursprünglich von Schwill definierten Kriterien zu finden, sondern wurde später hinzugefügt. Zendler und Spannagel sehen den Grund hierfür in der weiteren Auseinandersetzung Schwills zwischen den Begrifflichkeiten Konzept und Idee. In ihrer Sammlung zentraler Konzepte verwerfen Zendler und Spannagel das Zielkriterium, weil es schwer fassbar ist [ZS06]. In der Tat stellt sich nach den Erweiterungen nach Modrow zumindest die Frage, ob eine fundamentale Idee der Informatik „zur Annäherung an eine gewisse idealisierte Zielvorstellung dient“ [SS11, S. 65] (Zielkriterium), wenn sie keinen „Bezug zu Sprache und Denken des Alltags und der Lebenswelt besitzt“ [SS11, S. 65] und nicht „für das Verständnis des Faches notwendig ist“ [SS11, S. 65] (Sinnkriterium).

Da jede Idee einen Gegenstandsbereich hat, ist auch immer eine Intention oder ein Ziel vorhanden. Deswegen kann, nach Auffassung des Autors, jeder Sachverhalt mit dem Zielkriterium gerechtfertigt werden. Auch für das Gebiet der Entwicklung eingebetteter Systeme stellt dieses Kriterium deswegen keine Filterfunktion dar. Die Forderung nach einer konkreten Zielstellung ist implizit durch die später erläuterte Anpassung des Sinnkriteriums gegeben und würde sich somit doppeln. Das Zielkriterium wird nicht in den Kriterienkatalog übernommen.

4.2.5. Analyse des Sinnkriteriums

Im vorherigen Abschnitt wurde bereits auf die Wechselwirkungen zwischen Sinnkriterium und Zielkriterium hingewiesen. Modrows Erweiterung des Sinnkriteriums sichert eine *„didaktisch-curriculare“* [SS11, S. 65] Verankerung, die, zumindest bei dem hier vorliegenden Anwendungsbereich und der angesprochenen Zielgruppe, nicht vollständig übernommen werden muss. Die Begründung hierfür findet sich in der unterschiedlichen Zielsetzung der Ausbildung auf schulischer und universitärer Ebene, die im Falle letzterer deutlichen Bezug zu industriellen oder wissenschaftlichen Themengebieten aufweist. In der Regel liegt der Fokus der schulischen Ausbildung auf der Vermittlung allgemeiner Grundlagen, die später in einer Berufsausbildung einschlägig vertieft werden. Für die Ausbildung von Entwicklern eingebetteter Systeme stellt sich deswegen die Frage, ob sich statt einer didaktisch-curricularen nicht eine wissenschaftlich oder industrielle orientierte Verankerung anbietet. Immerhin folgt für die meisten Studierenden nach ihrem Abschluss entweder die Beschäftigung durch einen Arbeitgeber in der Industrie oder eine wissenschaftliche Stelle an einer Hochschule oder Universität. Im universitären Bildungsbereich ist das Verhältnis von allgemein nutzbaren und fachspezifischen

Wissen anders gewichtet als in der Schulbildung. Umso spezieller und konkreter eine Idee wird, desto weniger besitzt sie für das „Denken des Alltags und der Lebenswelt“ einen Bezug. Für eine spezielle Anwendung hingegen erhöht sich dieser Bezug (siehe Vertikal- bzw. Fortbildungskriterium).

Fuller et al. weisen darauf hin, dass das Studieren von Prozessen und das Lösen von Problemen der zentrale Gegenstand von Informatik sind [Ful+07]. Das Sinnkriterium sollte sich deswegen am Problemlösen im wissenschaftlichen und industriellen Kontext ausrichten. Die folgende Definition des Kriteriums fasst diese beiden Bedingungen zusammen:

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> „die einen Bezug zu einer praktischen Problemstellung oder einer wissenschaftlichen Fragestellung besitzt, die in der Entwicklung eingebetteter Systeme signifikant ist.“

4.2.6. Analyse des Repräsentationskriteriums

Hartmann, Näf und Reichert schlagen zusätzlich das *Repräsentationskriterium* vor, welches nur Ideen erfasst, die sich auf unterschiedliche Weise darstellen lassen [HNR06]. Gemeint sind insbesondere enaktive, ikonische und symbolische Darstellungsformen.

Schwill und Schubert entschieden sich dazu, das Repräsentationskriterium nicht in den Kriterienkatalog zur Ermittlung fundamentaler Ideen der Informatik aufzunehmen, da das Vertikalkriterium implizit eine Darstellung auf unterschiedlichen Niveaustufen vorsieht [SS11]. In dieser Arbeit wird das Repräsentationskriterium aus einem weiteren Grund nicht übernommen: Die zwangsläufige Kopplung einer Darstellungsform an den Inhalt einer potentiell fundamentalen Idee widerspricht der Auffassung des Autors nach der Definition von Ideen im weitesten Sinne. Eine Idee kann auch nur begrenzt darstellbar sein wie beispielsweise die Mehrdimensionalität in der Mathematik. Hier muss sich nicht die Idee den kognitiven Strukturen der Studierenden anpassen, sondern eine didaktisch sinnvolle Methode gefunden werden, entsprechende Lerninhalte zu vermitteln.

4.2.7. Zusatz: Das Varianzkriterium

Vier der fünf bisher analysierten Kriterien konnten, zumindest teilweise, übernommen werden. Ein darüber hinausgehendes Kriterium (das Varianzkriterium) soll sicherstellen, dass keine Idee in den Fundus aufgenommen wird, die nur unwesentliche Neuerungen gegenüber einer anderen Idee besitzt. Es ist pragmatisch begründet und stark an das in dieser Arbeit verwandte Vorgehen gekoppelt. Das Varianzkriterium unterstützt den Analyseprozess, indem es die Rahmenbedingungen validiert. Der Ausschluss doppelter Themen vor der Anwendung der Kriterien erfüllt eine gänzlich andere Funktion und ist deshalb nicht mit dem Varianzkriterium gleich zu

setzen. Bei der Analyse des Varianzkriteriums ist bedeutsam, dass unterschiedliche Themen auf die gleiche Idee verweisen können und somit zwar nicht begrifflich synonym, aber inhaltlich synonym verwendet werden (siehe Abschnitt 4.3).

Zwischen zwei Ideen muss deswegen ein gewisser Wissensabstand liegen, der die Fundamentalität der untersuchten Idee abändert oder deutlich erweitert, damit beide aufgenommen werden können. Die folgende Definition wird deswegen verwendet:

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> „die eine neue Facette gegenüber allen anderen Ideen besitzt oder diese auf neuartige Weise miteinander verbindet."

Mit einem besonders hohen Spezialisierungsgrad einer Idee lässt sich oft auch eine entscheidende Neuerung und somit die Erfüllung dieses Kriteriums rechtfertigen. Auf das in Abschnitt 4.3 hingewiesene, möglichst homogene Abstraktionsniveau in der Analyse ist deswegen zu achten. Neben der intendierten Filterfunktion kann das Kriterium dafür genutzt werden, die Alternativen zu einer Idee oder mit ihr artverwandte andere Ideen aufzuzeigen und schließlich zur Konzeption einer Lehrveranstaltung auf Basis des Konzeptes der fundamentalen Ideen beitragen (siehe Kapitel 6).

4.2.8. Zusammenfassung des Kriterienkataloges

Aufbauend auf den Vorarbeiten zum fachdidaktischen Ansatz der fundamentalen Ideen wurde ein angepasster Kriterienkatalog für eingebettete Systeme vorgestellt. Der folgende Abschnitt fasst die Änderungen und Zusätze zusammen.

> Eine fundamentale Idee der Entwicklung eingebetteter Systeme ist eine Methode oder Sichtweise:
>
> - (*Horizontalkriterium*) die im Bereich der Entwicklung eingebetteter Systeme breit anwendbar ist oder sich in mehreren Phasen des Entwicklungsvorgehens wiederfindet. Ideen anderer angrenzender Fachdisziplinen erfüllen das Horizontalkriterium, falls sie notwendiger Betrachtungsgegenstand eingebetteter Systeme sind.
> - (*Fortbildungskriterium*) die auf jedem für die Hochschullehre relevanten intellektuellem Niveau, also für Bachelor-, Master- und Diplomstudierende, erfolgreich vermittelt werden kann.
> - (*Zeitkriterium*) die in der historischen Entwicklung eingebetteter Systeme über die letzten 20 Jahre deutlich wahrnehmbar ist und längerfristig relevant bleibt.
> - (*Sinnkriterium*) die einen Bezug zu einer praktischen Problemstellung oder einer wissenschaftlichen Fragestellung besitzt, die in der Entwicklung eingebetteter Systeme signifikant ist.

- (*Varianzkriterium*) die eine neue Facette gegenüber allen anderen Ideen besitzt oder diese auf neuartige Weise miteinander verbindet.

4.3. Nachteile und Besonderheiten der Vorgehensweise

Obwohl die fundamentalen Ideen inzwischen in viele Disziplinen übernommen wurden und diverse Curriculaempfehlungen maßgeblich geprägt haben (bspw. die GI-Empfehlungen von 2011 [Hoc+11]), enthält das Vorgehen konzeptuell mehrere Schwachstellen. Die folgende Gegenüberstellung zeigt Wege auf, mit denen sich diese Probleme eingrenzen lassen.

Fundamentalität der Ideen Die Analyse von Methoden und Sichtweisen mit dem Konzept der fundamentalen Ideen ist durch die Bewertung eines oder mehrerer Experten subjektiv geprägt. Es ist somit unwahrscheinlich, dass eine Menge an fundamentalen Ideen existiert, mit der jeder Experte einverstanden ist. Eine offene, nachvollziehbare Analyse erleichtert eine Diskussion über die Fundamentalität der untersuchten Techniken. Aus diesem Grund werden alle Ideen im Anhang analysiert und nur eine Idee stellvertretend in Kapitel 5.5 vorgestellt. Der Prozess der Ideenanalyse und deren Ergebnis ist damit nachvollziehbar.

Vollständigkeit des Kataloges Wie in Abschnitt 4.1.2 dargestellt, orientiert sich Schwill am Software-Lifecycle und schließt somit den Großteil der Technischen und Theoretischen Informatik aus. Durch die normative Analyse von Themenschwerpunkten internationaler Konferenzen für das Anwendungsgebiet wird dieser Kritik begegnet (siehe Sektion 5.1).

Angemessenheit der Kriterien Ein weiterer Kritikpunkt betrifft die filternde Funktion der Kriterien. Da es bis auf die Definition des Zeitkriteriums kein Kriterium mit festen Grenzen gibt (hier die 20 Jahre), besteht die Gefahr, dass eine Analyse auch subjektiv geprägt sein könnte. Dieser Vermutung muss entgegnet werden, dass auch bei einer alternativen empirischen Befragungen, durch die individuelle Einschätzung der Befragten, immer Subjektivität enthalten ist. Die in Abschnitt 4.3 dargestellten Beispiele und Gegenbeispiele für jedes Kriterium zeigen, wie der Autor der Arbeit die Kriterien anwendet, und welche Grenzen zwischen einer positiven und negativen Evaluation liegen. Die geäußerte Kritik liegt sicherlich auch darin begründet, dass in den meisten Veröffentlichungen zum Konzept der fundamentalen Ideen nur die positiv analysierten Ideen dargestellt werden, nicht aber diejenigen, die die Kriterienanalyse abgelehnt hat. Die komplette Analyse aller Ideen ist im Anhang offengelegt.

Nichtspezifischer Transfer Die fundamentalen Ideen basieren aus pädagogischer Sicht auf dem Konzept des Wissenstransfers. Nach Gagne, Perkins und Salomon, Detterman und De Corte kann dieser folgendermaßen definiert werden: „[...] the degree to which behavior will be repeated in a new learning

situation“ [DS13, S. 2]. Für zukünftige Problem- und Lernsituationen ist die Adaption vorhandenen Wissens in neue Kontexte erforderlich. Transfer ist somit ein wichtiges Werkzeug zur Aneignung weiteren Wissens und zur Bewältigung zukünftiger (Arbeits-)Aufgaben. Man unterscheidet zwischen spezifischem und nichtspezifischem Transfer. Schwill argumentiert, dass ersterer nur „relativ lokale Effekte“ [Sch93, S. 2] hat. Nichtspezifischer Transfer bezieht sich seiner Auffassung nach auf langfristige Effekte, bei denen zusätzlich zu Fertigkeiten auch „grundlegende Begriffe, Prinzipien und Denkweisen (sog. fundamentale Ideen)“ [Sch93, S. 2] gelernt werden. Empirisch gestützte Forschungsergebnisse lassen Zweifel hinsichtlich der Wirksamkeit des nichtspezifischen Transfers erkennen [EW12], [BN91], [ARS96], [Det93].

Der vorliegende Ansatz berücksichtigt die Kritik an nichtspezifischem Transfer, indem keine Ideen mit besonders weitem, vom Anwendungsbereich unabhängigen Kontext untersucht worden sind. Alle Ideen haben einen Bezug zur Entwicklung eingebetteter Systeme und lassen sich deswegen zwischen spezifischem und nichtspezifischem Transfer einordnen.

Durchführung der Analyse Obwohl Schwills Forschungsarbeiten größtenteils als wichtig und begründet wahrgenommen worden sind, gab es auch Kritik hinsichtlich seines Vorgehens und der Auswahl der untersuchten Themenbereiche. Modrow plädiert für eine Sammlung generellerer Ideen als die von Schwill vorgestellte Sammlung aus 61 Themen und Überbegriffen. Seiner Begründung nach enthält Schwills Katalog „[...] (fast) alle in Frage kommenden Ideen, also viel zu viele“ [Mod06, S. 3]. Nach Meinung des Autors ist die Orientierung an informatischen Problemen, wegen der Berücksichtigung spezifischen Transfers, eher Vorteil als Nachteil.

Humbert kritisiert die mangelnde Diskussion der einzelnen Themen, auf die Schwill nur exemplarisch eingeht. Ebenso enthalten Schwills Arbeiten keine Beispiele für Ideen, die untersucht, aber nicht angenommen worden sind [Hum06]. Diese Kritik wird berücksichtigt, indem alle durchgeführten Analysen in einem umfangreichen Analyseprotokoll im Anhang dieser Arbeit zu finden sind. Zusätzlich werden gesondert Beispiele und Gegenbeispiele für jedes Kriterium erläutert.

Eine mögliche Subjektivität ist auch bei der Anwendung der Kriterien auf die zu untersuchenden Techniken, Konzepte und Methoden möglich. Die Argumentation für oder gegen eine Idee findet je nach Kriterium auf unterschiedlichen technischen Abstraktionsschichten statt. Die Abbildung 4.2 veranschaulicht, dass das Horizontalkriterium, das Fortbildungskriterium und das Zeitkriterium eher technikunabhängig gerechtfertigt werden können, während sich für das Sinnkriterium und das Varianzkriterium eine konkrete Instanz der Idee anbietet. Diese Charakteristika sind bei der Analyse selbst, bzw. bei der Wahl eines möglichst homogenen Abstraktionsniveaus, zu beachten und von Bedeutung, wenn eine Lehrveranstaltung auf Basis der fundamentalen Ideen konzipiert wird (siehe Abschnitt 6).

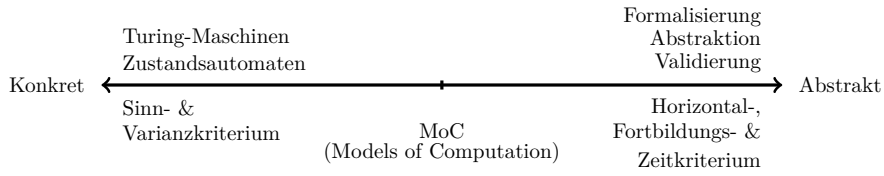

Abbildung 4.2.: Wechselwirkung bei der Ideenbeschreibung: Konkret vs. Abstrakt

Beispiele und Gegenbeispiele zur Kriterienanalyse

Um die teils abgeänderten, teils neuen Kriterien verständlich zu machen, sind im Folgenden Beispiele und Gegenbeispiele von Methoden und Sichtweisen geschildert.

Beispiele für das Horizontalkriterium *Models of Computation* (MoCs) gehören zu den Ideen, die das Horizontalkriterium positiv evaluiert. MoCs legen die Regeln fest, nach denen die Kommunikation und die Berechnung in einem Modell abläuft. Deswegen werden sie nicht nur beim struktur-, sondern auch beim verhaltensorientierten Entwurf eingebetteter Systeme genutzt. Sie sind eng mit dem grundlegenden Ansatz der Modellbildung elektrotechnischer und informatischer Entwurfstechniken verbunden und werden bspw. in Bereichen des Designs (bspw. SysML) und der Implementierung (endliche Automaten zu VHDL) eingebetteter Systeme eingesetzt.

Virtualisierungstechniken sind ein Themenbereich, der mit Hinblick auf das Horizontalkriterium nicht positiv evaluiert werden kann. Weder ist es ein verpflichtender Aspekt in der Entwicklung eingebetteter Systeme, noch lässt sich die Umsetzung des Konzeptes in mehreren Entwicklungsphasen beobachten.

Beispiele für das Fortbildungskriterium Die Idee paralleler und nebenläufiger Prozesse genügt dem Fortbildungskriterium. Es ist gleichermaßen Bestandteil der Software- und Hardwareentwicklung und notwendige Grundlage vieler weiterer Themen (bspw. FPGA-Programmierung bzw. rekonfigurierbare Architekturen). Zugleich ist das Verständnis nebenläufiger Anweisungen und Berechnungen eine wesentliche Grundvoraussetzung, um die inhärente Parallelität jeglicher Hardware zu verstehen. Diese Idee ist auf Bachelor- und Masterniveau vermittelbar.

Ein Gegenbeispiel für das Fortbildungskriterium sind massiv-parallele Architekturen. Trotz ihrer ähnlichen Namensgebung ist die zentrale Idee eine deutlich Andere. Massiv parallele Architekturen nutzen das Konzept der Parallelität, um die Leistungsaufnahme eines Systems zu verringern oder dessen Verlässlichkeit zu erhöhen, selbst wenn es in letzterem Fall aus einzelnen, unverlässlichen Komponenten zusammengesetzt ist [Ahm95]. Die Studierenden müssen demnach für das Verständnis des Konzeptes bereits ein umfangreiches Wissen über Parallelität, Nebenläufigkeit und damit verbundene Themenbereiche wie Datenfluss- oder Kontrollflussabhängigkeiten besitzen. Fehlertoleranz, Fehlererkennung und Fehlervermeidung sind

Lerninhalten, die ebenso vor der Vermittlung der Idee thematisiert werden sollten. Mit diesen speziellen Zielsetzungen und den damit verbundenen benötigten Vorwissen kann die Idee nicht als grundlegend und damit als dem Fortbildungskriterium genügend angesehen werden.

Beispiele für das Zeitkriterium Die Methode des Hardware/Software Co-Designs genügt dem Zeitkriterium. Die Idee des Ansatzes ist eine neue Strukturierung der Systementwicklung, bei der die Ansätze von Hardware und Software zusammengeführt und nebenläufig umgesetzt werden. Dass dieser Ansatz schon seit über 20 Jahren diskutiert wird, belegt unter anderem der Artikel von Wolf [Wol03].

Ein Gegenbeispiel für das Kriterium ist die Idee der modellbasierten Entwicklung. Das zentrale Merkmal der Modelltransformation sowie die iterative Verfeinerung von Modelleigenschaften auf unterschiedlichen Entwicklungsstufen lässt sich erst ab der Jahrtausendwende in breiterem Stil beobachten. Da auch die Bedeutung modellbasierter Entwicklung für die Zukunft kontrovers diskutiert wird (siehe Abschnitt A.18), kann das Kriterium nicht positiv evaluiert werden.

Beispiele für das Sinnkriterium Der Themenbereich der Co-Synthese und Co-Verifikation ist ein Beispiel für zwei Ideen, die vom Sinnkriterium positiv evaluiert werden. Im wissenschaftlichen und praktischen Kontext wird die gleichzeitige Verifikation von Hardware und Software häufig genutzt (siehe bspw. [HKM01]). Durch dieses Vorgehen lässt sich die Designzeit reduzieren und gleichzeitig alle Entwicklungsschritte vor der Systemintegration früher durchführen. Probleme bei der Abbildung von Software auf Hardware-Komponenten können damit frühzeitig entdeckt werden. Das Konzept ist deswegen maßgeblich an der Reduktion der Entwicklungszeit aktueller Industrieprodukte beteiligt.

Der probabilistische Entwurf ist ein Gegenbeispiel für das Sinnkriterium. Der Einsatzbereich ist zu eng, um für den Großteil der Zielgruppe relevant zu sein. Eine rege Diskussion findet ausschließlich im wissenschaftlichen Kontext statt. Mit dieser fehlenden Relevanz für die industrielle Praxis muss die Idee probabilistischer Systeme als nicht fundamental bewertet werden.

Beispiele für das Varianzkriterium Das Varianzkriterium dient der Separierung ähnlicher Ideen, wenn nicht mindestens eine grundlegend neue Facette im Vergleich zu bereits erfassten Ideen vorhanden ist. Es kann also immer nur auf zwei oder mehr Methoden oder Sichtweisen angewandt werden. Im Fall von Model-based Design und Component-based Design kann die Trennung aufrechterhalten werden. Die strukturelle oder verhaltensspezifische Abbildung eines Systems im Model-based Design funktioniert fast immer über Komponenten. Auch werden Komponenten oft bei modellgetriebenen Entwicklungen verwendet. Obwohl also beide Konzepte einen starken Bezug zueinander haben, vertreten sie verschiedene Ideen und genügen somit dem Varianzkriterium. Diese Art der Verbindung ist für die Gestaltung von Lehr-/Lernangeboten besonders wertvoll, da durch sie eine Verknüpfung aus didaktischer Sicht nahegelegt wird.

Ein Gegenbeispiel für dieses Kriterium wären die zwei Kernideen hinter den Begriffen FPGAs und rekonfigurierbaren Architekturen. Beide beschreiben ein und dieselbe Idee, jedoch auf unterschiedlichen Abstraktionsniveaus. Die Begriffe stehen also in einer hierarchischen Beziehung, da rekonfigurierbare Architekturen der abstrakte Überbegriff für die Kernidee hinter FPGAs sind. Deswegen kann die Idee nur einmal aufgenommen werden.

5. Anwendung des Kriterienkataloges auf fachspezifische Methoden und Sichtweisen

Durch die Anwendung des Kriterienkataloges auf eine Auswahl von Ideen kann deren Eignung als Grundlage der Fachdisziplin verifiziert werden. In einer breit aufgestellten Disziplin wie der Entwicklung eingebetteter Systeme ist es schwierig, eine Sammlung an wesentlichen Ideen zu finden, die möglichst viele Aspekte, im besten Fall den ganzen Entwicklungszyklus, abdeckt. Schwill hatte für die Informatik ein ähnliches Problem. Sein Ansatz bestand in der Beschreibung einer typischen Aufgabe innerhalb der Informatik - dem Softwareentwicklungsprozess. Dies ist einer der genannten Kritikpunkte an Schwills Vorgehen, da er Überlegungen zur Theoretischen Informatik oder zu ebenso informatikspezifischen Themen wie Rechnernetze oder Rechnerarchitekturen nicht analysiert hat.

Dieses Kapitel stellt einen alternativen Weg der Ideenauswahl vor, der in Abschnitt 5.1 erläutert und hinsichtlich der möglichen Umsetzungen analysiert wird. Darauf aufbauend werden in Sektion 5.2 die praktischen Rahmenbedingungen der Ideenauswahl (zum Beispiel Clusterung und Filterung) und in Abschnitt 5.3 die Kriterienbewertung auf einer dreistufigen Skala erläutert. Der darauf folgende Teil 5.4 stellt die Sammlung aller durch das Verfahren gewonnenen Ideen vor. In Abschnitt 5.5 wird exemplarisch für eine der vorher dargestellten Ideen die kriterienbasierte Analyse durchgeführt. Die Analyseprotokolle der restlichen Untersuchungen finden sich im Anhang (siehe Anhang A). Abschließend wird in Abschnitt 5.7 das Ergebnis des Forschungsschrittes in Form einer Zusammenfassung der gesamten Analyseergebnisse geschildert und ein Rückblick auf die Selektions- und Filterfunktionen der einzelnen Kriterien gegeben.

5.1. Vorgehensweise

Im Gegensatz zum Schwillschen Vorgehen soll für die Auswahl der Ideen eine durch Experten aufgestellte Liste verwendet werden. Dafür bieten sich zwei unterschiedliche Ansätze an. Die erste Möglichkeit liegt in der Betrachtung formeller Begriffskataloge wie denen der ISO, IEEE oder der ACM. Die Auswertung informeller Sammlungen, die das Themenfeld oder die Aufgabe nicht nur stichwortartig, sondern im Kontext beschreiben, stellen den zweiten Ansatz dar (Curricula oder „Call for Papers“). Der zweite Ansatz hat den Vorteil, dass nicht nur die begriffliche

Ebene betrachtet wird, sondern Ideen mit einem bestimmten Anwendungsfeldbezug, also einem Kontext, beschrieben werden, der die Idee für die Analyse mit Hilfe der Kriterien deutlicher eingrenzt. Der nachfolgende Abschnitt erläutert, mit welchen Ergebnis die zwei dargestellten Ansätze mit ihren jeweiligen Umsetzungen untersucht wurden.

Standards Catalogue ISO Die *Internationale Organisation für Standardisierung* (ISO) ist bei über 19.500 internationalen Standards auf ein Katalogsystem zur Ordnung und Strukturierung der Dokumente angewiesen. Die dafür genutzte, nur im Internet verfügbare, Struktur kann in die zwei Bereiche Standards und Technikkomitees unterteilt werden [Sta14]. Beide Bereiche unterscheiden sich nicht inhaltlich, sondern nur in der Organisation der Standards. Die Struktur der ISO enthält zwar viele für die eingebettete Systementwicklung relevante Themenfelder, ist aber nicht nach Technologien oder Ansätzen, sondern nach Anwendungsgebieten geordnet. Damit verbunden sind Mehrfachnennung von Themen und eine unübersichtliche Struktur. So werden die potentiell interessanten Gebiete Automobil-, Flugzeug- und Schienentechnik bereits auf der obersten Ebene unterschieden. Eine Untersuchung fundamentaler Ideen anhand der Inhalte verschiedener Standards ist sehr schwierig, da alle einen konkreten Anwendungsfall oder Kontext berücksichtigen, dessen Gegebenheit nur industrielle Rahmenbedingungen, nicht aber die einer Bildungseinrichtung berücksichtigen. Durch die problematische Begriffshierarchie wird die Anwendung des Horizontal- und des Fortbildungskriteriums erschwert, da sich andere Themenbereiche, die für beide Kriterienanalysen relevant sind, auf ganz anderen Ebenen dieser Baumstruktur befinden können.

IEEE-Thesaurus und IEEE-Keyword-List Das *Institute of Electrical and Electronics Engineers* (IEEE) ist als größter technisch-wissenschaftlicher Verband der Welt ähnlich wie die ISO auf eine nachvollziehbare und feingranulare Gliederungsstruktur für wissenschaftliche Beiträge angewiesen. Damit Autoren die Einteilung in die entsprechenden Themenbereiche selbst vornehmen können, bietet die IEEE zwei verschiedene Richtlinien an: das IEEE-Lexikon (engl. *Thesaurus*) und die IEEE-Taxonomie [EI14b], [EI14a]. Das Lexikon ist als Vokabularrichtlinie für technische, wissenschaftliche und IEEE-spezifische Begriffe zu sehen. Es wird von Experten der IEEE erstellt und reguliert. Der Katalog umfasst in der Version von 2013 580 Seiten mit verschiedenen Beziehungsarten zwischen den aufgeführten Begriffen. Durch die Unterteilung der Beziehungsarten in *„Überbegriff“*, *„Unterbegriff“*, *„verwandter Begriff“*, *„bevorzugter Begriff“* und*„genutzt für“* lässt sich eine mehrschichtige Baumstruktur erstellen. Da die Informatik und die Elektrotechnik sowie verwandte Disziplinen komplett betrachtet werden, ist der Beschreibungsgrad an Informationen auf unterster Ebene gering gehalten, damit das Dokument nicht unübersichtlich groß wird. Ebenso sind Kompromisse bei der Begriffswahl gemacht worden, welche die Klassifizierung teilweise negativ beeinflussen. So ist der Begriff *Embedded Systems* unter *Operating Systems* einsortiert, welcher wiederum unter *Software-Systems* und somit unter *Software* klassifiziert ist. Da eingebettete Systeme keine Software sind und auch nicht ausschließlich aus dieser bestehen, ist diese Einordnung zu überdenken.

Die IEEE-Taxonomie hat prinzipiell dieselben Stärken und Schwächen des Lexikons, da sie aus diesem generiert wird und deswegen auch nur eine etwas übersichtlichere, komprimierte Fassung darstellt. Das Hauptproblem beider Dokumente ist der grobgranulare Detailgrad und die Einordnung von Techniken und Konzepten an streitbaren Positionen innerhalb der Baumstruktur. Beide Ansätze sind somit nicht alleinig zur Auswahl von Ideen geeignet, sondern können in der aktuellen Fassung maximal unterstützend dienen.

ACM Computing-Classification-System Ähnlich zur IEEE benötigt auch die *Association for Computing Machinery* (ACM) eine Struktur zum Organisieren wissenschaftlicher Beiträge. Dafür benutzen sie jedoch kein Lexikon, sondern ein Computing-Classification-System (CCS), welches 2012 komplett überarbeitet wurde und vom Aufbau her der IEEE-Taxonomie ähnelt [Com12]. Die Kategorisierung der Einträge hat ein Team aus 85 Wissenschaftlern zusammengestellt. Leider ist die Datenbank zum Zeitpunkt des Analyseschrittes recht neu und deswegen unvollständig. Beiträge aus Themenbereichen, die bereits vorher als Klassifikation zur Verfügung standen, wurden nur sporadisch übernommen. Unter dem Eintrag „Embedded and cyber-physical systems“ sind lediglich fünf weitere Kategorien. Eine davon ist „Embedded systems“, welche die Einträge „Firmware“, „Embedded Software“ und „Embedded Hardware“ beinhaltet. Da diese drei die Blattknoten der Baumstruktur darstellen, sind ihnen direkt die neusten wissenschaftlichen Beiträge zugeordnet. Für den Bereich Firmware sind fünf Beiträge aufgelistet, wovon die Mehrzahl jedoch aus den 80er-Jahren ist. In den anderen beiden Bereichen stammen die neusten Publikationen von 2012 (auch noch im Jahr 2014) und somit ebenfalls nicht mehr aktuell. Durch diese unvollständigen Datensätze kann das CCS nicht für die Herleitung der für diese Arbeit interessanten Themen verwendet werden. Die Prüfung dieser Sammlung auf Vollständigkeit und Aktualität sollte auch für zukünftige Arbeiten durchgeführt werden, da das Kategoriesystem konzeptuell für die Zusammenstellung eines Ideenkatalogs geeignet scheint.

ACM Special Issues Die ACM Special Issues sind von der ACM organisierte Sammelbeiträge (Journals) zu besonders interessanten Themen der Forschung oder industriellen Praxis. Für den Bereich eingebetteter Systeme ist dies die *Transactions on Embedded Computing* (TECs) [Com14]. Im Gegensatz zu den bisherigen Ansätzen sind die ACM Special Issues also nicht für eine Strukturierung des Themenbereiches gedacht, sondern zur Zusammenfassung von Forschungsbeiträgen aus den jeweiligen Fachbereichen. Aus diesem Grund enthalten die Calls der Special Issues ausführlichere Beschreibungen der zu untersuchenden Thematik und die Beziehungen zwischen den verschiedenen Problemen und Herausforderungen. Neben der breiteren Betrachtung von möglichen Ideen ist das Anwendungsgebiet der TECs-Beiträge meist stark fokussiert, was sich im Umfang und der Detailtiefe der angegebenen Problemstellungen widerspiegelt. Durch die Weitergabe der Koordination entsprechender Tagungen sind die aufgeführten Themen nicht immer überschneidungsfrei. Dies stellt kein Problem dar, da neben einer vorgeschalteten Filterung doppelter Techniken auch Mehrfacheinträge auf inhaltlicher Ebene

durch das in dieser Arbeit vorgestellte Varianzkriterium vermieden werden (siehe Abschnitt 4.2.7). Mit 38 Special Issues im Bereich *Embedded Computing System* im Zeitraum von 2008 bis 2013 sind zudem genug Themen verfügbar, um eine breite Betrachtung der Disziplin zu gewährleisten.

Curricula Ein weiterer Ansatz ist die Untersuchung einer Curriculaempfehlung. Zu den am häufigsten referenzierten Curriculaempfehlungen zählen die international gültigen Empfehlungen der IEEE und der ACM [Cas+08], [ACM04]. Das Computer-Science-Curriculum von IEEE/ACM wurde 2013 komplett überarbeitet und stellt nach der Erstveröffentlichung von 2001 die zweite Version der Empfehlungen dar [SR13]. Eine Überarbeitung der ersten Veröffentlichung erfolgte 2008 (vgl. [Cas+08]). In beiden Versionen werden eingebettete Systeme nur oberflächlich behandelt, da sie inhaltlich zur Technischen Informatik, also dem Computer-Engineering, gezählt werden. Die letzte Version der Computer-Engineering-Empfehlungen ist aus dem Jahr 2004 [NAI04] und somit nicht mehr als Grundlage für die fachdidaktische Analyse fundamentaler Ideen geeignet. Im Dokument selbst wird darauf hingewiesen, dass durch den schnellen Wandel verwendeter Techniken und Konzepte eine regelmäßige Aktualisierung der Empfehlung notwendig ist. Andere Wissenschaftler haben das Computer-Engineering-Curriculum als Grundlage für ihre eigenen Vorschläge verwendet. Besondere Aufmerksamkeit erfuhren die ein Jahr später erschienenen Empfehlungen der Artist-Education-Group [Cas+05], da sie eine ausschließlich auf die Entwicklung von eingebetteten Systemen ausgerichtete Empfehlung darstellen. Das Artist-Team stellt neben grundlegenden Prinzipien fünf zentrale Themengebiete für die Technische Informatik im Bereich der Entwicklung eingebetteter Systeme vor. Je nach Umsetzung kann das Curriculum durch weitere optionale Lernbereiche erweitert werden. Die Autoren beziehen sich ausschließlich auf das Masterstudium, was die Verwendung der Empfehlungen im Kontext des hier vorgeschlagenen Konzeptes der fundamentalen Ideen erschwert. Für die Analyse des Fortbildungskriteriums ist dieser Umstand ungünstig, da explizit verlangt wird, dass eine Idee auf Bachelor- und Masterniveau vermittelbar ist. Ähnlich zu den Curriculaempfehlungen der IEEE/ACM der Technischen Informatik wurden sie seit der ersten Veröffentlichung nicht mehr überarbeitet.

Ein etwas neueres Curriculum haben Ricks, Jackson und Stapleton vorgestellt [RJS08]. Es basiert auf dem 2004er-Computer-Engineering-Curriculum und fügt inhaltlich keine Themen hinzu. Im Gegensatz zu den anderen Dokumenten orientieren sich die Autoren in ihrer Arbeit an den Rahmenbedingungen der Universität Alabama, was die Adaption auf eine generelle Lehr-/Lernempfehlung erschwert.

Die letzte für die Sammlung der Ideen berücksichtigte Curriculaempfehlung ist von der *Gesellschaft für Informatik* (GI) und bezieht sich auf die Technische Informatik auf Bachelor- und Masterniveau [Hoc+11], wobei der Schwerpunkt der Erläuterungen auf die Strukturierung des Bachelorstudiums ausgerichtet ist. Von den sieben vorgeschlagenen Themenbereichen beschäftigt sich lediglich einer mit eingebetteten Systemen, was den Raum für die Erhebung

fundamentaler Ideen einschränkt. Für die Curriculaempfehlung der GI spricht die Aktualität und der Fokus auf das Anwendungsgebiet.

Die Analyse der Quellensammlung ergab, dass die GI-Empfehlungen zur Technischen Informatik und die TECs-Sammlung als Quellen besonders interessant sind. Zwar wurde auch von den zwei einschlägigen Berufsverbänden der ACM und der IEEE der Bedarf festgestellt, die Forschung an der Entwicklung eingebetteter Systeme mit in ihr Portfolio aufzunehmen, jedoch ist dies bisher nicht in geeigneter Weise geschehen. Falsche Kategorisierung auf begrifflicher Ebene und unzureichende Übertragung aktueller Fachforschung in die entsprechenden Kategorien sind die zwei maßgeblichen Gründe dafür, dass die folgenden Forschungsschritte nicht auf diesem Fundament aufbauen können.

Die TECs-Sammlungen wie auch die Empfehlungen der Gesellschaft für Informatik sind aktuell und bieten geeigneten Umfang für eine normative Analyse. Während die TECs-Calls spezifisch auf eingebettete Systeme fokussiert sind, ist der Themenbereich in den GI-Empfehlungen nur ausschnittsweise dargestellt. Zusätzlich sind die Inhaltsbeschreibungen bei beiden Quellen heterogen hinsichtlich ihres Anwendungsbezuges, müssen also noch vereinheitlicht bzw. zusammengefasst werden. Der größte Nachteil bei den GI-Empfehlungen ergibt sich aus dem Umstand, dass diese bereits eine komprimierte Version einer Expertenanalyse darstellen. Es ist weder ersichtlich, wie der Entscheidungsprozess zur Gestaltung der Empfehlung ausgesehen hat, noch welche Inhalte aus welchen Gründen verworfen wurden. Unter Umständen sind einige Ideen bereits verworfen worden, die der Analyse durch das Konzept der fundamentalen Ideen standgehalten hätten. Ein Grund dafür könnte beispielsweise eine „günstigere“ Aufteilung der Themenbereiche im Curriculum sein, die erst einmal nichts mit der Aussage über die Fundamentalität einer Idee zu tun hat, sondern organisatorischer Natur ist. Aus diesem Grund wird der Korpus an zu analysierender Ideen durch die Untersuchung der TECs-Sammlung erarbeitet.

Die Abbildung 5.1 stellt die Vorgehensweise des Anwendungsprozesses dar. Ausgehend von der Themensammlung, deren Inhalte beispielhaft mit Buchstaben illustriert wurden (1), können relevante Ideen ausgewählt (2) und anschließend deren Tragweite und Kontext mit Hilfe externer Quellen beschrieben werden (3). Der Buchstabe „Z“ steht in der Grafik für eine Idee, die aus den in Abschnitt 5.2 genannten Gründen nicht mit in die Sammlung der zu analysierenden Methoden und Sichtweisen übernommen wurde. Im letzten Schritt erfolgt die Analyse mit Hilfe des bereits im letzten Kapitel vorgestellten Kriterienkataloges (4).

5.2. Filterung der Ideen

Die TECs-Sammlung ist so umfangreich, dass weder der komplette Zeitraum der Veröffentlichungen noch die gesamte Breite des diskutierten Themenbereiches in einer Arbeit abgedeckt werden können. Die Analyse wird deshalb für den Zeitraum von 2008 bis 2013 durchgeführt. Dies entspricht einer Themensammlung mit 277 Einträgen. Da dies immer noch eine zu große Menge an zu analysierenden Ideen

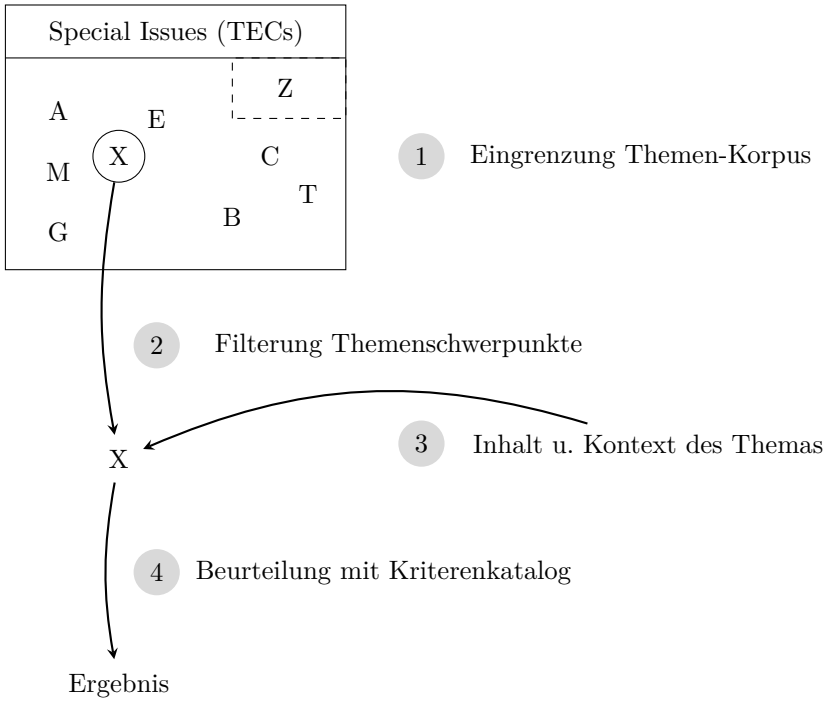

Abbildung 5.1.: Vorgehensweise zur Auswahl, Filterung und Bewertung der Ideenkollektion

ist, empfiehlt sich eine weitere Fokussierung, bei der die Ergebnisse des KOMINA-Projektes hilfreich sind. Die Expertenbefragung zeigte, dass die ersten Phasen des Systemdesigns auf Architektur- und Gesamtsystemebene von den Befragten als besonders wichtig eingestuft wurden [Jas+12]. Die Kompetenzbeschreibungen des EKSM geben auch auf Wortebene Hinweise auf die folgenden wichtigen Lehr-/Lerninhalte (im Text hervorgehoben):

- Sie erlernen *zielgerichtete* und *strukturierte Vorgehensweisen* beim *Entwurf* (C2.1)
- Sie kennen die besonderen *Randbedingungen* des *Entwurfs* eingebetteter Systeme (C2.2).
- Sie erwerben elementare Kenntnisse und grundlegendes Verständnis über die wichtigsten Technologien und über die wichtigsten Konzepte, die beim *Entwurf* und der *Analyse* von rechnergestützten Systemen benötigt werden (C2.3).
- Sie verstehen den *Aufbau* und die *Funktion* aller wichtigen Grundschaltungen und Rechenwerke (C2.3).
- Sie können Schaltungen mittels einer *Beschreibungssprache entwerfen* (C2.4).

- Sie erlernen den Umgang mit der *Programmiersprache* C und deren *Zusammenspiel mit der Hardware*. Dabei werden grundlegende Funktionalitäten und das Zusammenspiel der Basiskomponenten eines *Betriebssystems* mit dem Schwerpunkt auf effiziente *Ressourcenverwaltung* vermittelt (C3.1).

- Sie sind in der Lage, den *Zusammenhang zwischen Hardwarekonzepten und den Auswirkungen auf die Software* zu verstehen, um *effiziente Programme erstellen* und anwenden zu können sowie einen Rechner aus Grundkomponenten *aufbauen* zu können (C3.2).

Die Kompetenzbeschreibung C2.4 gehört im EKSM nicht zu den Kompetenzdimensionen, die im Tätigkeitsfeld des „Designs“ liegen. Die Beschreibung wurde für diese Arbeit trotzdem mit in die zur Fokussierung genutzten Liste an Kompetenzbeschreibungen aufgenommen, da sie explizit das Design auf Wortebene thematisiert und lediglich wegen der Umsetzungsmöglichkeit der Beschreibungssprachen (engl. *description languages*) im EKSM der Implementierung zugeordnet wurde. Da für einige der hervorgehobenen Wörter Synonyme gebräuchlich sind, werden diese ebenso der Liste der zur Analyse genutzten Begriffe zugeordnet. Somit ergibt sich die folgende Sammlung von englischen Fachausdrücken, nach denen der Katalog an zu untersuchenden Ideen zusammengestellt wird: Proceeding, Process, Method, Methodology, Constraints, Requirements, Description-Language, Synthesis, Design, Analysis, Operating System, Resource-Management.

Wie aus der Zusammenstellung und den Kompetenzbeschreibungen zu sehen ist, bewerteten die Experten häufig die Kompetenzbeschreibungen als „sehr wichtig“ (Bewertungsskala, siehe Abschnitt 3.2), welche die Prozessdimension „Design“ beinhalten. Die Argumentation für mehr Systemdesigner anstelle von Spezialisten der Implementierung lässt sich auch aus einer Reihe aktueller Publikationen ableiten [Sif11], [HS06], [JC05], [Cas+05]. Methoden und Sichtweisen, die folgend analysiert werden, sind durch ihren Bezug zu frühen Entwicklungsphasen gekennzeichnet und werden damit dem geschilderten Umstand gerecht. Dies bedeutet jedoch nicht, dass im Entwicklungsprozess nur die Designphase auf das Auftreten der Idee hin untersucht wird. Das Horizontalkriterium ist so definiert, dass auch Themensammlungen mit Bezügen zu anderen Abschnitten des Entwicklungsprozesses positiv evaluiert werden können. Kann der Kontext einer Idee aufgrund der Beschreibung im TECs-Call nicht abgesteckt werden, wird das Themengebiet aufgrund seiner unklaren Beschreibung nicht analysiert.

5.3. Beschreibung und Beurteilung der Ideen

Alle Kriterienanalysen werden mit Hilfe eines dreistufigen Schemas bewertet. Aufgrund der Diversität der Kriterien ist auch die dreistufige Bewertung nicht bei allen Kriterien gleich und wird im Folgenden erklärt.

Horizontalkriterium In der Definition des Horizontalkriteriums sind zwei Aspekte enthalten: die breite Anwendbarkeit und die phasenübergreifende Nutzung

im Entwicklungsvorgehen. Sind beide Aspekte bei einer Idee vorhanden, erfüllt die Idee das Kriterium vollständig. Wird nur eine der beiden erfüllt, kann eine Entscheidung für oder gegen das Kriterium nicht mehr zweifelsfrei erfolgen. Die schlussendliche Annahme oder Ablehnung hängt von der Gewichtung der jeweiligen Argumentation ab. Auf keinen Fall erfüllt ist das Horizontalkriterium, wenn keiner der beiden genannten Aspekte vorhanden ist.

Fortbildungskriterium Ähnlich zum Horizontalkriterium basiert auch das Fortbildungskriterium auf zwei zentralen Aspekten. Zum einen muss die Idee uneingeschränkt auf Bachelor-, Master- und Diplomniveau vermittelbar sein und zum anderen eine Basis für weitere Methoden und Sichtweisen darstellen, die für die Entwicklung eingebetteter Systeme wichtig sind. Wie beim Horizontalkriterium wird das Fortbildungskriterium vollständig erfüllt, falls beide Aspekte in einer Idee ausgemacht werden können. Sobald die Idee auf den angesprochenen Niveaustufen nicht vermittelbar ist, ist sie abzulehnen. Für die zweite Bewertungsstufe heißt dies, dass die Erfüllung des Kriteriums abzuwägen ist, falls es nur vermittelbar, nicht aber notwendiges Vorwissen für andere Ideen der Entwicklung eingebetteter Systeme ist. Hier ist also eine Unterscheidung zur Bewertungsstruktur des Horizontalkriteriums zu erkennen. Sollte eine Idee zwar ein Fundament für weitere Ideen bereitstellen, aber nicht auf Bachelor-, Master- oder Diplomniveau vermittelbar sein, wird sie vom Fortbildungskriterium abgelehnt, da eine solche Idee keinen fachdidaktischen Nutzen im Kontext der Hochschule bereitstellt.

Zeitkriterium Da die Erfüllung des Zeitkriteriums anhand einer definierten Zeitspanne ausgewertet wird, ist eine Dreistufung der Bewertung nicht nötig. Das Kriterium wird erfüllt, wenn die Idee bereits vor den in der Kriteriendefinition festgelegten 20 Jahren interdisziplinär beobachtbar war. Falls dies nicht der Fall ist, wird sie abgelehnt.

Sinnkriterium Für das Sinnkriterium können wie beim Fortbildungs- und Horizontalkriterium wieder zwei Aspekte ausgemacht werden. In der Definition ist festgehalten, dass eine Idee von praktischer wie auch von theoretischer Relevanz für das Anwendungsgebiet sein muss. Analog zum Horizontalkriterium wird deswegen festgehalten, dass das Kriterium voll erfüllt wird, falls beide Aspekte in einer Idee vorhanden sind. Sollte nur ein Aspekt vorhanden sein, muss die Idee im Hinblick auf die Argumentation als entweder erfüllt oder abgelehnt angesehen werden. Wenn weder eine theoretische noch praktische Revelanz zu erkennen ist, ist die Idee abzulehnen.

Varianzkriterium Die Bewertung des Varianzkriteriums unterscheidet sich von denen der anderen Kriterien insofern, dass es nicht nur eine Filterfunktion für Ideen mittels einer Bewertung darstellt, sondern gleichzeitig die Bezüge zu Ideen untereeinander aufdeckt. Konkret kann deswegen der Aspekt des Methodenbezugs von dem des Ideenbezugs unterschieden werden. Falls eine Idee besonders eng mit anderen Ideen in Verbindung steht (auf methodischer Ebene), ohne jedoch identisch zu diesen zu sein, besitzt sie einen hohen Methodenbezug. In diesem Fall ist das Varianzkriterium nicht abzulehnen, da

die untersuchte Idee, wie in der Definition des Kriteriums erläutert, eine eigene Idee darstellt. Die Erfüllung des Kriteriums ist abzuwägen, falls bestehende Ideen auf neuartige Weise miteinander verbunden werden. Je nach Zielsetzung und Umfang dieser Ideenverknüpfung ist eine Ablehnung oder Annahme durch das Kriterium gerechtfertigt. Sollte die Idee jedoch identisch zu einer anderen Idee sein, die bereits im Katalog analysiert wurde, ist das Kriterium als negativ zu evaluieren. Ein vielfältiger Methodenbezug ohne eine die Idee erweiternde Facette reicht nicht zur Begründung des Kriteriums aus.

Diese Skala wurde gewählt, um dem Umstand Rechnung zu tragen, dass die Bewertung fundamentaler Ideen einer Disziplin von Arbeitsschwerpunkten der Industrie und konkreten Zielstellungen der Hochschule abhängt. Ein einfaches *Negativ/Positiv*-System würde die Bewertung von Grenzfällen erschweren. Die Bewertung „2 (mit Einschränkung)“ erlaubt die Verlagerung in das jeweils andere Ergebnis, wenn besondere Umstände dies nahelegen. Ein solcher Umstand kann eine bereits bestehende Lehrveranstaltung an der jeweiligen Universität oder Fachhochschule sein, die als Pflichtveranstaltung für eine andere Lehrveranstaltung eingetragen ist, die einen fundamentalen Lehr-/Lerninhalt vermittelt. Regionale Gegebenheiten in Form von Industriepartnerschaften können durch die Möglichkeit von Praktika durchaus eine Verlagerung des Analyseergebnisses rechtfertigen. Diese Umstände sind in der Analyse jedoch nicht berücksichtigt, um hochschulunabhängige Empfehlungen aussprechen zu können. Bewertungen auf der niedrigsten und höchsten Stufe (eins bzw. drei) sollten hingegen auch bei veränderten Rahmenbedingungen nicht anders eingeordnet werden.

5.4. Sammlung von Themenschwerpunkten

Mit dem oben beschriebenen Vorgehen konnten 103 Themenschwerpunkte extrahiert werden. Da einige „Special Issues“ über die Jahre verteilt mehrmals mit sehr ähnlichen oder gleichen Themenschwerpunkten auftraten, wurden viele Begriffe zusammengefasst. Mit insgesamt sechs Vorkommen ist zum Beispiel der Themenschwerpunkt „Hardware/Software Co-Design“ häufig vertreten.

Es gibt eine Vielzahl an Themenschwerpunkten, die trotz Design- oder Architekturbezug unpassend oder zu unkonkret waren. Beispielhaft wird der Themenschwerpunkt *Embedded System-Architecture* betrachtet. Er ist in der Ausschreibung für die „Special Issues on Real Time, Embedded and Cyber-Physical Systems“ enthalten. Auch durch die Fließtextbeschreibung zur Aufzählung der Themenschwerpunkte war keine Konkretisierung des Begriffes möglich. Der Interpretationsspielraum reicht von der Architektur- über die Logik- bis hin zur Layout-Ebene. Eine Untersuchung der Idee mit dem Kriterienkatalog würde somit zu teilweise völlig unterschiedlichen Ergebnissen führen, je nachdem welche Interpretation des Begriffes man zugrunde legt. In Abschnitt 2.4 wurde für die Autoren Zendler und Spannagel der gleiche Kritikpunkt erörtert.

Andere Gründe für die Entfernung aus der Ideensammlung sind fälschliche Erfassungen aufgrund von Mehrdeutigkeiten im Begriff Architektur oder Design.

Nr.	Themenschwerpunkte
1	Berechnungsmodelle (Models of Computation)
2	Synthese und Steuerung paralleler Systeme
3	Controller-Synthese
4	Hardware/Software Co-Design
5	Platform-based Design
6	Component-based Design und Intellectual Property
7	Synchrone and asynchrone Konzepte
8	Ressourcen-Management
9	Fehlertoleranz und Quality-of-Service
10	Betriebssysteme, Middlewares, Laufzeitumgebungen und Echtzeit-Kernel
11	Virtualisierungstechniken
12	Design-Space-Exploration
13	Reliable Design
14	Virtuelles Prototyping
15	Domänenspezifische Anwendungen und Methoden
16	Cybersecurity
17	Methoden der formalen Verifikation und Validierung
18	Modellbasierte Entwicklung
19	Probabilistische Software-Entwicklung und Werkzeuge
20	System-On-Chip Design (MPSoC und NoCs)
21	Rekonfigurierbare Architekturen und reprogrammierbare Designs
22	Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen

Tabelle 5.1.: Sammlung der Themenschwerpunkte aus den TECs

Der Themenschwerpunkt „Router Microarchitecture“ aus „On-Chip and Off-Chip Network-Architectures“ befasst sich so zwar mit Architektur, aber nicht auf dem Abstraktionsniveau der Systemebene. Hierbei geht es um die hardwarenahe Umsetzung einer „Instruction Set Architecture“ (ISA).

Die Liste aller analysierten Themenschwerpunkte ist in Tabelle 5.1 dargestellt.

5.5. Analyse am Beispiel rekonfigurierbarer Architekturen

Der folgende Abschnitt stellt die beispielhafte Analyse einer Idee aus der im vorherigen Abschnitt beschriebenen Liste an Methoden und Sichtweisen dar. Der Themenbereich, der analysiert wird, ist „Rekonfigurierbare Architekturen und reprogrammierbare Designs“ (Nr. 21). Nach einer kurzen Einleitung, um den Rahmen der Idee zu definieren, werden die einzelnen Kriterienanalysen durchgeführt. Dabei steht hinter dem jeweiligen Kriteriennamen das Ergebnis der Analyse auf der in Abschnitt 5.3 beschriebenen dreistufigen Skala.

Tessier und Burleson haben 2001 eine kompakte Definition des Begriffes „Rekonfigurierbarkeit“ gegeben, die im Folgenden als Grundlage der Analyse angesehen wird:

> „In the context of re-configurable computing this term indicates that the logic functionality and interconnect of a computing system or device can be customized to suit a specific application through post-fabrication, user-defined programming.“ [TB01, S. 2]

Für Hardwarekomponenten ist die technische Umsetzung größtenteils in „Field Programmable Gate Arrays“ (FPGAs) zu finden [TB01]. Hauck weist jedoch darauf hin, dass FPGAs nicht synonym zur *Idee* einer rekonfigurierbaren Architektur verwendet werden sollten, sondern eine mögliche technische Umsetzung des Konzeptes darstellen [Hau98].

Unterschieden werden Rekonfiguration vor und während der Laufzeit des Systems. Rekonfiguration stellt damit die Basis des Konzeptes „Updates“ dar. Statt dem Begriff „Rekonfiguration“ wird in einigen Publikationen auch von „Reprogrammierung“ gesprochen [But95], [Ada01], [Hau98]. Aus diesem Grund wird der Term „reprogrammable Design“ synonym verwendet. Obwohl die Idee einen starken Bezug zur Hardwareentwicklung besitzt, soll zumindest erwähnt werden, dass die Idee auf Softwareebene ebenso bekannt und genutzt ist. Die technische Umsetzung zwischen Hardware und Software ist dabei grundverschieden. In Softwaresystemen lassen sich Module mit neuer Funktionalität an definierte Schnittstellen, sogenannte *„Application-specific Interfaces“* (APIs) anschließen. Eine weitere Möglichkeit, Updates für Softwaresysteme bereitzustellen, liegt im einfachen Austausch von Skripten oder Binärkompilaten. Ist der aktuelle Stand der Applikation bekannt, können auch differentielle Updates eingespielt werden, die nur einen Teil des Softwaresystems durch bereits für die Rechnerarchitektur kompilierte Anweisungen austauschen. Für traditionell entwickelte Hardwarekomponenten gibt es keine Möglichkeit, nachträglich Änderungen einzupflegen, die für eine Massenproduktion rentabel wären.

Auch wenn Rekonfiguration auf Softwareebene ebenfalls beobachtbar ist, widmet sich die folgende Analyse hauptsächlich der Hardwareebene.

Horizontalkriterium (3) Rekonfigurierbare Designs beeinflussen in vielfacher Weise die Entwicklung eingebetteter Systeme. Die häufig genutzte technische Umsetzung der Field Programmable Gate Arrays (FPGAs) ist als Prototypeninstrument sehr verbreitet (siehe [HD10]). Neben der eigentlichen Implementierungsphase, in der die Umsetzung eines Systemdesigns mit Beschreibungssprachen wie VHDL synthetisiert wird, ist deshalb auch die Phase der Validierung von der Idee betroffen. Die Möglichkeit, Systemcharakteristika wie Leistungsaufnahme oder Performanz durch einen Prototypen abzuschätzen, zeigt, wie die Idee rekonfigurierbarer Architekturen auch den Entwurf des Systems auf der Ebene des „Architectural Design Processes“ beeinflussen kann. Erst wenn der erstellte Prototyp zufriedenstellend umgesetzt ist, wird mit der Portierung auf die eigentliche, nicht-rekonfigurierbare Hardware begonnen. In manchen Fällen fällt der letzte Schritt insofern weg, dass ein FPGA

bereits als Zielplattform dient. Bei solchen Szenarien wird die Idee also nicht zum Prototyping, sondern für die direkte Produktentwicklung genutzt.

Die besondere Relevanz von rekonfigurierbaren Architekturen zum Erstellen eines zeitigen Systemprototypens lässt sich prinzipiell auf fast alle Entwicklungsprojekte eingebetteter Systeme übertragen. Die einzige Einschränkung in dieser Hinsicht sind die auf der rekonfigurierbaren Architektur verfügbaren Logikzellen. Bei sehr großen Projekten sind diese eine gegenüber dem ASIC-Entwurf limitierender Faktor. Der Großteil der Projekte, die im Anwendungsgebiet der eingebetteten Systeme liegen, wird davon jedoch nicht betroffen sein, sodass die Idee auch den zweiten Aspekt des Horizontalkriteriums erfüllt.

Fortbildungskriterium (2) Rekonfigurierbare Architekturen können in der Lehre eine Doppelrolle einnehmen. Wie bereits erläutert, sind sie zum einen eine relevante, ebenenübergreifende Technik für das Anwendungsgebiet und zum anderen lassen sich durch rekonfigurierbare Plattformen die Grundlagen von Digitalsystemen anschaulich darstellen. Die Fertigung eines integrierten Schaltkreises (ASIC) ist üblicherweise mit der Erstellung von Fertigungsmasken und dem anschließenden Beleuchten des Wafers mit ultraviolettem Licht in einem Reinraum verbunden. Die Kosten für die benötigten Werkzeuge und qualifiziertes Personal sind so hoch, dass sich nur wenige Lehrstühle die Fertigung leisten können. Zusätzlich ist der Zeitaufwand für die Fertigung nur schwer in eine reguläre Vorlesung oder ein Praktikum zu integrieren. Auf dieser Ebene durchführbar sind hingegen diskrete Aufbauten aus fertigen Komponenten oder die Verwendung von *printed circuit boards* (PCBs). Zu Übungszwecken ist die Nutzung von Hardwarebeschreibungssprachen jedoch besonders praktikabel. Demnach profitieren auch Lehrende, die keinen Fokus auf rekonfigurierbare Hardware setzen, aber sich in ihren Lehrveranstaltungen mit dem Entwurf oder mit der Implementierung von digitalen Systemen beschäftigen, von den Möglichkeiten der FPGA-Entwicklung. Eine weitere Synergie ergibt sich durch die Lehre von Hardwarebeschreibungssprachen (HDLs). HDLs sind Programmiersprachen aus der Softwaretechnik funktional ähnlich [KB09] und werden nicht nur für die Konfiguration von rekonfigurierbaren Architekturen, sondern auch für die Beschreibung des Hardwareaufbaus und des Verhaltens von ASICs verwendet. Ein ASIC ist ein integrierter Schaltkreis, der für eine bestimmte Aufgabe optimiert wurde, beispielsweise für eine Satellitensteuerung [Smi08]. Der große Unterschied zu rekonfigurierbaren Architekturen, der für die Betrachtung in dieser Arbeit relevant ist, ist, dass ASICs nach der Fertigung nicht mehr strukturell verändert werden können.

Die Abbildung 5.2 zeigt ein typisches Entwicklungsvorgehen aus Sicht einer FPGA-basierten und einer ASIC-basierten Hardwareumsetzung. Dabei ist ersichtlich, dass die Entwicklungsschritte bis zur Synthesephase nahezu identisch sind. Werden die bis dahin relevanten Kompetenzen vermittelt, können sie demnach in zweifacher Hinsicht verwendet werden.

Wichtige Voraussetzungen für die Vermittlung entsprechender Kompetenzen sind Grundlagen der Rechnerarchitektur und Grundlagen digitaler Schaltungen. Erste Erfahrungen in Programmiersprachen wie C oder C++ sind für die Umsetzung von

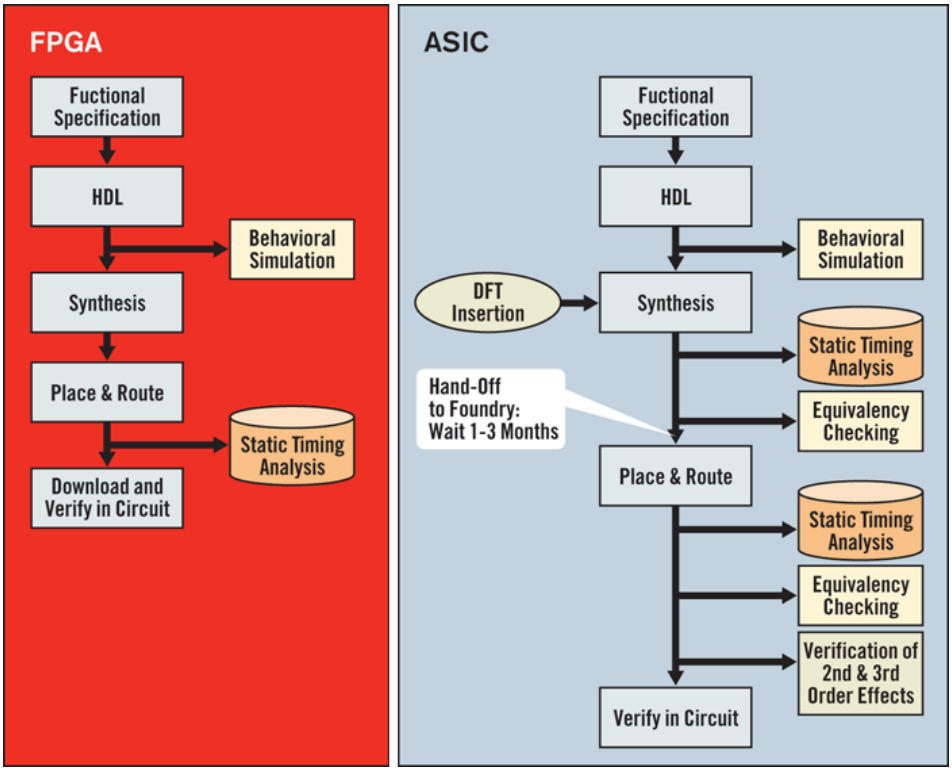

Abbildung 5.2.: Unterschiedliche Vorgehen bei der FPGA- und ASIC-Entwicklung, nach [Xil14]

HDLs empfehlenswert, da einige Konzepte wie Namensräume übernommen werden können. Kenntnisse domänenspezifischer Sprachen sind ebenso hilfreich, aufgrund der meist anwendungsfallspezifischen Ausrichtung aber nur begrenzt auf andere Sprachen übertragbar. Alternativ kann die Entwicklung anstatt mit HDLs teilweise auch auf MoCs verlagert werden, die sich, beispielsweise in Form von Automaten oder Zustandsdiagrammen, ebenso zur Beschreibung von Komponentenverhalten eignen und teilweise sogar synthetisierbar sind (auch in VHDL). Allerdings ist darauf hinzuweisen, dass VHDL verschiedene Berechnungsmodelle unterstützt, die sich nicht mit endlichen Automaten abbilden lassen. Hierzu zählt zum Beispiel die Spezifikation der Schnittstellen, also im weitesten Sinne Eingabe- und Ausgabesignale.

Aufgrund seiner Relevanz in der Hochschullehre ist das Themengebiet fachdidaktisch vergleichsweise gut erforscht (bspw. [SS05], [DeH+04]). Viele Universitäten und Fachhochschulen in Deutschland lehren die mit dem Themenkomplex verbundenen Inhalte allerdings auf Masterniveau. Die Universitäten Bielefeld, Kassel und Karlsruhe bieten je eine Lehrveranstaltung mit Fokus auf Masterstudierende an [Zip13], [Por13], [HB13].

Ebenso gibt es Universitäten wie Erlangen und Frankfurt, die das Themenge-

biet nicht als eigenständiges Modul, sondern im Rahmen der Vorlesung über Rechnerarchitekturen lehren. Beide Veranstaltungen lassen sich dem Modulhandbuch für Bachelorstudierende zuordnen [Wal14], [Fey13]. Das im Rahmen des KOMINA-Projektes entworfene „Entwurfs- und Anwendungspraktikum für eingebettete Mikrosysteme“ (EAP) hat gezeigt, dass grundsätzliche Entwicklungen mit rekonfigurierbaren Plattformen auch auf Bachelorniveau möglich sind [Jas+12]. Fortgeschrittene Themengebiete wie die dynamische Rekonfiguration zur Laufzeit sind der Komplexität wegen jedoch nicht genutzt worden.

Da die aufgezeigten Quellen keine eindeutige Einordnung auf Bachelor- oder Masterniveau zulassen, ist eine Kompetenzniveaustufung in Anlehnung an den Qualifikationsrahmen Deutscher Hochschulen (QDH) ratsam. Für Bachelorstudierende werden Lerninhalte empfohlen, mit denen sich die Kompetenz- bzw. Prozessdimensionen *Strukturieren, Erarbeiten* und *Anwenden* vermitteln lassen. Somit sind Grundlagen der Hardwarebeschreibung mittels HDLs auch für Bachelorstudierende geeignet, um zum Beispiel die bereits erwähnten Grundlagen digitaler Systeme anzuwenden [HD10]. Für fortgeschrittene Anwendungsszenarien wie beispielsweise der dynamischen Rekonfiguration zur Laufzeit müssen Studierende über ein umfängliches Wissen aus dem gesamten Bachelorstudium verfügen, wie es die Zugangsvoraussetzungen der aufgezeigten Universitäten beschreiben.

Sofern im Rahmen einer Veranstaltung auf Bachelorniveau über das *Anwenden* hinausgehend Kompetenzen gefragt sind, ist auf eine starke Hilfestellung durch die Lehrenden zu achten. Letzteres kann beispielsweise in Form von bereits entwickelten HDL-Komponenten geschehen, welche die Studierenden nutzen, ohne deren technische Umsetzung im Detail zu kennen. Dieses Konzept ist auch als *Didaktische Reduktion* bekannt [RS96]. Im Entwurfs- und Anwendungspraktikum wurde ein solches Rahmenwerk beispielsweise in Form von vorgefertigten Hardwarebeschreibungen vorgegeben, um die Studierenden bei der Konzeption und Entwicklung eigener Komponenten zu unterstützen. Die Prozessdimension „Entwickeln“ findet sich im Deutschen Qualitätsrahmen für die Hochschule erst auf Masterniveau.

Wie dargestellt, lassen sich nicht alle Anwendungsbezüge der Idee auf Bachelorniveau lehren. Eine Niveaustufung ist deswegen ratsam. Das Fortbildungskriterium erhält im Analyseprozess deswegen jedoch nicht mehr die höchstmögliche Wertung, sondern kann nur mit Einschränkungen (der Niveaustufung) positiv evaluiert werden. Da diese Wertung in der dreistufigen Skala der Stufe zwei entspricht, können rekonfigurierbare Architekturen und reprogrammierbare Designs trotzdem noch eine fundamentale Idee darstellen.

Zeitkriterium (3) Rekonfigurierbare Architekturen werden seit 1980, das Konzept an sich seit 1960, wissenschaftlich diskutiert [Hau98], [CH00]. Heute sind FPGAs die am häufigsten genutzte Technologie für die Umsetzung rekonfigurierbarer Architekturen [HD10]. Obwohl alternative Ansätze bestehen, hat sich bisher keines dieser Konzepte durchgesetzt. Guccione vermutet, dass dieser Umstand der schlechten Verfügbarkeit und Qualität der zugehörigen Entwicklungsumgebungen geschuldet ist [Guc10]. Durch das Einsatzgebiet des High-Performance-Computing scheinen neue Anwendungsgebiete erschlossen zu werden. Gleichzeitig zeigt die

UBM-Entwicklerumfrage von 2013 einen Abwährtstrend in der Nutzung von FPGAs in aktuellen Projekten [Tec13]. Neben der nicht benötigten Funktionalität stellen die Kosten von FPGAs im Vergleich zu mikroprozessbasierten Lösungen den Hauptgrund für diesen Trend dar [Tec13]. Die Gründe spiegeln also eine nachteilige technische Umsetzung, nicht aber Nachteile der Idee selbst, wider. Butts bezeichnet rekonfigurierbare Systeme als „Turn of the Century"-Technologie, weist aber darauf hin, dass die reine Betrachtung von Kosten immer zu Gunsten einer nichtrekonfigurierbaren Systemlösung ausfallen wird [But95].

Einige Autoren gehen davon aus, dass die Beschreibung der Systemkonfiguration einen weiteren Abstraktionsschritt nach „oben" gehen wird und demnach zukünftige Systeme nicht mehr auf Registertransferebene, sondern auf der Ebene von Algorithmen beschrieben werden [CH00], [ABP11]. Am Beispiel von HDLs wie VHDL und Verilog sind diese Ansätze bereits recht fortgeschritten. Die Verhaltensbeschreibung einer Komponente kann prinzipiell ähnlich wie in klassischen Softwareprogrammiersprachen gestaltet werden, inklusive von Kontrollflussmechanismen wie Schleifen (*while, for*) oder Verzweigungen (*if, case*). Da für die Idee rekonfigurierbarer Architekturen und reprogrammierbarer Designs die Relevanz in der Vergangenheit und in der Zukunft gezeigt werden konnte, wird das Zeitkriterium erfüllt. Zusätzlich lässt sich feststellen, dass es einige Indikatoren dafür gibt, dass das Konzept auch in Zukunft eine wesentliche Rolle in der Entwicklung eingebetteter Systeme spielen wird. Das Zeitkriterium ist deswegen vollständig erfüllt.

Sinnkriterium (2) Rekonfigurierbare Architekturen sind das Mittel der Wahl, um frühe Hardware-Prototypen zu erstellen. Dies schließt auch die Arbeitsschritte ein, in denen versucht wird, Systemcharakteristika des finalen Systems abzuschätzen. Erst wenn Funktionalität und die Erfüllung der Rahmenbedingungen des Entwurfes zufriedenstellend sind, wird dieser auf nicht-rekonfigurierbare Hardware portiert. Von praktischer Bedeutung ist die Eigenschaft rekonfigurierbarer Systeme, mehr Funktionalität in einem System zu integrieren, als eigentlich aktuell benötigt wird. Je nach Anwendungskontext wechselt das System die Konfiguration der Hardwareblöcke und aktiviert die entsprechend notwendigen Funktionen. Um den Zeitaufwand für diese Rekonfiguration gering zu halten, haben sich unterschiedliche Ansätze zur Rekonfiguration etabliert. Hier sind Techniken wie die *partielle Rekonfiguration* oder die *Multikontext-Konfiguration* zu nennen [HD10]. Als Beispiel kann ein GSM-Modul dienen, welches nur die Hardwarekonfiguration aktiviert, die für den aktuellen Aufenthaltsort bzw. dessen Frequenzband erforderlich ist. Die Laufzeitrekonfiguration ermöglicht neben dem Beheben von Fehlern auch die dynamische Anpassung der Hardware an spontane Anforderungen der Umgebung und enthält damit ein signifikantes Optimierungspotential. In diesem Zusammenhang nennen Compton und Hauck Verschlüsselungs- oder Entschlüsselungskomponenten auf Basis eines rekonfigurierbaren Systems, das speziell für einen gegebenen Schlüssel optimiert wurde [CH02]. Trotzdem behält das System weiterhin die Funktionalität, auch andere Schlüssel zu nutzen, dann aber deutlich ineffizienter (bis zu einer weiteren Rekonfiguration). FPGAs werden zudem in Anwendungsbereichen eingesetzt, die in sehr großem Stil parallele Bearbeitung benötigen [Loc+07]. Weitere

Anwendungsbereiche, die im Kontext des Sinnkriteriums interessant sind, können in [Tod+05] nachgelesen werden.

Rekonfigurierbare Systeme haben zusammengefasst für einige Nischenanwendungen, insbesondere aber für das Prototyping eingebetteter Systeme, eine signifikante Bedeutung. In der zitierten UBM-Studie zeigt sich der Trend, dass FPGAs nicht mehr in dem Umfang für Projekte des Anwendungsbereiches genutzt werden, wie dies noch vor einigen Jahren der Fall war. Das Sinnkriterium wird aus diesem Grund mit Einschränkung erfüllt.

Varianzkriterium (3) Rekonfigurierbare Architekturen beinhalten viele Bezüge zu anderen Themengebieten. Zu nennen sind hier deswegen die Idee der Parallelität, Berechnungsmodelle bzw. „Models of Computation“ oder „Virtual Prototyping“. Die Verknüpfung zum Themengebiet der Parallelität findet sich in der Möglichkeit, die Struktur- oder Verhaltensbeschreibung mit nebenläufigen Komponenten und Prozessen zu gestalten. An dieser Stelle zeigt sich ebenso der Bezug zu Berechnungsmodellen. Dem Entwickler ist es überlassen, in welcher Weise er das System spezifiziert. Zum einen können die LUTs und Flip-Flops im FPGA direkt durch die Registertransferebene beschrieben werden, zum anderen aber auch durch eine abstraktere Verhaltensbeschreibung, die Konstrukte wie Schleifen oder Verzweigungen nutzt. Bei letzterer übernimmt das Synthesewerkzeug das Mapping auf die entsprechende Zielplattform. Verknüpfungen zu Middlewares und Betriebssystemen sind sichtbar, wenn man die Laufzeitumgebung, die zur dynamischen (Re-)Konfiguration von Hardwareverbindungen dient, als abstrakte Vermittlungsschicht auffasst. Die Laufzeitumgebung ist dann ein Hardware-/Softwaresystem, welches die Neustrukturierung des restlichen Systems nach externen Anforderungen hin durchführt. Um ein rekonfigurierbares System umzusetzen kann, neben Verilog und VHDL auch SystemC benutzt werden. SystemC ist keine eigene Sprache, sondern eine Programmbibliothek, die C unter anderem um Funktionen wie Ports und eine vierwertige Logik erweitert. Hardwareseitig können CPLDs (complex programmable logic devices) von FPGAs unterschieden werden. Allerdings ändert sich durch diese alternative Umsetzung nichts an der grundlegenden Idee der Rekonfiguration. Alle genannten Themengebiete und Konzepte sind als verwandt, aber nicht als synonym einzustufen. Der Aspekt der Rekonfiguration nach Beendigung des Fertigungsprozesses findet sich in keiner anderen analysierten Idee wieder und stellt, wie im Horizontal- und Sinnkriterium beschrieben, das zentrale Merkmal der Idee dar. Das Varianzkriterium ist deswegen erfüllt.

Da keines der Kriterien schlechter als zwei, also mit Einschränkung, bewertet wurde, handelt es sich bei der Idee „Rekonfigurierbare Architekturen und reprogrammierbare Designs“ um einen fundamentalen Lehr-/Lerninhalt.

5.6. Übersicht fundamentaler Ideen

Der Prozess der Ideenanalyse mit Hilfe der in Abschnitt 4.2.8 vorgestellten Kriterien bildet den Kern der Arbeit und ist mit der Erfüllung des Teilziels II gleichzusetzen.

Der folgende Abschnitt fasst die Ergebnisse der Ideenanalyse zusammen, um einerseits einen praktischen Leitfaden für die Auswahl von Lerninhalten bereitzustellen, und andererseits, um eine Evaluation der Kriterienanwendungen zu ermöglichen.

Alle analysierten Ideen wurden auf einer dreistufigen Skala bewertet (siehe Abschnitt 5.3). In Tabelle 5.2 ist für jede Idee die entsprechende Einschätzung hinsichtlich jedes Kriteriums aufgelistet. Alle Ideen, deren Analyse auf einen fundamentalen Lehr-/Lerninhalt für die Entwicklung eingebetteter Systeme hindeutet, sind farblich hervorgehoben.

Insgesamt erwiesen sich die zehn Ideen *Berechnungsmodelle, Synthese und Steuerung paralleler Systeme, Hardware/Software Co-Design, Component-based Design und Intellectual Property, Betriebssysteme (abgekürzt), Reliable Design , Domänenspezifische Anwendungen und Methoden, Methoden der formalen Verifikation und Validierung, Rekonfigurierbare Architekturen und reprogrammierbarer Designs* sowie *Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen* als fundamental.

Bei der Analyse hat sich gezeigt, dass nur *Models of Computation* allen Kriterien vollumfänglich gerecht werden konnte. Berechnungsmodelle sind deswegen jedoch nicht „fundamentaler“ als die restlichen akzeptierten Ideen. Besonders wichtige Ideen ordnete Schwill der Gruppe an Masterideen zu [SS11]. Laut Schwill sind Masterideen phasenübergreifende Ideen, die in allen Stadien der Entwicklung eine besondere Rolle spielen [SS11]. Schwill sieht die Idee der Algorithmisierung als eine von drei Masterideen zu anderen Ideen aus den Bereichen der Entwurfsparadigmen, der Programmierkonzepte, der Ablaufsteuerung und der Evaluation. Schwills Beschreibung einer Masteridee kann für das in dieser Arbeit angewandte Konzept nicht übernommen werden, da bereits das Horizontalkriterium Aspekte dieser „phasenübergreifenden“ Eigenschaft besitzt. Damit wurde Modrows Kritik Rechnung getragen, dass zu viele Konzepte und Methoden im Ideenkatalog von Schwill eingetragen sind. Masterideen im Rahmen dieser Arbeit sind Ideen, die fachspezifische fundamentale Ideen funktional auf einer abstrakten Ebene bündeln. Beispielhaft ist das Themengebiet rund um nicht-funktionale Anforderungen eher prädestiniert eine Masteridee darzustellen als die Synthese und Steuerung paralleler Systeme, auch wenn letztere die Kriterien umfänglicher erfüllen konnte. Dieses Beispiel zeigt, dass nicht direkt aus den Analysewertungen geschlossen werden kann, welche Idee für eine Masteridee prädestiniert ist.

Generell wird die Gliederung in normale und Masterideen in dieser Arbeit vermieden, da der Autor das Konzept hinsichtlich einer Umsetzung in einem Lehr-/Lernkonzept erforscht. Für diese Aufgabe spielt die Benennung von Masterideen keine Rolle, da in Kapitel 6 ein Ansatz vorgestellt wird, mit dem sich jede mit der angewandten Methode analysierte Idee in ein Lehrkonzept integrieren lässt. Ideen mit fundamentalen Eigenschaften sind aufgrund ihrer vielschichtigen Bezüge zu anderen Themengebieten jedoch einfacher umzusetzen. Prinzipiell ist der Ansatz auch für Ideen geeignet, die nicht jedes Kriterium erfüllen können.

Idee	Horizontal	Fortbildung	Zeit	Sinn	Varianz
1. Berechnungsmodelle (Models of Computation)	3	3	3	3	3
2. Synthese und Steuerung paralleler Systeme	2	3	3	3	3
3. Controller-Synthese	2	3	3	2	1
4. Hardware/Software Co-Design	3	3	3	3	2
5. Platform-based Design	2	1	3	2	2
6. Component-based Design und Intellectual Property	3	3	2	3	3
7. Synchrone and asynchrone Konzepte	2	3	3	3	1
8. Ressourcen-Management	3	1	1	3	2
9. Fehlertoleranz und Quality of Service	3	1	3	3	1
10. Betriebssysteme, Middlewares, Laufzeitumgebungen und Echtzeit-Kernel	3	3	3	3	2
11. Virtualisierungstechniken	1	1	1	2	1
12. Design-Space-Exploration	3	1	3	3	2
13. Reliable Design	3	3	3	3	2
14. Virtuelles Prototyping	3	1	2	3	2
15. Domänenspezifische Anwendungen und Methoden	2	3	3	2	2
16. Cybersecurity	2	2	3	1	2
17. Methoden der formalen Verifikation und Validierung	3	3	3	3	2
18. Modellbasierte Entwicklung	2	1	1	1	1
19. Probabilistische Software-Entwicklung und Werkzeuge	1	2	3	1	2
20. System-On-Chip Design (MPSoC und NoCs)	2	1	2	3	2
21. Rekonfigurierbare Architekturen und reprogrammierbare Designs	3	2	3	2	2
22. Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen	3	2	3	3	3

Tabelle 5.2.: Analyseergebnisse der kriterienbasierten Ideenprüfung

5.7. Beurteilung der Kriterien und Bemerkungen zur Analyse

Wie in den Forschungszielen erläutert (siehe Kapitel 1.1), besitzt die vorliegende Arbeit mehrere Teilziele. Es soll ein Katalog an fundamentalen Ideen der Entwicklung eingebetteter Systeme erstellt werden. Durch die Analyse der TECs-Themenschwerpunkte und der in diesem Abschnitt dargestellten Auswertung ist dieses Teilziel erfüllt. Der Prozess zur Ermittlung fundamentaler Lehr-/Lerninhalte ist für zukünftige Arbeiten jedoch ebenso wichtig, um eine exemplarische Vorlage zur Adaptionen in anderen Anwendungsgebieten zu besitzen. Rückblickend ist deswegen interessant, wie geeignet die fünf Kriterien und das angesetzte Vorgehen im Analyseprozess waren. Eventuelle Konzeptionsfehler können aufgedeckt und in zukünftigen Forschungsvorhaben vermieden werden. Die möglichen Fehlerquellen sind dabei ein zu starkes Abstraktionsniveau (siehe Abschnitt 4.3) oder eine zu stark einschränkende Definition und Bewertung der fünf Kriterien.

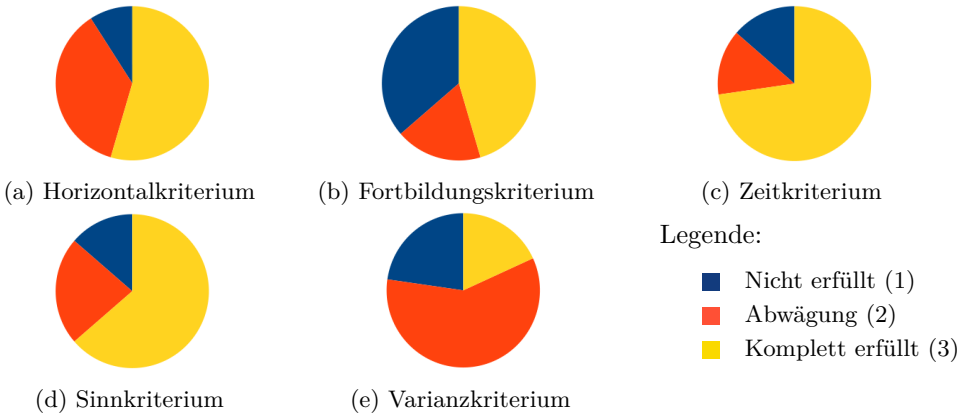

Abbildung 5.3.: Analysewertung der fünf Kriterien

Die Häufigkeit der Analysewertungen aller Kriterien ist in Abbildung 5.3 dargestellt.

Das Horizontalkriterium wurde mit Hinblick auf den interdisziplinären Charakter des Anwendungsbereiches definiert und ist deswegen nicht nur auf den Bereich der Informatik bezogen. Diese Einteilung war sinnvoll, da ein Teil der analysierten Ideen anderen Anwendungsfeldern entspringen. Im dargestellten Katalog sind dies beispielsweise die Themenbereiche „Controller-Synthese“ mit ihrem Ursprung im Control-Engineering, dem „Reliable-Design“ als Inhalt des System-Engineering und der Design-Automation, oder dem Bereich der „Cybersecurity“, die aus der Informationssicherheit stammt. Die Idee des „Virtual Prototyping“ kann nach der in der Analyse verwendeten Definition von Wang der Mechatronik zugeordnet werden [Wan02]. Eine striktere Definition des Horizontalkriteriums hätte diese Methoden und Sichtweisen ausgeschlossen und zu einer kleineren Menge an Ideenverbindungen geführt. Das Horizontalkriterium hat keine untersuchte Idee abgelehnt,

die mit einer besseren Bewertung als fundamental angenommen worden wäre. Wie in Abbildung 5.3a zu sehen, wurde das Kriterium in den meisten Fällen vollständig erfüllt.

Beim Fortbildungskriterium genügt ebenfalls ein Großteil der Ideen beiden in der Definition erläuterten Aspekten (siehe Abbildung 5.3b). Allerdings ist das Kriterium das restriktivste aller Kriterien, auch wenn bei den besonders schlecht bewerteten Analysen oft andere Kriterien ebenfalls negativ ausgewertet wurden. Bei den Themengebieten „Platform-based Design“, „Design-Space Exploration“, „Virtual Prototyping“ und „System-on-Chips“ ist es hingegen der einzige Grund für die Ablehnung als fundamentale Idee.

Die Analyse des Zeitkriteriums zeigt, dass trotz der fest gewählten Zeitspanne von 20 Jahren keine große Abweichung zu den anderen Kriterien hinsichtlich der Ablehnquote besteht (siehe Abbildung 5.3c). Rückblickend ist zu überlegen, ob die im Zeitkriterium definierte Zeitspanne für technische Disziplinen nicht kürzer angesetzt werden kann. Einige der analysierten Ideen sind nur knapp positiv evaluiert worden, da sie die 20-Jahre-Grenze gerade so erfüllten (zum Beispiel „System-On-Chip Design“ oder die modellbasierte Entwicklung). Das Zeitkriterium hat keine Idee abgelehnt, die bei einer besseren Bewertung fundamental gewesen wäre.

Bei der Untersuchung der Analyseergebnisse des Sinnkriteriums fällt auf, dass das Kreisdiagramm sehr ähnlich zum Horizontalkriterium ist (siehe Abbildung 5.3d). Wie auch beim Horizontalkriterium erfüllt ein Großteil der Ideen die geforderte praktische und theoretische Relevanz für das Anwendungsgebiet. Lediglich bei der Idee zu „Cybersecurity“ hat das Sinnkriterium eine Idee abgelehnt, die ansonsten einen fundamentalen Lehr-/Lerninhalt dargestellt hätte.

Die in Abbildung 5.3e illustrierte Auswertung des Varianzkriteriums zeigt, dass sich insbesondere die Anzahl der komplett erfüllten und der nur teileweise erfüllten Analysen zu allen anderen Kriterienauswertungen unterscheidet. Mit Hinblick auf die Definition des Kriteriums (siehe Abschnitt 4.2.7) und die Anwendung der dreistufigen Bewertung (siehe Abschnitt 5.7) zeigt sich, dass viele Ideen eine neue Facette gegenüber anderen Ideen beinhalten oder diese auf neuartige Weise miteinander verbinden. Die wenigsten Ideen sind also komplett neu. Dieses Ergebnis ist verständlich und zeigt die starke Verknüpfung zwischen den untersuchten Themengebieten.

Diese Themenverbindung kann ebenfalls grafisch dargestellt werden und ist in Abbildung 5.4 zu sehen. Aus Platzgründen wurden alle Bezeichnungen abgekürzt.

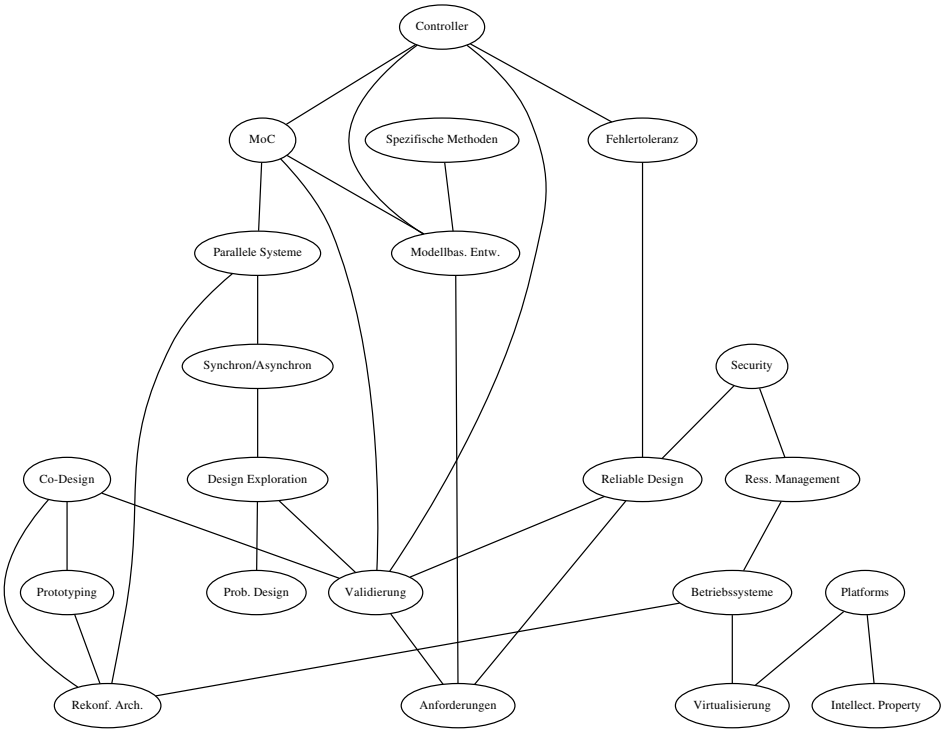

Abbildung 5.4.: Bezüge zwischen den Ideen; aufgedeckt durch das Varianzkriterium

6. Ableitung eines Lehrkonzeptes aus den Kriteriendefinitionen

Das Ziel dieses Kapitels ist die Konzeption einer auf den fundamentalen Ideen basierenden Lehrveranstaltung. Dies beinhaltet die Erarbeitung der übergeordneten fachdidaktischen Organisation der Lehrveranstaltung, den Aufbau der Lehrveranstaltung sowie der Ausformulierung der einzelnen Veranstaltungseinheiten mit ihren zugehörigen Lernzielen. Die Konzeption der Lehrveranstaltung ist, im Gegensatz zum Rest der Arbeit, an den Curricula und Organisationsstrukturen der Universitiät Siegen ausgerichtet, um beispielhaft zu zeigen, wie sich eine an der Forschungsmethodik orientierte Lehrveranstaltungskonzeption in ein bestehendes Hochschulsystem einbetten lässt. Aus Gründen der Nachvollziehbarkeit orientiert sich die Konzeption der Lehrveranstaltung am bereits in Abschnitt 5.5 vorgestellten Themengebiet „rekonfigurierbarer Architekturen und reprogrammierbarer Designs“.

6.1. Erfassung des fachdidaktischen Lehrveranstaltungsaufbaus

Bei der Umsetzung einer Lehrveranstaltung kann das Lehrpersonal inzwischen aus einer großen Sammlung an fachdidaktischen Ansätzen auswählen. Obwohl die Erforschung vieler Methoden sinnvoll ist, muss die Lehrperson nun ein, zumindest grobgranulares, Verständnis der meisten dieser Konzepte besitzen, um einen geeigneten Ansatz. Die Aufzählung und Analyse aller möglichen Umsetzungen wird in diesem Abschnitt nicht diskutiert, sondern nur die Konzepte eingehend betrachtet, die als fachdidaktische Leitlinie für die entworfene Lehrveranstaltung in Frage kommen.

Das so genannte *Constructive Alignment* ist eine didaktische Methode, um die wichtigen Bereiche Lernergebnisse, Lehr-/Lernmethoden und Prüfungsmethoden zu verbinden. Biggs hat die auf dem Konstruktivismus basierende und speziell für die Hochschullehre ausgerichtete Methode 1996 vorgestellt [Big96]. Ausgehend von einem portfoliobasierten Ansatz, in dem die Studierenden erläutern, was sie in der Veranstaltung gelernt haben, fällt Biggs auf, dass Studierende ihre Lernanstrengungen nicht zwangsläufig an den Themen ausrichten, die den Inhalt der Lehrveranstaltung darstellen, sondern den der Prüfung beschreiben. Diese unterschiedliche Wahrnehmung von Lehr-/Lernzielen seitens der Studierenden und der Lehrperson hat weitreichenden Einfluss auf die Durchführung der Veranstaltung und der anschließenden Prüfung der Studierenden. Das *Constructive Alignment* ist

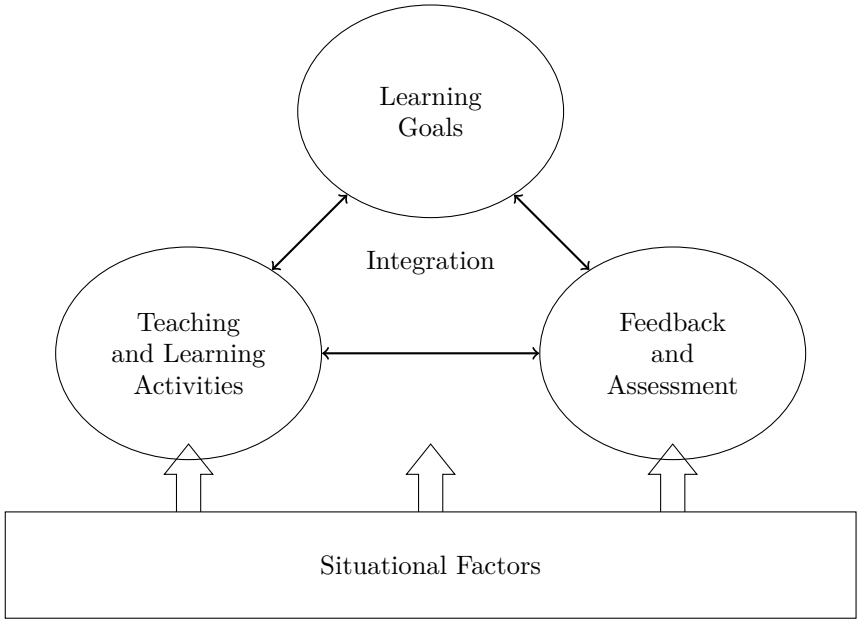

Abbildung 6.1.: Abhängigkeiten zwischen den Aspekten einer integrierten Lehrveranstaltung [Fin13]

eine Systembeschreibung kompetenzorientierter Lehrveranstaltungen, bei denen sich Relationen zwischen den drei dargestellten Aspekten (Lernergebnisse, Lehr-/Lernmethoden und Prüfungsmethoden) herstellen lassen [Bra08], [BT10]. Die Lernergebnisse sind die Kompetenzen, die die Studierenden nach dem erfolgreichen Absolvieren der Lehrveranstaltung besitzen. Lehr-/Lernmethoden beschreiben die Umsetzung der Lehrveranstaltung (bspw. Gruppenarbeit, Vortrag, projektbasiertes Lernen, etc.) und die Prüfungsmethoden geben an, welche Art von Aufgabe an die Studierenden gestellt wird, um den Erfolg der Kompetenzaneignung zu bestimmen. Fink hat die Berücksichtigung dieser Abhängigkeiten als „integrierte Lehrveranstaltung“ bezeichnet [Fin13].

Diese, zuerst trivial anmutende Unterteilung, zeichnet sich durch starke Verbindungen aus, die sich in Form eines Dreiecks darstellen lassen und auf verschiedene Lehrszenarien angewendet werden können (siehe Abbildung 6.1).

Die Abbildung ist folgendermaßen zu verstehen: Ist eine Verbindung unterbrochen, bzw. nicht abgestimmt, können nur noch zwei der drei Aspekte „integriert“ umgesetzt werden. Sind bspw. die Prüfungsmethoden nicht auf den intendierten Kompetenzgewinn abgestimmt, wird eine Überprüfung trotz geeigneten Lehr-/Lernmethoden kein objektives Ergebnis über den Lernerfolg ergeben, da die Kompetenzen, die geprüft werden, nicht die Kompetenzen sind, die in der Lehrveranstaltung vermittelt werden [Fin13]. Die Studierenden orientieren sich für die nächste Lehrveranstaltungsdurchführung dann nicht mehr an den aufgestellten Lernzielen, sondern an den Prüfungsaufgaben, was wiederum die Lehr-/Lernmethoden

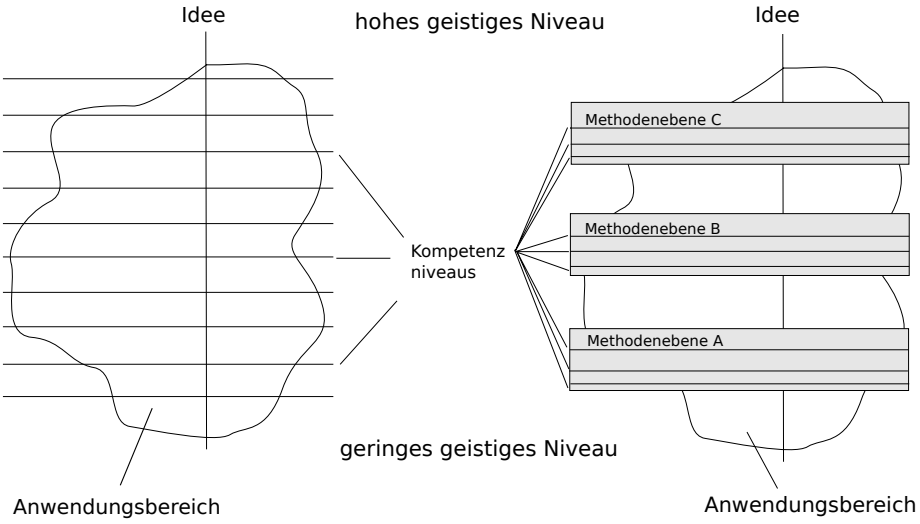

Abbildung 6.2.: Vertikale Gliederung des Lernfortschrittes durch Methodenebenen

beeinträchtigt, da diese von den Studierenden nicht mehr als relevant für ihre Zielstellung angesehen werden.

Die intendierten Kompetenzgewinne können besonders gut mit dem Konzept der fundamentalen Ideen verbunden werden, da beide eine implizite Niveaustufung darstellen. Biggs weißt darauf hin, dass die Kompetenzen eine Niveaustufung beinhalten können, die insofern mit den Prüfungsmethoden verbunden ist, als dass sie die Bewertungsskala vorgibt [BT10]. Die Studierenden wissen demnach im Vorhinein, dass bspw. die Prozessdimension *Aufzählen* im Anwendungskontext immer niedriger bewertet wird als die Prozessdimension *Entwerfen*. In diesem Zusammenhang kann von deklarativem und funktionalem bzw. prozeduralem Wissen gesprochen werden (vgl. [McC97]).

Beide Ansätze lassen sich zusammenführen. Ausgangspunkt für die folgende Erläuterung bildet die Veranschaulichung von Schwill zum Vertikalkriterium, die auf der linken Seite der Abbildung 6.2 zu sehen ist. Dabei strukturiert Schwill die verschiedenen geistigen Niveaus einer Idee hinsichtlich deren Anwendungsgebiet. Die rechte Seite der Abbildung zeigt eine Verfeinerung des Ansatzes, in der die Ebenen des geistigen Niveaus nicht fließend, sondern schrittweise erreicht werden. Dabei ist die Ebene der fundamentalen Ideen, wie auch bei Schwill, durch die vertikale Achse dargestellt. Von besonderer Bedeutung sind hierbei die Methodenebenen, welche die Wiederholung eines Ideenaspektes im Rahmen einer Lehrveranstaltung darstellen. Die Gliederung des geistigen Niveaus ist dann an die Kompetenzniveaus der jeweiligen Methodenebenen gebunden. Aus diesem Grund ist die feingliedrige

Stufung nur in den Methodenebenen aufgeführt. Der Abstand zwischen zwei dieser Ebenen sollte möglichst gering sein, um einen einfachen Übergang von einer Niveaustufe zur nächsten zu ermöglichen. Somit bilden die Methodenebenen mit ihren Kompetenzzielen die Stufen zum Erreichen eines höheren geistigen Niveaus hinsichtlich einer Idee. Für die Niveaustufung innerhalb der Methodenebenen kann bspw. die Taxonomie von Anderson und Krathwohl verwendet werden. Hier bildet sich der Zusammenhang der Niveaustufung zwischen einer fundamentalen Idee und der Niveaustufung des Constructive Alignments auf Veranstaltungsebene. Somit kann auch der Wechsel der Kompetenzansprüche zwischen dem Bachelor- und Masterniveau gut nachvollzogen werden. Wird eine fundamentale Idee zum ersten Mal in einer Lehrveranstaltung behandelt (Bachelorniveau) ist ein Großteil des Kompetenzniveaus auf der Ebene des deklarativen Wissenserwerbs anzusetzen, um Grundlagen für die Vermittlung weiterer Inhalte zu schaffen. Wie auch aus dem Qualitätsrahmen für Deutsche Hochschulen ersichtlich, verlagert sich dieser Aspekt im Masterstudium zugunsten des funktionalen Wissen und den Prozessdimensionen *Integrieren, Problemlösen* und *Entwickeln* (siehe Abschnitt 4.2.2).

Um dies zu verdeutlichen, stellt die Aufzählung 6.3 drei Methodenebenen dar, die nicht nur verschiedene Kompetenzniveaus, sondern auch unterschiedliche Themengebiete abdecken. Methodenebene A kann stellvertretend für den Anfang des Studiums auf Bachelorniveau und Methodenebene B für das Ende des Bachelorstudiums bzw. den Anfang des Masterstudiums gesehen werden. Die letzte Methodenebene würde schließlich das Ende des Masterstudiums und damit den Eintritt in die industrielle oder wissenschaftliche Laufbahn bedeuten.

Die fundamentale Idee, die in allen Methodenebenen den sie verknüpfenden Aspekt darstellt, sind Berechnungsmodelle. In der Aufzählung der Kompetenzniveaus zu den Methodenebenen wurde auf eine entsprechende Verteilung von deklarativen und funktionalen Kompetenzbeschreibungen im zuvor geschilderten Sinne geachtet. Dies bedeutet selbstverständlich nicht, dass es in der Methodenebene A keine funktionalen Prozessdimensionen gibt, sondern dass diese im Vergleich zu späteren Studienabschnitten deutlich reduziert sind.

6.2. Herleitung der Lehrveranstaltungsgliederung

Für die Lehrveranstaltung an der Universität Siegen wird vorausgesetzt, dass die Grundkurse in Informatik bzw. Elektrotechnik bereits erfolgreich absolviert wurden. Dementsprechend befinden sich die Studierenden ungefähr in der Mitte ihres Bachelorstudiums.

Das Konzept der fundamentalen Ideen wurde bisher auf der Curriculumsebene angewandt. Durch die Formulierung entsprechend feingranularer Kompetenzbeschreibungen lässt sich auch eine Anwendung auf der Ebene einer Lehrveranstaltung sinnvoll umsetzen. Der Kern der Überlegung ist dabei, dass ein ausgewogenes Verhältnis zwischen Tiefen- und Breitenwissen gefördert wird, indem die fünf Kriterien die Dimensionen vorgeben, in denen der Lerngegenstand betrachtet werden soll. Dadurch kann der von Sifakis und Caspi geäußerten Kritik, dass zu viele Spezialisten, aber

Methodenebene A:	**Methodenebene B:**	**Methodenebene C:**
• Studierende können erläutern, warum Berechnungsmodelle als ein Fundament der Informatik angesehen werden.	• Studierende können Beispiele für Berechnungsmodelle nennen, die bei der Konzeption von Rechnernetzen sinnvoll sind.	• Studierende können ihr Wissen über Berechnungsmodelle nutzen, um eine Systembeschreibung mit formalen Methoden zu verifizieren.
• Studierende können den Bezug von Programmiersprachen zu Berechnungsmodellen erläutern.	• Studierende können einen Kommunikationsablauf zwischen Client und Server mit einem Berechnungsmodell ihrer Wahl (bspw. Petri-Netze) modellieren.	• Studierende können die Verbindung von Komponenten mit verschiedenen Berechnungsmodellen mittels SysML spezifizieren.

Abbildung 6.3.: Beispielhafte Kompetenzbeschreibungen für die Methodenebenen

zu wenig Generalisten ausgebildet werden, Rechnung getragen werden [Cas+05], [Sif11]. Die Relationen zwischen Themengebieten kann aus der Analyse der Kriterien extrahiert werden und erweitert das Konzept der fundamentalen Ideen damit insofern, dass nicht nur das Ergebnis der Analyse, sondern auch der Prozess selbst einen fachdidaktischen Beitrag besitzt. Für die Erläuterung der Umsetzung wird das in Abschnitt 5.5 durchgeführte Analyseverfahren für *Rekonfigurierbare Architekturen und reprogrammierbarer Designs* genutzt. Die Kriteriendimensionen können folgendermaßen für die Konzeption einer Lehrveranstaltung verwendet werden:

Horizontalkriterium und Fortbildungskriterium: Das Horizontal- und das Fortbildungskriterium eignen sich zum Einstieg in eine Lehrveranstaltung, da sie das Fundament eines Themenbereiches beschreiben. Studierenden kann durch die Anwendungsgebiete einer Idee, die in der Analyse zum Horizontalkriterium beschrieben sind, eine Motivation zum Erlernen des Themenbereiches gegeben werden. Gleichzeitig veranschaulicht es die breite Applikation der Idee bzw. umgekehrt die technischen Realisierungsmöglichkeiten. Das Fortbildungskriterium kann in zweifacher Weise für die Ausgestaltung einer Lehrveranstaltung genutzt werden. Zum einen enthält es das für die Idee notwendige Vorwissen, das in einer kurzen Zusammenfassung (bspw. eine Vorlesungseinheit) zum Annähern der Kompetenzniveaus der Studierenden benutzt werden kann. Zum anderen ist die zweite Facette des Kriteriums das notwendige Fundament für andere Ideen. Studierende erhalten hierdurch eine Motivation für das Themengebiet, wenn sie verstehen, dass es eine sinnvolle Grundlage für andere interessante Anwendungsgebiete ist. Beispielsweise ist das theoretische Wissen über parallele und nebenläufige Prozesse wichtig, um später Hardwarebeschreibungssprachen wie VHDL effektiv nutzen zu können. Diese Unterscheidung deckt sich demnach mit dem in Abschnitt 6.1 erläuterten

Zusammenhang zwischen Kompetenzniveaustufen einer Lehrveranstaltung und dem Verständnis einer Idee auf verschiedenen geistigen Niveaus.

Zeitkriterium: Die Strukturierung einer Lehrveranstaltung kann theoretisch durch alle Kriterien erfolgen, wobei das Zeitkriterium in den meisten Fällen nach historischen Fakten hin analysiert wird, und deswegen nur der Teil der Zeitkriteriumsanalyse relevant ist, der die Bezüge des Lerngegenstandes hinsichtlich zukünftig zu erwartender Paradigmenwechsel betrachtet. Da Aussagen über zukünftige Entwicklungen nicht zu belegen sind, bleibt es der Lehrperson überlassen, in welcher Form dort gewonnene Erkenntnisse in die Lehrveranstaltungskonzeption übernommen werden. In der hier vorgestellten beispielhaften Umsetzung wird das Zeitkriterium aus den genannten Gründen nicht berücksichtigt.

Sinnkriterium: Das Sinnkriterium beschreibt, ähnlich wie das Horizontalkriterium, eine die Studierenden motivierende Dimension, da die praktischen Umsetzungen der Idee betrachtet werden können. Deswegen können die in der Analyse ermittelten Themenbezüge für die Umsetzung von Praktika, Laborveranstaltungen oder Seminaren geeignet sein. Zwei didaktische Konzepte, die oft bei praktischen Anwendungen verwendet werden, sind der „projektbasierte Ansatz“ und der „problemorientierte Ansatz“. Beide zeichnen sich durch eine deutlich höhere Motivation bei den Lernenden aus, als dies bei traditionellen Vorlesungskonzepten der Fall ist [Blu+91], [NS92]. Norman analysierte verschiedene Beiträge zum problemorientierten Ansatz und kommt zu folgendem Schluss: „*There is also a strong theoretical basis for the idea that PBL students may be better able to transfer concepts to new problems, and there is some preliminary evidence to this effect*“ [NS92, S. 563]. Demzufolge ist die Anwendung einer Idee auf eine in der Wissenschaft oder industriellen Praxis vorhandene Problemstellung im Rahmen der fundamentalen Ideen empfehlenswert. Das Hauptproblem dabei ist, ein passendes Niveau zu finden, auf dem die Studierenden problembasiert lernen können, ohne dabei überfordert zu werden. Da für eine praktische Umsetzung einer Idee das im Fortbildungskriterium analysierte Wissen notwendig ist, empfiehlt sich die Durchführung der praktischen Aufgabe erst nach der Vermittlung dieser Grundlagen. Als motivierender Ausblick kann das zu gestaltende Projekt am Anfang der Lehrveranstaltung präsentiert werden, auch wenn die Umsetzung erst später erfolgt.

Varianzkriterium: Das Varianzkriterium kann eng mit dem Sinnkriterium verknüpft werden, da auf dieser Dimension Alternativen zur eigentlichen Idee darstellbar sind. Ähnlich wie beim Sinnkriterium eignet sich die Betrachtung entsprechender Lerninhalte erst zu einem späteren Zeitpunkt der Lehrveranstaltung, da die Studierenden bereites über Kompetenzen hinsichtlich der Idee verfügen sollten, um die Differenzierung zu anderen Ideen nachvollziehen zu können.

Im Gegensatz zu traditionellen Lehrkonzepten wird den Studierenden der Rahmen einer Idee bzw. deren Schnittstellen zu anderen Themenbereichen aufgezeigt. Hieraus ergibt sich die Strukturierung der Lehrveranstaltung, da Horizontal- und

Fortbildungskriterium am Anfang und das Sinn- bzw. Varianzkriterium am Ende einer Lehrveranstaltung am sinvollsten umzusetzen sind.

Bevor auf die Konzeption der Lehrveranstaltung eingegangen werden kann, muss deren methodische Umsetzung spezifiziert werden.

6.3. Integration der Lehrveranstaltungsmethodik

Aus den Erläuterungen der vorherigen Abschnitte lassen sich Anforderungen an die Integration der Lehrveranstaltungsmethodik ableiten. Dies wäre zum einen, dass ein Vorlesungskonzept ohne Praktikum für die Umsetzung der fundamentalen Ideen in einer Lehrveranstaltung nicht empfehlenswert ist. Zum anderen wurde festgestellt, dass sich die Struktur der Veranstaltung im Groben durch die Art der Kriterien ableiten lässt. Da die Grundlagen zu einer Idee zum Großteil aus Wissenselementen bestehen, bietet sich kein alleiniges Praktikum an. Besonders schwierig ist, die Forderung nach Generalisten in einen motivierenden Veranstaltungskontext einzubetten, da eigentlich Breiten- statt Tiefenwissen gefordert ist. Problem- und projektorientierte Konzepte werden vorrangig in praktischen Veranstaltungen umgesetzt, die tendenziell mehr fachspezifische als fachübergreifende Kompetenzen fördern. Deswegen empfiehlt sich für die Umsetzung des Lehrveranstaltungskonzeptes keine durchgehende Aufgabenstellung wie im Entwurfs- und Anwendungspraktikum (EAP), sondern mehrere kleinere Projekte, die zwar thematisch zusammenhängen können, aber nicht aufeinander aufbauen. Die Konzeption der Einzelveranstaltungen kann damit auf die jeweiligen Kriterienschwerpunkte (projektorientiert oder problemorientiert) angepasst werden. Der Schlusstermin der Veranstaltung kann die Zusammenhänge der Kriterien und den damit verbundenen Lehrveranstaltungen noch einmal aufgreifen. Durch die Praxisphasen und den umfangreichen Grundlagenteil lässt sich die Veranstaltung nur schwer in einem Block organisieren. Eine semesterbegleitende Veranstaltung, ähnlich zum EAP, ist deswegen sinnvoller (siehe Abschnitt 3.3).

Marwedel und Engel haben ein Lehrveranstaltungskonzept für den Bereich der Entwicklung eingebetteter Systeme umgesetzt, das die genannten Anforderungen erfüllt, jedoch keine inhaltliche Strukturierung nach dem Forschungsansatz der fundamentalen Ideen verfolgt [ME14]. Sie verwenden dabei den Ansatz des „Flipped Classroom“ (teilweise auch „Inverted Classroom“), der eine Vertauschung von Präsenz- und Heimarbeitsphasen vorsieht. Durch diesen dualen Charakter kann das Konzept dem „blended Learning“ zugeordnet werden, das Präsenzphasen und Onlinephasen beinhaltet [GV08, S. 5]. Durch den Flipped Classroom werden die typischerweise in der Präsenzphase vermittelten Lerninhalte in die Onlinephase verschoben. Dadurch können Aufgaben, Analysen und Diskussionen gemeinschaftlich in der eigentlichen Vorlesungszeit an der Universität oder Fachhochschule durchgeführt werden. Die Lehrperson ist dabei anwesend und kann direktes Feedback zu Lösungsprozessen oder Fragestellungen geben. Dabei greift die Lehrperson möglichst wenig in die Veranstaltung ein und wird somit zum „Guide on the Side“ [Kin93], also einer Hilfsperson, anstatt einem Dozenten. Der üblichen Vorlesung folgen die Studierenden von zuhause aus. Die dafür benötigten Unterlagen werden

typischerweise als Videos von den Lehrpersonen bereitgestellt. Es existiert eine Vielzahl an wissenschaftlichen Beiträgen, die belegen, dass sich die videobasierte Lehrgestaltung positiv auf die Motivation, das Verhalten und die Leistung der Studierenden auswirkt [HS13].

In den Vorbereitungsphasen des EAPs wurde festgestellt, dass sich trotz umfassender Lehrmaterialien kein oder nur ein sehr flüchtiger Wissenstransfer in die Präsenzphase eingestellt hat (siehe Abschnitt 3.4.1). Die Lehrmaterialien, die im EAP verwendet wurden, waren ausschließlich Textmaterialien. Eine entsprechende Umgestaltung in ein Video- oder Präsentationsformat ist zu überdenken. Um zu überprüfen, wie viel der zur Verfügung gestellten Lehrmaterialien die Studierenden verstanden haben, kann am Anfang der darauf folgenden Präsenzphase ein kurzes Quiz durchgeführt werden. Dies dient nicht zur Bewertung der Studierenden, sondern gibt ihnen ein Feedback zu ihrem Wissenserwerb. Im EAP wurde die Möglichkeit zur Überprüfung des Verständnisses mit Moodle-Tests durchgeführt [Jas+12]. Wie Herreid und Schiller anmerken, ist die professionelle Gestaltung von Lehrvideos eines der größten Probleme, das bei der Umsetzung des Flipped Classroom-Ansatzes auftritt [HS13]. Ebenso ist der Aufwand, um diese erstmalig zu erstellen, höher als der Aufwand, der gebraucht wird, um textbasierte Materialien zu erstellen. Alternative Methoden zur Darstellung des Veranstaltungsinhaltes sind Powerpointpräsentation und Audio-Dateien [HS13], [MSC13].

Hinsichtlich der in dieser Arbeit betrachteten Zielgruppe und der im vorherigen Abschnitt herausgearbeiteten Anforderungen an eine Lehrveranstaltung ist beim vorgestellten Konzept besonders von Vorteil, dass nicht eine Vorlesung *und* ein Praktikum für die Durchführung entworfen werden muss und die Verbindung des theoretischen und des praktischen Teils enger aneinander gebunden ist.

Der Flipped Classroom-Ansatz kann nur bei Veranstaltungen gewählt werden, bei denen eine Präsenzveranstaltung organisatorisch möglich ist. Hier unterscheiden sich das Konzept deutlich von anderen aktuell diskutierten Ansätzen wie dem „Massiv open online courses“ (MOOCs), das auf Veranstaltungen mit bis zu mehreren tausenden Studierenden ausgelegt ist, jedoch ohne Präsenzphasen durchgeführt wird (vgl. [Wed13]). Mason, Shuman und Cook haben in einer Lehrveranstaltung für Ingenieure den traditionellen und den Flipped Classroom-Ansatz verglichen [MSC13]. Sie kommen zu dem Schluss, dass der Umfang des Lehrmaterials, die Leistung der Studierenden und die Akzeptanz der Veranstaltung der Studierenden mindestens auf dem Niveau der traditionellen Vorlesung liegen, oft sogar darüber. Zu bedenken ist jedoch, dass der Kurs nur 20 Teilnehmer hatte und die geschilderten Ergebnisse für Lehrveranstaltung mit mehr Teilnehmern erst noch überprüft werden muss. Marwedel berichtet, dass in einer Lehrveranstaltung nach dem Flipped Classroom-Vorbild 30 Studierende in den Präsenzphasen kein Problem darstellten [ME14]. Seiner Erfahrung nach schätzt er, dass die maximal Teilnehmeranzahl für ein solches Veranstaltungskonzept bei 120 Studierenden liegt. Dabei sollten bereits mehrere nebenläufige Präsenzveranstaltungen durchgeführt werden (jeweils ca. 30 Teilnehmer).

6.4. Beschreibung der Lehrveranstaltung

Die Lehrveranstaltung wird in Anlehnung an die von Marwedel vorgestellte Veranstaltung strukturiert [ME14]. Marwedel führte die ersten zwei Veranstaltungen nach traditionellem Vorbild durch, um die Studierenden mit der Thematik vertraut zu machen. Auch in Verbindung mit der Methode der fundamentalen Ideen macht dieses Vorgehen Sinn, da die ersten Vorlesungen für einen Überblick, bzw. das Schaffen der Grundlagen (Fortbildungskriterium) genutzt werden sollten.

Ab der dritten Veranstaltung folgt die Präsenzphase dem folgendem Ablauf [ME14]:

1. Es wird ein kurzer Überblick über die aktuellen Folien gegeben (fünf Minuten) für den Fall, dass es Unterschiede zu den Videoaufnahmen gibt.
2. Darauf folgend arbeiten die Studierenden in Kleingruppen (zwei bis drei Personen) an bisher unbekannten Aufgaben. Die Tutoren helfen den jeweiligen Gruppen bei Problemen. Falls mehrere Gruppen Verständnisschwierigkeiten haben, wird der Sachverhalt im Plenum erläutert. Die Aufgaben (teilweise auch die Lösungen) können durch die Studierenden in einem Kursportal im Internet abgerufen werden.
3. Am Ende einer Präsenzveranstaltung wird ein Ausblick auf die nächste Veranstaltung gegeben (fünf Minuten).

Ausgehend von den Darstellungen der Abschnitte 6.1, 6.2 und 6.3 ergibt sich ein inhaltliches und organisatorisches Gesamtkonzept der Lehrveranstaltung (siehe Abbildung 6.4). Aufgrund der Erfahrungen aus dem EAP wurden je zwei Veranstaltungstermine für die praktischen Präsenzphasen zusammengefasst, um den Studierenden mehr Zeit für die Durchführung zu geben.

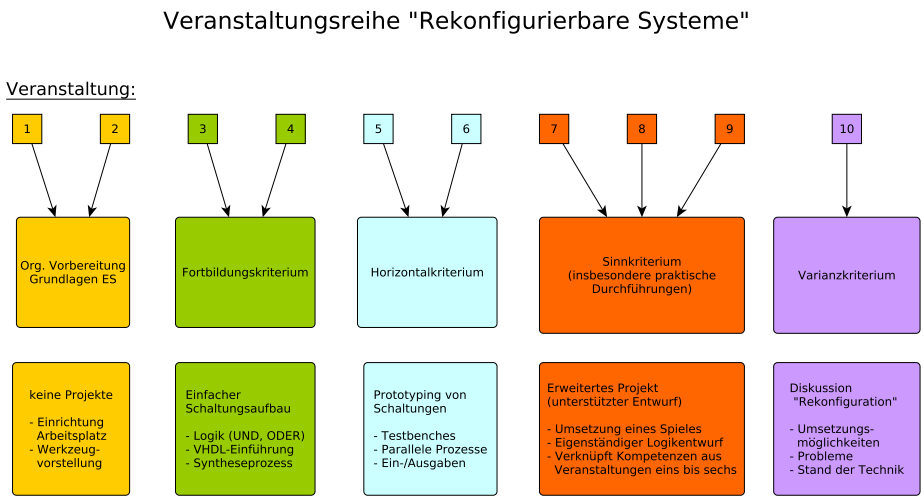

Abbildung 6.4.: Lehrveranstaltungskonzept basierend auf vier Kriterien fundamentaler Ideen

Die ersten beiden Veranstaltungen bilden die Einführung zum Themenbereich eingebettete Systeme und erläutern den Studierenden die Strukturierung der Lehrveranstaltungsreihe. Die Grundlagen rekonfigurierbarer Architekturen werden in Veranstaltung drei und vier anhand einfacher digitaler Schaltungen eingeführt. Durch den Umgang mit der Entwicklungsumgebung Xilinx ISE eignen sich die Studierenden Kompetenzen an, die für die späteren Projekte notwendig sind (bspw. Testbenches lesen). Die Veranstaltungen fünf und sechs behandeln die Themengebiete, die im Horizontalkriterium erarbeitet wurden. Dies ist insbesondere der Themenbereich zum Prototyping digitaler Systeme mit einem FPGA. Dazu gehören auch die Ansteuerung von Ausgängen und Eingängen, die zusammen mit Grundlagen der Systembewertung in den zugehörigen Präsenzveranstaltungen erarbeitet werden. In den Veranstaltungen sieben, acht und neun wird ein umfangreiches Projekt zur Verdeutlichung paralleler und nebenläufiger Konzepte durchgeführt. Die Studierenden bekommen die Aufgabe, einen gut parallelisierbaren Algorithmus erst sequentiell, dann teilweise parallel umzusetzen. Den Abschluss der Veranstaltung bildet die Veranstaltung zehn, deren inhaltlicher Schwerpunkt auf den theoretischen Grundlagen zur Rekonfigurierbarkeit von Rechensystemen liegt. Wie in der Analyse in Kapitel 5.5 nachzulesen ist, ist die Vermittlung von praktischen Kompetenzen zu diesem Anwendungsfeld auf Bachelorniveau schwierig. Auf ein umfangreiches Projekt wird deswegen an dieser Stelle verzichtet, um die Studierenden nicht zu überfordern. Die Relevanz rekonfigurierbarer Architekturen für die wissenschaftliche und industrielle Praxis ist durch die vorhergehenden Veranstaltungstermine bereits deutlich geworden.

Die folgende Erläuterung der Lehrveranstaltung ist konzeptueller Art und soll vor allem die Möglichkeiten aufzeigen, wie die Analyse der Kriteriendimensionen für die Strukturierung einer Lehrveranstaltung genutzt werden kann. Somit ist der folgende Abschnitt zwischen einer sehr abstrakten Curriculaempfehlung und einer konkreten Lehrverantstaltungsbeschreibung einzuordnen. Es werden aus diesem Grund nur die Implementierungsansätze gezeigt, die einen besonders günstigen didaktischen Ansatz darstellen. Neben der konzeptuellen Ebene erheben die Veranstaltungsbeschreibungen deswegen keinen Anspruch an die technische Vollständigkeit, sondern sind anhand der erläuterten Konzeption fachspezifisch auszugestalten. Dies ist beispielsweise im Kontext der Grundlagen eingebetteter Systeme relevant, deren technische Umsetzungen zum einen in der Literatur ausführlich behandelt wurden und zum anderen den Umfang dieser Arbeit sprengen würde.

Veranstaltung 1: Grundlagen der Entwicklung eingebetteter Systeme

Die ersten Vorlesungen werden, wie im Lehrkonzept von Marwedel, traditionell ohne einen „Flip“ durchgeführt [ME14]. In Zusammenhang mit den fundamentalen Ideen können diese zwei Einführungsveranstaltungen zum Wiederholen grundlegenden Wissens verwendet werden, um eine bessere Verflechtung mit den bereits gehörten Fächern zu erreichen. Da der Aufbau der Lehrveranstaltung mit mehreren kleineren Projekten ungewöhnlich ist, bietet es sich an, den Studierenden die Strukturierung durch die vier verschiedenen Kriteriendimensionen im Vorhinein zu erläutern.

Inhaltlich orientiert sich die erste Veranstaltung an den grundlegenden Überlegungen zum Themenbereich der eingebetteten Systeme als übergeordnetem Anwendungsbereich für viele rekonfigurierbare Rechnerarchitekturen. Aus dem Studienverlaufsplan der Universität Siegen für Informatikstudiengänge geht hervor, dass die Studierenden zum Zeitpunkt der Veranstaltung noch keine andere Veranstaltung besucht haben, in denen eingebettete Systeme als Anwendungsbereich thematisiert wurden (Stand: Modulhandbuch, 2013). Vor dem Studienbeginn kennen nur sehr wenig Studierende den Begriff der eingebetteten Systeme. Diesen Umstand belegt eine im Jahr 2011 bei Siegener Erstsemesterstudierenden (N=30) durchgeführte Befragung durch die Wissenschaftler des KOMINA-Projektes. Wie in Tabelle 6.1 zu sehen ist, begannen im Sommersemester 2011 besonders viele Informatiker mit Nebenfach Elektrotechnik bzw. Medienwissenschaften ihr Studium. Die meisten Studienanfänger gaben an, nicht zu wissen, was ein eingebettetes System ist. Auch die Kenntnisse in domänenunspezifischen Programmiersprachen wie C oder C++ sind nicht bei allen Studienanfängern vorhanden, wobei die damit verbundenen Kompetenzen durch die Grundlagenfächer im Studium abgedeckt werden. Für die Universität Siegen sind dies „Algorithmen und Datenstrukturen“ sowie „objektorientierte und funktionale Programmierung“. Hardwarebeschreibungssprachen wie VHDL, die in einem Großteil der Präsenzphasen verwendet werden, sind mehr als 80% der Studienanfänger unbekannt. Ähnlich verhält sich der Vorwissensstand zum Themenbereich „FPGAs“, der ebenfalls erst im EAP vertieft wird. Die Grundlagen zur Digitaltechnik sind hingegen ab dem ersten Semester in den Vorlesungen „Digitaltechnik“ und „Rechnerorganisation“ thematisiert. Grundlegende Konzepte aus diesen zwei Bereichen können demnach in der Lehrveranstaltung als bekannt vorausgesetzt werden.

Ausgehend vom zu erwartenden Vorwissen sollten folgende Kompetenzen in der ersten Veranstaltung vermittelt werden.

1. Die Studierenden können ein eingebettetes System auf Begriffsebene *definieren* und schematisch die im System üblicherweise vorkommenden Komponenten und deren Verbindungen *skizzieren*.
2. Die Studierenden können anhand eines Beispiels *erläutern*, wofür ein eingebettetes System verwendet wird und auf welchen Ebenen es sich zu traditionellen PC-Plattformen *unterscheidet*.
3. Die Studierenden können einen Entwicklungsprozess eines eingebetteten Systems mit mindestens fünf verschiedenen Entwicklungsetappen *beschreiben*.

Die in den Kompetenzbeschreibungen verwendeten Verben sind hervorgehoben, um leichter die Kompetenzstufung von Fuller et al. anwenden zu können [Ful+07]. Wie zu erkennen ist, befindet sich der Kompetenzerwerb nur auf der X-Achse, besitzt also noch keinen Bezug zu den „produzierenden“, sondern nur zu den *interpretierenden* Kompetenzen [Ful+07] (siehe Abschnitt 3.2).

Auf die geeigneten Lehrveranstaltungsmaterialien wird an dieser Stelle nicht eingegangen, da sich reichlich Literatur zu den Grundlagen der Entwicklung eingebetteter Systeme finden lässt, welche die Vermittlung der genannten Kompetenzen fördert. Beispielsweise seien hier genannt: [Mar11], [Kop11], [BST10].

Frage	Antwort(%)
Mit welchem Nebenfach studieren Sie?	
Automotive Systems Engineering	03,33
Elektrotechnik	46,67
Mathematik	10,01
Maschinenbau	03,33
Medienwissenschaften	36,67
Kennen Sie FPGAs?	
Ist mir unbekannt	78,57
Ist mir bekannt	17,86
Nutze ich	03,57
Kennen Sie C/C++?	
Ist mir unbekannt	13,33
Ist mir bekannt	56,67
Nutze ich	20,00
Beherrsche ich	10,00
Kennen Sie VHDL?	
Ist mir unbekannt	82,14
Ist mir bekannt	14,29
Nutze ich	03,57
Wissen Sie was ein eingebettetes System ist?	
Ja	23,33
Nein	76,67

Tabelle 6.1.: Erstsemesterbefragung zu eingebetteten Systemen (2011)

Die Überprüfung der ersten Kompetenz kann leicht anhand der eingeführten Definition eines eingebetteten Systems erfolgen. Beispiele hierfür wurden bereits in Abschnitt 2.1 genannt. Das Skizzieren der Komponenten innerhalb eines eingebetteten Systems soll den Studierenden die besondere Wichtigkeit der Kommunikation mit der Umgebung verdeutlichen und kann anhand des in Abbildung 2.1 gezeigten Aufbaus erfolgen (siehe Kapitel 2.1). Die zweite Kompetenz lässt sich anhand von Beispielen aufzeigen. Insbesondere kann hier bereits ein erster Verweis auf nicht-funktionale Anforderungen erfolgen, die ebenso einen fundamentalen Aspekt der Entwicklung eingebetteter Systeme darstellen (siehe Abschnitt 5.6). Wie das Konzept für eine Softwareumsetzung eines Lehr-/Lernwerkzeuges aussehen kann, das die umfangreichen Bezüge zwischen Komponenten eines eingebetteten Systems verdeutlicht, wurde von Büchner gezeigt [Büc13]. Der vorgestellte Ansatz nutzt symbolische Schieberegler, um ein Attribut eines Systems oder einer Systemkomponente zu verändern. Mehrere auf diese Weise dargestellte Komponentenattribute können miteinander verknüpft werden. Ändert der Benutzer nun einen Schieberegler, ändern sich transitiv, also auch über mehrere Verbindungen hinweg, die anderen Schieberegler. Ein Beispiel illustriert die Anwendung des Konzeptes:

Beispiel zur Schiebereglermetapher Für eine Temperaturüberwachung ist in einer Produktionsanlage jedes Produkt mit einem Temperatursensor und einer Steuerungseinheit verbunden. Sobald eine kritische Temperatur überschritten wird, meldet die Steuerungseinheit einen akustischen oder visuellen Alarm.

Der besseren Veranschaulischung wegen wird im Folgenden nicht das ganze System in Komponenten mit Attributen aufgeschlüsselt, sondern nur die drei Attribute und deren Beziehungen untersucht, die direkten Einfluss auf die Temperaturmessung haben. Diese sind: *Temperaturauflösung*, *Geschwindigkeit der Temperaturmessung* und *Genauigkeit der Temperaturmessung*. An den Attributen kann gezeigt werden, dass die Geschwindigkeit einer Temperaturmessung vom Temperatursensortyp abhängt (Thermale Konstante) und zusätzlich auch von der Ansteuerung des ADCs (siehe Grafik 6.5). Die Ansteuerung des ADCs kann auf unterschiedliche Weisen erfolgen. Eine Möglichkeit ist, dass mehrere Messwerte zusammengefasst und als ein durchschnittlicher Wert ausgegeben werden. Zur Steigerung der Temperaturauswertungsgeschwindigkeit kann demnach entweder ein schnellerer Temperatursensor eingesetzt werden oder weniger Auswertezyklen beim ADC durchgeführt werden. Die Studierenden können so *ausprobieren*, welche Auswirkungen eine Parametrisierung auf mehrere Attribute und Komponenten des Systems hat. Lehrmethodisch sind verschiedene Nutzungen des Konzeptes denkbar. Die Studierenden könnten in einer Art Wettkampf die Aufgabe bekommen, durch eine geeignete Komponentenkonfiguration eine gestellte Parameterzielstellung zu erreichen. Eine andere Möglichkeit ist die gemeinschaftliche Erstellung eines Systemmodells. Besonders wertvoll sind dabei grundlegende Fragestellung wie bspw. die physikalische Einheit, in der ein Temperatursensor misst. Dies ist eben nicht direkt die Temperatur, sondern der Widerstand, der sich auch über die abfallende Spannung ermitteln lässt. Für Studierende, die keine Vorkenntnisse in der Elektrotechnik besitzen, ist dieses Verständnis unter Umständen nicht intuitiv.

Das in [Büc13] vorgestellte Konzept nutzt das Gestaltgesetz „Gleiches Schicksal“, um die Zugehörigkeit von Komponenten zu verdeutlichen. Diesem psychologischen Gesetz zu Folge werden Elemente, die sich zur gleichen Zeit bewegen, als zusammengehörig wahrgenommen [MKB03], [NF02]. Die sonst verwendeten Netzstrukturen ohne interaktive Elemente werden von Benutzern typischerweise langsamer aufgenommen als eine entsprechend interaktive Umsetzung mittels Gestaltgesetzen [MKB03].

Die dritte in der Lehrveranstaltung zu vermittelnde Kompetenz ist auf einer geeigneten Abstraktionsebene zu vermitteln, ohne die Studierenden durch die Einführung neuer Begriffe zu überfordern. Im Wesentlichen können die in den ersten beiden Kompetenzdefinitionen enthaltenen Wissensebenen aufgegriffen und in Form einer Abbildung zusammengefasst werden. Eine vereinfachte Darstellung des in Abbildung 5.2 gezeigten Entwicklungsprozesses ist dafür geeignet. Diese Grafik kann ebenso für die folgende Veranstaltung verwendet werden, um technische Bezüge zu den Kriterien herzustellen und im Entwicklungsprozess einzuordnen.

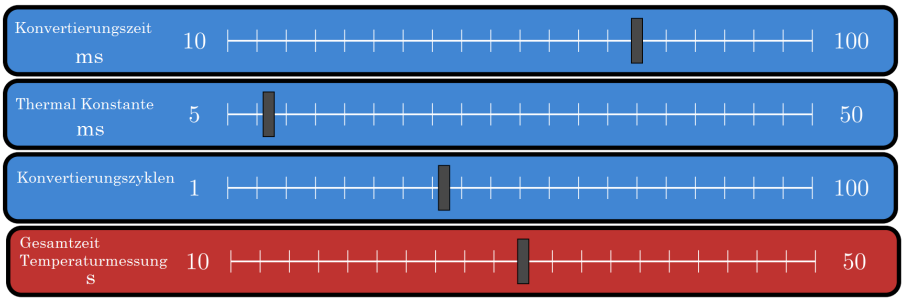

Abbildung 6.5.: Beispiel zur Darstellung komplexer Attribute von eingebetteten Systemen [Büc13]

Veranstaltung 2: Einführung Lehrveranstaltungskonzeption und rekonfigurierbare Architekturen

In der zweiten Veranstaltung gilt es, den Studierenden den Veranstaltungsaufbau hinsichtlich der verschiedenen Dimensionen fundamentaler Ideen aufzuzeigen. Ausgehend von den in Lehrveranstaltung eins geschaffenen Grundlagen zu eingebetteten Systemen sollten nun die Voraussetzungen für die Einführung des Themengebietes „rekonfigurierbarer Architekturen und reprogrammierbarer Designs“ geschaffen sein. Die folgende Kompetenzförderung wird angestrebt:

1. Studierende können die Unterschiede zwischen klassischen eingebetteten Systemen und Systemen mit rekonfigurierbaren Architekturen oder reprogrammierbaren Designs *erläutern*.

2. Studierende können rekonfigurierbare Architekturen und reprogrammierbare Designs zu anderen Methoden und Sichtweisen des Anwendungsgebietes *in Bezug setzen*.

Neben den zwei genannten Kompetenzen ist spätestens in dieser Veranstaltung der organisatorische Rahmen für die späteren praktischen Durchführungen zu schaffen. Dies betrifft insbesondere die Erläuterung der verschiedenen Software- und Hardwarewerkzeuge sowie die Aufteilung in Gruppen, die sich aufgrund der meist nur in geringen Stückzahlen verfügbaren FPGA-Plattformen anbietet. In Kapitel 3.4.2 wurde der Virtual Workspace-Ansatz vorgestellt, dessen Anwendung wegen der getrennten Heimarbeits- und Präsenzphasen auch beim vorliegenden Lehrkonzept sinnvoll ist. Die Verteilung und Erklärung des dazugehörigen USB-Sticks sollte ebenfalls in dieser Veranstaltung erfolgen, um mehr Zeit für die praktischen Anwendungen in den folgenden Veranstaltungen zur Verfügung zu haben.

Ähnlich zur ersten Veranstaltungskonzeption gibt es empfehlenswerte Literatur, auf der die Präsenzphase aufgebaut werden kann. Allen voran ist das Buch von

Hauck und Dehon zu nennen, das technikunabhängig die Grundlagen der rekonfigurierbaren Architekturen erläutert [HD10]. Eine etwas technischere Einführung in das Themengebiet gibt das Buch von Kesel und Bartholomä [KB09]. Letzteres wird für die später folgenden Veranstaltungen zum Themenbereich VHDL noch häufiger verwendet werden.

Die erste Kompetenz kann durch die Vorstellung rekonfigurierbarer Architekturen vollzogen werden. Bisher wurden diese Architekturen den Studierenden als „Blackbox“ dargestellt. Um die in der ersten Kompetenz angesprochenen Unterschiede zu klassischen eingebetteten Systemen begreifen zu können, ist der Blick auf die technische Realisierung eines rekonfigurierbaren Systems notwendig. Zum einen wegen seiner industriellen Relevanz und zum anderen wegen des Einsatzes als Plattform für die praktische Durchführung sollte dieser Lerninhalt an einem FPGA erklärt werden. Die dabei maßgebliche Fragestellung nach den universellsten Komponenten, die einen FPGA ausmachen, kann in Gruppenarbeit recherchiert und im Plenum diskutiert werden. Um den Studierenden einen Eindruck über die verfügbaren Ressourcen zu geben, bietet es sich an, die später zu verwendende Plattform als Beispiel einzusetzen. Im Fall des EAPs war dies ein Xilinx Spartan 3-FPGA, der in dieser Ausführung jedoch nicht mehr von der aktuellen Entwicklungsumgebung Xilinx ISE unterstützt wird.

Die Diskussion um den Aufbau eines rekonfigurierbaren Systems orientiert sich inhaltlich an den Fragestellungen, die die fünf Kriterien der fundamentalen Ideen ausmachen. Somit ist es eine Überleitung zur zweiten Kompetenz und damit zur Vermittlung der Lehrveranstaltungsstruktur. Die Teilnehmer können dabei selbst recherchieren, inwieweit es Bezüge zwischen rekonfigurierbaren Architekturen und reprogrammierbaren Designs und anderen, bspw. in einer Liste dargestellten, Methoden und Sichtweisen der eingebetteten Systementwicklung gibt. Diese Liste kann die Lehrperson direkt aus den untersuchten Ideen in dieser Arbeit ableiten. Es ist zu erwarten, dass Studierende die Sinnzusammenhänge der einzelnen Veranstaltungsteile dadurch besser miteinander verknüpfen können und in Verbindung mit der ersten Kompetenz das Anwendungsgebiet rekonfigurierbarer Architekturen und reprogrammierbarer Designs umfassender einordnen können.

Veranstaltung 3 und 4: Grundlagen zur Entwicklung rekonfigurierbarer Architekturen (Fortbildungskriterium)

Beginnend mit der dritten Veranstaltung ist das Lehrkonzept nach dem beschriebenen Leitgedanken des „Flipped Classroom“ organisiert. Demnach gibt es nun nicht nur eine Präsenz-, sondern auch eine Vorbereitungsphase. Die Kompetenzen, über die die Studierenden nach den jeweiligen Veranstaltungen verfügen sollen, sind nicht zweigeteilt dargestellt, da der Kompetenzzuwachs erst über einen längeren Zeitraum, in diesem Falle also der Zeit von Beginn der Vorbereitung bis zum Abschluss der Präsenzphase, zu erwarten ist. Wie bereits vorher erläutert, werden ab der dritten Veranstaltung alle Kriteriendimensionen im Rahmen einer Doppelveranstaltung behandelt, damit die Studierenden mehr Zeit für die praktische Umsetzung in den Präsenzphasen haben.

Die dritte und vierte Lehrveranstaltung beziehen Inhalt und Zielstellung aus der Analyse zum Fortbildungskriterium, die das notwendige Vorwissen für die Vermittlung rekonfigurierbarer Architekturen und reprogrammierbarer Designs aufgedeckt hat. Die folgenden Kompetenzen sind deswegen der „Learning-Outcome" der beiden Veranstaltungen:

1. Die Studierenden können eine einfache Hardwarebeschreibung in VHDL funktional *erläutern* und für die Architektur einer Komponente Struktur-, Verhaltens- und Registertransferbeschreibungen *einsetzen*.

2. Die Studierenden sind in der Lage, die Nutzung von Simulationstechniken zu *begründen* und in Form von Testbenches *anzuwenden*.

3. Die Studierenden können grundlegende Konzepte (bspw. Prozesse, Sensitivitätslisten, Takte) der Hardwarebeschreibung erläutern.

Wie bereits in der Analyse der Idee erläutert wurde, werden FPGAs in der Praxis häufig zum Erstellen von Prototypen genutzt. Die Beschreibung der FPGA-Funktionalität wird durch die Verwendung einer Hardwarebeschreibungssprache (bspw. VHDL oder Verilog) erreicht. Als somit zentrales Element der Entwicklung rekonfigurierbarer Architekturen wird ihnen in der Lehre ein großer Stellenwert eingeräumt. Verschiedene Lehrveranstaltungskonzepte und Praxisberichte sind hierzu veröffentlicht worden (bspw. [AG93], [Hua+97], [Duc05], [Are01]). Duckworth betont, dass es wichtig sei, die Studierenden am Anfang der Lehre nicht mit den „Besonderheiten" von VHDL hinsichtlich synthetisierbarer und nichtsynthetisierbarer Anweisungen zu überfordern [Duc05]. Ebenso argumentiert er für eine detaillierte Reflexion der Unterschiede zwischen klassischen Programmiersprachen und Beschreibungssprachen im Allgemeinen. Studierende müssten verstehen, dass sie eine Beschreibung über das Verhalten des Systems anfertigen, die vom Synthesewerkzeug interpretiert wird und somit nicht einer festgesetzten Anweisungsabfolge entspricht [Duc05]. Die Studierenden kennen diesen Umstand eventuell von Optimierungsalgorithmen in Compilern, die ebenso Quellcodebeschreibungen uminterpretieren.

In der dritten und vierten Lehrveranstaltung werden diese Punkte systematisch aufgegriffen. Die Vorbereitungsphase der dritten Veranstaltung ist durch eine Einführung in die Hardwarebeschreibungssprache VHDL gekennzeichnet. Die Sprache VHDL wird gewählt, da sie die in Europa am häufigsten genutzte Sprache zur Hardwarebeschreibung ist (vgl. [Ran94]). Die Studierenden sollten sich mit den grundlegenden Ansätzen von VHDL beschäftigen und bspw. die Trennung von Schnittstellen- zur Architekturbeschreibung erläutern können. Dafür notwendige Informationen befinden sich in der bereits erwähnten Literatur von Hauck und Dehon, bzw. Marwedel. Wie aus der ersten Kompetenzbeschreibung ersichtlich ist, wird in den beiden Veranstaltungen das erste Mal das Kompetenzniveau des *Anwendens* erreicht (Einsetzen). Die Kompetenz ist also am ehesten durch praktische Arbeiten zu erlangen. Eine reine Rezeption von vorbereitetem Wissen genügt nicht. In der Präsenzphase kann beispielsweise die Aufgabe gegeben werden, eine UND-Komponente mit vier Eingängen zu entwerfen. Dabei geht es erst einmal nicht um die Komplexität der Aufgabenstellung. Ausgehend von der Unterscheidung von

Schnittstellen- und Architekturbeschreibung einer Komponente sollen die Studierenden die Logik mit unterschiedlichen Implementierungsmöglichkeiten umsetzen (Verhaltens-, Struktur- und Registertransferebene). Ein Beispiel für die funktional gleiche Umsetzung einer Struktur-, Verhaltens- und Registertransferbeschreibung findet sich in Listing 3.

```
-- Entity Deklaration
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity And4Bit is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
           z : out  STD_LOGIC);
end And4Bit;

-- Verhaltensbeschreibung
architecture behavioral of And4Bit is

begin
z <= a and b and c and d;
end behavioral_And4Bit;

-- RTL-Beschreibung
architecture rtl of And4Bit is

begin
z <= '1' when a='1' and b='1'
     and c='1' and d='1' else '0';
end rtl_And4Bit;
```

```
-- Strukturelle Beschreibung
architecture structural of And4Bit is
component andgate is
        port (
        a: in STD_LOGIC;
        b: in STD_LOGIC;
        c: out STD_LOGIC
        );
end component;

signal and_a_out_sig: STD_LOGIC;
signal and_b_out_sig: STD_LOGIC;

begin
and_a: andgate
port map (
 a => a,
 b => b,
 c => and_a_out_sig
 );

and_b: andgate
port map (
 a => c,
 b => d,
 c => and_b_out_sig
 );

and_c: andgate
port map (
 a => and_a_out_sig,
 b => and_b_out_sig,
 c => z
 );
end structural;
```

Listing 3: Entity-Deklaration einer Vier-Bit-UND-Komponente mit drei funktional gleichen Umsetzungen

Der Quellcode ist in vier Bereiche aufgeteilt, von denen der erste die Deklaration der Ein- und Ausgänge der Komponente beschreibt (Schnittstellendefinition). Die übrigen drei Quellcodeteile sind gleichbedeutend, da sie dieselbe Funktionalität auf unterschiedlichen Ebenen beschreiben. Welche dies jeweils ist, ist mit einem Kommentar am Anfang des Quellcodes eingeleitet. Wie eine Vier-Bit-UND-Logik aus drei Zwei-Bit-UND-Bausteinen aufgebaut werden kann, ist aus den Grundlagen der

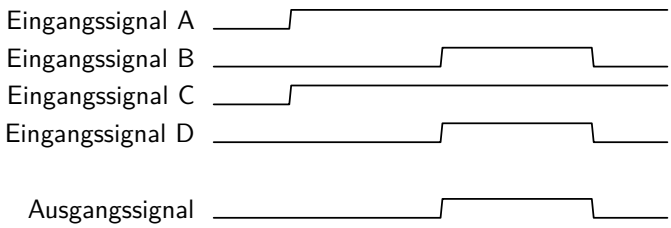

Tabelle 6.2.: Signal-/Zeitdiagramm für eine einfache Vier-Bit-UND-Komponente

Digitaltechnik bekannt. Die Studierenden sollen also anhand einer schematischen Zeichnung der Komponente deren Verhalten in VHDL auf den genannten Ebenen beschreiben (Präsenzphase). Aus didaktischer Sicht ist die Strukturbeschreibung die am schwersten zu vermittelnde Umsetzungsvariante, da zum einen der Quellcodeumfang gegenüber den anderen dargestellten Implementierungsmöglichkeiten deutlich höher ausfällt, und zum anderen das Konzept der *Signale* bekannt sein muss. Es empfiehlt sich daher, erst die Verhaltensbeschreibung und die Beschreibung auf Registertransferebene in der Präsenzphase zu bearbeiten. Da Programmieranfänger Quellcode typischerweise Zeile für Zeile interpretieren, ist darauf zu achten, Beispiele mit geringem Umfang zu wählen [RRR03].

Die in Listing 3 dargestellten Architekturbeschreibungen sind möglichst einfach gehalten, um weiterführende Konzepte wie Synchronisierung, Sensitivitätslisten und nebenläufige Abarbeitung zu vermeiden. Diese Themen sind Inhalt der nächsten Veranstaltungen. Damit die Studierenden ihre in der Präsenzphase entwickelten Entwürfe auch testen können, muss ihnen die zweite Kompetenz vermittelt werden. Die erste Facette dieser Kompetenzbeschreibung betrifft das Verständnis darüber, dass die *Simulation* der entworfenen Komponente neben der Implementierung auf Hardwareebene die einzige Möglichkeit darstellt, den Entwurf zu testen. In VHDL werden dafür so genannte Testbenches verwendet, die ein System (oder eine Komponente) instanziieren und dessen Eingänge in Abhängigkeit der in der Simulation genutzten Modellzeit mit frei definierbaren Signalen belegen. In einem Signal/Zeit-Diagramm werden die entsprechenden Eingangs- und Ausgangssignale dargestellt (siehe Abbildung 6.2). Diese Art des Diagramms lässt sich recht leicht mit einer Wahrheitstabelle abgleichen, die im Vorhinein, bspw. für den Vier-Bit-UND-Baustein, angefertigt wurde. Die Kompetenzbeschreibung wurde so formuliert, dass die Studierenden vorerst keine eigene Testbench entwerfen, sondern diese nur anwenden sollen. Hierbei können weiterführende Aufgaben erstellt werden, die die Studierenden dafür sensibilisieren, die Modellzeit in ihren Entwürfen zu berücksichtigen. Wird für die Testbench der falsche Clockzyklus gewählt, kann auch das Verhalten in der Simulation fehlerhaft sein.

Für die vierte Veranstaltung ist auf die neunwertige Logik in VHDL einzugehen. Mit dem zuvor erarbeiteten Verständnis von Testbenches sollten die am Anfang häufig auftretenden Fälle „undefined“ (U) und „multiple driver“ (X) erklärt werden, um den Studierenden zu helfen, ihre Entwürfe auf typische Fehler zu untersuchen. Testbenches sind auch geeignet, um den Studierenden die Unterschiede zwischen Signalen und Variablen in VHDL verständlich zu machen. In VHDL werden alle

Signale zum Ende eines Prozesses mit dem zuletzt zugewiesenen „Wert“ belegt. Variablen hingegen nehmen sofort den ihnen zugewiesenen Wert an, können jedoch nicht zur Verbindung von Komponenten genutzt werden. Signale können zudem nicht aus unterschiedlichen Prozessen heraus „getrieben“ werden, ohne eine sogenannte Auflösungsfunktion bereitzustellen, die angibt, welches Signal in diesem Fall auf der Leitung vorherrscht. Um dieses Verhalten verständlich zu vermitteln, bietet es sich an, die in VHDL beschriebene Schaltung selbst nachzubauen. Die Studierenden erkennen somit den Konflikt, der entsteht, wenn zwei Signale auf einer Leitung anliegen: die beiden Signale kollidieren. Eine entsprechende Literaturstelle zur Unterscheidung von Signalen und Variablen findet sich mit Beispielen im Buch von Kesel und Bartholomä [KB09]. Die Überprüfung des Verständnisses der Studierenden kann mit verschiedenen Quellcodebeispielen bereits in der Vorbereitungsphase stattfinden. Der Quellcodeauszug in Abbildung 4 zeigt ein Beispiel zur Unterscheidung von Signalen und Variablen in VHDL.

```
architecture behavior of Test_A is

signal sum: Unsigned \
    (3 DOWNTO 0):="0000";

begin
    process
    begin
    sum<=sum+1;
    sum<=sum+1;
    sum<=sum+1;
    wait for 100ns; --Sum = 0001
    sum<=sum+1;
    wait;           --Sum = 0010
    end process;
end;
```

```
architecture behavior of Test_B is

begin
    process
    variable sum: Unsigned \
        (3 DOWNTO 0):="0000";
    begin
    sum:=sum+1;     --Sum = 0000
    sum:=sum+1;     --Sum = 0001
    sum:=sum+1;     --Sum = 0010
    wait for 100ns; --Sum = 0011
    sum:=sum+1;
    wait;           --Sum = 0100
    end process;
end;
```

Listing 4: Einfaches Beispiel für die Unterschiede zwische Signalen und Variablen in VHDL

In der Präsenzphase kann aufbauend auf dem Verständnis von Variablen und Signalen das Prinzip der Synchronität und Asynchronität eingeführt werden. So kann den Studierenden anhand eines Beispiels gezeigt werden, warum ein getakteter Prozess mit Signalzuweisung an unterschiedlichen Takten alle Zuweisungen behält, dies in ungetakteten Beschreibungen aber nicht der Fall ist.

Veranstaltung 5 und 6: Prototyping und Hardwareentwurf mit rekonfigurierbaren Architekturen (Horizontalkriterium)

Die Inhalte der fünften und sechsten Veranstaltung können aus der Analyse des Horizontalkriteriums abgeleitet werden. Als zentral für diese Doppelveranstaltung sind deswegen die Kompetenzen hinsichtlich der Themengebiete *Simulation* und

Prototyping zu sehen. Mit den in der letzten Doppelveranstaltung vermittelten Kompetenzen können erste Abschätzungen von Systemcharakteristika vorgenommen werden. Zusammengefasst sind die folgenden Kompetenzen zu fördern:

1. Die Studierenden sind in der Lage, verschiedene Architekturbeschreibungen hinsichtlich ihres Ressourcenverbrauchs auf einem spezifischen FPGA zu *bewerten*.

2. Die Studierenden können anhand eines vorgegebenen VHDL-Quellcodes nebenläufige und sequentielle Anweisungen *unterscheiden*.

3. Die Studierenden sind in der Lage, eine digitale Schaltung prototypisch auf eine rekonfigurierbare Hardware *umzusetzen*.

4. Die Studierenden können Ein- und Ausgabekomponenten eines rekonfigurierbaren Systems mit Signalen der VHDL-Beschreibung *verknüpfen*.

In der ersten und zweiten Veranstaltung haben die Studierenden einen oberflächlichen Überblick eingebetteter Systeme und rekonfigurierbarer Architekturen erhalten. Für die Förderung der ersten Kompetenz dieser Doppelveranstaltung muss dieses Wissen hinsichtlich der Umsetzung einer Hardwarebeschreibung auf die tatsächliche Hardware vertieft werden. Insbesondere ist dabei das Verständnis der Prozessabfolge von der Beschreibung eines Systems mit VHDL bis hin zur Generierung des Bitstreams wichtig, um später die Verbindung zwischen Beschreibung auf HDL-Ebene und der vom Synthesewerkzeug vorgeschlagenen Umsetzung der CLBs nachvollziehen zu können. Während die Xilinx ISE bisher hauptsächlich als umfangreicher Texteditor genutzt wurde, ist ein solideres Verständnis dieses Werkzeugs für die folgenden Versuche zwingend notwendig. Für die, zumindest grobe, Abschätzung der Hardwareumsetzung ist konkret die Zusammenfassung des Systemdesigns in der Xilinx ISE wichtig. Die Studierenden können damit erfahren, wie viele Slices und LUTs für ihren Entwurf benötigt werden und diese Informationen für ein tieferes Verständnis ihres Entwurfes nutzen. Beispielhaft kann in der Präsenzphase die Aufgabe erteilt werden, die drei genannten Umsetzungsmöglichkeiten einer Vier-Bit-UND-Komponente zu synthetisieren und in der Zusammenfassung nach Unterschieden zu suchen. Wie dann zu erkennen ist, optimiert die Xilinx ISE selbst bei Strukturbeschreibungen Komponenten „weg“. Im Falle der Vier-Bit-UND-Logik wird jedes Mal eine LUT erzeugt, auch wenn die Strukturbeschreibung drei UND-Bausteine vorgibt. Die Studierenden können mit Hilfe der in der Übersicht aufgelisteten Informationen abschätzen, wie hoch die Ressourcenauslastung für den spezifischen FPGA ist (bspw. Spartan-3). Für ein besseres Verständnis der Anzahl von Slices und LUTs sollte ein Teil des Handbuches in einer Präsentation aufbereitet und den Studierenden in der Vorbereitungsphase zur Vertiefung ausgehändigt werden.

In Abschnitt 3.4.1 wurde die Beobachtung des EAP beschrieben. Dabei zeigte sich, dass die Studierenden, die mit VHDL arbeiteten, wesentliche Verständnisschwierigkeiten bei der Unterscheidung zwischen sequentiellen und nebenläufigen Programmabschnitten hatten. Die dritte Kompetenzbeschreibung greift diesen Umstand auf. Die Studierenden sollen sich die Grundlagen nebenläufiger und

sequentieller Verhaltensbeschreibungen in VHDL in der Vorbereitungsphase aneignen. Hierzu empfiehlt sich eine generelle Einführung der Begriffe *Parallelität*, *Nebenläufigkeit* und *Pipelining*. An dieser Stelle bietet sich die Verknüpfung zu Berechnungsmodellen, welche ebenfalls eine fundamentale Idee sind, an. Mit Hilfe von Petri-Netzen kann über die Problematik von synchronisierenden Prozessabschnitten diskutiert werden. Diese Diskussion lässt sich in der Präsenzphase auf Beispiele für VHDL übertragen.

In der Präsenzphase sollen die Studierenden das erste Mal ihre bisher gewonnenen Erkenntnisse in der Entwicklung eines Prototypen anwenden. Anders als beim Beispiel der Vier-Bit-UND-Komponente wird weder eine schematische Vorgabe der Systemschnittstellen noch eine empfohlene Umsetzungsart vorgegeben. Die Studierenden können mit Umsetzungen auf Registertransferebene oder mit Struktur- und Verhaltensbeschreibungen beginnen. Zusätzlich wird erwartet, dass sie eine insgesamt fehlerfreie syntaktische Beschreibung einer Komponente, d.h. Entities, Libraries und Architectures selbstständig implementieren können. In der sechsten Veranstaltung bilden die Themenbereiche Ein- und Ausgabe den Schwerpunkt der Präsenzveranstaltung. Die Studierenden sollen nach entsprechender Vorbereitung ein einfaches „User-Constraint-File“ erstellen können, in dem die Aus- und Eingabe-Pins den Eingängen und Ausgängen in der VHDL-Beschreibung zugewiesen werden. In der Vorbereitungsphase der Veranstaltung sind deswegen Grundlagen zur Analog-/Digitalwandlung empfehlenswert. Hierbei kann auf die Lehrmaterialien der EAP-Veranstaltungen eins bis vier zurückgegriffen werden (vgl. [Jas+12]).

Veranstaltung 7, 8 und 9: Eigenständige Systementwicklung und Nutzung der Hardwareparallelität

Durch die vorhergehenden Veranstaltungen wurden den Studierenden die Grundlagen rekonfigurierbarer Architekturen gezeigt. In der Doppelveranstaltung fünf und sechs haben die Teilnehmer einen eigenständigen Prototypen einer einfachen Komponente umgesetzt und mit Ein- und Ausgabemöglichkeiten des FPGAs verbunden. Die Themenschwerpunkte der folgenden Veranstaltungen leiten sich aus der Analyse zum Sinnkriterium ab. In dieser wurde untersucht, wie die Rekonfigurierbarkeit neben dem Prototyping von eingebetteten Systemen praktisch nutzbar ist (vgl. Abschnitt 5.5). Mit Verweis auf das Fortbildungskriterium wurde jedoch erwähnt, dass Rekonfigurierbarkeit zur Laufzeit nur schwer auf Bachelorniveau zu vermitteln ist und der Aspekt der Prototypenbildung deswegen weiter in den Vordergrund rücken sollte. Dieser wurde bereits teilweise in der vorherigen Doppelveranstaltung zum Horizontalkriterium behandelt. Ein bisher noch nicht in die Veranstaltungsreihe mit eingeflossenes Anwendungsgebiet befasst sich mit der hochgradig parallelen Prozessausführung, die durch FPGAs möglich ist. Die zugehörigen Begriffe wie Parallelität, Nebenläufigkeit und Pipelining wurden für das grundlegende Verständnis von VHDL-Prozessen bereits besprochen, jedoch noch nicht umgesetzt. Sie bilden den Schwerpunkt der Veranstaltungen sieben, acht und neun. In den vorherigen Veranstaltungen haben die Studierenden die grundlegenden Kompetenzen erworben, um eine eigene Entwicklung umzusetzen. Aufgrund des Umfangs des zu bearbeitenden Projektes wird eine Veranstaltungseinheit bestehend

aus drei, statt den üblichen zwei, Veranstaltungsterminen empfohlen. Diese drei Veranstaltungen sollen die folgenden Kompetenzen bei den Studierenden fördern:

1. Die Studierenden können mehrere Hardwarekomponenten in einer Hardwarebeschreibungssprache miteinander *verbinden*, um ein funktionales System zu *erstellen*.
2. Die Studierenden können die unterschiedlichen Arten von Datenabhänigkeiten *erläutern* und dadurch Aussagen über die Möglichkeiten der Parallelisierung eines Entwurfes treffen.
3. Die Studierenden sind in der Lage, den Entwurf eines einfachen sequentiellen Algorithmus parallel *umzusetzen*.

Ein beispielhaftes Projekt, das zum einen anschaulich und zum anderen gut auf rekonfigurierbaren Architekturen umzusetzen ist, ist Conways „Spiel des Leben“. In diesem Projekt wird einfaches Zellverhalten simuliert. Jede Zelle kann entweder den Zustand *Lebendig* oder *Tot* besitzen. Der folgende Katalog aus vier Spielregeln legt fest, wie sich eine Sammlung von lebenden und toten Zellen im nächsten Zeitabschnitt verändert:

- Eine tote Zelle wird im nächsten Zeitabschnitt wiederbelebt, wenn genau drei ihrer Nachbarn aktuell lebendig sind. Ansonsten bleibt sie tot.
- Eine lebendige Zelle ist im nächsten Zeitabschnitt tot, wenn sie aktuell weniger als zwei lebendige Nachbarn besitzt (Einsamkeit).
- Eine lebendige Zelle mit zwei oder drei lebendigen Nachbarn bleibt auch im darauf folgenden Zeitabschnitt lebendig.
- Eine lebendige Zelle mit mehr als drei lebendigen Nachbarn stirbt im nächsten Zeitabschnitt (Überbevölkerung).

Als Grundlage für die Zellsimulation wird eine zweidimensionale Matrix vorausgesetzt, die alle Zellen enthält und am Rand üblicherweise durch tote Zellen begrenzt ist. Dies ist notwendig, damit sich die Zellproduktion nicht über die Grenzen der Matrix hinaus fortsetzt. Als Nachbarn werden alle Zellen bezeichnet, die in der Moore-Nachbarschaft liegen, also die aktuelle Zelle an mindestens einer Ecke berühren. Für die Berechnung des Zellzustandes in Zeitabschnitt t_{n+1} sind ausschließlich Zellinformationen aus Zeitabschnitt t_n notwendig.

Conways Spiel des Lebens ist aus mehreren Gründen für die Lehrveranstaltung gut geeignet. Erstens sind entsprechende Visualisierungen eines Zellmusters durchaus motivierend, da sie eine eigene Dynamik aufzeigen, die auf Detailebene der einzelnen Regeln nicht erkennbar ist. Bevor die Studierenden mit der eigentlichen Arbeit an einem FPGA beginnen, können die Regeln und prinzipiellen Strukturen des Spieles gut auf einem PC gezeigt werden. Unter den vielen Umsetzungen des Game of Life finden sich auch Programme für Linux (bspw. *„Golly“*), die zusätzlich einen umfangreichen Katalog an Regeln bereithalten, die dynamisch ausgetauscht werden können. Diese Regeln sind der zweite Grund für die gute Eignung des Beispiels in einer Lehrveranstaltung. Sie sind sehr kompakt, einfach

zu verstehen und lassen sich deswegen entsprechend schnell in Pseudocode umsetzen oder auch verbal beschreiben. Modrow verwendet einen ähnlichen Ansatz, um eine fundamentale Idee der Theoretischen Informatik am Beispiel gekoppelter Automaten zu erläutern [Mod03]. Der Schwerpunkt liegt bei ihm deswegen auf dem theoretischen Konzept und seiner grafischen Modellierung und nur in geringem Umfang auf der eigentlichen Implementierung. Hier zeigt sich ganz deutlich die Verknüpfung zwischen der fundamentalen Idee rekonfigurierbarer Architekturen und reprogrammierbarer Designs und Berechnungsmodellen als ein Kernelement der Theoretischen Informatik. Mehr Informationen zum Game of Life finden sich in [Lif14].

Ein einfacher Ansatz zur Umsetzung der Spiellogik kann mit endlichen Automaten direkt in eine Hardwarebeschreibung mit VHDL erfolgen. Das Projekt bietet gleichzeitig jedoch genug Tiefe, um die Studierenden auch über die Lehrveranstaltung hinaus mit der Umsetzung zu beschäftigen. Beispielhaft ließe sich ein neues Regelwerk implementieren, das die von Conway geäußerten Grundregeln erweitert oder eine besonders sparsame Implementierung für die Hardwareplattform vorsieht. Bereits bei der „einfachen“ Umsetzung muss darauf geachtet werden, die Studierenden nicht zu überfordern (siehe Fortbildungskriterium). Folgendes Vorgehen zur Strukturierung der drei Veranstaltungen scheint deswegen sinnvoll:

Veranstaltung 7 In der Vorbereitungsphase zur siebten Veranstaltung erhalten die Studierenden das VHDL-Quellcode-Gerüst ohne die Umsetzung der Logikkomponente, um die Struktur des Hardwareentwurfes zu analysieren. Daraus sollen sie schließen, welche Signale von der Logikkomponente verarbeitet werden sollen und an welcher Stelle diese eine Rückmeldung an die anderen Komponenten und Prozesse geben müssen. In der Präsenzphase sollen die Studierenden als zweite Aufgabe ein entsprechendes Programmgerüst als Pseudocode erstellen, das auf seine Eignung hin gemeinsam diskutiert wird.

Neben der Diskussion der Logikkomponente des Systems sollen die Studierenden die Umsetzungen in VHDL beginnen und erste Analysen des synthetisierten Entwurfes mittels selbst erstellten Testbenches überprüfen. Der Umfang dieser Tätigkeiten ist gegenüber der Einführung zu Testbenches wesentlich erhöht, da die Anzahl und die Verknüpfung der Signale deutlich umfangreicher sind. Die Studierenden müssen zudem bestimmen, welche Signale dazu geeignet sind, die Berechnung ihres Entwurfes zu kontrollieren. Neben den Signalen der Komponente kann dies beispielsweise auch durch die Analyse des Speichers geschehen, der die Zustände der Zellen anschaulich mit Nullen und Einsen darstellt.

Veranstaltung 8 Die Hauptaufgabe in der achten Veranstaltung ist die Umsetzung des kompletten Projektes auf einer FPGA-Plattform. Deshalb sollen die Studierenden eine Aufstellung an VHDL-Elementen erhalten, die nicht synthetisierbar sind, um ihren bisherigen Entwurf in dieser Hinsicht zu überarbeiten.

In der Präsenzphase versuchen die Studierenden das Projekt zu synthetisieren und auf die Hardware zu übertragen. Das in der Vorbereitungsphase erworbene Wissen bzgl. synthetisierbaren und nicht-synthetisierbaren Elementen

muss dafür praktisch angewandt werden. Es wird davon ausgegangen, dass in Veranstaltungen sieben und acht Bezüge zu den Präsenz- und Vorbereitungsphasen der vorherigen Veranstaltungen notwendig sind, um grundlegende Konzepte erneut aufzugreifen. Hierzu gehören beispielsweise die neunwertige Logik von VHDL (*std_logic*), die Unterscheidung von Variablen und Signalen mit ihren entsprechenden Auswirkungen für die Synthese und die Synchronisation von Prozessen innerhalb einer Komponente.

Veranstaltung 9 In der neunten Veranstaltung sollen die Studierenden ein Konzept zur parallelen Abarbeitung des Game of Life entwerfen. Hierzu bieten sich auf abstrakter Ebene Petri-Netze an, welche den Ablauf der Hardwareprozesse auf konzeptueller Ebene wiedergeben. Für den intuitiven Ansatz, die Anzahl der Logikkomponenten zu erhöhen, müssen auch Hardwareconstraints wie die maximal mögliche Anzahl an Slices und Lookuptables berücksichtigt werden. Während die Durchführung Teil der Präsenzphase ist, soll das Konzept zur Parallelisierung von den Studierenden vorbereitet und gemeinsam in der Präsenzphase diskutiert werden. Hierbei sind die Betreuer in besonders intensiver Weise eingebunden, um durch ihr Fachwissen unpraktikable Ansätze aufzudecken und für alle Veranstaltungsteilnehmer einen gemeinsamen Parallelisierungsansatz vorzugeben. Auch wenn nicht alle Teilnehmer die Umsetzung im zeitlichen Rahmen realisieren können, ist damit eine gemeinsame Basis gegeben, die am Ende der Präsenzveranstaltung vorgeführt werden kann. Ansätze, die einzelnen Zellen parallel auszuwerten, sind gut umsetzbar, da Conways Spiel des Lebens zu den Prozessen gehört, die umgangssprachlich als „embarrasingly parallel“ bezeichnet werden. Damit ist gemeint, dass sich die Umwandlung von sequentiellen zu nebenläufigen Implementierungen sehr gut vollziehen lässt, da jede Aktualisierung einer Zelle unabhängig von anderen Zellen des gleichen Zeitabschnittes erfolgen kann.

In der Analyse zum Sinnkriterium wurde ebenso der Aspekt der Optimierung als besonders sinnvolles Einsatzgebiet rekonfigurierbarer Architekturen herausgestellt. Anstatt einer parallelen Umsetzung lässt sich das vorliegende Lehrkonzept auch in diese Richtung abändern. Dabei ist die Zielstellung, eine möglichst ressourcensparsame Umsetzung der Logikkomponente zu erreichen. Darüber hinaus können beide Aufgaben natürlich auch miteinander verbunden werden.

Im Anhang dieser Arbeit ist eine einfache sequentielle Umsetzung von Conways Game of Life zu finden, die als Basis für die geschilderte Projektaufgabe in Lehrveranstaltung sieben, acht und neun dienen kann. Die Aufteilung der Komponenten und deren Kommunikationsstruktur ist in Abbildung 6.6 zu sehen.

Bei der vorliegenden VHDL-Umsetzung wurde auf die explizite Trennung von Simulationslogik, Datenverarbeitung und Datenspeicherung geachtet. Damit können die Studierenden die eigenen Entwicklungsarbeiten an definierten Schnittstellen anbinden und benötigen für ihre Arbeiten kein besonders tiefes Verständnis der restlichen Architektur. Für die Weiterentwicklung der VHDL-Beschreibung hin zu einer parallelen Auswertung der Zellstatus ist zumindest die Änderung der Datenstrukturkomponente notwendig. Die referenzierte VHDL-Beschreibung ist

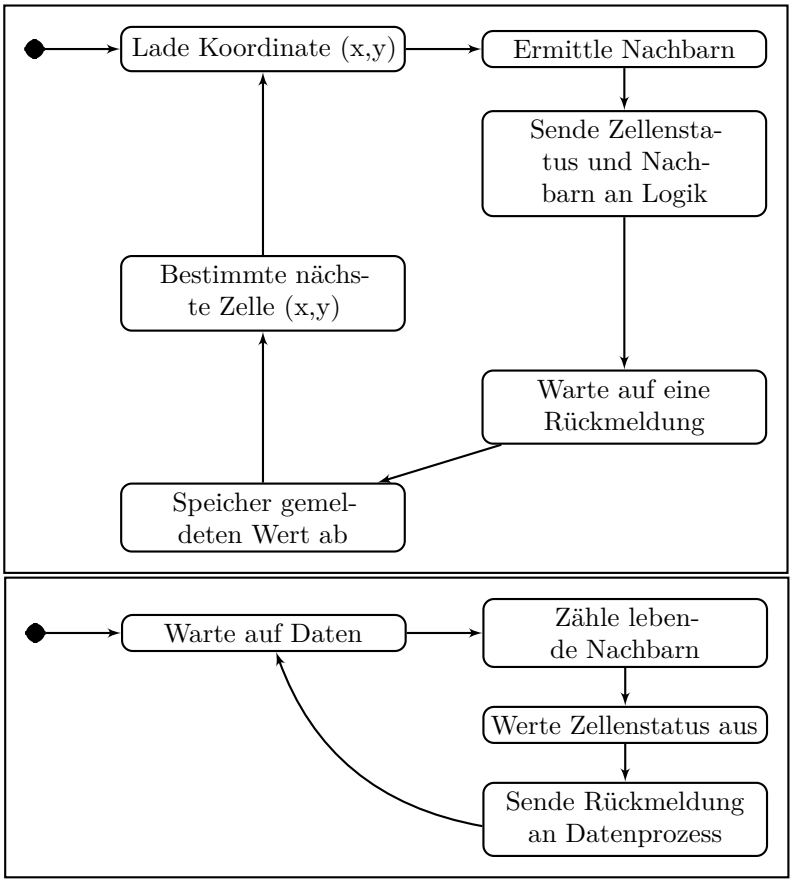

Abbildung 6.6.: Vereinfachtes Aktivitätsdiagramm zur Umsetzung des Spiels des Lebens

um ein User-Constraint-File zur Spezifikation der PIN-Belegung zu erweitern. Dieses wurde nicht angegeben, da die jeweiligen Belegungen abhängig von der vorhandenen FPGA-Plattform sind. Weitere Informationen zur Umsetzung finden sich in Abschnitt B.

Veranstaltung 10: Alternativen und dynamische Rekonfiguration

Die zehnte Veranstaltung orientiert sich an der Analyse zum Varianzkriterium. Im Fall von rekonfigurierbaren Architekturen ergab die Analyse, dass zwar keine gleichwertige Idee innerhalb der Technischen Informatik oder verwandten Disziplinen existiert, jedoch die Umsetzungsmöglichkeiten auf Software- und Hardwareebene variieren. Ebenso wurde in der Analyse auf das Zusammenspiel von rekonfigurierbarer Architektur und Betriebssystemen hingewiesen, das in der letzten Veranstaltung wei-

ter vertieft werden sollte. Beispielhaft können folgende Kompetenz als Zielstellung für den Veranstaltungsentwurf gesehen werden:

1. Die Studierenden sind in der Lage, verschiedene Umsetzungsmöglichkeiten von Rekonfiguration auf technischer Ebene zu *erklären*.
2. Die Studierenden können die besonderen Charakteristka von Betriebssystemen für eingebettete Systeme im Allgemeinen und rekonfigurierbaren Systemen im Speziellen *erläutern*.

Die erste Kompetenz betrifft insbesondere alternative Umsetzungen wie CPLDs oder Hardwarebeschreibungssprachen wie Verilog oder SystemC. Im Vergleich zu vorherigen Veranstaltungen ist bereits an den Kompetenzbeschreibungen zu erkennen, dass die Studierenden auf den unteren Ebenen der Fuller-Taxonomie gefördert werden sollen. In den Vorbereitungsphasen können entsprechende Rechercheaufgaben an die Studierenden verteilt werden, die in den Präsenzphasen gemeinschaftlich ausgewertet und diskutiert werden. In der Präsenzphase können kleinere Beispielaufgaben das grundlegende Verständnis zu alternativen technischen Umsetzungen fördern. Die in Veranstaltung zwei vorgeschlagene Umsetzung eines UND-Bausteins kann auch in dieser Veranstaltung für einen Überblick in SystemC oder Verilog genutzt werden. Marwedel erläutert in seinem Bericht zur Realisierung des Flipped Classroom-Ansatzes, dass teilweise auch Fragenkataloge in den Präsenzphasen an die Studierenden ausgegeben wurden [ME14]. Dieses Modell scheint für diese Veranstaltung ebenso sinnvoll, da der Umfang an praktischen Aufgaben aufgrund der Breite der Themen gering gehalten wird. Den Studierenden sollte schnell ersichtlich sein, dass Hardwarebeschreibungssprachen untereinander konzeptionelle Ähnlichkeiten besitzen.

Ein weiterer Aspekt, der in der Analyse zum Varianzkriterium genannt wurde, ist der der Betriebssysteme. Für rekonfigurierbare Architekturen im Allgemeinen hat ein Betriebssystem, bzw. eine Laufzeitumgebung, oft die Aufgabe, das System nach der Verarbeitung äußerer Signale oder Bedingungen dynamisch neu zu konfigurieren. Während die Entwicklung eigener dynamisch rekonfigurierbarer Hardwareeinheiten auf Bachelorniveau aufgrund des mangelnden Vorwissens schwierig ist, kann zumindest auf theoretischer Ebene darauf eingegangen werden, welche technischen Realisierungen des Konzeptes praktikabel sind. Es sollte darauf hingewiesen werden, dass die Möglichkeiten zur Rekonfiguration zumeist herstellerabhängig sind. Einen guten Überblick über die Vor- und Nachteile verschiedener Rekonfigurationsmöglichkeiten geben Hauck und Dehon [HD10].

Zusammenfassung und Einordnung des Veranstaltungskonzeptes

Die in den vorherigen Abschnitten erläuterte Veranstaltung bezieht ihren Lerninhalt und intendierten Kompetenzgewinn aus den Analysen der fünf Kriterien. Wie im Konzept zu sehen ist, besteht die Veranstaltung aus einer Reihe praktischer und theoretischer Lehr-/Lernelemente, die didaktisch aufeinander aufbauen, dabei aber

unterschiedliche Kriteriendimensionen und damit unterschiedliche Kompetenzen abdecken.

In Abschnitt 2.2 wurde eine Taxonomie vorgestellt, mit der Praxisberichte von Lehrveranstaltungen zum Anwendungsgebiet eingebetteter Systeme besser vergleichbar gestaltet werden können. Diese Taxonomie kann auch auf die beschriebene Lehrveranstaltungsreihe angewandt werden, um den didaktischen Kontext geeignet zu definieren. Die Einordnung der Lehrveranstaltung zeigt Abbildung 6.7.

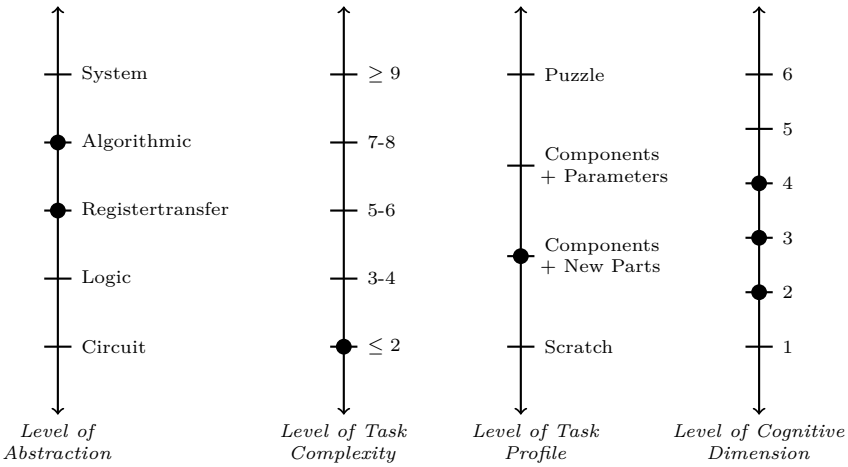

Abbildung 6.7.: Einordnung der Lehrveranstaltung mit Hilfe der vorgestellten Taxonomie

Für das Abstraktionsniveau sind die Veranstaltung zum Horizontal- und zum Sinnkriterium maßgebend, da diese den höchsten Anteil typischer Entwicklungsarbeiten enthalten. Die Veranstaltungen zum Fortbildungs- und Varianzkriterium zeichnen sich durch Kompetenzniveaus aus, die hauptsächlich auf deklarativen Wissensebenen einzuordnen sind. Durch die Verwendung von VHDL auf der einen und Berechnungsmodellen auf der anderen Seite liegt das Abstraktionsniveau entsprechend auf der algorithmischen Ebene und der Registertransfer-Ebene.

Die Einordnung der Aufgabenkomplexität deckt sich mit der Zielsetzung der Veranstaltung, weniger Tiefen- und mehr Breitenwissen zu fördern, um die Verknüpfung möglichst vieler fundamentaler Ideen zu ermöglichen. Aus diesem Grund sind nur zwei der in Abschnitt 2.2 erwähnten Charakteristika in den Beispielaufgaben der Lehrveranstaltungskonzeption zu finden. Eng verbunden mit der Idee rekonfigurierbarer Systeme ist die „Nutzung paralleler oder nebenläufiger Prozesse“. Ebenso kann die Berücksichtigung nicht-funktionaler Anforderungen den Aufgaben zugeordnet werden, wenn für die Hardwareplattformen Ressourcenconstraints zu beachten sind. Für die meisten der vorgeschlagenen Aufgaben ist dies jedoch nicht zutreffend und deren Komplexität deswegen niedrig.

Das Aufgabenprofil der Lehrveranstaltung ist hauptsächlich auf der zweiten Ebene, also dem Erstellen neuer Komponenten mit Hilfe bestehender Komponenten und

Module, einzuordnen. Wie in der Analyse zum Fortbildungskriterium erläutert wurde, kann die didaktische Begleitung der Studierenden bei der Komponentenentwicklung durch bereits vorgefertigte Module geschehen. In Bezug auf das in Veranstaltung sieben, acht und neun vorgestellte Projekt des Spiels des Lebens wurde dies durch umfangreiche Vorarbeit der Speicher-, Anzeige- und Datenstrukturkomponenten gewährleistet. Die Aufgabe der Studierenden ist, mit Hilfe dieser vorgefertigten Komponenten die Logikkomponente selbstständig umzusetzen. Die Einstufung auf die vierte Ebene dieser Achse wäre durch die Einführungsveranstaltungen und das Konzept der Schieberegler ebenso möglich gewesen. Da dies jedoch nicht den Schwerpunkt der Lehrveranstaltungskonzeption darstellt, wurde darauf verzichtet.

Die Kompetenzbeschreibungen der fünf Veranstaltungsblöcke enthalten eine Zuweisung zu kognitiven Kompetenzdimensionen. Besonders wichtig sind dabei die Kompetenzzielstellungen für die Veranstaltungen zum Horizontal- und Sinnkriterium, da hier keine weiteren Grundlagen mehr zu vermitteln sind und die Aufgaben im Kontext typischer Entwicklerhandlungen liegen. Die Ebenen *Anwenden* und *Analysieren* sind deswegen hervorzuheben. Ersteres betrifft die Umsetzung von Pseudocode in eine synthetisierbare VHDL-Beschreibung und Letzteres die Auswertung der Simulationsdaten hinsichtlich einer korrekten Funktionsweise. Die Veranstaltungen des Fortbildungskriteriums und des Varianzkriteriums fördern hauptsächlich Kompetenzen, die auf den unteren zwei Ebenen der Niveaustufung nach Anderson und Krathwohl, bzw. Fuller et al. anzuordnen sind. Aus diesem Grund ist ein zusätzlicher dritter Eintrag auf der Ebene des *Verstehens* gerechtfertigt.

7. Zusammenfassung, Fazit und Ausblick

In Abschnitt 7.1 dieses Kapitels wird ein kurzer Überblick über die wesentlichen Meilensteine des Forschungsprozesses und den aus diesen resultierenden Ergebnissen und Schlussfolgerungen gegeben. Der darauf folgende Abschnitt 7.2 vergleicht die Forschungsergebnisse und den geleisteten Forschungsbeitrag mit den ursprünglich definierten Forschungszielen und lässt somit ein Fazit über den Erfolg der Arbeit zu. Im Ausblick wird erläutert, welche Aufgaben die zukünftige Fachforschung verfolgen sollte, um die Hochschuldidaktik für Technische Informatik und angrenzende Disziplinen im Sinne der Forschungsmethodik weiter zu fundieren (siehe Abschnitt 7.3).

7.1. Zusammenfassung

Wie in Kapitel eins erläutert, sind eingebettete Systeme national und international ein wichtiger Anwendungsbereich, mit einer dementsprechend großen Nachfrage an gut ausgebildeten Fachkräften. Die Ausbildung dieser Mitarbeiter wird besonders von der Frage nach langfristig relevanten Fachinhalten und Kompetenzen bestimmt. Mit der in dieser Arbeit vorgestellten Forschungsmethodik wird ein Beitrag zur Beantwortung dieser Fragestellung geliefert.

Im Rahmen des DFG-Projektes KOMINA konnte die Grundlage für die in dieser Arbeit vorgestellten Forschungsansätze gelegt werden, indem ein empirisch verfeinertes Kompetenzstrukturmodell erstellt wurde. Aufbauend auf diesem Strukturmodell wurde eine Laborveranstaltung an der Universität Siegen grundlegend neu strukturiert und kompetenzorientiert ausgerichtet. Um den Erfolg dieser Umstrukturierung nach wissenschaftlichen Kriterien zu analysieren, war eine Beobachtung der Durchführung notwendig. In dieser Beobachtung zeigte sich, dass viele Studierende Probleme bei der richtigen Schreibweise von C-Quellcode und Registerzuweisung hatten. Um die entsprechenden Kompetenzen zu fördern, wurde die Applikation „ELVE“ entwickelt (siehe Abschnitt 3.4.3). Wie auch von anderen Wissenschaftlern im Bereich der Technischen Informatik angemerkt, wird die Lehre in diesem Bereich durch die mangelnde Verfügbarkeit von Laborinstrumenten und Hardwaresystemen beeinträchtigt. Der in dieser Arbeit entwickelte Virtual Workspace greift den Umstand auf und stellt Studierenden einen virtuellen Arbeitsplatz mit allen nötigen Entwicklungswerkzeugen zur Verfügung. Er kann von zuhause, wie auch bei der Durchführung im Labor genutzt werden (siehe Abschnitt 3.4.2).

Insgesamt zeigte sich aus der Beobachtung und der Evaluation der Hilfsmittel, dass nur die Struktur von Kompetenzbeziehungen durch die KOMINA-Ergebnisse beschrieben werden konnte, diese für eine praktische Umsetzung in einer Lehrveranstaltung jedoch nicht optimal geeignet waren. Ausgehend von diesem Umstand wurde ein Konzept gesucht, das die KOMINA-Ergebnisse aufgreift und feiner strukturiert. Neben der Frage nach einem geeigneten methodischen Vorgehen war dabei zu bedenken, dass konkretere Umsetzungsvorschläge als die des KOMINA-Forschungsprojektes Gefahr laufen, so technikspezifisch zu sein, dass sie aufgrund der Schnelllebigkeit der Disziplin in kurzer Zeit überholt und damit fachdidaktisch von geringem Nutzen sein werden. Nach einer Recherche bestehender fachdidaktischer Vorgehensweisen zur Lösung dieser Aufgabenstellung konnten die „fundamentalen Ideen der Informatik“ von Schwill als geeignet identifiziert werden. Kern des Vorgehens ist es, nicht mehr eine spezifische Technik, sondern die mit ihr verbundene Idee an die Studierenden zu vermitteln. Die Relevanz dieser Idee wird mit Hilfe eines Kriterienkataloges bestimmt. Eine Adaption auf die Zielgruppe und das Anwendungsgebiet des Forschungsprozesses sowie eine grundlegende Umstrukturierung des Vorgehens waren aus Gründen der Problemstellung und neuerer wissenschaftlicher Erkenntnisse ratsam (siehe Kapitel 4). Hinzu kommt, dass die bisherigen Umsetzungen des Verfahrens ausschließlich auf schulischem Niveau durchgeführt wurden und zudem nur einen exemplarischen Charakter besaßen. Deswegen hat sich der Autor dazu entschieden, eine Durchführung für das Anwendungsgebiet anhand einer normativ ermittelten Themensammlung durchzuführen. Im Forschungsprozess wurden hierfür die ACM „Transactions on Embedded Computing“ der letzten fünf Jahre nach Forschungsschwerpunkten hinsichtlich dem Entwurf eingebetteter Systeme analysiert. Die vorher durch das KOMINA-Projekt als besonders wichtig ermittelten Kompetenzen wurden auf einschlägige Fachbegriffe und Prozessdimensionen untersucht, die im folgenden Forschungsabschnitt als Auswahlkriterien für die zu analysieren Themensammlung genutzt wurden.

Jedes der in früheren Arbeiten vorgestellten Kriterien wurde auf die Eignung hinsichtlich des Anwendungsgebietes und der Zielgruppe untersucht und abgeändert. Für die Zielgruppe war maßgeblich, dass der Bezug der fundamentalen Ideen nicht mehr im Alltags-, sondern im Berufsleben verankert sein sollte. Dies bezieht sich für Studierende hauptsächlich auf die Bereiche der praktischen Arbeit in Unternehmen und den eher theoretisch orientierten Kompetenzen in Wissenschaft und Forschung. Die Interdisziplinarität des Anwendungsgebietes verlangt eine breite Betrachtung aller Kriterien. Hierbei sind artverwandte Fachtraditionen wie die Luft- und Raumfahrttechnik, Regelungstechnik, Physik oder Elektrotechnik in einem höheren Maße zu berücksichtigen, als dies im ursprünglichen Ansatz von Schwill der Fall war. Im Kriterienkatalog dieser Arbeit sind das Horizontalkriterium, das Fortbildungskriterium, das Zeitkriterium, das Sinnkriterium und das Varianzkriterium enthalten. Das Horizontalkriterium sichert die breite Anwendbarkeit der Idee und ihren Bezug zu den Ebenen des Entwicklungsvorgehens. Im Fortbildungskriterium wird analysiert, inwiefern die Vermittlung einer Methode oder Sichtweise auf Bachelorebene möglich ist und welche Grundlage sie für andere Ideen bereitstellt. Das Zeitkriterium überprüft, ob eine Idee im Anwendungsbereich auch in der Vergangenheit beobachtbar war und welche Tendenz sich daraus für die kommenden Jahre ableiten lässt. Da Studierende nach ihrer Fachausbildung

entweder im industriellen oder im wissenschaftlichen Bereich arbeiten werden, analysiert das Sinnkriterium die Relevanz der Idee für diese beiden Berufswege. Neben den Kriterien wurde ebenso die Vorgehensweise zu deren Anwendung weiterentwickelt, um den Katalog an zentralen Konzepten theoretisch zu stützen. Hierfür ist zusätzlich das Varianzkriterium eingeführt worden, um Ideen mit starke Bezügen untereinander aufzufassen und zu differenzieren.

Die im Anschluss durchgeführte Anwendung der entwickelten Kriteriensammlung dient der Veranschaulichung der Methodik und stellt gleichzeitig einen Katalog an zehn zentralen Ideen und Auffassungen für das Anwendungsgebiet dar, die für die inhaltliche Strukturierung einer Lehrveranstaltung oder einer Curriculaempfehlung hilfreich sind (siehe Kapitel 5). Die fundamentalen Ideen für die Entwicklung eingebetteter Systeme sind:

- Berechnungsmodelle,
- Synthese und Steuerung paralleler Systeme,
- Hardware/Software Co-Design,
- Component-based Design und Intellectual Property,
- Betriebssysteme, Middlewares und Real-Time Kernel,
- Reliable Design,
- Domänenspezifische Anwendungen und Methoden,
- Methoden der formalen Verifikation und Validierung,
- Rekonfigurierbare Architekturen und reprogrammierbarer Designs,
- Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen

Im letzten Schritt der Forschungsmethodik wurde untersucht, inwiefern aus dem Ansatz zur Bestimmung zentraler Themengebiete auch eine organisatorische Struktur für eine Lehrveranstaltung abgeleitet werden kann (siehe Kapitel 6). Dazu wurden aktuell diskutierte fachdidaktische Ansätze wie der „Flipped Classroom“ oder das „Constructive Alignment“ mit den fundamentalen Ideen verbunden und in ein Lehrkonzept überführt. Eine exemplarische Ausgestaltung des Konzeptes wird anhand des Themengebietes der rekonfigurierbaren Architekturen und reprogrammierbaren Designs gegeben und durch in VHDL implementierte Aufgaben verdeutlicht.

7.2. Fazit

Wie in Abschnitt 1.1 erläutert, sind die folgenden Forschungsziele maßgeblicher Gegenstand für die Evaluation der Arbeit:

1. Beobachtung des Hardwarelabors und darauf aufbauende Entwicklung fachdidaktisch begründeter Lehr-/Lernunterstützungen.

2. Erforschung einer fachdidaktischen Methode zur Bewältigung der Schnelllebigkeit technischer Konzepte und Sichtweisen für die Entwicklung eingebetteter Systeme.
3. Konzeptionelle Umsetzung von Lehr-/Lernempfehlungen auf Basis der vorgestellten fachdidaktischen Forschungsmethode.

Die folgende Diskussion der Ergebnisse orientiert sich inhaltlich an den drei genannten Zielstellungen und zeigt auf, wie das in der Arbeit vorgestellte Forschungsprojekt einen Beitrag zu deren Beantwortung liefert.

Beobachtung des Hardwarelabors

Die Beobachtungen der Studierenden innerhalb des Praktikums sind ein wichtiger Forschungsbeitrag für die stetige Verbesserung der Kompetenzstrukturierung und der Begründung von zu entwickelnden Lehr-/Lernsystemen. Unter anderem hat sich hieraus der Bedarf einer weiteren Verfeinerung der Kompetenzbeschreibungen bzw. der inhaltlichen Erweiterung der Kompetenzbeschreibungen durch die fundamentalen Ideen ergeben.

Die Beobachtung hat den praktischen Bedarf an einer organisatorischen Unterstützung der Studierenden hervorgebracht, die mit dem Virtual Workspace umgesetzt wurde. Dazu wurde ein USB-Stick an die Studierenden verteilt, auf dem sich ein virtueller Arbeitsplatz befindet, der sowohl in der Vorbereitungsphase als auch in der eigentlichen Labordurchführung genutzt werden konnte. In zwei Evaluationen am Ende der jeweiligen Praktikumsdurchführung zeigte sich, dass ein Großteil der Studierenden das Konzept für sinnvoll hält. Hierdurch wurde der in der wissenschaftlichen Diskussion oft geäußerte Umstand bekräftigt, dass die fachdidaktische Unterstützung von Lehrveranstaltungen im Bereich der Technischen Informatik auf Hilfsmittel angewiesen ist, die den Studierenden den Zugang zu Laborinstrumenten und Hardwareplattformen erleichtern.

Mit der Lernumgebung ELVE konnte ein zweites Hilfsmittel für die Studierenden zur Verfügung gestellt werden. Dieses basiert darauf, dass in der Beobachtungsphase der Veranstaltung Defizite bei den Studierenden hinsichtlich der hardwarenahen Programmierung aufgedeckt wurden. Die Lernumgebung ist dabei für die Problemstellung im EAP ausgelegt und kann eine aufwendige Umstellung zwischen Lernszenarien für die Studierenden vermeiden. Des Weiteren ist ELVE browserbasiert und kann, genau wie der Virtual Workspace auch, außerhalb des Laborraumes genutzt werden. ELVE leistet einen Beitrag zur kompetenzförderlichen Vorbereitung einer Lehrveranstaltung. Die Beobachtungen im Umgang mit ELVE zeigten, dass mehr Arbeit notwendig ist um eine fachdidaktische Stützung nicht nur in der Lernumgebung, sondern auch in später verwendeten Entwicklungsumgebungen bereit zu stellen. Aufgrund seines modularen Aufbaus kann ELVE auch für andere Lehrveranstaltungen genutzt werden.

Zudem wurde eine Taxonomie mit vier Dimensionen vorgestellt, die den Vergleich zwischen Lehrveranstaltungen erleichtert. Die Dimensionen beschreiben das Abstraktionsniveau, die Aufgabenkomplexität, das Aufgabenprofil und die intendierte

Ebene der Kompetenzförderung. Die Taxonomie wurde auf die Lehrveranstaltungskonzeption in Kapitel 6 angewandt, um folgenden Forschungsarbeiten die Bewertung und Vergleichbarkeit des fachdidaktischen Ansatzes zu ermöglichen. Auch wenn die Taxonomie keinen direkten Einfluss auf die durchgeführten Laborveranstaltungen hat, ist damit zu rechnen, dass die gestiegene Vergleichbarkeit von praktischen Durchführungen im Anwendungsgebiet der eingebetteten Systeme auf lange Sicht dazu führt, dass Lehr-/Lernunterstützungen entwickelt werden, die auch in Veranstaltungen wie dem HaPra oder dem EAP sinnvoll einsetzbar sind. Dies liegt darin begründet, dass alle praktischen Durchführungen hinsichtlich der in der Taxonomie verwendeten Dimensionen verglichen werden können und somit ein Urteil darüber möglich ist, welche Veranstaltungsvarianten besonders oft auftreten und deswegen primär durch Lernumgebungen zu unterstützen sind.

Forschung zur technologischen Schnelllebigkeit

Die Ausgangslage für die Bestimmung eines geeigneten fachdidaktischen Ansatzes ist die Zielstellung, langfristig relevante Kompetenzen an die Lernenden zu vermitteln. Bestehende fachdidaktische Konzepte sind in den meisten Fällen allgemeindidaktisch und auf die Schulbildung konzipiert gewesen. Empfehlungen für Lehr-/Lerninhalte auf Hochschulebene in Form von Curriculaempfehlungen existieren zwar, sind jedoch durch einen starken Technologiefokus nur zeitlich eingeschränkt für die Veranstaltungskonzeption nutzbar. Die Methode der *fundamentalen Ideen* wurde als Ausgangslage für die Entwicklung eines eigenen Konzeptes gewählt.

Der Ansatz der fundamentalen Ideen wurde in mehrfacher Hinsicht erweitert. Durch die Konzeption von Beurteilungsmöglichkeiten der Kriterien ist ein Beitrag geleistet worden, um das Konzept auch für andere Anwendungsgebiete und Disziplinen objektiver durchzuführen. Durch die Verwendung einer durch Experten erstellten Sammlung an Themenbereichen konnte zudem gewährleistet werden, dass alle Bereiche der Entwicklung eingebetteter Systeme berücksichtigt worden sind. Dieses Vorgehen kann somit als Vorlage für artverwandte Forschungsprojekte dienen und zeigt, wie den Kritikpunkten im Vorgehen von Schwill begegnet werden kann.

Frühere Forschungsprojekte haben lediglich die Konzepte beschrieben, die einen fundamentalen Aspekt einer Disziplin bilden. Die Mehrzahl der im Anhang analysierten Themenbereiche sind keine fundamentalen Ideen und zeigen somit auf, wo die Trennung zwischen positiv und negativ ausgewerteten Themenbereichen liegt.

Ein weiterer wichtiger Aspekt wurde durch die Adaption des Konzeptes für Hochschulen beigetragen. Im Gegensatz zur schulischen Bildungsebene ist ein Spiralcurriculum, wie es ursprünglich mit den fundamentalen Ideen verbunden ist, auf Hochschulniveau schwierig umzusetzen. Es wurde erläutert, welche Rahmenbedingungen insbesondere das Fortbildungskriterium garantieren muss, um eine Niveaustufung im Sinne des Spiralcurriculums auch auf Hochschulniveau zu ermöglichen. Als Basis dieser Untersuchung wurde der Qualitätsrahmen Deutscher Hochschulen gewählt, der eine anwendungsgebiet- und technikunabhängige Einteilung von Prozessdimensionen ermöglicht. Zukünftige Forschungsprojekte, die das Konzept der

fundamentalen Ideen auf Hochschulebene einsetzen wollen, profitieren ebenso von der Weiterentwicklung des Sinnkriteriums, das nun ebenfalls die Berufswege von Bachelor- und Masterstudierenden berücksichtigt, indem die wissenschaftliche und praktische Relevanz einer Idee gefordert wird.

Konzeptionelle Lehrveranstaltungsumsetzung

Mit Hilfe des weiterentwickelten Forschungsansatzes der fundamentalen Ideen konnte eine Sammlung an besonders wichtigen und langfristig relevanten Lehr-/Lerninhalten bestimmt werden. Im Gegensatz zu den Forschungsarbeiten im KOMINA-Projekt steht damit nicht nur ein Ergebnis für zukünftige wissenschaftliche Arbeiten zur Verfügung, sondern gleichzeitig ein theoretisches Werkzeug, mit dem die inhaltliche Struktur einer Lehrveranstaltung konzipiert werden kann. Dabei helfen die fünf Kriterienanalysen, indem sie die Themenbezüge aufdecken, die Studierende für das Verständnis der zu vermittelnden technischen Idee und deren Einsatzzweck benötigten. Es konnte gezeigt werden, dass die logische Struktur einer beispielhaften Veranstaltung zur fundamentalen Idee „rekonfigurierbarer Systeme und reprogrammierbarer Designs“ mit den Fachinhalten beginnen sollte, die in der Analyse des Fortbildungskriteriums aufgedeckt wurden. Folgend erhalten die Studierenden in den nächsten Veranstaltungen einen Überblick über mögliche Anwendungszenarien der Idee, indem eine Veranstaltung zu den Themenbereichen durchgeführt wird, die in der Analyse zum Horizontalkriterium aufgedeckt wurden. In der dargestellten Lehrveranstaltung besitzt der Veranstaltungsblock zum Sinnkriterium den zeitlich größten Umfang. Die Studierenden können die wesentliche Idee rekonfigurierbarer Systeme ausführlich praktisch testen und erproben. Die letzte Veranstaltung bezieht ihre fachlichen Inhalte aus der Analyse zum Varianzkriterium, die Alternativen und artverwandte Ideen aufgezeigt hat.

Die in dieser Arbeit vorgestellte Lehrveranstaltungskonzeption ist komplett aus dem Prozess der Kriterienanalyse abgeleitet und stellt somit einen neuartigen Beitrag zur Fachdidaktik der Technischen Informatik dar. Die Methodik der fundamentalen Ideen wurde in der bisherigen Forschung ausschließlich für die Ebene des Curriculaentwurfes angewandt und noch nie direkt auf Lehrveranstaltungsniveau umgesetzt. Thematisch ist das vorgestellte Konzept nicht an das Anwendungsgebiet gebunden und lässt sich voraussichtlich auch auf weitere Bereiche übertragen. Das dritte Teilziel dieser Arbeit wurde deswegen um eine entscheidende Facette erweitert, indem ausgehend von der Forschungsmethode erstmals ein Konzept auf Lehrveranstaltungsebene vorgestellt wurde, das sich direkt aus dem Analyseprozess der fundamentalen Ideen ableitet.

7.3. Ausblick

Die vorliegende Arbeit stellt neben der theoretischen Fundierung besonders wichtiger Themengebiete für die Disziplin auch eine Variante der Umsetzung vor, die Synergieeffekte zur Forschungsmethodik aufzeigt. Eine im Kontext fundamentaler

Ideen zu wenig erforschte Fragestellung ist, wie Kompetenzen, die durch die Methodik gefördert wurden, evaluiert werden können. Im Gegensatz zu traditionellen Vorlesungskonzepten richten sich fundamentale Ideen explizit auf einen langfristigen Kompetenzzuwachs aus. Die Evaluation oder Beobachtung einer einzelnen Veranstaltungsreihe würde diesem Umstand nach Ansicht des Autors nicht genug Rechnung tragen. Dafür scheinen Forschungsprojekte, die die Studienzeit und die spätere Berufswelt von Studierenden umfassen, eher für eine fundierte Aussage geeignet zu sein. Vermutlich lässt sich über eine solche Forschung auch ein Bezug zu artverwandten Forschungsfragen aus dem Bereich des lebenslangen Lernens herstellen, da hierbei die Kompetenzbeurteilung ebenfalls über einen längeren Zeitraum sinnvoll ist.

Ein für die weitere Forschung interessanter Problembereich wäre die Untersuchung von anderen Entwicklungsphasen in der Entwicklung eingebetteter Systeme. Die vorliegende Arbeit berücksichtigt insbesondere die Aspekte der Systemarchitektur und des Systementwurfes, da hierbei übergeordnete Kenntnisse aller weiteren Entwicklungsphasen notwendig sind.

Wie bereits geschildert, zeichnet sich das Anwendungsgebiet zudem durch viele Einflüsse verwandter Disziplinen aus, die sich schlussendlich auch in den vielfältigen Anwendungsgebieten eingebetteter Systeme niederschlagen. Eine interessante Fragestellung mit großer praktischer Relevanz ist, ob es für bestimmte Anwendungsgebiete wie der Automobiltechnik oder der Luft- und Raumfahrt einen spezifischeren Katalog an fundamentalen Ideen gibt. Durch weitere Forschung in dieser Richtung sind zusätzliche Bezüge zu anderen Themengebieten zu erwarten, welche helfen können, die interdisziplinäre Struktur des Anwendungsgebietes auch außerhalb eines fachdidaktischen Kontextes besser zu verstehen.

Die Lernumgebung „ELVE“ ist auf die Programmierung von Mikrocontrollern ausgerichtet und könnte in einem weiteren Forschungsprojekt für FPGA-basierte Praxisveranstaltungen umgestellt werden. Gerade bei komplexen industriellen Werkzeugen, wie der Entwicklungsumgebung von Xilinx zur Beschreibung rekonfigurierbarer Architekturen, ist es naheliegend, dass auch die Vorbereitung einer Veranstaltung durch fachdidaktisch begründete Informatiksysteme gefördert werden sollte.

A. Auswertung der Ideenanalysen

Nachdem die Menge der zu untersuchenden Ideen in Kapitel 5.4 eingegrenzt wurde, gilt es im Folgenden, den Kriterienkatalog auf alle ermittelten Themenschwerpunkte anzuwenden. Dabei wird zuerst immer eine Definition oder Zusammenfassung der Begriffe gegeben, die darauf folgend auf eine „Idee dahinter“ zu untersuchen sind, sofern diese nicht direkt ersichtlich ist. Die Bewertung jeder Idee hinsichtlich des analysierten Kriteriums wird mit Hilfe der in Abschnitt 5.3 eingeführten dreistufigen Skala vollzogen.

Wie in Kapitel 4.3 geschildert, ist bei der Analyse eines Themas oder einer Idee der Wechsel des Detailgrads unumgänglich. Der Themenbereich „Models of Computation“ (MoCs) bspw. wird sich mit der konkreten Technik einerseits (Petri-Netze, Kahn-Process-Networks, etc.), aber vor allem mit den dahinter befindlichen Ideen wie *Modellbildung* oder *Formalisierung von Anforderungen* befassen. Bei der nachfolgend aufgeführten Analyse ist dies zu berücksichtigen, um als Leser die Wechsel der Technik- und Abstraktionsebenen nachvollziehen zu können.

Wenn ein Kriterium nicht erfüllt wurde, ist dieser Umstand am Ende des jeweiligen Abschnitts deutlich beschrieben. In den restlichen Fällen ist auch ohne einen expliziten Hinweis von der Erfüllung des gerade diskutierten Kriteriums auszugehen. Schließlich ist zu beachten, dass viele der im Folgenden analysierten Themengebiete Oberbegriffe für komplette Forschungszweige der Elektrotechnik, der Informatik, der Mathematik, der Regelungstechnik oder verwandter Disziplinen darstellen und entsprechend vielschichtig untersucht werden können. Insofern ist die Analyse als eine Strukturierung hinsichtlich der Entwicklung eingebetteter Systeme zu sehen, die für andere Anwendungsgebiete zu unterschiedlichen Ergebnisse führen kann.

A.1. Berechnungsmodelle

„Models of Computation“ (dt. Berechnungsmodelle) sind ein maßgeblicher Bestandteil des Entwurfs eingebetteter Systeme. Wie in den meisten Anwendungsgebieten informatischer Methoden und Werkzeuge ist ein zu erstellendes System vor der Umsetzung in ein Modell mit allen wichtigen Eigenschaften zusammenzufassen, um nicht auf implizites Wissen einzelner Projektteilnehmer angewiesen zu sein und damit eine einheitliche Entwicklungsbasis zu besitzen [AP04]. Lavagno et al. geben eine kompakte Definition des Themenschwerpunktes:

> „A fundamental aspect of system design is the specification process. We advocate using an unambiguous formalism to represent design specifications and design choices. This facilitates tremendously efficiency

> of specification, formal verification and correct design refinement, optimization, and implementation. This formalism is often called model of computation.“ [LSS99, S. 1]

Die meisten klassischen „Models of Computation“ enthalten ursprünglich keine syntaktischen Komponenten zur Beschreibung von Nebenläufigkeit (siehe [Jan04]). Als Gegenstand wissenschaftlicher Diskussion wird deswegen der Nebenaspekt „Concurrency“ aus der folgenden Analyse ausgeschlossen, zumal dieser, in größerem Umfang, auch in der in Kapitel A.2 geschilderten Idee „Synthese und Steuerung paralleler Systeme“ enthalten ist.

Horizontalkriterium (3) Meistens sind für die unterschiedlichen Hardware- und Softwarevarianten, die für die Implementierung des Endproduktes herangezogen werden, verschiedene formalisierte Berechnungsmodelle notwendig. Deswegen ist die Idee eng mit der Anschauung von Informatiksystemen verknüpft. Zu unterscheiden sind bspw. kontinuierliche und diskrete Systeme sowie die entsprechenden Untergruppierungen (bspw. eventdiskret, zeitdiskret). Je nach Blickwinkel auf ein zu entwerfendes System sind die Berechnungsmodelle auszuwählen, die Eigenschaften vom System vereinfachen, ohne dabei wesentliche Charakteristika zu vernachlässigen [Jan04]. Zu Berechnungsmodellen in diesem Sinne zählen unter anderem „Process Networks“, Datenflussdiagramme, Petri-Netze oder endliche Automaten.

Modelle für andere Entwicklungsphasen finden sich in der Anforderungserhebung oder in Schnittstellenspezifikationen zu externen Komponenten (zugehörig zur Systemintegration). Eine detaillierte Übersicht, inwieweit sich gängige Berechnungsmodelle unterscheiden und in welchen Anwendungsgebieten sie zu finden sind, geben Jantsch und Sander [JS05].

Fortbildungskriterium (3) Berechnungsmodelle eignen sich in besonderem Maße für die Vermittlung auf Bachelorniveau, da Informatikbasiswissen kaum ohne sie zu erlernen ist. Schwill und Schubert bezeichnen die Informatik auch als *„Wissenschaft von der Modellbildung“* [SS11, S. 75]. Die Fähigkeit zur Modellbildung in allen Kontexten setzt voraus, dass die Syntax und Semantik der Modellsprache verstanden wurden. Das Verständnis darüber, dass jede Ansicht über ein System nur modellhaft sein kann, ist eine wichtige Voraussetzung für das Verständnis von Begrifflichkeiten wie *valide* und *verifiziert*.

Die Kompetenz zur Beurteilung der „Mächtigkeit“ eines Modells ist für die Auswahl von Spezifikationstechniken und Programmiersprachen entscheidend. Einige Sprachen, wie VHDL, können sogar mehrere Berechnungsmodelle umsetzen [Edw+01].

Zeitkriterium (3) Berechnungsmodelle sind in der Vergangenheit der Informatik leicht zu finden, da sie einen Grundstein der Disziplin bilden. Neben generellen Fragen zur Berechenbarkeit sind weitere Themenbereiche mit deutlichen Bezügen zu Modellbildung, die Komplexitätsberechnungen paralleler (60er-Jahre) und verteilter Prozesse (80er-Jahre) oder die Kryptographie (90er-Jahre) [Sav97]. Für

eingebettete Systeme sind neue Arten von Berechnungsmodellen notwendig, die berücksichtigen, dass das System abhängig von äußeren Einflüssen (seien es Signale oder Umwelteinflüsse) ist (vgl. [Agh85], [CM81]).

Sinnkriterium (3) Berechnungsmodelle sind für die Entwicklung eingebetteter Systeme unerlässlich geworden. Zwei Themenbereiche sollen hier stellvertretend genügen. Die Abbildung paralleler oder nebenläufiger Prozesse in Modellsprachen ist im Zeitalter von Mehrkernsystemen oder massiv-parallelen Architekturen für die Designphase äußerst wichtig. In seiner Analyse verschiedener MoCs und Programmiersprachen betont Marwedel die Notwendigkeit dieser Eigenschaft [Mar11]. Marwedel weist auf die Verbindung von Berechnungsmodellen und Sprachen im weiteren Kontext hin und zeigt, dass solche Modelle auch für die Spezifikation von Systemanforderungen geeignet sein können [Mar11]. Aus wissenschaftlicher Sicht ist insbesondere die Modellierung von Zeit eine Fragestellung, die sich auf die Entwicklung zukünftiger Berechnungsmodelle auswirkt [JS05].

Varianzkriterium (3) Es ist keine andere Methode oder Sichtweise bekannt, die den gleichen Anwendungsbezug aufweist wie Berechnungsmodelle. Mit dem Themenbereich eng in Verbindung stehende Entwicklungstätigkeiten sind die Modellbildung selbst und Ansätze der Co-Simulation und -Verifikation. Formale Methoden im Allgemeinen sind in einem Großteil der Ideen zu finden, die auf Konzepten der Analyse oder Synthese aufbauen [Nie98]. Eine natürliche Überlappung ergibt sich auch zur Idee paralleler oder nebenläufiger Prozesse (siehe Abschnitt A.2), die in Berechnungsmodellen festgehalten werden. Keine der beiden Ideen ist synonym zu „Models of Computation“ zu verwenden und verletzt daher auch nicht das Varianzkriterium.

A.2. Synthese und Steuerung paralleler Systeme

Parallele und nebenläufige Prozesse sind essentiell für eingebettete Systeme, die oft aus verschiedenen, miteinander kommunizierenden Komponenten bestehen. Dieser Aspekt findet sich nicht nur auf Ebene der Systemarchitektur, sondern stellt ebenso ein Merkmal vieler Hardwareentwürfe dar, deren nebenläufige Komponenten häufig auf Synchronisation angewiesen sind.

Der Schwerpunkt der folgenden Analyse liegt nicht auf der Synthese, sondern der Steuerung paralleler und nebenläufiger Systeme, da der Aspekt *Nebenläufigkeit* und *Parallelität* in keiner anderen analysierten Idee in diesem Umfang enthalten ist.

Lohstroh definiert Nebenläufigkeit folgendermaßen:

> „Taking the notions parallel and connected together we arrive at the notion of concurrency, which is a property of systems in which several computations are excecuting simultaneously, and potentially interact with each other.“ [Loh11, S. 2]

Horizontalkriterium (2) Während in den meisten Softwareanwendungen auf dem PC Parallelität zur Optimierung genutzt wird und somit oft ein zweitrangiges Ziel darstellt, ist es in der Entwicklung eingebetteter Systeme in großem Umfang zu bedenken. Eingebettete Systeme sind per Definition (siehe Abschnitt 2.1) mit Schnittstellen zur Umgebung, in der sie eingebettet sind, ausgestattet. Sie nutzen Sensoren zum Empfangen von Signalen und Messwerten, die für die korrekte Abarbeitung ihrer Aufgabe notwendig sind. Oft lassen sich keine zeitlichen Vorhersagen über das Auftreten dieser äußeren Einflüsse machen, sodass diese parallel zu anderen Prozessen überwacht werden müssen. Während die breite Anwendbarkeit der Idee ersichtlich ist, findet sie sich hauptsächlich in den Entwicklungsphasen des Designs und der Implementierung, weswegen das Horizontalkriterium nicht voll erfüllt wird.

Fortbildungskriterium (3) Das sichere Verständnis der Begriffe Pipelining, Parallelität und Nebenläufigkeit ist wichtig für die Entwicklung eingebetteter Systeme [WM00]. Für die Vermittlung entsprechender Kompetenzen auf Bachelorniveau können die Breite des Konzeptes und damit verbundene Synergieeffekte vorteilhaft sein. Bezüge zum Themengebiet Middlewares und Betriebssysteme lassen sich recht einfach über die Begriffe Scheduling, Interrupt-Service-Routinen oder Prozesse herstellen. Ein weiteres Fundament bilden die Konzepte hinsichtlich der Themenbereiche der synchronen und asynchronen Methoden oder der rekonfigurierbaren Architekturen. Als Überleitung zur Programmierung in C oder VHDL eignet sich die Vermittlung der Thematik deswegen ebenso (siehe [JD93], [Gom+07]).

Einfache Aufgaben zur Kompetenzförderung der Idee lassen sich bereits mit endlichen Automaten oder Petri-Netzen vermitteln. Spätere Verfeinerungen, zum Beispiel hin zu gefärbten oder hierarchischen Petri-Netzen, zeigen, wie eine Niveaustufung für fortgeschrittene Studierende bis hin zum Masterstudium umgesetzt werden kann.

Zeitkriterium (3) Die Konzepte paralleler oder nebenläufiger Prozesse reichen in der Historie der Informatik wie auch der Elektrotechnik weit zurück. Beispiele finden sich unter anderem in Kommunikationsprotokollen (bspw. [RE92]), Sprachkonstrukten (OpenMP [DM98], MPI [Gab+04]) und formalen Methoden (Petri-Netze [Mur89]). Die steigende Parallelisierung im Bereich der Computer-Hardware, die Forschung an massiv-parallelen Architekturen sowie die damit verbundene Anforderung, immer mehr heterogene, nebenläufige Kompetenzen zu steuern, können als Beleg für die zukünftige Relevanz der Idee gesehen werden.

Sinnkriterium (3) Wie bereits im Horizontalkriterium erläutert, ist die Parallelisierung von Algorithmen ein häufiger Ansatz zur Optimierung sequentieller Software. Ebenso wurde erläutert, dass Parallelität eine typische Eigenschaft von Hardwareaufbauten und Hardwarekomponenten ist. Die Berücksichtigung dieser parallelen Architektur auf Datenflussabhängigkeiten und Kommunikationsprotokolle ist essentiell, um eine korrekte Systemfunktionalität gewährleisten zu können. In den letzten Jahren zeigt sich mehr und mehr, dass nicht ein sehr leistungsfähiger

Chip, sondern mehrere durchschnittlich leistungsfähige Chips die größten Geschwindigkeitsvorteile ermöglichen. Dies spiegelt sich schon seit einigen Jahren im Trend zu Mehrkernprozessoren im Consumer-Bereich wider und wird voraussichtlich in Zukunft durch massiv-parallele Architekturen (bspw. Many-Core-Architekturen) weiter an Bedeutung gewinnen [Sha07]. Die Nutzung hochgradig paralleler Recheneinheiten oder moderner Grafikkarten zur Simulation komplexer physikalischer und logischer Aufgaben (bspw. Wettervorhersage), sind die praktischen Umsetzungen dieser Erkenntnis.

Varianzkriterium (3) Parallelität und Nebenläufigkeit ist in keiner anderen analysierten Idee so stark vertreten wie in der vorliegenden. Weder in der Analyse zu MoCs, noch in der Analyse zu Ressourcen-Management, Verifikation und Validierung, Fault-Tolerance oder Betriebssysteme, Middlewares und Real-Time-Kernel kann der Blickwinkel auf die Idee so weit gefasst werden. Auch wenn Bestandteile der Idee in den angesprochenen Themen vorkommen, rechtfertigt die besonders abstrakte, und damit breite, Betrachtung der Konzepte *Nebenläufigkeit, Pipelining* und *Parallelität* die Erfüllung des Varianzkriteriums.

A.3. Controller-Synthese

Controller-Synthese beschreibt die Erstellung der Controllereinheit aus einer gegebenen Systemspezifikation. Der Controller hat die Aufgabe, das System in allen möglichen Kombinationen von Eingangssignalen aus der Umwelt funktional integer zu halten. Eine besonders wichtige Komponente ist der Controller in Systemen mit hohen Anforderungen an Zuverlässigkeit. Baier et al. schlagen die folgende Definition vor:

> „Controller synthesis addresses the question of how to limit the internal behavior of a given implementation to meet its specification, regardless of the behavior enforced by the environment.“ [Bai+04, S. 1]

Neben der methodischen Komponente der Controller-Synthese ist ein weiterer Aspekt im Konzept erhalten: die prinzipielle Aufteilung von Rechnerarchitekturen in Steuerwerk und Datenpfad als zentrale Komponenten eines Prozessors. Zusätzlich ist ein Steuerwerk in vielen Hardwareplattformen eine Schnittstelle zu externen Komponenten.

Horizontalkriterium (2) Die Konzepte „Datenfluss“ und „Kontrollfluss“ sind für alle Informatiksysteme die einen Prozessor verwenden von Bedeutung. Wie in der grundlegenden Erläuterung eingebetteter Systeme geschildert (siehe Abschnitt 2.1), werden Mikroprozessoren häufig als Berechnungseinheiten in Mikrocontrollern genutzt und implementieren somit ebenfalls ein Steuerwerk und einen Datenpfad. Das Verständnis über das Zusammenwirken der zwei Konzepte ist aber auch wichtig, wenn mit anderen Hardwareplattformen aus dem Bereich der eingebetteten Systeme gearbeitet wird. Dies sind beispielsweise rekonfigurierbare Architekturen wie

FPGAs, die typischerweise genutzt werden, um eine Berechnung datenflussorientiert auszuführen. Ein „Controller“ ist zudem die Schnittstelle zweier interagierender Systeme und findet sich beispielsweise in einem Netzwerk- oder Speicherinterface als Netzwerkcontroller oder Speichercontroller. Die Breite des Anwendungsgebietes ist damit vorhanden, da die Konzepte ein Bestandteil der meisten Rechnerarchitekturen sind.

Im Gegensatz zum breiten Anwendungsgebiet sind die zwei Konzepte nur spärlich in den verschiedenen Entwicklungsphasen eines Projektes sichtbar. Hauptsächlich ist dies in der Phase des Architekturdesigns und der Implementierung der Fall. In ersterer geht man der Frage nach, welche Systemplattform die Umsetzung eines angefertigten Systemmodells prinzipiell erlaubt, also welche Operationen der Controller ausführen kann. In der Implementierungsphase sind Entwickler häufig auf das Verständnis eines Controllers bei der Interaktion mit anderen Komponenten angewiesen, um die richtige Funktionalität des Systemes zu gewährleisten (Netzwerk- oder Speichercontroller).

Da nur ein Aspekt des Horizontalkriteriums vollständig erfüllt wurde, kann die Analyse nicht vollständig positiv evaluiert werden.

Fortbildungskriterium (3) Das Wissen über die Zusammenhänge zwischen Kontrollfluss und Datenfluss gehören thematisch in die Lehrveranstaltungen zu Rechnerarchitekturen. Wie die Curriculaempfehlung der GI zeigt, ist diese Veranstaltung typischerweise schon frühzeitig auf Bachelorniveau angesetzt. Die Vermittlung entsprechender Kompetenzen sollte deswegen kein Problem darstellen und der erste Aspekt des Forbildungskriterium damit erfüllt sein.

Wie in der Einleitung zu dieser Analyse erläutert, ist die Controller-Synthese ein Beispiel für ein Themengebiet, das auf dem Verständnis über Rechnerarchitekturen aufbaut. Eine Unterscheidung der Verarbeitungstypen „Single-Instruction/Multiple-Data“ und „Multiple-Instruction/Multiple-Data“ ist nicht nur die Grundlage moderner PC-Architekturen, sondern auch gleichzeitig die theoretische Fundierung für Vektorrechner und rekonfigurierbare Architekturen die meist datenflussorientiert beschrieben werden. Letztere sind, wie in Abschnitt 5.5 gezeigt, ein wichtiger Bestandteil vieler Entwicklungsprojekte.

Das Fortbildungskriterium wird damit hinsichtlich seiner beiden Aspekte erfüllt.

Zeitkriterium (3) Die ersten Überlegungen zu einem Steuerwerk nach heutigem Verständnis können bereits aus den Vorschlägen zur „Von-Neumann-Architektur“ von 1945 entnommen werden. Darin heißt es:

> „2.3 Second: The logical control of the device, that is the proper sequencing of its operations can be most efficiently carried out by a central control organ.“ [S. 2 Neu93, Neuauflage]

Controller-Synthese ist als wissenschaftliches Themengebiet seit den 80er-Jahren in der Diskussion [Dor95], [SCS87]. Wie bereits ausgeführt, gehört die Idee in den

Themenbereich der *Regelungstechnik*, die eine eigene Disziplin darstellt und ebenfalls länger besteht als die 20 Jahre, die für das Zeitkriterium angesetzt wurden.

Sinnkriterium (2) Die praktische Relevanz für den Themenbereich ist partiell durch seine Fundamentalität für Rechnerarchitekturen aller Art gegeben. Die Idee enthält jedoch keine Aspekte für Methoden oder Sichtweisen, die ganz konkret zur Lösung praxisrelevanter Aufgabenstellung dienen.

Forschung zu den beiden Ideen findet sich auf verschiedenen Ebenen. Die namensgebende Controller-Synthese basiert auf der Spieltheorie der Theoretischen Informatik und ist in diesem Kontext Ziel des wissenschaftlichen Diskurses. Die automatische Controller-Synthese ist bei fehlertoleranten Systemen und bei Systemen mit starken Anforderungen an die Zuverlässigkeit notwendig.

Wie in der Analyse geschildert, kann nur der theoretische Aspekt des Kriteriums erfüllt werden, da die Signifikanz für das Anwendungsgebiet im praktischen Kontext nicht zu beobachten ist.

Varianzkriterium (1) Die Synthese des Controllers aus Beschreibungen der Umgebung sowie Anforderung an die eigentliche Aufgabenstellung besitzt viele Aspekte anderer Ideen. Die Nähe zu formalen Methoden und MoCs ist offensichtlich. Ebenso existieren starke Bezüge zu fehlertoleranten und zuverlässigen Systemen. Die Transformation von Modell- und Spezifikationsartefakten in einen Controller erinnert ebenso an die Grundprinzipien des modellbasierten Entwicklungsansatzes.

Die Ideen des Kontrollflusses und des Datenflusses stellen zudem keine eigenen Ideen dar, sondern lediglich das Fundament für andere Themengebiete. Ebenso ist nicht ersichtlich, dass bestehende Ideen auf neuartige Weise miteinander verbunden oder um wesentliche Elemente erweitert werden. Das Varianzkriterium wird abgelehnt, da somit keiner der beiden geforderten Aspekte erfüllt wurde.

A.4. Hardware/Software Co-Design

Beim Design eines eingebetteten Systems kann der Entwickler prinzipiell zwischen mehreren Umsetzungsvarianten (Software- und Hardwareaufteilung; Eigenentwicklung oder Fremdleistung) wählen. Dabei unterscheiden sich diese Möglichkeiten durch Parameter wie Leistungsfähigkeit, Energieeffizienz, Leistungsdichte, Entwurfskomplexität, Entwicklungskosten und begrenzte Entwicklungszeiten [Sch10]. In diesem Zusammenhang sind es Methoden und Werkzeuge der Analyse, Synthese und Simulation, die für die Beantwortung solcher Fragestellungen von Entwicklern genutzt werden. De Micheli und Gupta bieten folgende Definition des Begriffes an:

> „Hardware/Software co-design means meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design.“ [DG97, S. 1]

Die Idee bei diesem Ansatz ist nicht wie bei früheren Methoden, erst die Hardware und darauf folgend die Softwarekomponenten zu entwickeln, sondern beides parallel zu entwerfen. Das beinhaltet die gegenüber traditionellen Entwicklungsverfahren flexiblere Zuweisung von Funktionalität auf Hardware- und Softwarekomponenten.

Horizontalkriterium (3) Hardware/Software Co-Design wird in Industriebereichen wie der Automobiltechnik, Luft- und Raumfahrttechnik, industriellen Produktionsanlagen, Hausautomatisierung und vielen mehr eingesetzt [Tei12]. Aus methodischer Sicht umfasst Hardware/Software Co-Design neben der namensgebenden Design-Phase auch alle Entwicklungsschritte bis hin zum „Operation Process". Begrifflich sind deswegen die Termini *Co-Design*, *Co-Synthese* und *Co-Verifikation* dem Konzept untergeordnet [WM00]. Teich nennt weitere Bestandteile des Konzepts, wie beispielsweise die *Koordinierung* von Designetappen, *simultanes* Arbeiten an Software- und Hardwaredesigns oder die *Überprüfung* der Kommunikation und Integration von Systemkomponeten [Tei12]. Die breite Anwendung und die Relevanz der Idee erfüllen somit das Horizontalkriterium.

Fortbildungskriterium (3) Wolf spricht sich dafür aus, dass die grundlegenden Ansichten des Hardware/Software Co-Designs auch auf Bachelorniveau vermittelt werden können [WM00]. Wie im Abschnitt zum Varianzkriterium und teilweise auch im Horizontalkriterium zu sehen ist, bildet die Idee das Fundament einiger speziellerer Sichtweisen, die durch das Sinnkriterium rechtfertigbar sind. Mit Bezug auf den in Sektion 4.2.2 erwähnten Qualtitätsrahmen deutscher Hochschulen ist festzustellen, dass die Erarbeitung und Sammlung von Informationen zur Idee auch auf Bachelorniveau möglich und die Anwendung entsprechender Konzepte für die Studierenden umsetzbar ist. Im kleineren Rahmen kann die Umsetzung in einer Lehrveranstaltung oder einem Praktikum bspw. mit einer FPGA-Plattform realisiert werden (siehe bspw. [BKR10] und [Wan11]).

Zeitkriterium (3) Der Begriff ist zuerst Anfang der 1980er-Jahre im Zusammenhang mit integrierten Schaltkreisen aufgetreten (siehe [SFC85]). Maßgeblich für die damaligen Anforderungen sind Überlegungen gewesen, wie die Leistungsfähigkeit verschiedener Komponenten (häufig ASICs und CPUs) am besten genutzt werden kann. Nach einer mittlerweile 30 Jahre dauernden Historie ist Hardware/Software Co-Design eine weit verbreitete Designpraxis. Seitens der Forschung werden neue Partitionierungsalgorithmen oder Vorgehensweisen entwickelt (bspw. [Ped+12] und [Zha+13b]). Wolf prognostiziert, dass die Herausforderungen für Hardware/Software Co-Design in Zukunft ansteigen und das Themenfeld aus diesem Grund noch lange ein „lebendiges" Gebiet sein wird [Wol03, S. 42].

Sinnkriterium (3) Das Ziel von Hardware/Software Co-Design ist die möglichst nebenläufige Entwicklung und die flexible Verwendung von Hardware- und Softwarekomponenten. Zentrale Gründe für die Notwendigkeit einer solchen Entwicklungsmethodik sind:

- Kürzere Entwicklungszeit, unter anderem durch Rapid-Prototyping [CAS11],
- Höhere Performanz durch Software-Unterstützung von hardwareeigenen Eigenschaften sowie früherer Verifikations- und Testmöglichkeiten [Mar11], [JW05],
- Wiederverwendung von Hardware- und Softwarebestandteilen und somit eine geringe Entwicklungszeit [Mar11], [Ha+07].

Das Ziel des Hardware/Software Co-Designs ist aber vor allem die praktikable Durchführung von Entwicklungsprozessen moderner eingebetteter Systeme. Mit herkömmlichen, hauptsächlich sequentiellen, teils sogar Ad-hoc-Entwicklungsmethoden, ist dieses Ziel nicht mehr erreichbar [HS07], [PEP06]. Die praktische Relevanz ist damit gegeben. Aus wissenschaftlicher Sicht ist Hardware/Software Co-Design sogar für andere informatische Disziplinen ein interessanter Forschungsgegenstand (bspw. Supercomputing [SQJ11]).

Varianzkriterium (2) Wie aus der bisherigen Analyse ersichtlich, ist der Begriff Hardware/Software Co-Design eine Mischung aus Methodik und Sichtweise in einem. Ein möglicher Überschneidungspunkt liegt im Bereich der Verifikation. Ob das Varianzkriterium die Idee ablehnt, hängt damit stark von der Gewichtung des Aspektes der Formalisierung ab. Die in Kapitel A.17 geschilderte Analyse betont formale Methoden besonders. Der Autor ist der Auffassung, dass Formalisierung nicht der Kern der hier besprochenen Idee ist, sondern dieser in der gleichzeitigen Ausführung mit anderen Entwicklungsschritten liegt. Somit ist die Idee des Hardware/Software Co-Designs nur partiell synonym zu formalen Verifikationsmethoden zu sehen und erfüllt damit das Varianzkriterium teilweise.

A.5. Platform-based Design

„Platform-based Design“ ist ein Konzept, welches im Bereich der eingebetteten Systementwicklung von Autoren wie Chang [Cha99] und Sangiovanni-Vincentelli vorgestellt wurde [SM01]. Die zentrale Idee des Ansatzes ist die Trennung von vertikalen Schichten in Plattformen und deren Zugangs-APIs, um die Plattformfunktionalitäten auch für externe Komponenten zugänglich zu machen. Ähnlich zu Computer-Plattformen wie „i386“ oder „i686“ kapselt eine Plattform Funktionalitäten, die für eine ganze Systemfamilie gelten.

Kumar et al. schildern die Intention des Ansatzes in folgender Zusammenfassung:

> „The basic idea is that an architecture, which is suitable and efficient for one application will also be suitable and efficient for many similar applications.“ [Kum+02, S. 2]

Bailey fügt in seiner Definition des Begriffes hinzu, dass nicht nur typische Komponenten im Sinne der Hardware- oder Softwareentwicklung eingebunden werden, sondern auch Business-Pläne oder Support-Praktiken [BMA05].

Horizontalkriterium (2) „Platform-based Design“ findet nicht nur in der Entwicklung eingebetteter Systeme Anwendung, sondern auch bei herkömmlichen PCs. Aus diesem Bereich bekannte Plattformen sind bspw. Intel Celeron oder AMD Athlon. Plattformen für eingebettete Systeme finden sich in den verschiedenen Hardware-Realisierungen wie ASICs, FPGAs oder Mikrocontrollern, wobei auch innerhalb dieser Gruppen verschiedene Plattformen definiert werden können. So ist in der Untergruppe der ARM-Mikrocontroller die ATiny-Reihe deutlich von der ATmega-Reihe zu unterscheiden. Die entsprechenden technischen Systeme und die dazugehörige APIs sehen für beide Varianten unterschiedlich aus. Auch während eines Entwicklungsprozesses gibt es Plattformen, die sich zwischen den eigentlichen Software- und Hardwareeinheiten befinden, wie zum Beispiel die System- oder Netzwerk-Plattform. Weitere Anwendungen des Konzeptes finden sich in breiter Form in der Automobil- und Luftfahrttechnik [Boe10]. Eine Mischform des Ansatzes ist das von Sangiovanni-Vincentelli, Damm und Passerone eingeführte „Contract-based Design“ [SDP12b].

Auf den Entwicklungsprozess bezogen, findet sich das „Platform-based Design“ insbesondere bei der Auswahl einer geeigneten Plattform („Achitectural Design Process“) und der Nutzung dieser („Implementation Process“). In einer Erläuterung des Konzeptes stellen Sangiovanni-Vincentelli und Martin eine Abbildung der Methodik vor, aus der sich ableiten lässt, dass insbesondere die zwei genannten Ebenen zentral durch den neuartigen Ansatz beeinflusst werden [SM01]. Da nicht ersichtlich ist, dass mehr Phasen des Entwicklungsprozesses durch die Idee abgedeckt werden, ist das Horizontalkriterium nur teilweise erfüllt.

Fortbildungskriterium (1) „Platform-based Design“ schließt Ideen und Applikationen ein, die auf unterschiedlichen kognitiven Niveaus betrachtet werden können. Grundlegende Elemente wie *Kapselung* und *Hierarchisierung* können auf Bachelorniveau dargestellt und in praktischen Einheiten mit weitreichenden Hilfestellungen auch angewandt werden. Eine eigenständige Konzeption und Umsetzung empfiehlt sich aufgrund der Antizipation von notwendigen Systemfamilienattributen nicht, da hierfür nicht nur die Modellierung eines einzelnen Systems, sondern einer ganzen Systemfamilie notwendig wäre. Wie aus den Lernzielbeschreibungen des QDH ersichtlich ist, gehört der Entwurf eines Systems bereits zu den höchsten Kompetenzniveaus, die auf Masterniveau angesetzt sind. Ein Vorteil bei der Lehre des Konzeptes ist die reichhaltige Auswahl an greifbaren Beispielen anderer Disziplinen, an denen Überlegungen zum Entwurf leicht verständlich dargestellt werden können (bspw. i386er-Systemplattform).

Zu hinterfragen ist allerdings, ob „Platform-based Design“ eine notwendige Grundlage für weitere Ideen ist, die ohne ein solides Verständnis der Thematik kaum oder nicht vermittelbar sind. Der Autor vertritt die Meinung, dass die in der Thematik enthaltenen Ideen (Kapselung, Plattformen von Systemfamilien, Schichtentrennung) in dieser Weise fundamental sind, während „Platform-based Design“ als technische Umsetzung keine Grundlage weiterer Technik für die Ausbildung zukünftiger Entwickler ist. Die angesprochenen Metakonzepte sind unter Umständen sogar in anderen Kontexten besser zu vermitteln (bspw. Betriebssysteme oder komponen-

tenbasierte Entwicklung). Das Fortbildungskriterium wird aus diesem Grund nicht erfüllt.

Zeitkriterium (3) Sangiovanni-Vincentelli, Damm und Passerone zufolge stammt das „Platform-based Design“ aus den späten 80er-Jahren [SDP12a] und erfüllt somit deutlich das Zeitkriterium. Um die Jahrtausendwende erfuhr die Technik einen Aufschwung (siehe bspw. [Cha99]). Im Computer-Science-Curriculum von 2013 der ACM/IEEE ist „Platform-based Design“ eine der vier neu aufgenommenen Knowledge-Areas, was die Notwendigkeit von Experten mit den damit verbundenen Kompetenzen unterstreicht [SR13]. Diese Empfehlungen sehen den Themenschwerpunkt jedoch in einem informatisch weit gefassten Rahmen, zu dem Anwendungsbezüge wie Mobil- und Webplattformen gehören.

Sinnkriterium (3) „Platform-based Design“ versucht die Wiederverwendung bereits entwickelter Systemkomponenten (bzw. Plattformen) zu erhöhen und damit langfristig die Entwicklungszeit zu verringern [San+04]. Plattformen werden üblicherweise nur für ähnliche Architekturen erstellt und bieten deswegen auch lediglich in diesem Rahmen die angesprochenen Vorteile. Madni beschreibt den Nachteil des Ansatzes damit, dass Produktpaletten sich nicht in dem Maße weiterentwickeln könnten wie in individuellen Projekten, da sie immer an die Möglichkeiten der Plattform gebunden sind [Mad12]. Simpson et al. berichten von Unternehmen wie Black & Decker, Sony, Seagate oder Volkswagen, die „Platform-based Design“ erfolgreich einsetzen [Sim+06]. Problematisch sehen einige Experten die vermeintlich vollständige Modularisierung von Plattformen, die wegen den produktspezifischen Rahmenbedingungen (bspw. Leistungsaufnahme, Abmessungen, Zeit) nicht vollständig umsetzbar sind [Sim+06].

Ein wissenschaftlicher Diskurs zur Idee des „Platform-based Designs“ ist ebenso beobachtbar (siehe [Kum+02]), sodass beide Aspekte des Sinnkriteriums erfüllt werden.

Varianzkriterium (2) Bezüge des „Platform-based Designs“ finden sich unter anderem zu den Begriffen Wiederverwendung, Abstraktion, Schnittstellen und Schichtentrennung und damit zur Idee des „Component-based Design“. Eine Plattform ist verschieden zu einer Komponente, da sie nicht nur einen Teil der Systemfunktionalität vorgibt, sondern auch die anwendungsfeldrelevanten Rahmenbedingungen in die Umsetzung mit einbezieht. Auch die Sichtweise auf die zwei zugehörigen Ideen ist partiell unterschiedlich. Das übergeordnete Ziel von „Component-based Design“ ist Design-Flexibilität durch Austauschbarkeit. Gegensätzlich dazu versucht „Platform-based Design“ ein festes, anwendungsfeldspezifisches Fundament für eine Familie von Systemen zu schaffen [BMA05]. Eine neue Idee ist es wegen den genannten Themenüberschneidungen jedoch nicht. Deswegen wird das Varianzkriterium nur teilweise erfüllt.

A.6. Component-based Design und Intellectual Property

„Component-based Design“ teilt das zu entwerfende System nicht vertikal, sondern horizontal, um eigenständige Entitäten zu erhalten, die über definierte Schnittstellen miteinander kommunizieren [LS11a]. Der ursprünglich nicht vorhandene Zusatz des „Intellectual Property“ (IP) wurde in die Analyse mit aufgenommen, da es sich dabei um das gleiche Konzept handelt. Im Gegensatz zu „Component-based Design“ ist die Begriffshistorie nicht mit der Informatik, sondern mit der Elektrotechnik und der Disziplin des Systementwurfs verknüpft. In der folgenden Analyse sind deswegen beide Begriffe synonym zu verstehen. Die Verbindung zwischen IP-Modulen und Komponenten wird durch den Wortgebrauch wissenschaftlicher Publikationen bestätigt (bspw. [Fil+98], [Gaj+00b]).

Die zentrale Idee hinter „Component-based Design“ ist eine erhöhte Wiederverwendbarkeit einzelner Designartefakte (Komponenten) mit eigenständiger Funktionalität und entsprechenden Schnittstellen. Im Gegensatz zum „Platform-based Design“ wird bei der vorliegenden Idee oft die Schnittstellen-Konzeption und -Umsetzung zwischen heterogenen Komponenten stärker betont (vgl. [Ces+02]).

Horizontalkriterium (3) Die ursprüngliche Idee ist bereits aus der Softwareentwicklung bekannt, in der sich zeigte, dass Objektorientierung zu feingranular für weitreichende Wiederverwendung sein kann [Szy02]. In der eingebetteten Systementwicklung ist diese Idee in vielen verschiedenen Bereichen beobachtbar. Die Modellierung mit MoCs hat bspw. sehr häufig eine hierarchische Strukturierung, über die Komponenten und Subkomponenten definiert sind. Im System-Design werden einzelne Funktionalitäten in Form von Komponenten eingekauft (IP) und in bestehende Systeme integriert. Diese sind im Kontext der Softwareentwicklung Bibliotheken und im Kontext der Hardwareentwicklung einzelne Komponenten wie ADCs, Muxer, DSPs oder komplette System-on-Chips (SoCs). Die breite Anwendbarkeit ist deswegen gegeben.

Die komponentenbasierte Entwicklung ist besonders für den „Architectural Design Process“ wichtig, in dem der Aufbau des Gesamtsystems oder einzelner Teile entworfen wird. Neben der Implementierung, also der Umsetzung dieses Entwurfes, ist auch die Phase der Integration von dieser Idee beeinflusst, wenn die Kommunikation zwischen selbst- und fremdentwickelten Komponenten abgestimmt werden muss. Komponentenschnittstellen können ebenfalls als Grenzen dienen, die für eine Systemvalidierung oder -verifikation von Nutzen sind [AH01].

Fortbildungskriterium (3) Komponenten bilden, wie das Verständnis zur objektorientierten Denkweise in der Softwaretechnik, einen wesentlichen Baustein für Vorgehensweisen im Entwurf eingebetteter Systeme. Die Idee hinter „Component-based Design“ ist generell genug, um sie ohne tieferes Technikverständnis an Bachelorstudierende vermitteln zu können. Durch die erprobte Anwendung im Bereich der Softwaretechnik sind bei der Vermittlung entsprechender Kompetenzen

Synergieeffekte zu erwarten. Eine Niveaustufung hin zu Lehrkonzepten für Masterstudierende ist zum Beispiel in Form von heterogenen Komponentenbeschreibungen und dem damit zusammenhängenden, schwierigen Entwurf von Schnittstellen denkbar. Während die Arbeit von Bachelorstudierenden in Praktika auf die *Anwendung* von Komponenten und IP beschränkt bleibt, können auf Masterniveau auch eigene *Entwicklungen* gefordert werden (vgl. QDH).

Zeitkriterium (2) Obwohl der Begriff des „Component-based Designs“ bereits in Publikationen der 90er-Jahre auftaucht [SL94], beziehen sich diese primär auf Forschung im Bereich der Mikrosystemtechnik und nicht auf das Konzept, das heute mit dem Begriff verbunden wird. Erst um die Jahrtausendwende wurde das Konzept präzisiert und mit formalen Methoden unterstützt (bspw. [Kop98]). Brown veröffentlichte im Jahr 1996 erste Bücher zur komponentenbasierten Entwicklung in Software [Bro96]. Wie bereits in der Einführung zu dieser Analyse beschrieben, findet sich komponentenbasierte Entwicklung für Hardwareteile selten auf dieser Ebene (bspw. [AMO05]). Unter dem Begriff IP hingegen lässt sich eine längere Historie ausmachen. In diesem Zusammenhang zu nennende Arbeiten stammen unter anderem von Gajski [Gaj+00a] und Bhatia et al. [BGV96].

Das Zeitkriterium kann nicht für die vollen 20 Jahre, die in der Definition angesetzt wurden, erfüllt werden. Wie ebenfalls bei der generellen Analyse des Kriteriums beschrieben, ist speziell beim Zeitkriterium auf dessen Gewichtung zu achten. Mit Verweis auf die rapide Entwicklung von System-on-Chips und damit dem gestiegenen Interesse an wiederverwendbaren (zum Teil virtuellen) Komponenten kann das Kriterium auch mit einer geringen Abweichung von circa drei bis fünf Jahren eingeschränkt bestätigt werden.

Sinnkriterium (3) Wissenschaftlich ist das Themengebiet stark erforscht. Um die Inkompatibilität von Komponentenbeschreibungen zu lösen, werden in der aktuellen wissenschaftlichen Debatte abstrakte Semantiken diskutiert. Diese geben eine Abstraktionsschicht für die Integration unterschiedlicher MoCs vor [LS11a]. In praktischer Hinsicht ist die Integration heterogener Komponenten im Entwicklungsprozess relevant. Durch das Aufkommen von „Intellectual Property“ gilt es, blackboxartige Komponenten mit vorhandenen oder noch zu entwickelnden Teilen des Systems zu verbinden und dabei funktionale und nicht-funktionale Anforderungen zu gewährleisten. Im Hardwarepraktikum an der Universität Siegen wurde beobachtet, dass viele Studierende wesentliche Parameter externer Komponenten, vor allem Hardwarebauteile, nicht extrahieren und in Bezug zum Anwendungsprogramm setzen konnten (siehe Abschnitt 3.4.1). Es kann davon ausgegangen werden, dass diese Aufgabe oft in der industriellen Praxis vorkommt und damit verbundene Kompetenzen wichtig für zukünftige Entwickler eingebetteter Systeme sein werden.

Varianzkriterium (2) „Component-based Design“ hat starke Bezüge zum „Platform-based Design“. Ideen, die beide Konzepte miteinander verbinden, sind die Abstraktion von Anwendungsgebieten und die Zusammenführung funktionaler Einheiten

in über Schnittstellen ansprechbare Komponenten und Plattformen. Beide Methoden empfehlen die Schichtentrennung (engl. „Separation of Concerns“). Auf die zentralen Unterschiede zwischen den beiden Ansätzen wurde bereits in der Varianzkriteriumsanalyse zu „Platform-based Design“ hingewiesen (siehe Abschnitt A.5). Wegen der partiellen Überschneidung kann das Kriterium nur teilweise erfüllt werden.

A.7. Synchrone und asynchrone Konzepte

Synchrone und asynchrone Konzepte stellen Möglichkeiten dar, Berechnungs- und Kommunikationsabläufe in einem System zu gestalten. Beide Konzepte finden sich im Software- und im Hardware-Design. Die zentrale Idee des asynchronen Designs ist der Verzicht auf eine globale Koppelung unabhängiger Komponenten. Synchronität meint dementsprechend die zeitliche Koppelung unabhängiger Komponenten. Eine Unterscheidung der beiden Paradigmen hinsichtlich Programmiersprachen findet sich in [Cas+05].

Horizontalkriterium (2) Die Anwendung synchroner und asynchroner Prinzipien findet sich in vielen Bereichen der Entwicklung eingebetteter Systeme wie in analogen Schaltungen oder Softwarekomponenten (bspw. Interrupt-Service-Routinen). Da beide Methoden zueinander komplimentär sind, wird bei jeglicher Kommunikation immer eine der beiden Methoden verwendet. Die sorgfältige Abschätzung von Anforderungen und die technischen Rahmenbedingungen geben an, ob synchrone oder asynchrone Verfahren verwendet werden können. Die Modellierung von externen asynchronen Komponenten in einem synchronen System ist eine Facette der als schwierig angesehenen Verbindung heterogener Komponenten [HS06].

Der Vorteil synchroner Methoden und Komponenten liegt oft im einfachen Nachvollziehen der Funktionsweise und der damit verbundenen einfacheren Implementierung. Asynchrone Varianten können diesen Umstand evtl. durch eine theoretisch höhere Leistung oder einem verbesserten Durchsatz wettmachen.

Synchrone und asynchrone Methoden sind vor allem bei der Modellierung von Kommunikation oder Parallelität und der entsprechenden Implementierung zu finden. Damit sind nur zwei Entwicklungsphasen eng mit der Idee verknüpft, weswegen das Horizontalkriterium nur für die Breite des Anwendungsgebietes positiv beurteilt werden kann.

Fortbildungskriterium (3) Losgelöst von den vielfältigen technischen Umsetzungen und den dazugehörigen Einsatzgebieten lassen sich die Unterschiede zwischen synchronen und asynchronen Verfahren an Bachelor- wie auch Masterstudierende vermitteln. Ein einfaches Beispiel wäre in diesem Zusammenhang die Konstruktion eines Addierers als synchrone und einmal als asynchrone Komponente. Unterschiede in Wartbarkeit und Performanz sollten den Studierenden gut vermittelbar sein. Ein gutes Beispiel für ein Konzept, das in Software wie auch in Hardware von Nöten

ist, ist das des atomaren Zugriffs auf geteilte Ressourcen. Auf Ebene der Anwendung kann dies bspw. eine Datenbank sein, während asynchrone Hardwaresysteme dieselbe Problematik beim Zugriff auf Register oder Bussysteme haben.

Zeitkriterium (3) Synchrone und asynchrone Verfahren sind Bestandteil elektrotechnischer und informatischer Disziplinen. Den Zeitpunkt auszumachen, an dem die Paradigmen in das Zentrum wissenschaftlicher Debatten gerückt sind, ist schwierig. Unabhängig vom Anwendungsbereich sind die meisten Diskussionen dieser Idee deutlich älter als die vom Zeitkriterium geforderten 20 Jahre. Beispiele hierfür finden sich in den Arbeiten von [Cha84], [KG82] und [HLR94].

Sinnkriterium (3) Wie bereits aus den vorherigen Kriterienanalysen ersichtlich ist, steigt mit dem Voranschreiten komponentenbasierter Entwicklungsvorgehen auch die Notwendigkeit für das Verständnis synchroner und asynchroner Methoden, welche die Systemteile miteinander verbinden. Besonders vielversprechend sind asynchrone Prinzipien im Low-Power-Computing oder bei sehr umfangreichen Chips, bei denen die Verteilung des Clock-Signals bereits Probleme bereitet. Synchrone Ansätze sind in der Regel besser wartbar, da der Ablauf von Berechnungs- und Kommunikationsschritten nachvollziehbarer abläuft. Das Sinnkriterium kann bei der vorliegenden Idee positiv ausgewertet werden, da es nur wenig Entwicklungsprojekte gibt, bei denen sich keine Ansätze von synchronen und asynchronen Prinzipien finden lassen. Bei Hardwareaufbauten, Hardwarebeschreibungssprachen, Mikrocontrollerimplementierungen oder Softwareprojekten sind Überlegungen zu den zwei genannten Ideen zumindest immer enthalten, wenn eine Kommunikation oder Regelung mit externen Prozessen notwendig ist. Aufgrund der Definition eines eingebetteten Systems wird dies in der Mehrzahl der Anwendungsfälle gegeben sein (siehe Abschnitt 2.1). Die Kenntnis über Vor- und Nachteile der Ansätze wie auch deren praktische Umsetzung ist deswegen essentiell.

Varianzkriterium (1) Das Themengebiet hat einen starken Bezug zu parallelen und nebenläufigen Prozessen, da Synchronität und Asynchronität immer in eine Form der Kommunikation eingebettet sind und diese zwischen zwei oder mehr Prozessen bzw. Systemen stattfindet. Die meisten nebenläufigen Prozesse besitzen am Ende einer Berechnung oder einer Kommunukation die Notwendigkeit für eine Form der Synchronisation. Selbst bei hochgradig parallelen Algorithmen, wie beispielsweise der parallelen Berechnung der Mandelbrotmenge, ist spätestens dann eine Synchronisation erforderlich, wenn auf ein gemeinsames Medium wie einen Datenträger oder einen Bildschirm geschrieben werden muss. Synchrone und asynchrone Konzepte sind deswegen ein bedeutender Bestandteil paralleler Prozesse und Systeme und teilen damit einen Großteil der Idee. Aus diesem Grund lehnt das Varianzkriterium die Idee ab.

A.8. Ressourcen-Management

Die optimale Handhabung verfügbarer Systemressourcen ist bei eingebetteten Systemen sehr viel kritischer als bei traditionellen Softwaresystemen [IKF06]. Kritisch bedeutet in diesem Kontext, dass Entscheidungen über den Entwurf und die Implementierung eines Systems stärker von verfügbaren Ressourcen abhängen. Die Idee hinter dem Konzept ist die Erfassung, Modellierung und Umsetzung von Abhängigkeitsbeziehungen zwischen Systemkomponenten untereinander und in Abhängigkeiten zu Attributen der Umwelt. Ausgehend von der textuellen Beschreibung im TECs-Call widmet sich die folgende Analyse der dynamischen Neubewertung- und Zuordnung von funktionalen und nicht-funktionalen Systemparametern und nicht dem statischen Ressourcenmanagement. Die Fundamentalität anderer Attribute und Tätigkeiten (wie *Echtzeitfähigkeit* oder *Scheduling*) werden in Sektion A.10 gesondert analysiert.

Horizontalkriterium (3) Dynamisches Ressourcen-Management wird oft synonym zum Ressourcen-Management zur Laufzeit gesehen. Es wird prinzipiell auf verschiedenen Systemebenen eingesetzt, bspw. der Hardware (FPGAs) oder der Vermittlungsebene von Hardware und Software in Form eines Betriebssystems. Eine besondere Herausforderung bei der Neubewertung von veränderlicher Ressourcennutzung ist die Wahrung funktionaler und nicht-funktionaler Anforderungen. Die Abschaltung bestimmter Prozessorkerne zur Reduzierung der Stromaufnahme darf beispielsweise nicht zu Lasten von Echtzeitbedingungen gehen. Beginnend mit der Anforderungserhebung hat die Planung von dynamischer Ressourcenverwaltung Einfluss auf die Phasen des Systemdesigns, der Implementierung und auch der Validierung bzw. dem Test des Gesamtsystems. Eine entsprechend breite Sichtbarkeit des Konzeptes ist damit gegeben.

Fortbildungskriterium (1) Dynamisches Ressourcen-Management überschneidet sich mit schwergewichtigen Themenkomplexen wie der Abschätzung von Laufzeitverhalten und mit der Kommunikation in und dem Ablauf von nebenläufigen Prozessen. Wie Buttazzo erläutert, ist die Bandbreite an Möglichkeiten zur dynamischen Rekonfiguration schwierig zu koordinieren [But06]. Auch wenn es potentiell wichtig für weitere Ideen im Design eingebetteter Systeme ist, nimmt der Autor an, dass Bachelorstudierenden die Grundkonzepte dynamischen Ressourcenmanagements nur schwer zu vermitteln sind, da hierfür erst das Verständnis über statische Ressourcen vorhanden sein muss. Statisches Ressourcemanagement in eingebetteten Systemen ist hingegen auch auf Bachelorniveau vermittelbar. Ein praktisches Beispiel, das sich gut in Lernsituationen integrieren lässt, betrifft die höchstmögliche Anzahl an nutzbaren Speicherzellen in eingebetteten Systemen oder die verfügbaren Logikblöcke in einem FPGA. Ein Demoboard für einen Mikrocontroller verfügt typischerweise nur über wenige Kilobyte RAM, was die Umsetzung von größeren Projekten stark einschränkt. Das Kriterium wird mit den genannten Einschränkungen abgelehnt.

Zeitkriterium (1) Die durch das Zeitkriterium festgelegte Spanne von 20 Jahren kann teilweise für die untersuchte Idee belegt werden [Zav82], [NS95]. Allerdings ist eine breite Beobachtbarkeit nicht gegeben, da sich zu wenige Hinweise auf eine rege wissenschaftliche Debatte der Idee im angesprochenen Zeitraum finden. Auch wenn ersichtlich ist, dass die Verwaltung verfügbarer Ressourcen auch für die kommenden Jahre ein notwendiger Umstand sein wird, ist diese Aussage nicht spezifisch für das dynamische Ressourcenmanagement zu sehen, sondern auch mit statischen Methoden umsetzbar. Ein anschauliches Beispiel ist die Stromaufnahme mobiler Systeme, die laut Lee, Leung und Son zu einem der kritischsten Constraints für entsprechende Systeme geworden ist [LLS07]. Hier vermischen sich in der Regel beide Arten von Ressourcenverwaltung. Das Zeitkriterium kann aus den genannten Gründen nicht erfüllt werden.

Sinnkriterium (3) Durch die große Bandbreite der Thematik und der Tatsache, dass es (noch) keine vorgefertigten Schemata gibt, um eine optimale Ressourcenverteilung zu ermitteln und umzusetzen, ist der Themenbereich aus wissenschaftlicher Sicht interessant. Verschiedene Techniken zum dynamischen Verwalten von Ressourcen umfassen bspw. das „Dynamic-Voltage-Scaling“ (DVS) oder „Dynamic-Resource-Management“ (DRM) [Kim+08].

Im Bereich von Betriebssystemen und Middlewares verteilter Systeme ist die Bewertung von Ressourcenverfügbarkeit und die entsprechende Zuweisung an Teilsysteme von praktischer Relevanz. Damit ist für die Idee die wissenschaftliche und praktische Bedeutung belegt und das Kriterium somit vollständig erfüllt.

Varianzkriterium (2) Für die analysierte Idee sind deutliche Bezüge zu Themengebieten wie der „Design-Space-Exploration“, rekonfigurierbaren Architekturen oder Betriebssystemen und Middlewares zu erkennen. Alle genannten Ideen setzen jedoch andere Schwerpunkte und sind methodisch auf einem anderen Abstraktionsniveau als die vorliegende Idee zu sehen. Eine eigene Facette wird durch das Ressourcen-Management in eingebetteten Systemen jedoch nicht hergestellt, weswegen das Kriterium nur teilweise erfüllt ist.

A.9. Fehlertoleranz und Quality of Service

Die Idee der Fehlertoleranz ist sehr eng mit der Definition eines zuverlässigen Systems verbunden (siehe Varianzkriterium). Die Aufgabe des Konzeptes ist es, einen Weg zu finden, Fehler zu tolerieren, die in der Entwicklung entstanden sind, aber erst im Betrieb auffallen [Lap85]. Damit verbunden ist die Anforderung, die Operation des Systems auf einem definiertem Niveau zu gewährleisten, meist mit dem Hintergrund, Schaden von Mensch und Maschine abzuhalten. Neben Fehlererkennung und der Bestimmung von Fehlerparametern sind auch Konzepte wie „Fail-Over“ oder Redundanz als technische Realisierungen in der Idee enthalten.

Nach Laprie lassen sich Bezüge zwischen dem Begriff „Verlässlichkeit“ und „Quality of Service“ herstellen. Er schreibt dazu:

> „Computer system dependability is the quality of the delivered service such that reliance can justifiably be placed on this service“ [Lap85, S. 2].

Weiterführend definiert er Fehlertoleranz als:

> „Fault-tolerance: How to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring“ [Lap85, S. 2].

Horizontalkriterium (3) Fault-Tolerance ist in verschiedenen Ebenen der Entwicklung eingebetteter Systeme beobachtbar. Beispiele hierfür sind die Kommunikation auf Bussystemen oder die redundanten Auslegungen von Komponenten auf Hardware- und Softwareebene [Hua+12]. Für die Implementierung eines sicherheitskritischen Systems ist Fehlertoleranz nur ein Aspekt, der neben Fehlervermeidung und Fehlererkennung zu bedenken ist [Lap85].

Ähnlich zur Idee des „Reliable-Designs“ ist Fehlertoleranz eine der zentralen nichtfunktionalen Anforderungen, die typisch für eingebettete Systeme sind, da das Verhalten physikalischer Prozesse nicht zuverlässig vorhergesagt werden kann. Ebenso regeln eingebettete Systeme physikalische Prozesse, die schädlich für Mensch und Umwelt sein können, falls eine Fehlfunktion auftritt. Deswegen ist die Idee bei vielen eingebetteten Systemen zu beobachten und gewährleistet damit die vom Horizontalkriterium geforderte breite Anwendbarkeit.

Fehlertoleranz ist in allen Entwicklungsphasen bis auf den „Disposal Process“ eine Anforderung die bedacht werden muss, und deswegen einen Einfluss auf die entsprechende Systemumsetzung besitzt. Der zweite Aspekt des Horizontalkriteriums ist deshalb ebenso als erfüllt anzusehen.

Fortbildungskriterium (1) Das Fortbildungskriterium sieht der Autor für die analysierte Idee als nicht erfüllt an. Durch die starken Bezüge zur Spezifikation funktionaler und nicht-funktionaler Anforderungen und dem nötigen Vorwissen im Bereich „Reliable-Design“ und „Dependability“ ist die Vermittlung entsprechender Kompetenzen an Bachelorstudierende als schwierig zu bewerten. Das Verständnis über die groben Zusammenhänge fehlertoleranter Systeme ist unter Umständen noch lehrbar, die Kompetenz, ein eigenes, fehlertolerantes System zu entwickeln, aufgrund des nötigen Vorwissens erst nach umfangreicher Praxiserfahrung möglich. Die Idee formaler Methoden zur Verifikation und Validierung der Systemspezifikation, genau wie der Bereich der Co-Simulation und Co-Verifikation, stehen in Wechselwirkung mit fehlertoleranten Systemen. Fay et al. stellen einen Kurs im Bachelorstudium zum Thema „Fault-Tolerance“ überblickartig vor [Fay+07]. Eine Evaluation wird jedoch nicht erwähnt, da der Kurs noch nicht durchgeführt wurde und somit die geäußerten Überlegungen als Konzept einzuordnen sind. Da für Bachelorstudierende das *Strukturieren, Erarbeiten* und *Anwenden* (vgl. Qualitätsrahmen Deutscher

Hochschulen) nicht allgemein als für diesen Themenbereich möglich angesehen werden kann, ist das Fortbildungskriterium mit dieser Einschränkung abzulehnen.

Zeitkriterium (3) Das Zeitkriterium für die analysierte Idee wird erfüllt. Der dafür notwendige Beleg ist durch Publikationen wie [SJ95], [WRS95] und [Kop+89] gegeben. Noch weiter zurück reicht die Diskussion über fehlertolerante PC-Anwendungen, wobei nur selten die Software-Ebene eingebetteter Systeme der Betrachtungsgegenstand ist (siehe bspw. [Lev91]). Ähnlich zum Themengebiet des „Reliable-Designs“ ist davon auszugehen, dass die Fehleranfälligkeit zukünftiger Systeme durch die Änderungen von Fertigungsgrößen der Hardware zunehmen wird [Lee08]. Verlässliche Systeme müssen deswegen Fehler erkennen, vermeiden und wenn nicht anders möglich auch tolerieren können. Es ist somit auch in Zukunft wichtig zu erkennen, dass Hardware nicht fehlerfrei ist und Softwarekomponenten von Fehlern der verwendeten Plattformen ausgehen müssen.

Sinnkriterium (3) Auf wirtschaftlicher Ebene erfordert die Entwicklung eines sicherheitskritischen Systems grundverschiedene Ansätze zu üblichen Vorgehensweisen. Ein sehr anschauliches Beispiel gibt Lee für die Erstellung eines Mikrocontrollers im Luftfahrtbereich. Damit verknüpft ist die Anforderung, die gleichen Hardwarebauteile der gleichen Fertigungsreihe für die nächsten 50 Jahre vorrätig zu halten, um sich kostspielige Verifizierungsmaßnahmen eines Echtzeitsystems zu sparen [Lee06]. Die Verbindung von Echtzeitanforderungen, Fehlertoleranz und der Einhaltung anderer nicht-funktionaler Anforderungen wie bspw. der Leistungsaufnahme stellt sich als besondere Herausforderung dar, die im wissenschaftlichen Kontext insbesondere in Bezug auf verteilte Systeme untersucht wird. Einfache Techniken wie die Redundanz einiger Bauteile können den genannten Anforderungen nur selten gerecht werden. Begrenzende Faktoren sind dabei die Leistungsaufnahme, Kosten und die Baugröße.

Fehlertoleranz stellt deswegen eine wissenschaftlich wie praktisch relevante Idee dar, die das Sinnkriterium vollständig erfüllt.

Varianzkriterium (1) Das Varianzkriterium lehnt die Idee fehlertoleranter Systeme ab, da es lediglich eine Verfeinerung der allgemeinen Anforderung hinsichtlich verlässlicher Systeme ist, die bereits in „Reliable-Design“ in größerem Umfang enthalten sind. Starke Ähnlichkeiten zu anderen Ideen wie der Controller-Synthese und den bereits angesprochenen formalen Verifikationsmethoden bekräftigen diese Entscheidung.

A.10. Betriebssysteme, Middlewares, Laufzeitumgebungen und Echtzeit-Kernel

Im folgenden Abschnitt sind die Begriffe Betriebssystem, Middleware, Laufzeitsystem und Echtzeitkernel zusammengefasst, da sie die Idee des abstrakten Res-

sourcenzugriffs und der Bereitstellung von grundlegenden Systemdiensten (eine Form der Abstraktion und vertikalen Partitionierung) teilen. Eine Definition für moderne Betriebssysteme gibt Henderson:

> „An operating system is an overarching program that manages the resources of the computer. It runs programs and provides them with access to memory (RAM), input/output devices, a file system, and other services.“ [Hen09, S. 352]

Horizontalkriterium (3) Im Anwendungsgebiet eingebetteter Systeme erfüllen Betriebssysteme die grundlegenden Funktionen der Ressourcenvermittlung, hauptsächlich zwischen Hardware und Software. In der Implementierungsphase arbeiten die Entwickler aus diesem Grund nicht mehr direkt mit der Hardware oder Software, sondern einer vom Betriebssystem abstrahierten Schnittstelle. Das hat den Vorteil, dass insbesondere die Softwarekomponenten nicht mehr an die Hardware angepasst werden müssen, solange das Betriebssystem die Vermittlung übernimmt. In der Design-Phase ist jedoch die Ressourcennutzung durch das Betriebssystem selbst zu berücksichtigen und auf die Hardwareanforderungen zu übertragen. Die Anforderungen an ein Betriebssystem für eingebettete Systeme unterscheiden sich deutlich zu denen von Desktop-Computern (siehe Sinnkriterium). Die Breite der Anwendbarkeit der Idee ist mit der geschilderten Argumentation ersichtlich.

Die Entwicklungsphasen, in der die Idee berücksichtigt wird, sind vielfältig. Neben dem Design und der Implementierung muss ein Echtzeitbetriebssystem beispielsweise in der Simulation oder der Verifikation von nicht-funktionalen Anforderungen berücksichtigt werden.

Fortbildungskriterium (3) Die Ideen der Vermittlungsschicht und des Ressourcenzugriffs lassen sich auf verschiedenen Niveaustufen erklären. Das Konzept ist auch Bachelorstudierenden aus dem täglichen Umgang mit Computersystemen bekannt. Es wird erwartet, dass dieses praktische Vorwissen die Vermittlung zugehöriger Konzepte günstig beeinflusst, indem Erfahrungen einfacher mit dem neu aufgenommenen Wissen verknüpft werden können. Für weitere Ideen, wie das „Platform-based Design“, ist das Themengebiet als Basis zu sehen. Wie bei allen Ideen mit vergleichbar breiter Anwendbarkeit gibt es auch bei der vorliegenden Idee Aspekte, die erst später im Studium sinnvoll zu vermitteln sind. Für diese Idee sind dies beispielsweise die Themenbereiche „Scheduling“ und „Echtzeitbetriebssysteme“.

Zeitkriterium (3) Belege für die Erfüllung des Zeitkriteriums aus dem Bereich „Real-Time Kernels“ finden sich in [Spi90] und [JLT85]. Des Weiteren hat Tanenbaum vor 20 Jahren darauf hingewiesen, dass Betriebssysteme in batteriebetriebenen eingebetteten Systemen eine entscheidende Rolle spielen werden [Tan92]. Auch Teilbereiche der eingebetteten Systementwicklung wie Sensor-Netzwerke und andere verteilte Systeme hat er als möglichen Einsatzort für diese Art vermittelnder Abstraktionsschichten gesehen [Tan92]. Die Historie um die Diskussion von

Middlewares, Betriebssystemen und Real-Time-Kernel in eingebetteten Systemen genügt somit den Anforderungen des Zeitkriteriums.

Sinnkriterium (3) Der UBM-Studie zu Folge werden Vermittlungsschichten, wie die in dieser Sektion vorgestellten, bei 70% aller Projekte eingesetzt [Tec13]. Die praktische Relevanz der Idee ist demnach deutlich wahrnehmbar. Aus wissenschaftlicher Sicht ist insbesondere die Forschung an Echtzeitbetriebssystemen und verteilten eingebetteten Systemen immer noch ein aktuelles Forschungsgebiet (siehe [JLT85], [GLV13]). Zusammen mit der Sicherstellung funktionaler- und nicht-funktionaler Anforderungen ergeben sich aus dem Forschungsgebiet neue Problemklassen, deren Lösung und Beherrschung Gegenstand aktueller Forschung sind (siehe auch Abschnitt A.9). Ein weiteres Forschungsfeld sind Betriebssysteme für rekonfigurierbare Hardware (siehe [SWP04]).

Varianzkriterium (2) Insbesondere die Berücksichtigung der Echtzeitfähigkeit stellt ein Alleinstellungsmerkmal dar, das in keiner anderen analysierten Idee einen ähnlich großen Einflussfaktor besitzt. Aus diesem Grund ähneln sich auch das „Platform-based Design“ und die hier analysierten Betriebssysteme und Real-Time-Kernel nicht genug, um das Varianzkriterium negativ zu bewerten.

Wie aus den Erklärungen ersichtlich, bestehen starke Verknüpfungen zum Themenkomplex Ressourcen-Management. Der Autor ist jedoch der Auffassung, dass die Begriffe Ressourcen-*Verwaltung* und Ressourcen-*Anforderung* nicht synonym sind. Da beide Ideen Aspekte teilen, kann das Varianzkriterium nicht mehr vollständig positiv evaluiert werden. Bei Betriebssystemen ist insbesondere durch den Aspekt der Echtzeit eine Facette vorhanden, die bei der Ressourcen-Verwaltung noch nicht berücksichtigt wurde.

A.11. Virtualisierungstechniken

Virtualisierung ist die Erzeugung einer virtuellen Schicht, typischerweise zwischen Betriebssystem und Hardware, die einen Teil der tatsächlich verfügbaren Schichtenfunktionen implementiert. Die Schicht wird durch einen sogenannten *Hypervisor* bereit gestellt, der entweder direkt auf der Hardware oder aber auf dem Betriebssystem aufsetzt. Die Kernidee, die hierbei zum Ausdruck kommt, ist die Abstraktion von Softwarekomponenten oder Applikationen von der unterliegenden Hardwareplattform und gleichzeitig die Reduktion auf die notwendigsten Eigenschaften und Methoden, um einheitliche Ressourcen-Schnittstellen und eine flexiblere Konfiguration zu ermöglichen.

Horizontalkriterium (1) Heiser sieht eines der Einsatzgebiete von Virtualisierungstechniken in der Trennung verschiedener Funktionalitäten wie Consumer-Betriebssystem und Echtzeitbetriebssystem auf demselben Prozessor [Hei08]. Mit

dem stetigen Voranschreiten der Multi-Core-Technologien bezweifelt Heiser jedoch, dass dieses Anwendungsszenario für längere Zeit relevant sein wird, denn die Betriebssysteme könnten dann genauso gut auf verschiedenen Kernen laufen. Sicherheitsbedenken können teilweise durch die logische Trennung von virtuellen Maschinen, die durch den Hypervisor angesprochen werden, gelöst werden. Somit ist zumindest erkennbar, dass Überlegungen zur Umsetzung der Idee auf Design- und Implementierungsebene erfolgen sollten. Eine breite Anwendung der Idee ist jedoch weder im Anwendungsfeld noch in den Entwicklungsphasen zu erkennen. Das Horizontalkriterium ist deswegen als nicht erfüllt anzusehen.

Fortbildungskriterium (1) Es ist fraglich, ob grundlegende Kenntnisse über Betriebssysteme und Middlewares bei den Studierenden vorhanden sein müssen, um das weiterführende Konzept der Virtualisierung zu verstehen und anwenden zu können. Typ-2-Hypervisoren setzen, im Gegensatz zu Typ-1-Hypervisoren, immer auf Betriebssystemen auf [Man13]. Zumindest für diesen Fall müssen also zentrale Funktionen eines Betriebssystems und die damit zusammenhängende Idee einer ressourcenorientierten Vermittlungsschicht verstanden sein. Virtualisierung stellt zudem für Bachelorstudierende keinen notwendigen Baustein dar, der für das Erlernen weiterer Ideen zwingend erforderlich ist. Masterstudierende mit einem Fokus auf sicherheitskritische Anwendungsgebiete könnten hingegen auf die Idee angewiesen sein (siehe Horizontalkriterium). Durch die eingeschränkte Vermittelbarkeit und den fraglichen Nutzen für Bachelorstudierende wird das Fortbildungskriterium nicht erfüllt.

Zeitkriterium (1) Einige Quellen erwähnen hauptsächlich militärische oder für die Luftfahrt relevante Virtualisierungsplattformen und Betriebssysteme, die Mitte der 90er-Jahre veröffentlicht wurden [Jon11]. Cohen und Rohou sind der Auffassung, dass auch die Java-Virtual-Machine (JVM) und ihre Portierung auf eingebettete Systeme als Virtualisierung anzusehen ist [CR10]. Dies würde bedeuten, dass die Idee schon seit den 90er-Jahren im Anwendungsgebiet beobachtbar ist. Allerdings scheint die Auffassung einer solche Virtualisierung verschieden von der in dieser Arbeit analysierten Idee zu sein, da es sich dabei um Paravirtualisierung handelt. Das Zeitkriterium wird deswegen nicht erfüllt.

Sinnkriterium (2) Die Fachforschung analysiert aktuell Möglichkeiten, sicherheitskritische Anforderungen eingebetteter Systeme durch die Nutzung von virtuellen Maschinen zu erfüllen. Ein typisches Beispiel ist die Trennung von beruflichen und privaten Kontexten bei Smartphones [Bra+08]. Hypervisoren werden dafür genutzt, eine zusätzliche Abstraktionsschicht zwischen Betriebssystem und Hardware einzuführen, um auch die eigentliche Vermittlungsschicht auf andere Hardware-Plattformen portieren zu können und somit geringere Kosten bei einer erneuten Verifikation zu haben (bspw. [CRM10]).

Es ist jedoch zu erwarten, dass ein Großteil der Studierenden in ihrem späteren Arbeitsgebiet nicht mit dem Themenbereich konfrontiert werden (siehe auch Zeitkriterium), da Virtualisierung für Industriecontroller und nicht sicherheitskritische

Systeme kaum Anwendung findet. Die Teilnehmer der UBM-Studie unterstützen diese These. Mehr als 80% der befragten Entwickler gaben an, keine Virtualisierung oder Hypervisoren in den nächsten 12 Monaten verwenden zu wollen. Ebenso planen sie keinen Einsatz der genannten Techniken nach diesem Zeitraum [Tec13].

Die praktische Signifikanz der Idee kann damit nicht beobachtet werden. Allerdings ist die Idee, wie erläutert, Teil eines regen wissenschaftlichen Diskurses. Das Sinnkriterium wird deswegen nur teilweise erfüllt.

Varianzkriterium (1) Das Varianzkriterium kann ebenfalls nicht für den Anwendungsbereich bestätigt werden. Der Hauptzweck einer Virtualisierung besteht aktuell in der Trennung von Kontexten zur Steigerung der Sicherheit eines Systems. Dieser Aspekt ist nur ein Bruchteil der Überlegungen, die bei Cybersecurity und der Vertraulichkeit eingebetteter Systeme eine Rolle spielen (siehe Abschnitt A.16). Auch die Idee der vermittelnden Abstraktionsschicht ist bereits deutlich vorher durch Betriebssysteme und Middlewares etabliert worden. „Virtual Prototyping“ und „Platform-based Design“ beinhalten ebenso Aspekte der Idee.

A.12. Design-Space-Exploration

In der Entwicklung eingebetteter Systeme sind viele variable Parameter wie die Wahl der Hardwareplattform oder der Systemarchitektur gegeneinander abzuwägen, um ein optimales System fertigen zu können. Anzahl und Wertebereich der Parameter geben die Menge an Zuständen und Designmöglichkeiten vor, deren Kombination eine noch größere und nicht mehr handhabbare Menge an Systemkonfigurationen ist. Dieses Problem nennt man „Design-Space-Explosion“. „Design-Space-*Exploration*“ beschreibt die Methode, diesen Raum einzugrenzen und dabei funktionale und nicht-funktionale Anforderungen zu berücksichtigen. Die folgende Definition von Sigdel unterstützt diese Beschreibung:

> „DSE is the process of analyzing functionally equivalent alternative design points, either architecture and/or application, in order to identify the 'optimal' design point.“ [Sig+08, S. 27]

Horizontalkriterium (3) Die Entscheidung, welche Verhaltens-, Architektur- oder Kommunikationsstruktur für die Umsetzung eines Systems am besten genutzt wird, ist ein Suchproblem im Raum aller möglichen Umsetzungen. Insbesondere für große Industriezweige (bspw. Automobilindustrie oder Luft- und Raumfahrttechnik) ist die Ermittlung geeigneter Systemkonfigurationen ein Kerngebiet der Systementwicklung [RB13]. „Design-Space-Exploration“ kann nicht nur genutzt werden, um eine mögliche Konfiguration an Komponenten für ein System zu finden, sondern auch, um funktionale oder nicht-funktionale Eigenschaften wie den Stromverbrauch, die minimale Prozessorgeschwindigkeit o. Ä. abzuschätzen [Hoc+13]. Auch wenn ein formaler Ansatz zur „Design-Space-Exploration“ bei kleineren Systemen eine

untergeordnete Rolle spielen kann, ist doch davon auszugehen, dass die Abschätzung und Bewertung von Design-Alternativen in jedem Entwicklungsprojekt für eingebettete Systeme vorhanden ist. Entsprechend ist die breite Anwendung der Idee nicht auf die genannten Industriezweige beschränkt, sondern auch in kleineren Entwicklungsprojekten vorhanden.

Die Methode ist in Entwicklungsphasen wie der Systemintegration, der Optimierung und natürlich dem Systemdesign einsetzbar [KJS11] und erfüllt damit den zweiten Aspekt des Horizontalkriteriums.

Fortbildungskriterium (1) Die analysierte Idee baut teilweise stark auf Vorwissen zu formalen Methoden auf und ist deswegen nur begrenzt für die Vermittlung auf Bachelorniveau geeignet. Ebenso werden verschiedene Analyseaspekte wie Sicherheit, Leistungsverbrauch, Echtzeitverhalten oder Kosten miteinander vernetzt, deren Einschätzung schwierig ist [Bas+10]. Die Idee bietet neben den erlernten Grundlagen zu formalen Methoden kein Wissen, welches als Basis für weiterführende Ideen genutzt werden kann. Neben einer Umsetzung mit formalen Methoden gibt es auch Ansätze, die „Design-Space-Exploration“ durch exzessive Simulation virtueller Prototypen durchzuführen. Vermittelbar und wichtig ist in Zusammenhang mit dieser Idee das grundlegende Verständnis darüber, dass Anforderungen oft wegen der Anzahl an möglichen Kombinationen nicht in einer einfachen Liste gegeneinander abgewogen werden können und das formale Methoden sowie Simulation geeignete Konfigurationen von Implementierungen und Komponenten herausfiltern können (siehe Varianzkriterium).

Auf Bachelorniveau ist die Anwendung formaler Methoden kaum möglich, da neben den Grundlagen der Theoretischen Informatik (bspw. Beweisführung) auch Fachkenntnisse über funktionale und nicht-funktionaler Anforderungen und deren Gewichtung im Rahmen der Anwendung vorliegen müssen. Es ist davon auszugehen, dass eine praktische Veranstaltung das Verständnis über die Art der Anforderungen fördert und als Basis für eine theoretische Vertiefung im Masterstudium dienen kann.

Rover et al. stellen zwar ein thematisch verwandtes Kurskonzept auf Bachelorniveau vor, setzen dabei jedoch eine Menge an Vorwissen voraus, das in Studiengängen, die nicht spezifisch auf eingebettete Systeme ausgerichtet sind, nur schwerlich abgedeckt werden kann [Rov+08]. Die Analyse des Fortbildungskriteriums kann aus den genannten Gründen nicht positiv evaluiert werden.

Zeitkriterium (3) „Design-Space-Exploration“ ist bereits 1973 für Register-Transfer-Systeme wissenschaftlich diskutiert worden [BS73]. Der Ausgangspunkt war damals die funktionale Beschreibung des zu entwickelnden Systems und nicht wie heute die zusätzliche Abschätzung von nicht-funktionalen Parametern. „Design-Space-Exploration“ nach heutigem Verständnis erfüllt das Zeitkriterium trotzdem. Exemplarische Quellen, die eine wissenschaftliche Diskussion um das Themengebiet belegen, sind [COB95], [DH94] und [TMW94].

Sinnkriterium (3) Ohne Techniken zur automatischen „Design-Space-Exploration“ wären komplexe eingebettete Systeme allein aus zeitlichen Gründen kaum mehr möglich. Eine Alternative, die in den kommenden Jahren voraussichtlich an Wichtigkeit gewinnen wird, ist die Simulation [Tec13]. Zumindest im Bereich der Analyse von Systemattributen sind virtuelle Prototypen nutzbar und auch ohne vertieftes Vorwissen formaler Methoden anwendbar. Ob Studierende zukünftig auf die „Design-Space-Exploration“ mit formalen Methoden oder der Simulation eines Prototyps zurückgreifen, hängt stark vom Einsatzgebiet des eingebetteten Systems und Fortschritten in der werkzeugunterstützten Entwicklung ab. Wie in der Analyse zum Horizontalkriterium erläutert, ist eine abgeschwächte Form der „Design-Space-Exploration“ in jeder Entwicklung eingebetteter Systeme zu finden, und deswegen erst einmal unabhängig von der konkreten Umsetzung.

In der wissenschaftlichen Debatte ist Design-Space-Exploration, allein wegen ihrer Nähe zur theoretischen Informatik, wichtig. Gemischte Ansätze aus Simulation und formalen Methoden werden genauso diskutiert wie die Integration der Methode in System-Level-Design mit technikunabhängigeren Beschreibungskonzepten [Müh+11], [Pim+08].

Da die theoretische und praktische Relevanz nachgewiesen werden konnte, wird das Sinnkriterium vollständig erfüllt.

Varianzkriterium (2) Die analysierte Idee besitzt erwartungsgemäß einen starken Bezug zu formalen Methoden. Die in Abschnitt A.17 analysierte Idee der formalen Verifikation und Validierung nutzt formale Methoden jedoch nicht vordergründig zur Abschätzung von Systemeigenschaften und der darauf aufbauenden Auswahl von Komponenten. Je nach Gewichtung von nicht-funktionalen Anforderungen kann auch ein Bezug zu „Reliable Design“ hergestellt werden, der jedoch nicht die Ideenebene betrifft und somit keinen negativen Einfluss auf die Beurteilung des Varianzkriteriums hat. Kang, Jackson und Schulte weisen darauf hin, dass „Design-Space-Exploration“ ebenso für das Prototyping eines Systems und der damit verbundenen Simulation von Systemeigenschaften nützlich sein kann [KJS11].

Damit stellt Design-Space-Exploration keine eigenständige, von anderen Ideen unabhängige, Methode oder Sichtweise mehr dar. Das Varianzkriterium wird mit Einschränkung erfüllt, da die Verbindung formaler Methoden für die Abschätzung von Systemattributen und Designalternativen eine Erweiterung anderer Ideen darstellt.

A.13. Reliable Design

Burns versteht „Reliability“ als Verlässlichkeit eines Systems [BL91]. Eine nicht-funktionale Anforderung, die mit der Verlässlichkeit verwandt ist, ist die Robustheit. Nach Henzinger ist ein System robust, wenn es trotz Schwankungen der Betriebsparameter das intendierte Verhalten zeigt [Hen08]. Weitere, in diesem Kontext relevante Schlagwörter wie Sicherheit und Fehlertoleranz implizieren in der Regel

weitere nicht-funktionale Anforderungen. Die zentrale Idee hinter diesem Themenbereich ist im Gegensatz zum Großteil der anderen analysierten Ideen nicht eine bestimmte Methode oder ein Lösungsansatz, sondern eine Sichtweise und Methode zum Problemraum *Verlässlichkeit*.

Horizontalkriterium (3) Wie bereits in der Einführung zur Analyse dieser Idee beschrieben, ist Verlässlichkeit eine nicht-funktionale Anforderung, die eine thematische Verbindung zu anderen nicht-funktionalen Attributen wie *Verfügbarkeit, Integrität, Fehlertoleranz* oder *Wartbarkeit* besitzt [Avi+04]. Diese Attribute sind für ein breites Spektrum der Hardware- und Softwareumsetzungen von Bedeutung. Im Kontext des Entwicklungsvorgehens sind es insbesondere die Design- und Implementierungsphase, in der sich maßgeblich mit der Umsetzung dieser Qualitätsbegriffe beschäftigt wird. Weitere Auswirkung hat die Idee auf eine eventuelle Systemoptimierung und natürlich auf die Verifikations- und Simulationsphasen (Testphase), bei denen überprüft wird, ob das entwickelte System wirklich verlässlich operiert. Der erste Aspekt des Horizontalkriteriums ist damit erfüllt.

In der Definition zum Horizontalkriterium wird besonders der Umstand betont, dass eine Idee neben der Relevanz für verschiedene Entwicklungsphasen auch eine breite Anwendbarkeit besitzen kann. Bei der vorliegenden Idee ist dieser Umstand in besonderer Weise erfüllt, denn im Gegensatz zu vielen anderen Anwendungsgebieten und der Softwaretechnik im Allgemeinen sind eingebettete Systeme ganz besonders auf einen verlässlichen Betrieb angewiesen. Diese Systeme können in der Regel nicht im laufenden Betrieb aktualisiert werden und besitzen damit einen deutlich höheren Anspruch auf fehlerfreies und damit verlässliches und robustes Verhalten. Henzinger fügt hinzu, dass die Grundlagen der Softwaretechnik durch mathematische Ansätze automatisch robust sind [Hen08]. Es lässt sich beweisen, ob ein Softwaresystem die Aufgabe erfüllt oder nicht. Bei eingebetteten Systemen ist diese Form von Robustheit laut Henzinger nicht automatisch gegeben und deswegen eine besondere Herausforderung für den Entwurf eingebetteter Systeme [Hen08]. Damit ist auch der zweite Aspekt und somit das Horizontalkriterium insgesamt vollständig erfüllt.

Fortbildungskriterium (3) Das Spektrum an Anforderungen, die zu einem verlässlichen System gehören, ist sehr weitläufig und muss differenziert betrachtet werden. Echtzeitsysteme bspw. besitzen oft Anforderungen im Kontext der Systemverlässlichkeit und sind auf Bachelorniveau gut zu vermitteln (siehe bspw. [Hoc+11]). Während im Bachelorstudiengang die Wichtigkeit verlässlicher Systeme und Grundattribute zur Einschätzung eines solchen Systems behandelt werden, können sich Masterstudierende mit der Überprüfung eines virtuellen Prototyps oder einer formalen Spezifikation der Systemfunktionalität befassen und so weitere Kompetenzen in der praktischen Entwicklung erlangen. Die Empfehlungen der Gesellschaft für Informatik zeichnen einen ähnlichen Weg, indem einige Module, wie beispielsweise Echtzeitsysteme, mehrfach im Bachelor- und im Masterstudium genannt werden (siehe [Hoc+11]). Auch das Thema Verlässlichkeit wird als Themengebiet im Bachelorstudiengang vorgeschlagen.

Die grundlegende Vermittelbarkeit und die Relevanz für andere Ideen kann in diesem Fall also bestätigt werden. Damit ist das Fortbildungskriterium vollständig erfüllt.

Zeitkriterium (3) Der Themenkomplex genügt dem Zeitkriterium. Einige beispielhafte Publikationen, die belegen, dass der Themenbereich seit mindestens 20 Jahren breit beobachtbar ist, finden sich in [GGA95], [Wol94], [MGA94] und [AD94]. Als Unterschiede von damaligen zu heutigen Vorgehensweisen ist neben dem technischen Fortschritt auch die gestiegene Integration von Hardware- und Software-Komponenten hin zum Hardware-/Software Co-Design zu nennen. In Anbetracht zukünftiger Fertigungsverfahren im Nanometerbereich ist mit einer wachsenden Bedeutung für die Erstellung verlässlicher Systeme zu rechnen [NX06].

Sinnkriterium (3) Verlässlichkeit ist mit wechselnder Gewichtung oft ein wichtiges Attribut der Entwicklung eingebetteter Systeme und deswegen auch für zukünftige Experten relevant. In sicherheitskritischen Systemen mit potentiell gesundheitsgefährdenden Mechanismen ist dieser Umstand besonders wichtig (bspw. Airbags). Nicht-funktionale Anforderungen, zu denen auch die Verlässlichkeit gehört, werden bei eingebetteten Systemen im Consumer-Bereich ebenfalls wichtiger [Koo07], [NX06].

Aus wissenschaftlicher Perspektive ist damit zu rechnen, dass Verlässlichkeit für zukünftige Systeme wichtiger wird. Durch die schrumpfenden Fertigungsgrößen der Hardware ist mit einer deutlich höheren Fehleranfälligkeit zu rechnen, der auf der Ebene des Systemdesigns, der Implementierung und der System-Integration begegnet werden muss. Maßnahmen zur Steigerung der Verlässlichkeit solcher Systeme werden zur Zeit vermehrt erforscht (siehe bspw. [Zhu06], [Hen+14]).

Beide Aspekte des Sinnkriteriums sind somit erfüllt.

Varianzkriterium (2) „Reliable Design“ ist ein Entwicklungsparadigma und als „*Reliability*“ zeitgleich eine nicht-funktionale Anforderung. Die Überlappung mit der gleichnamigen Idee der Synthese und Analyse nicht-funktionaler Anforderungen ist direkt ersichtlich. Um einen verlässlichen Systementwurf zu erhalten, sind Techniken wie die Design-Space-Exploration oder Ansätzen zur Fault-Tolerance nötig. Weitere Themengebiete, die in Relation zur analysierten Idee stehen, sind beispielsweise die Controller-Synthese oder formale Methoden zur sicherheitskritischen Verifikation von Systemeigenschaften. Die methodische Ebene hat deswegen sehr viele Bezüge zu anderen Ideen, die eine vollständig positive Evaluation des Varianzkriteriums ausschließen. Das Kriterium wird jedoch mit Einschränkungen erfüllt, da Verlässlichkeit eine besondere Stellung unter allen nicht-funktionalen Anforderungen im Bereich der Entwicklung eingebetteter Systeme einnimmt. Wie im Horizontalkriterium erläutert, ist Verlässlichkeit und Robustheit ein zentraler Unterschied zwischen klassischen Computern und eingebetteten Systemen. Wie

auch aus der Argumentation der methodischen Ebene zu sehen, sind diese nichtfunktionalen Anforderungen gleichzeitig der Ursprung erweiterter Konzepte wie Fehlertoleranz oder formaler Verifikation.

A.14. Virtuelles Prototyping

Ein virtueller Prototyp dient zur Präsentation, zur Analyse und zum Testen eines zu entwickelnden Systemes, bevor dieses physikalisch gefertigt wird. Dazu wird ein virtueller Prototyp erstellt, der hinsichtlich aller für die Entwicklung relevanten Attribute dem physikalischen System gleicht. Nach einer Analyse bestehender Definitionen des Begriffs definiert Wang *„Virtual Prototyping“* als:

> „Virtual prototype, or digital mock-up, is a computer simulation of a physical product that can be presented, analyzed, and tested from concerned product life-cycle aspects such as design/engineering, manufacturing, service, and recycling as if on a real physical model. The construction and testing of a virtual prototype is called virtual prototyping (VP).“ [Wan02, S. 4]

Horizontalkriterium (3) Das Erstellen eines Prototypen ist ein in der Softwaretechnik etabliertes Verfahren. Techniken wie das *Rapid Prototyping* verfolgen prinzipiell ähnliche Zielsetzungen wie die vorliegende Idee: Abschätzung von Systemparametern und Machbarkeitsstudien mit einem virtuellen Objekt, das nur in Bezug auf relevante Attribute vollständig nachgebaut wurde. Im Bereich der Entwicklung eingebetteter Systeme ist oft ein virtueller Hardware-Prototyp gewünscht, um die Software bereits vor der Fertigung der Hardware testen zu können. Trotz der Gleichheit auf Wortebene sind beide Ansätze jedoch verschieden. Virtuelle Prototypen beinhalten auch die Abbildung mechanischer Komponenten und nicht nur die Verhaltenssimulation, die sonst mit dem Begriff verbunden wird.

Der erste Aspekt des Horizontalkriteriums erfordert eine breite Anwendbarkeit der Idee in vielen Entwicklungen eingebetteter Systeme. Auch wenn, wie in der Analyse zum Varianzkriterium geschildert, viele Entwickler zukünftig vermehrt virtuelle Prototypen einsetzen wollen, ist nicht klar, welche begriffliche Definition bei der entsprechenden Umfrage zugrunde gelegt wurde. Üblicherweise werden die eigentlichen Hardware- und Softwareentwickler nur in geringem Umfang Einfluss auf Servicepläne und Optionen zur Wiederverwertung des Systems haben. Nach der Definition von Wang sind diese Überlegungen jedoch auch ein Bestandteil des virtuellen Prototypings. Viel eher ist zu vermuten, dass die Umfrage auf rekonfigurierbare Plattformen wie FPGAs abzielte, die genutzt werden, um einen Systementwurf zu testen.

Der zweite Aspekt des Horizontalkriteriums wird erfüllt, da die Idee auf vielen Schichten des Entwicklungsprozesses relevant ist (siehe Definition). Zwar wird der Prototyp nur auf der Ebene des „Implementation Process“ umgesetzt, dieses Prototypensystem dann jedoch auf alle für das *echte* Produkt wichtigen Entwicklungsphasen hin verwendet und analysiert.

Fortbildungskriterium (1) Williams et al. haben zwei Kurse zum virtuellen Prototyping durchgeführt, geben in ihrer Zusammenfassung jedoch nur einen begrenzten Einblick in die Rückmeldungen der Studierenden [WKA03]. Inwiefern Studierende auf Bachelorniveau für die Lerninhalte genug Kompetenzen besitzen, kann aus dieser Publikation deshalb nicht abgeleitet werden. Zhang erläutert Herausforderungen und notwendige Qualifikationen für Mikrosystemtechniker, jedoch mit Fokus auf Problemstellungen im thermisch-mechanischen Bereich von Hardwarekomponenten [Zha03]. Da die Idee vornehmlich in den späten Entwicklungsphasen beobachtet werden kann, ist davon auszugehen, dass Studierende solide Kenntnis über die vorherigen Entwicklungsschritte besitzen müssen. Um die Anforderungen für einen virtuellen Prototypen zu extrahieren, müssen zuerst die Anforderungen an das zu erstellende System und die technischen Möglichkeiten einer Umsetzung für den virtuellen Prototyp erarbeitet worden sein. Zusätzlich kommen hierzu die von Wang erwähnten entwicklungsspezifischen Überlegungen hinsichtlich der Fertigung, der Systembetreuung und der Wartung [Wan02].

Da nicht davon ausgegangen werden kann, dass die dazugehörigen Kompetenzen bei den Studierenden bereits auf Bachelorniveau vorhanden sind, kann die Idee das Kriterium nicht erfüllen.

Zeitkriterium (2) Das allgemeine Prototyping ist in der Software- wie auch in der Hardwareentwicklung etabliert (bspw. [SB95]). Belege für die Erfüllung des Zeitkriteriums sind für das virtuelle Prototyping (wie von Wang definiert) schwer zu finden. Auch wenn verschiedene Publikationen wie [ME95] oder [HNM95] am für das Zeitkriterium relevanten Zeitpunkt vorhanden sind, ist die breite wissenschaftliche Diskussion des Konzeptes tendenziell erst um die Jahrtausendwende beobachtbar (bspw. [Ern98]). Das Zeitkriterium kann nur mit einer Abweichung von drei bis fünf Jahren erfüllt werden. Ob die Idee einen fundamentalen Kern besitzt, ist deswegen abzuwägen.

Sinnkriterium (3) Die Aberdeen-Gruppe hat eine Marktanalyse zum Themenbereich simulationsgetriebene Entwicklung durchgeführt, in der auch die Praxisrelevanz des virtuellen Prototypings untersucht wird [Inc06]. Diese wird als eine der grundlegenden Änderungen gegenüber traditionellen Vorgehensweisen beschrieben, die notwendig ist, um mit den kürzeren Time-To-Market-Zeiten mithalten zu können [Inc06]. Knapp ein Drittel der Teilnehmer (32%) der Embedded-Market-Study von UBM schätzen, dass virtuelles Prototyping in den nächsten Jahren an Relevanz für ihre Arbeit zunehmen wird [Tec13]. Aus praktischer Sicht erhoffen sich Entwickler eine deutliche Zeiteinsparung gegenüber herkömmlichen Entwicklungsansätzen (vgl. [Sin+01]).

Die wissenschaftliche Gemeinschaft diskutiert seit einigen Jahren über mögliche Verbesserungen des virtuellen Prototyping, besonders im Kontext von Co-Simulation und -Verifikation [HNM95], [Pal+13]. Die Signifikanz für wissenschaftliche und praktische Fragestellung kann also nachgewiesen werden.

Varianzkriterium (2) „Virtual Prototyping“ besitzt viele Ähnlichkeiten mit Ideen aus dem Bereich des Hardware/Software Co-Designs. Insbesondere die zeitgleiche Erstellung eines Hardwareprototypen zum Test oder der Integration anderweitig entwickelter Komponenten (bspw. Software) sind identisch. Die vorliegende Idee ist eine mögliche Umsetzung dieser Aufgabe. Durch die Einbindung mechanischer Komponenten und der über die Entwicklung des Systems hinausgehenden Nutzung des Prototyps ist dennoch eine Erweiterung des Hardware/Software Co-Designs zu erkennen. Das Varianzkriterium kann deswegen mit Einschränkungen erfüllt werden.

A.15. Domänenspezifische Anwendungen und Methoden

Die Kernidee der domänenspezifischen Entwicklung ist gleichermaßen Gegensatz und Erweiterung zur Idee *Abstraktion*. Einerseits wird durch die Spezifität auf den Anwendungsfall die mögliche Wiederverwendbarkeit für andere Anwendungsbereiche eingeschränkt, andererseits für gleichartige Anwendungsgebiete erhöht. Einige Experten vertreten die Auffassung, dass vollständige Abstraktion nicht lohnenswert sei, da anschließend wieder viel Arbeit in die Anpassung für den konkreten Anwendungsfall erfolgen müsse [Hud96]. In diesem Sinne ist eine domänenspezifische Sprache die *„ultimative Abstraktion“* [Hud96, S. 1].

Horizontalkriterium (2) Eine domänenspezifische Anpassung von Entwicklungsvorgehen und Werkzeugen findet sich größtenteils in den Phasen des Designs und der Implementierung. Beispiele hierfür sind domänenspezifische „Design-Space-Explorations“ [Cat+09] oder anwendungsgebiets-spezifische Sprachen [HM03]. Im Bereich der Programmier- und Beschreibungssprachen ist der Einfluss domänenspezifischer Anpassungen am deutlichsten zu sehen. Die Sprachen VHDL, Verilog, SysML und SystemC sind alle spezifisch auf ein Anwendungsgebiet oder eine Aufgabenstellung ausgerichtet. Im Gegensatz dazu gibt es die allgemeinen Sprachen wie bspw. C, C++ oder Java, die für verschiedenste Einsatzgebiete gleich gut geeignet sind.

Das Horizontalkriterium kann nicht vollständig erfüllt werden, da es hauptsächlich die zwei genannten Entwicklungsphasen sind, in denen die Idee beobachtbar ist. Durch die große Verbreitung domänenspezifischer Sprachen und Werkzeuge wie VHDL, Verilog oder Matlab/Simulink ist die breite Anwendbarkeit hingegen gut zu belegen.

Fortbildungskriterium (3) In der Softwareentwicklung ist der Schlüssel zur Wiederverwendung Abstraktion. Auch dort ist es nicht sinnvoll, zu viele Metaebenen einzuführen, da diese mit großem Aufwand wieder spezifischer auf den Anwendungsfall angepasst werden müssen. Diese Abwägung zu verstehen, ist auch für die

Entwicklung eingebetteter Systeme wichtig. Domänenspezifische Konzepte begegnen den Studenten in vielfacher Weise und sind für das Verständnis weiterführender Themenbereiche (bspw. „Reconfigurable Computing“, FPGAs) eine wichtige Voraussetzung. In der Softwareentwicklung gibt es nur wenig didaktische Forschung zu diesem Themenbereich, auf dem erste Ansätze zur Übernahme für die Technische Informatik und vergleichbare Gebiete aufbauen könnten. Synergieeffekte bei der Vermittlung von Kompetenzen zu domänenspezifischen Ansätzen können erwartet werden, da die Grundlagen der Programmierung und Modellierung in der Regel zu den ersten Vorlesungen der Zielgruppe gehören. Die für das Fortbildungskriterium wichtige Eigenschaft, dass der mit der Idee verbundene Lehrinhalt wichtig für darauf aufbauende Ideen ist, kann damit als gegeben angesehen werden.

In Hinsicht auf die Komplexität klassischer Programmiersprachen spricht aus Sicht des Autors kein Grund gegen die Vermittlung ähnlich komplexer domänenspezifischer Sprachen. Beispielhaft seien hier VHDL oder Verilog genannt. An der Universität Siegen und an anderen Hochschulen werden entsprechende Kurse seit Längerem auch für Bachelorstudierende erfolgreich angeboten (siehe bspw. [Mad+12], [RR13] und [Abi+10]). Da die Idee vermittelbar ist, wird auch der zweite Aspekt des Fortbildungskriteriums erfüllt.

Zeitkriterium (3) Domänenspezifische Anpassungen im Entwicklungsprozess sind seit den vom Zeitkriterium angesetzten 20 Jahren im interdisziplinären Umfeld beobachtbar. Neben der Hardwarebeschreibungssprache VHDL, die bereits 1987 von der IEEE standardisiert wurde, sind SQL für Datenbankanwendungen und Erlang als Programmiersprache für Hochverfügbarkeitssysteme wohlbekannt. Beide sind in den 70er- bzw. 80er-Jahren entstanden. Domänenspezifische Methoden werden voraussichtlich für die nächsten Jahre in der Systementwicklung relevant bleiben, da sie zusammen mit modellgetriebener Entwicklung die logische Fortsetzung der Abstraktion darstellen [LT09].

Sinnkriterium (2) Domänenspezifische Entwicklung hat laut Liggesmeyer und Trapp großes Potential, parallel zur modellgetriebenen Entwicklung die bisher weitreichenste Form der Abstraktion darzustellen [LT09]. Durch die problemnahe Systembeschreibung ist weniger Vorwissen zu klassischen Programmierkonstrukten und informatischen Konzepten, aber mehr domänenspezifisches Fachwissen gefragt. In diesem Zusammenhang hat sich das Konzept der „End-User-Entwicklung“ hervorgetan, das Anwendungs- und Systementwicklungen beschreibt, die komplett durch den späteren Abnehmer des Systems erstellt werden (vgl. [Fis+04], [Lie+06]). Aus praktischer Sicht sind die State-of-the-Art-Techniken VHDL, SystemC oder Matlab/Simulink zu nennen. Letzteres wird als De-facto-Standard in Bereichen der Signalverarbeitung oder der Automobil- und Luftfahrttechnik angesehen [Zha+13a], [BB04]. Die praktische Relevanz der Idee für das Anwendungsgebiet kann damit belegt werden.

In der Wissenschaft ist ein Teil der mit der Idee verbundenen Forschung auf die Entwicklung neuer domänenabhängiger Sprachen ausgerichtet („domain specific languages“ (DSL)). Im Vergleich zur Auswertung der praktischen Relevanz ist

dies aber ein vergleichbar kleines Forschungsgebiet, weswegen die Signifikanz für wissenschaftliche Fragestellungen nicht vollständig gegeben ist. Das Sinnkriterium wird deswegen als nur partiell erfüllt angesehen.

Varianzkriterium (2) Die Idee der domänenspezifischen Entwicklung eines eingebetteten Systems ist ein gutes Beispiel für die Auslegung des Varianzkriteriums. Bisherige Ideen enthielten oft neuartige Ansätze, Methoden und Techniken. Die aktuelle Idee hingegen lässt sich als Zusatz zu bereits eingeführten Methoden und Techniken sehen und erweitert diese um eine neue Facette. So beginnt die Entwicklung nicht mehr bei der Beschreibung von Algorithmen und Strukturen, die eine gestellte Aufgabe lösen sollen, sondern bei der domänenspezifischen Problemmodellierung selbst. Dies schließt die Transformation der entsprechenden Modelle (MoCs) auf die darunter liegenden Ebenen mit ein (modellbasierte Entwicklung). Methoden und Sichtweisen beider Themengebiete sind nicht als Synonym zu betrachten, auch wenn sie oft gegenseitig bedingt sind. Die modellgetriebene Entwicklung muss nicht mit dem domänenspezifischen Modell einer Aufgabe oder Problemstellung beginnen. Da jedoch deutliche Bezüge zu den Grundlagen von Berechnungsmodellen zu erkennen sind, kann das Varianzkriterium nicht vollständig erfüllt werden.

A.16. Cybersecurity

Die Klassen von Sicherheitsanforderungen für eingebettete Systeme sind ähnlich zu denen traditioneller PCs. Allerdings haben eingebettete Systeme in den wenigsten Fällen die gleichen Möglichkeiten, um diese Sicherheitsanforderungen umzusetzen. Dies bezieht sich auf die Leistungscharakteristika durchschnittlicher eingebetteter Systeme, wie auch auf funktionale und nicht-funktionale Anforderungen, die oft mit dem Bedarf an Sicherheit in Konflikt stehen. Beispiele für solche Parameter sind minimaler Energieverbrauch und damit eingeschränkte Rechenleistung [Koc+04]. Da eingebettete Systeme oft physikalische Prozesse regeln, kann eine Sicherheitslücke unter Umständen genutzt werden, um Menschen oder der Umwelt direkten Schaden zuzufügen.

Die zentrale Idee dabei ist, Sicherheit gegenüber traditionellen Softwareanwendungen neu zu bewerten und gleichzeitig die geringeren Leistungscharakteristika eingebetteter Systeme zu berücksichtigen. Die Idee besitzt also keinen direkten methodischen Aspekt. Sie ist eine Sichtweise.

Die Definition des Begriffes „Cybersecurity“ ist noch nicht einheitlich geprägt, weswegen die begriffliche Basis für die folgende Analyse aus den Teilwörtern abgeleitet wird. Der Begriff „Security“ lässt sich folgendermaßen zusammenfassen:

> „The term 'security' is used in the sense of minimizing the vulnerabilities of assets and resources. An asset is anything of value. A vulnerability is any weakness that could be exploited to violate a system or the information it contains. A threat is a potential violation of security.“ [TC91, S. 32]

Laut Hansen und Nissenbaum besteht bei „*Cyber*security“ die besondere Brisanz in der weitreichenden Verknüpfung der potentiell gefährdeten Systeme [HN09].

Horizontalkriterium (2) Sicherheitsaspekte im Design eingebetteter Systeme sind auf allen Ebenen, von der Bestimmung der Sicherheitsanforderungen über deren Umsetzung bis hin zur Einhaltung im eigentlichen Betrieb, zu finden. Laut Kocher et al. ist bereits die Verwendung von Programmiersprachen wie C oder C++ schwierig, da es zu viele Möglichkeiten gibt, unbeabsichtigt sicherheitskritischen Code zu erzeugen [Koc+04]. Dieser Aspekt fällt somit noch in die Phasen vor dem Systementwurf, also dem „Stakeholder Requirements and Definition Process“. Damit ist ein Aspekt des Horizontalkriteriums, die Verwendung der Idee in mehreren Entwicklungsphasen, gegeben.

Die breite Anwendbarkeit von Cybersecurity kann, wie in der Analyse zum Sinnkriterium zu sehen, nicht zweifelsfrei belegt werden. Der Aspekt ist hauptsächlich bei SCADA-Systemen (engl.: *Supervisory Control and Data Acquisition*) zu beobachten und spielt für kleinere Entwicklungsprojekte oft keine Rolle.

Das Horizontalkriterium kann deswegen nur teilweise erfüllt werden.

Fortbildungskriterium (2) Dornseif et al. beschreiben die Umsetzung eines zweisemestrigen Studiengangs mit dem Themenschwerpunkt „Cybersecurity“, der für Masterstudierende konzipiert wurde und aufgrund des notwendigen Vorwissens nicht im Bachelorstudium behandelt werden kann [Dor+05]. Die an der Universität Siegen angebotene Vorlesungsreihe „Kryptographische Verfahren und Anwendungen“ beinhaltet große Theorieteile der Idee. Sie wird in zwei getrennten Veranstaltungen angeboten, wobei der letzte Kurs, wie auch ein dazugehöriges Praktikum erst für Studierende ab dem fünften Semester empfohlen werden. Bei beiden Veranstaltungskonzepten kann demnach nicht von einer einfachen Vermittlung zu Beginn des Bachelorstudiums ausgegangen werden.

Ein Beispiel *für* die Vermittlung auf Bachelorniveau findet sich in [Luk+14]. Lukowiak et al. schildern eine Lehrveranstaltung, die nach den Prinzipien des aktiven Lernens „Cybersecurity“ in der Software- und Hardwareentwicklung sowie auf der algorithmischen Ebene thematisiert. Die Autoren schildern, dass eine begleitende Veranstaltung notwendig war, um die teilnehmenden Studierenden auf einen vergleichbaren Wissensstand zu bringen. Trotz der guten Rückmeldungen der Teilnehmer gab es auch eine kritische Anmerkung hinsichtlich der Vielzahl an Voraussetzungen für die Veranstaltung.

Gerade in den USA werden vermehrt fachdidaktische Beiträge zum Themenkomplex beschrieben (bspw. [MP14], [TPB14] und [Che+11]). Ebenso hilfreich ist das auf 32 Arbeitsbeschreibungen basierende NICE-Framework, das sogar kompetenzorientiert formulierte Lernziele vorgibt (siehe [ST10]).

Gesellschaftlich wird der Ruf nach einer besseren Ausbildung in diesem Bereich laut. Die Royal Society (UK) veröffentliche 2013 ein Stellungspapier zum Themenkomplex „Cybersecurity“, in dem dessen besondere Wichtigkeit hervorgehoben wurde [Cyb13].

Von Regierungsseite sind Verknüpfungen in den Lehrbetrieb, wie beispielsweise die sogenannten „Common Criteria“, zu beobachten (bspw. [VWW02]).

Auch wenn bisher wenige wissenschaftliche Publikationen zum Themenbereich „Cybersecurity“ in eingebetteten Systemen veröffentlicht wurden, gibt es durch die vorher genannten Beiträge eine Fundierung der Grundlagen, die als Voraussetzung für die Kompetenzvermittlung zu sehen sind. Das Fortbildungskriterium wird somit eingeschränkt positiv beurteilt.

Zeitkriterium (3) Je nach Definition und Anwendungsgebiet von „(Cyber-)Security“ ist das Zeitkriterium gänzlich erfüllt (siehe bspw. [Rus81]). Das trifft insbesondere für den Consumerbereich, weniger jedoch für die sogenannten SCADA-Systeme im industriellen Umfeld zu. „Cybersecurity“ im Bereich eingebetteter Systeme ist im Vergleich zur Sicherheit von Consumer-PCs erst spät in die wissenschaftliche Debatte aufgenommen worden und nicht in einer ausreichenden Breite im Anwendungsgebiet beobachtbar.

Aufgrund der Bedeutung von „Cybersecurity“ für die kommenden Jahre und der Brisanz kürzlich aufgetretener Fälle (bspw. dem Stuxnet-Wurm) wird das Zeitkriterium nach Abwägung erfüllt.

Sinnkriterium (1) Die UBM-Marktstudie unterstreicht die Relevanz sicherheitskritischer Aspekte für die Praxis nicht vollständig. Lediglich zehn Prozent der befragten Teilnehmer entwickeln eingebettete Systeme, die hauptsächlich sicherheitsrelevante Aufgaben umsetzen [Tec13]. Die Bedenken an die Erfüllung notwendiger Sicherheitskriterien bei den Projektverantwortlichen stiegen derselben Quelle nach in den letzten drei Jahren um je 4% und stellen damit nur ein untergeordnetes Ziel dar [Tec13].

Die Bewertung von Forschung über „Cybersecurity“ für eingebettete Systeme ist aufgrund der sensiblen Thematik problematisch. Präzedenzfälle wie der Stuxnet-Virus, der Kraftwerke attackierte, haben diesen Umstand verschärft. Laut Hansen und Nissenbaum ist die Beurteilung über die Häufigkeit von Angriffen auf die „Cybersecurity“ eines Systems deswegen schwierig, weil „Cybersecurity“ ein nationales Anliegen ist und somit in den Bereich militärischer Angelegenheiten fällt [HN09].

Während erfahrene Entwickler in der Regel keinen Bedarf an der Berücksichtigung von Cybersecurity in ihren Projekten sehen, forcieren Großunternehmen und Regierungen die Ausbildung eben dieser Entscheidungskompetenz. Zum aktuellen Zeitpunkt kann nicht eindeutig festgestellt werden, ob die Idee des „Cybersecurity“ im theoretischen oder im praktischen Kontext der Entwicklung eingebetteter Systeme eine signifikante Relevanz besitzt. Aus diesem Grund kann das Sinnkriterium für die vorliegende Idee nicht erfüllt werden.

Varianzkriterium (2) Die vorliegende Idee besitzt wenige konkrete Bezüge zu anderen Themenschwerpunkten. Am deutlichsten zeigt sich die Verbindung zu „Reliable Design“ und Ressourcen Management. Kein Aspekt dieser beiden Bereiche deckt die Idee genügend ab. „Cybersecurity“ stellt deswegen zumindest eine neuartige Facette dar und erfüllt dadurch das Varianzkriterium teilweise.

A.17. Methoden der formalen Verifikation und Validierung

Verifikation und Validierung werden fälschlicherweise häufig synonym verwendet. Edwards et al. erläutern in folgender Textstelle neben einer umgangssprachlichen Deutung des Begriffes *valide* auch den Zusammenhang zwischen den Begriffen *Simulation* und *Verifikation*:

> „Validation loosely refers to the process of determining that a design is correct. Simulation remains the main tool to validate a model, but the importance of formal verification is growing, especially for safety-critical embedded systems.“ [Edw+01, S. 30]

Die Idee der formalen Verifikation und Validierung ist mehrschichtig. Zum einen beinhaltet es das Formalisieren von Anforderungen und somit die Möglichkeit der Überprüfung selbiger und zum anderen das Differenzieren von Validierung und Verifikation im vorher geschilderten Sinne. Somit ist ein methodischer Aspekt und eine Sichtweise in der Idee enthalten.

Horizontalkriterium (3) Verifikation und Validierung sind im Entwicklungsprozess auf verschiedenen Ebenen beobachtbar, beispielsweise bei der Konzeption der Hard- und Software, der Anforderungsanalyse zur Erstellung einer Spezifikation oder bei der Implementierung, die sogar aus einer formalen Beschreibung des Systems abgeleitet werden kann [Edw+01]. Im Entwicklungsprozess, wie er durch den ISO-Standard ISO-15288 vorgegeben wird, sind natürlich die namensgebenden Prozessebenen „Validation Process“ und „Verification Process“ zentrales Anwendungsgebiet der Idee. Zur Durchführung der Verifikation bieten sich Konzepte wie das „Model-Checking“ oder das Theorembeweisen (engl. „Theorem-Proving“) an [CW96]. Formale Methoden werden häufig durch MoCs dargestellt, die für den Beweis oder die Widerlegung von Systemeigenschaften genutzt werden [SLS00]. Die Synthetisierung einer formalen Beschreibung ist in diesem Zusammenhang auch als „Correct-by-Construction“ bekannt [Edw+01]. Clarke betont, dass alle Entwickler, unabhängig von Software oder Hardware, mit den Grundprinzipien formaler Methoden in Berührung kommen [CW96]. Damit ist nicht nur die Relevanz der Idee für verschiedene Entwicklungsphasen, sondern auch für eine breite Anwendbarkeit der Idee belegt. Das Horizontalkriterium kann deswegen vollständig erfüllt werden.

Fortbildungskriterium (3) Das notwendige mathematische Vorwissen zur Kompetenzaneignung von formalen Methoden sollte in den ersten Veranstaltungen des Grundstudiums bzw. Bachelorstudiums behandelt worden sein (bspw. Mengenlehre und Beweisführung aus der Einführung in die Theoretische Informatik).

Finney führte eine Studie durch, in der 62 Bachelor- und Masterstudierende Teile einer mathematischen Systemspezifikation interpretieren sollten [Fin96]. Die Ergebnisse dieser Studie belegen, dass Vorwissen aus den Bereichen Mengenlehre und Logik lediglich ein Fundament an Kompetenzen bedeutet, deren Anwendung als weitere Qualifikation vielen Teilnehmern schwer gefallen ist. An der Universität Siegen sind diese Voraussetzungen nach dem Abschluss des zweiten Semesters gegeben, da dort die Veranstaltung „Grundlagen der Theoretischen Informatik“ für Bachelorstudierende angesetzt ist. Beispiele für weitere Durchführungen auf Bachelorniveau finden sich in [LG02] und [Gib09].

Wie gezeigt, lassen sich auf Bachelorniveau Bezüge zu formalen Methoden lehren [Win00]. Neben den bereits genannten Einsatzgebieten der Idee ist sie auch eine Basis für Bereiche wie Datenstrukturen, Compiler oder Programmiersprachen.

Zeitkriterium (3) Die Idee formaler Verifikation und Validierung ist bereits vor 20 Jahren im Bereich der eingebetteten Systeme deutlich wahrnehmbar gewesen [Rus93], [Ost92], [BS93]. Weitere Anwendungsfelder, in denen die Technik ähnlich genutzt wird, finden sich zum Beispiel bei der Verifikation paralleler Prozesse und Betriebssysteme [Kel76].

Sinnkriterium (3) In der modellgetriebenen Entwicklung ist die Verwendung formaler Methoden zur Validierung gut umsetzbar, allerdings praktisch gesehen nur für wenige Anwendungsbereiche gefordert [Vaa06]. Kopetz beschreibt, dass bis zu 50% der Entwicklungskosten von Echtzeitsystemen nur dafür verwendet werden, zu prüfen, ob das System prinzipiell den Anforderungen genügt [Kop11]. Obwohl die Signifikanz des Themenbereiches somit ersichtlich ist, berichtet Miller, dass Methoden zur formalen Verifikation nur langsam in der Industrie angenommen werden [MC06], obwohl Universitäten vermehrt Bildungsangebote auf Masterebene schaffen [CW96].

Die Kompetenzbeschreibung des EKSM, die sich der Idee zuordnen lässt (C2.3), wurde von den Experten als B-Kompetenz, also der zweithöchsten möglichen Kategorie, bewertet. Zusammenfassend kann das Sinnkriterium mit Einschränkungen erfüllt werden, auch wenn die praktische Umsetzung formaler Methoden eher im wissenschaftlichen Bereich anzunehmen ist.

Varianzkriterium (2) Die Idee, die Validierungs- und Verifikationsphasen formal zu gestalten, um mathematisch belastbare Tests und Analysen zu erhalten, steht thematisch gesehen in Zusammenhang mit „Reliable Design“ und mit „Models of Computation“. Allerdings ist die zentrale Idee der Formalisierung unterschiedlich genug, um eine eigene Berechtigung als fundamentaler Lehr-/Lerninhalt zu besitzen.

Die Verifikation eines Systems geschieht immer in Bezug zu erarbeiteten Anforderungen. Deswegen ist die größte Überschneidung die Idee der Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen. Das Varianzkriterium wird eingeschränkt erfüllt, allerdings die Verifikation in der Analyse der verwandten Ideen nicht mehr berücksichtigt, da sie in besonderem Umfang für die Rechtfertigung dieser Idee genutzt wurde.

A.18. Modellbasierte Entwicklung

Der modellbasierte Entwicklungsansatz beinhaltet die zentralen Konzepte *Transformation* und *Verfeinerung*. Erstere findet sich in der Umwandlung einer Modellbeschreibung in eine andere. Idealerweise wird nur ein Dokument (die Spezifikation der Systembeschreibung) erstellt, das sich dann auf eine andere Detailebene, bspw. die des Systemdesigns, transformieren lässt. Die Verfeinerung ist im Hinzufügen und Entfernen von Modelldetails, je nach Abstraktionsstufe und Modellintention, sichtbar [Edw+01]. Durch die Änderung des Ausgangsdokuments kann in modellbasierten Ansätzen ein Großteil der daraus entstehenden Entwicklungsartefakte (Quellcode, VHDL-Beschreibungen, UML-Diagramme, etc.) automatisch aktualisiert werden. Die Fehleranfälligkeit beim Wechsel der Abstraktionsschichten und Entwicklungssphasen kann so, genau wie der Arbeitsaufwand, reduziert werden.

Modellbasierte Entwicklung ist nicht klar definiert und umfasst, je nach Ansicht, lediglich die Transformation eines Modells in eine andere Repräsentationsform (bspw. ein anderes Modell oder ein Entwicklungsartefakt) oder einen kompletten Entwicklungsprozess. Ausgangspunkt ist in jedem Fall eine ausführbare Spezifikation [SPF07].

Horizontalkriterium (2) Wie Schätz et al. ausführen, kann die modellgetriebene Entwicklung bei der Anforderungserhebung und -verfolgung und bei der Dokumentation des Entwicklungsprozesses eingesetzt werden [Sch+02]. Die Ebene der Implementierung kann für das Anwendungsgebiet eingebetteter Systeme mehrfach abgedeckt werden. Mit Hilfe der klassischen Transformation einer Modellbeschreibung in Quellcode oder direkt in eine Hardwarebeschreibung wie VHDL. Die Transformation einer Hardware-Beschreibung in die entsprechende Netzliste nennt sich Synthese und ist eine der State-of-the-Art-Techniken in der aktuellen Hardwareentwicklung (bspw. bei FPGAs). Modellbasierte Entwicklung kann als eine Fortführung der Abstraktionsschicht von Assembler zu höheren Programmiersprachen und in letzter Instanz schließlich zu Modellen gesehen werden [Sch+02].

Die breite Anwendung von modellbasierten Entwicklungsprozessen ist, unter anderem durch die in der Analyse zum Sinnkriterium geschilderten Positionen, nicht gegeben. Das Horizontalkriterium kann deswegen nur eingeschränkt positiv bewertet werden.

Fortbildungskriterium (1) Obwohl modellbasierte Entwicklung nicht nur eine graphische, domänenspezifische Systembeschreibung darstellt [Sch+02], kann diese Basis hilfreich für die Vermittlung der Idee auf Bachelorniveau sein, da die Studierenden fachspezifische Methoden und Werkzeuge auf eine ebenso fachspezifische Problemstellung anwenden können. Bei einer domänenunabhängigen Sprache müssen sich die Studierenden ihre *Werkzeuge* (Klassen, Methoden und Funktionen) für den Anwendungsfall erst noch erstellen. Da den Studierenden die Grundlagen von Berechnungsmodellen bereits durch die Grundlagenveranstaltung bekannt sind, können sie beispielsweise auf Vorwissen zu endlichen Automaten oder Petri-Netzen aufbauen.

An der Vanderbilt-Universität haben Sztipanovits et al. ein Praktikum auf Bachelorniveau durchgeführt, das die Grundlagen der modellbasierten Entwicklung für eingebettete Systeme vermittelt [Szt+05]. Im von Caspi et al. vorgestellten Curriculum finden sich ebenso Veranstaltungen zur Idee der modellbasierten Entwicklung [Cas+05]. Allerdings werden diese Lerneinheiten nicht auf Bachelor-, sondern auf Masterniveau vermittelt. Aufgrund der jungen Historie der Idee (siehe Zeitkriterium) finden sich nur wenige fachdidaktisch begründete Veranstaltungsbeschreibungen zum genannten Themenbereich. Neben diesen unterschiedlichen Niveaustufungen ist ebenso unklar, für welche Bereiche die modellbasierte Entwicklung eine Grundlage darstellt. Auch wenn einige Erhebungen, meist von Herstellern entsprechender Werkzeuge, die Überlegenheit von modellbasierten Entwicklungsverfahren aufzeigen (bspw. [ST08]), existieren ebenso viele Gegenargumente [Lee08], [JCL11]. Eine Rechtfertigung für das Fortbildungskriterium ist deswegen nicht möglich.

Zeitkriterium (1) Zwei Quellen belegen erste Gedanken zum angesprochenen Themenbereich, auch wenn praktische Umsetzungen erst später folgten [Wym93], [Suh98]. Dickerson und Mavris sehen die Ursprünge noch wesentlich früher im mathematischen Prinzip der First-Order-Model-Theory begründet [DM13], aber auch dort ist der Begriff diffus. Sie argumentieren zudem häufig nur für die informatische Modellbildung, aber so gut wie nie über deren Verfeinerung oder Transformation als Vorgehensweise. Eine sichere Abschätzung der Ideenrelevanz für die Vergangenheit und Zukunft kann daher nicht erfolgen. Das Zeitkriterium kann deswegen nicht positiv evaluiert werden.

Sinnkriterium (1) Die modellbasierte Entwicklung hat aus wissenschaftlicher Sicht einige theoretische Vorteile gegenüber traditionellen Entwicklungsvorgehen wie bspw. das automatische Verfolgen von Anforderungen, die korrekte Transformation zwischen verschiedenen System- und Komponentenbeschreibungen und die lückenlose Dokumentation des Entwicklungsprozesses [Sch+02]. Idealerweise umspannt eine Modelltransformation den gesamten Entwicklungsprozess. Die Entwicklungseinheiten *Anforderungsdokument*, *Systemspezifikation* und *Implementierung* werden zu einem „Entwicklungsartefakt“ zusammengefasst.

Modellgetriebene Entwicklung und die damit verbundene Idee der Modelltransformation werden von einigen Wissenschaftlern als essentiell angesehen, um bisher

getrennte Entwicklungsschritte wie das mathematische Beschreiben der Systemfunktionalität, die Simulation heterogener Komponenten oder die Softwaresynthese miteinander zu verbinden [JCL11]. Sifakis und Henzinger zeigen auf, dass das Hauptanliegen eines modellbasierten Entwicklungsvorgehens in der Trennung zwischen Modellierung und Implementierung liegt [HS07]. Dadurch muss sich der Designer vorerst nicht auf technikabhängige Ausführungs- und Kommunikationsmethoden festlegen [HS07]. Wie bereits im Fortbildungskriterium erläutert, ist die Idee Gegenstand einer kontroversen Diskussion, deren Ausgang momentan nicht vorauszusagen ist. Aus diesem Grund kann die Signifikanz der Idee und damit das Sinnkriterium für die praktische Anwendung nicht zweifelsfrei belegt werden.

Varianzkriterium (1) Die Idee modellbasierter Entwicklung hat einen starken Bezug zu den Ideen der Domänenspezifität, der Berechnungsmodelle als Abstraktion gegebener Aufgaben- und Problembeschreibungen und Syntheseprozessen im Allgemeinen. Ersteres lässt sich noch von dieser Idee unterscheiden, da hier die Transformation von Modellrepräsentation und deren iterative Verfeinerung die Kernidee darstellen. „Models of Computation“ enthalten die prinzipielle Idee, einen Sachverhalt anhand von verschiedenen Modelltypen darzustellen, die sich, je nach Zielstellung, unterscheiden. Kritischer muss der Zusammenhang zu Synthesetechniken gesehen werden, denn in beiden Ansätzen wird, ausgehend von einer hochsprachlichen, abstrakten Beschreibung, eine konkretere Systembeschreibung erzeugt. Aufgrund dieser starken Verknüpfung kann das Varianzkriterium nicht erfüllt werden.

A.19. Probabilistische Softwareentwicklung und Werkzeuge

Die folgende Analyse betrachtet Software- und Hardwareaspekte der Idee, da diese sich im Wesentlichen durch den gleichen Ansatz auszeichnen, also die gleiche Idee beinhalten. Softwaresysteme können verschiedene Anforderungen zur Laufzeit verletzen und damit Fehler beinhalten, die sich im Vorfeld nicht eindeutig bestimmen lassen [LPS01]. In der Hardwarefertigung stellt sich das Problem, dass Komponenten in aktuellen, verstärkt aber wahrscheinlich in zukünftigen Fertigungsgrößen nicht mehr so präzise hergestellt werden können, dass für alle gefertigten Komponenten das exakt gleiche Verhalten garantiert werden kann [Pal+09].

Die Systemkomponenten werden in diesem Ansatz durch Fehlerwahrscheinlichkeiten beschrieben, die, je nach Design, zu unterschiedlichen Bewertungen hinsichtlich der Zuverlässigkeit des Gesamtsystems gelangen. Der Entwickler kann über die möglichen Varianzen der Komponenten, und somit des gesamten Systems, eine Analyse ausführen, deren Ziel die Bestimmung des Entwurfes ist, der am wenigsten unvorhersagbares Verhalten beinhaltet.

Palem et al. definieren den Begriff folgendermaßen:

> „By contrast, the central theme of our work which we refer to as *probabilistic and approximate Design* ist to *design computing systems using circuit components which are susceptible to perturbations*“ [S. 2 Pal+09, Hervorhebung im Original]

Horizontalkriterium (1) Wie aus der Bezeichnung der Idee ersichtlich ist, findet sich deren Anwendung in der Entwicklungsetappe „Systementwurf“. Das Anwendungsgebiet der Idee ist auf Systeme begrenzt, die entweder äußerst zuverlässig sein müssen (bspw. Projekte aus dem Bereich Bauingenieurwesen oder der Luft- und Raumfahrttechnik) oder aber unzuverlässige Komponenten verwenden. Der Sinn, diese Art von Komponenten den „korrekt“ funktionierenden vorzuziehen, ist mehrfach begründbar. Eine Motivation für das absichtliche Nutzen solcher Bauteile kann im günstigeren Verhältnis von Energie- und Leistungsparametern zu Fertigungskosten bestehen [Pal+09]. Ähnlich zur „Design-Space-Exploration“ kann das Verfahren genutzt werden, um den Bedarf an Prototyping zu reduzieren oder das Design hinsichtlich Material- oder Platzverbrauch zu analysieren und optimieren [GBM05]. Eine breite Anwendbarkeit und die Relevanz für verschiedene Entwicklungsphasen sind für die vorliegende Idee nicht gegeben. Das Horizontalkriterium wird deswegen nicht erfüllt.

Fortbildungskriterium (1) Probabilistisches Software-Design setzt schwergewichtige Themenkomplexe wie Modellierung von Zufälligkeiten, Fehlerquellen, verifizierende Methoden und grundsätzliche Aspekte von Systemzuverlässigkeit voraus. Bei den wenigen Beispielen der Lehre der Idee auf Hochschulniveau sind die entsprechenden Kurse ausschließlich auf Master- und nicht auf Bachelorniveau ausgerichtet [Had13]. Fachdidaktisch begründete, wissenschaftliche Erkenntnisse sind dem Autor nicht bekannt. Empfehlungen für die Lehre von „Zufälligkeit“ an sich finden sich in [LL09].

Das Fortbildungskriterium ist nicht erfüllt, da das notwendige Vorwissen zu umfangreich für Bachelorstudierende ist und die wenigen fachdidaktischen Quellen die Ausrichtung auf den Masterstudiengang empfehlen.

Zeitkriterium (3) Die Autoren Goh, Booker und McMahon erwähnen, dass die Idee seit mehr als 40 Jahren diskutiert wird [GBM05]. Im Zusammenhang mit Verlässlichkeit von Systemen behandelt Voges die Idee bereits 1988 [Vog88]. Auch Lyu et al. gehen in ihrem Buch zur „Software-Fault-Tolerance“ auf die Grundlagen ein [Lyu+96]. Durch die Verkleinerung von Fertigungsgrößen erhöht sich deren Fehleranfälligkeit gegenüber Störeinflüssen und anderen Fehlerquellen, weswegen zu erwarten ist, dass auch im Zeitalter der Nanotechnologien die System- und Komponentenzuverlässigkeit von Bedeutung sein werden (siehe bspw. [Har+01]).

Sinnkriterium (1) Probabilistische Architekturen sind *eine* Möglichkeit, die Fehlertoleranz eines Systems zu erhöhen. Praktische Anwendungen gibt es in spezialisierten Anwendungsgebieten wie der Luftfahrt- und Raumfahrttechnik und

im Schienenverkehr; also den Domänen, in denen Systemzuverlässigkeit eine entscheidende Rolle spielt [LPS01]. Die Idee ist weder wissenschaftlich noch praktisch unabdingbar und stellt keine einzigartige Lösung zur Steigerung der Systemzuverlässigkeit zur Verfügung. Goh, Booker und McMahon weisen darauf hin, dass die Kernprobleme bei der Anwendung probabilistischer Entwicklungsmethoden hauptsächlich in der Zuverlässigkeit der Methoden selbst und in deren Auswahl, je nach Anwendungsfall, liegen [GBM05]. Ohne belastbare Daten über die Zuverlässigkeit von Komponenten kann nicht auf die Zuverlässigkeit des Gesamtsystems geschlossen werden. Durch den eingeschränkten Nutzen der Idee aus praktischer wie industrieller Sicht kann das Kriterium nicht erfüllt werden.

Varianzkriterium (2) Die Idee ist eng verknüpft mit den Themenbereichen *Verlässlichkeit*, *Fehlertoleranz* und *Verifikation und Validierung*. Ein besonders starker Bezug zeigt sich hinsichtlich der „Design-Space-Exploration“, die, genau wie die vorliegende Idee, die Abschätzung von Designalternativen als Kernelement hat. Unterschiedlich ist hier lediglich die Art der Systemspezifikation mittels Wahrscheinlichkeiten und die Zielsetzung, ein verlässliches System aus potentiell nicht verlässlichen Komponenten zu konstruieren [LPS01]. „Probabilistic-Design“ wie auch der verwandte Begriff der „Design-Diversity“ sind mögliche Lösungstechniken und stehen somit in einer „Bestandteil von“-Beziehung zu den oben genannten Themenbereichen. Die Idee stellt somit maximal eine Erweiterung der genannten Ideen dar und kann das Varianzkriterium somit nicht mehr vollumfänglich erfüllen.

A.20. System-On-Chip Design (MPSoC und NoCs)

Multi-Processor-System-on-Chips und Network-On-Chips vereinen den Ansatz, ein System auf einem Chip zu integrieren. Sie stellen demnach unterschiedliche *System-on-Chip (SoC)* Paradigmen dar [BD02]. Im Vergleich zum klassischen Prozessor erweitern sie die Recheneinheit um Schnittstellen, Busstrukturen und Speicherbausteine, also Komponenten, die in traditionellen Entwürfen außerhalb der Berechnungseinheit liegen. Das Vorgehen, ein System auf einem Chip zu integrieren, ist im Wesentlichen durch die Vorteile hinsichtlich Baugröße und Preisgestaltung bei der Massenproduktion gerechtfertigt [LS03]. Verschiedene Teile des SoCs (IP-Cores) werden üblicherweise von externen Dienstleistern bereitgestellt und mit den selbstentwickelten Komponenten verbunden.

Bailey, Martin und Anderson schlagen die folgende Definition des Begriffes vor:

> „A single piece of silicon containing multiple VCs (Anmerkung des Autors: Virtual Components) to perform a certain defined function.“ [BMA05, S. 86]

Wie aus der zitierten Textstelle ersichtlich, wird zum einen die Integration verschiedener Komponenten betont und zum anderen die Zusammenführung auf einem Chip. Damit enthält die Technik die Idee der *Miniaturisierung*, die eines der grundlegenden Prinzipien der Mikrosystemtechnik darstellt [Hsu08].

Horizontalkriterium (2) Thematisch liegt die Idee näher an der Hardwareentwicklung als an der Softwareentwicklung, weswegen die Vorteile einer entsprechenden Umsetzung meist nur mit Hinblick auf Hardwareeinheiten geschildert werden (bspw. [LS03]). Dies ist verständlich, da *Miniaturisierung* auf Softwareebene so gut wie nicht beobachtbar ist. Da die Miniaturisierung und die Integration von Systemkomponenten einen weitreichenden Einfluss auf aktuelle eingebettete Systeme besitzt, kann ein Aspekt des Horizontalkriteriums erfüllt werden. In den Entwicklungsphasen ist die Idee der Miniaturisierung jedoch kaum beobachtbar, weswegen der zweite Aspekt des Kriteriums nicht erfüllt werden kann. Damit ist das Horizontalkriterium nur teilweise erfüllt.

Fortbildungskriterium (1) Das Fortbildungskriterium lehnt die Idee ab, da Bachelorstudierende nicht in nötigem Umfang Vorwissen aus Rechnerarchitekturen, „Component-based Design“, Hardware/Software Co-Design und objektorientierter Entwicklung verfügen, um SoC-Design *anwenden* zu können. Eine auf Bachelorniveau ausgerichtete Lehrveranstaltung würde Kompromisse hinsichtlich der Tiefe und Breite des behandelten Themengebietes und der zu erarbeitenden Lösung akzeptieren müssen. Ähnlich kann auch für Masterstudierende argumentiert werden, wobei zu beachten ist, dass der QDH bei diesem Studienniveau explizit die Möglichkeit erwähnt, wissenschaftlich aktuelle Themengebiete mit einem gegenüber Bachelorniveau gestiegenen Komplexitätsgrad zu lehren (siehe Sektion 4.2.2).

Gleichzeitig ist die Idee nur teilweise Grundlage für weitere Lerninhalte. Insbesondere der Aspekt des „Component-based Designs“ und der damit verbundenen „Divide-and-Conquer“-Ansätze können in anderen Anwendungsgebieten hilfreich sein.

Deman sieht die Ausbildung von SoC-Architekten problematisch, da aktuelle Curricula auf einzelne Themengebiete und nicht auf deren vertikale Verbindungen ausgerichtet sind, die sich für das Verständnis des Konzeptes empfehlen [De 99]. Das Fortbildungskriterium kann nicht als erfüllt angesehen werden, da die Vermittlung von Kompetenzen (in Hinblick auf den QDH) durch die Analyse nicht gerechtfertigt wird.

Zeitkriterium (2) Die Diskussion über die Integration mehrere Systemkomponenten auf einem Chip ist in der Historie der Disziplin wahrnehmbar (bspw. [MYT93]). Mit einer leichten Verlagerung des Methodenumfangs kann auch [PBC95] als Quelle betrachtet werden. Der Autor bezieht sich ausschließlich auf ASICs und empfiehlt den diskutierten Ansatz lediglich für geringe Mengen an analogen Funktionalitäten, die auf dem Chip integriert werden sollen. Der Umfang wissenschaftlicher Arbeiten vor 1995 zu diesem Thema ist jedoch dürftig.

Wenn, wie in der Einleitung der Analyse erläutert, die Idee der Miniaturisierung als Kern des Ansatzes ausgemacht wird, ist das Zeitkriterium in jedem Fall erfüllt, da die Verkleinerung der Platinen- und Chipgrößen seit mindestens 50 Jahren beobachtbar ist [Hsu08]. Die Bewertung des Zeitkriteriums sollte in diesem Fall keinen Grund zur Ablehnung der Idee geben, wird aber auch nicht vollumfänglich erfüllt.

Sinnkriterium (3) Die technische Umsetzung der Idee ist neben der Miniaturisierung von Hardwarekomponenten besonders aufgrund der gesteigerten Wiederverwendbarkeit gegenüber traditionellen Methoden attraktiv. Im Forschungsbereich von „System-on-Chips“ ergeben sich zusätzliche Vorteile durch die Optimierung der Leistungsparameter hinsichtlich verkürzter Verbindungswege oder gesunkener Leistungsaufnahme (vgl. [Boh09]). Die Kehrseite dieser Entwicklung sind gestiegene Anforderungen an die, die Komponenten verbindende, Netzwerkinfrastruktur [BM06]. An dieser Stelle beginnt der auf der Problemstellung aufbauende Forschungsbereich der „Networks-on-Chip“.

Während traditionelle Ziele im Systementwurf früher die Leistungssteigerung und die schnelle Verfügbarkeit am Markt waren, ist inzwischen der Markt mehr an kleinen, stromsparenden Lösungen interessiert [LS03]. In diesem Zusammenhang sind SoCs für mobile Systeme wie Smartphones besonders relevant. Die Auswirkungen der Idee und entsprechender technischer Umsetzung auf industrielle Gegebenheiten sind deutlich wahrnehmbar [Sal+06].

Die Relevanz im theoretischen und praktischen Kontext ist also gegeben und das Sinnkriterium somit erfüllt.

Varianzkriterium (2) Die Idee hat auf konzeptueller Ebene viele Ähnlichkeiten zum „Component-based Design“. Beispiele hierfür sind die Architektur von Komponenten (*GALS*-Prinzip [BM06]) und der grundlegende Vorteil der höheren Wiederverwendbarkeit (siehe auch [Sal+06]), die sich in beiden Ansätzen wiederfindet.

Gleichzeitig ist der Einfluss von SoCs auf die Praxis deutlich umfangreicher und stark von der Idee der *Miniaturisierung* geprägt. Weitere Entwicklungsziele, wie im Sinnkriterium diskutiert, unterscheiden sich deutlich zur Kriteriumsanalyse für die komponentenbasierte Entwicklung. Die gute Ergänzung eines komponentenorientierten Entwicklungsvorgehens bei der Umsetzung von System-on-Chips lässt sich auch an der wissenschaftlichen Zusammenführung beider Konzepte ablesen. Mehrere wissenschaftliche Beiträge schlagen ein solches Entwicklungsvorgehen vor (siehe [Ces+02], [BD04], [ZS10]).

Da die Idee nicht durchweg überschneidungsfrei mit der komponentenbasierten Entwicklung ist, diese jedoch um eine entscheidende Facette erweitert, wird das Varianzkriterium teilweise erfüllt.

A.21. Rekonfigurierbare Architekturen und reprogrammierbare Designs

Tessier und Burleson haben 2001 eine kompakte Definition des Begriffes „Rekonfigurierbarkeit“ gegeben, die im Folgenden als Grundlage der Analyse angesehen wird:

> „In the context of re-configurable computing this term indicates that the logic functionality and interconnect of a computing system or device can be customized to suit a specific application through post-fabrication, user-defined programming.“ [TB01, S. 1]

Für Hardwarekomponenten ist die technische Umsetzung größtenteils in „Field Programmable Gate Arrays“ (FPGAs) zu finden [TB01]. Hauck, 1998, weist jedoch darauf hin, dass FPGAs nicht synonym zur *Idee* einer rekonfigurierbaren Architektur verwendet werden sollten, sondern eine mögliche technische Umsetzung des Konzeptes darstellen [Hau98].

Unterschieden werden Rekonfiguration vor und während der Laufzeit des Systems. Rekonfiguration stellt damit die Basis des Konzeptes „Updates“ dar. Statt dem Begriff „Rekonfiguration“ wird in einigen Publikationen auch von „Reprogrammierung“ gesprochen [But95], [Ada01], [Hau98]. Aus diesem Grund wird der Term „reprogrammable Design“ synonym verwendet. Obwohl die Idee einen starken Bezug zur Hardwareentwicklung besitzt, soll zumindest erwähnt werden, dass die Idee auf Softwareebene ebenso bekannt und genutzt ist. Die technische Umsetzung zwischen Hardware und Software ist dabei grundverschieden. In Softwaresystemen lassen sich Module mit neuer Funktionalität an definierte Schnittstellen, sogenannte *„Application-specific Interfaces“* (APIs) anschließen. Eine weitere Möglichkeit, Updates für Softwaresysteme bereitzustellen, liegt im einfachen Austausch von Skripten oder Binärkompilaten. Ist der aktuelle Stand der Applikation bekannt, können auch differentielle Updates eingespielt werden, die nur einen Teil des Softwaresystems durch bereits für die Rechnerarchitektur kompilierte Anweisungen austauschen. Für traditionell entwickelte Hardwarekomponenten gibt es keine Möglichkeit, nachträglich Änderungen einzupflegen, die für eine Massenproduktion rentabel wären.

Auch wenn Rekonfiguration auf Softwareebene ebenfalls beobachtbar ist, widmet sich die folgende Analyse hauptsächlich der Hardwareebene.

Horizontalkriterium (3) Rekonfigurierbare Designs beeinflussen in vielfacher Weise die Entwicklung eingebetteter Systeme. Die häufig genutzte technische Umsetzung der Field Programmable Gate Arrays (FPGAs) ist als Prototypeninstrument sehr verbreitet (siehe [HD10]). Neben der eigentlichen Implementierungsphase, in der die Umsetzung eines Systemdesigns mit Beschreibungssprachen wie VHDL synthetisiert wird, ist deswegen auch die Phase der Validierung von der Idee betroffen. Die Möglichkeit, Systemcharakteristika wie Leistungsaufnahme oder Performanz durch einen Prototypen abzuschätzen, zeigt, wie die Idee rekonfigurierbarer Architekturen auch den Entwurf des Systems auf der Ebene des „Architectural Design Processes“ beeinflussen kann. Erst wenn der erstellte Prototyp zufriedenstellend umgesetzt ist, wird mit der Portierung auf die eigentliche, nicht-rekonfigurierbare Hardware begonnen. In manchen Fällen fällt der letzte Schritt insofern weg, dass ein FPGA bereits als Zielplattform dient. Bei solchen Szenarien wird die Idee also nicht zum Prototyping, sondern für die direkte Produktentwicklung genutzt.

Die besondere Relevanz von rekonfigurierbaren Architekturen zum Erstellen eines zeitigen Systemprototypens lässt sich prinzipiell auf fast alle Entwicklungsprojekte

eingebetteter Systeme übertragen. Die einzige Einschränkung in dieser Hinsicht sind die auf der rekonfigurierbaren Architektur verfügbaren konfigurierbaren Logikzellen. Bei sehr großen Projekten sind diese eine gegenüber dem ASIC-Entwurf limitierender Faktor. Der Großteil der Projekte die im Anwendungsgebiet der eingebetteten Systeme liegen, wird davon jedoch nicht betroffen sein, sodass die Idee auch den zweiten Aspekt des Horizontalkriteriums erfüllt.

Fortbildungskriterium (2) Rekonfigurierbare Systeme können in der Lehre eine Doppelrolle einnehmen. Wie bereits erläutert, sind sie zum einen eine relevante, ebenenübergreifende Technik für das Anwendungsgebiet und zum anderen lassen sich durch rekonfigurierbare Plattformen die Grundlagen von Digitalsystemen anschaulich darstellen. Die Fertigung eines integrierten Schaltkreises (ASIC) ist üblicherweise mit der Erstellung von Fertigungsmasken und dem anschließenden Beleuchten des Wafers mit ultraviolettem Licht in einem Reinraum verbunden. Die Kosten für die benötigten Werkzeuge und qualifiziertes Personal sind so hoch, dass sich nur wenige Lehrstühle die Fertigung leisten können. Zusätzlich ist der Zeitaufwand für die Fertigung nur schwer in eine reguläre Vorlesung oder ein Praktikum zu integrieren. Auf dieser Ebene durchführbar sind hingegen diskrete Aufbauten aus fertigen Komponenten oder die Verwendung von *printed circuit boards* (PCBs). Zu Übungszwecken ist die Nutzung von Hardwarebeschreibungssprachen jedoch besonders praktikabel. Demnach profitieren auch Lehrende, die keinen Fokus auf rekonfigurierbare Hardware setzen, aber sich in ihren Lehrveranstaltungen mit dem Entwurf oder mit der Implementierung von digitalen Systemen beschäftigen, von den Möglichkeiten der FPGA-Entwicklung. Eine weitere Synergie ergibt sich durch die Lehre von Hardwarebeschreibungssprachen (HDLs). HDLs sind Programmiersprachen aus der Softwaretechnik funktional ähnlich [KB09] und werden nicht nur für die Konfiguration von rekonfigurierbaren Architekturen, sondern auch für die Beschreibung des Hardwareaufbaus und des Verhaltens von ASICs verwendet. Ein ASIC ist ein integrierter Schaltkreis, der für eine bestimmte Aufgabe optimiert wurde, beispielsweise für eine Satellitensteuerung [Smi08]. Der große Unterschied zu rekonfigurierbaren Systemen, der für die Betrachtung in dieser Arbeit relevant ist, ist, dass ASICs nach der Fertigung nicht mehr strukturell verändert werden können.

Wichtige Voraussetzungen für die Vermittlung entsprechender Kompetenzen sind Grundlagen der Rechnerarchitektur und Grundlagen digitaler Schaltungen. Erste Erfahrungen in Programmiersprachen wie C oder C++ sind für die Umsetzung von HDLs empfehlenswert, da einige Konzepte wie Namensräume übernommen werden können. Kenntnisse domänenspezifischer Sprachen sind ebenso hilfreich, aufgrund der meist anwendungsfallspezifischen Ausrichtung aber nur begrenzt auf andere Sprachen übertragbar. Alternativ kann die Entwicklung anstatt mit HDLs teilweise auch auf MoCs verlagert werden, die sich, beispielsweise in Form von Automaten oder Zustandsdiagrammen, ebenso zur Beschreibung von Komponentenverhalten eignen und teilweise sogar synthetisierbar sind (auch in VHDL). Allerdings ist darauf hinzuweisen, dass VHDL verschiedene Berechnungsmodelle unterstützt, die sich nicht mit endlichen Automaten abbilden lassen. Hierzu zählt zum Beispiel die Spezifikation der Schnittstellen, also im weitesten Sinne Eingabe- und

Ausgabesignale.

Aufgrund seiner Relevanz in der Hochschullehre ist das Themengebiet fachdidaktisch vergleichsweise gut erforscht (bspw. [SS05], [DeH+04]). Viele Universitäten und Fachhochschulen in Deutschland lehren die mit dem Themenkomplex verbundenen Inhalte allerdings auf Masterniveau. Die Universitäten Bielefeld, Kassel und Karlsruhe bieten je eine Lehrveranstaltung mit Fokus auf Masterstudierende an [Zip13], [Por13], [HB13].

Ebenso gibt es Universitäten wie Erlangen und Frankfurt, die das Themengebiet nicht als eigenständiges Modul, sondern im Rahmen der Vorlesung über Rechnerarchitekturen lehren. Beide Veranstaltungen lassen sich dem Modulhandbuch für Bachelorstudierende zuordnen [Wal14], [Fey13]. Das im Rahmen des KOMINA-Projektes entworfene „Entwurfs- und Anwendungspraktikum für eingebettete Mikrosysteme“ (EAP) hat gezeigt, dass grundsätzliche Entwicklungen mit rekonfigurierbaren Plattformen auch auf Bachelorniveau möglich sind [Jas+12]. Fortgeschrittene Themengebiete wie die dynamische Rekonfiguration zur Laufzeit sind der Komplexität wegen jedoch nicht genutzt worden.

Da die aufgezeigten Quellen keine eindeutige Einordnung auf Bachelor- oder Masterniveau zulassen, ist eine Kompetenzniveaustufung in Anlehnung an den Qualifikationsrahmen Deutscher Hochschulen (QDH) ratsam. Für Bachelorstudierende werden Lerninhalte empfohlen, mit denen sich die Kompetenz- bzw. Prozessdimensionen *Strukturieren, Erarbeiten* und *Anwenden* vermitteln lassen. Somit sind Grundlagen der Hardwarebeschreibung mittels HDLs auch für Bachelorstudierende geeignet, um zum Beispiel die bereits erwähnten Grundlagen digitaler Systeme anzuwenden [HD10]. Für fortgeschrittene Anwendungsszenarien, wie beispielsweise der dynamischen Rekonfiguration zur Laufzeit, müssen Studierende über ein umfängliches Wissen aus dem gesamten Bachelorstudium verfügen, wie es die Zugangsvoraussetzungen der aufgezeigten Universitäten beschreiben.

Sofern im Rahmen einer Veranstaltung auf Bachelorniveau über das *Anwenden* hinausgehend Kompetenzen gefragt sind, ist auf eine starke Hilfestellung durch die Lehrenden zu achten. Letzteres kann beispielsweise in Form von bereits entwickelten HDL-Komponenten geschehen, welche die Studierenden nutzen, ohne deren technische Umsetzung im Detail zu kennen. Dieses Konzept ist auch als *Didaktische Reduktion* bekannt [RS96]. Im Entwurfs- und Anwendungspraktikum wurde ein solches Rahmenwerk beispielsweise in Form von vorgefertigten Hardwarebeschreibungen vorgegeben, um die Studierenden bei der Konzeption und Entwicklung eigener Komponenten zu unterstützen. Die Prozessdimension „Entwickeln“ findet sich im Qualitätsrahmen für deutsche Hochschulen erst auf Masterniveau.

Wie dargestellt, lassen sich nicht alle Anwendungsbezüge der Idee auf Bachelorniveau lehren. Eine Niveaustufung ist deswegen ratsam. Das Fortbildungskriterium erhält im Analyseprozess deswegen jedoch nicht mehr die höchstmögliche Wertung, sondern kann nur mit Einschränkungen (der Niveaustufung) positiv evaluiert werden.

Zeitkriterium (3) Rekonfigurierbare Architekturen werden seit 1980, das Konzept an sich seit 1960, wissenschaftlich diskutiert [Hau98], [CH00]. Heute sind FPGAs die am häufigsten genutzte Technologie für die Umsetzung rekonfigurierbarer Systeme [HD10]. Obwohl alternative Ansätze bestehen, hat sich bisher keines dieser Konzepte durchgesetzt. Guccione vermutet, dass dieser Umstand der schlechten Verfügbarkeit und Qualität der zugehörigen Entwicklungsumgebungen geschuldet ist [Guc10]. Durch das Einsatzgebiet des High-Performance-Computing scheinen neue Anwendungsgebiete erschlossen zu werden. Gleichzeitig zeigt die UBM-Entwicklerumfrage von 2013 einen Abwährtstrend in der Nutzung von FPGAs in aktuellen Projekten [Tec13]. Neben der nicht benötigten Funktionalität stellen die Kosten von FPGAs im Vergleich zu mikroprozessbasierten Lösungen den Hauptgrund für diesen Trend dar [Tec13]. Die Gründe spiegeln also eine nachteilige technische Umsetzung, nicht aber Nachteile der Idee selbst, wider. Butts bezeichnet rekonfigurierbare Systeme als „Turn of the Century"-Technologie, weist aber darauf hin, dass die reine Betrachtung von Kosten immer zu Gunsten einer nicht rekonfigurierbaren Systemlösung ausfallen wird [But95].

Einige Autoren gehen davon aus, dass die Beschreibung der Systemkonfiguration einen weiteren Abstraktionsschritt nach „oben" gehen wird und demnach zukünftige Systeme nicht mehr auf Registertransferebene, sondern auf der Ebene von Algorithmen beschrieben werden [CH00], [ABP11]. Am Beispiel von HDLs wie VHDL und Verilog sind diese Ansätze bereits recht fortgeschritten. Die Verhaltensbeschreibung einer Komponente kann prinzipiell ähnlich wie in klassischen Softwareprogrammiersprachen gestaltet werden, inklusive von Kontrollflussmechanismen wie Schleifen (*while, for*) oder Verzweigungen (*if, case*). Da für die Idee rekonfigurierbarer Architekturen und reprogrammierbarer Designs die Relevanz in der Vergangenheit und in der Zukunft gezeigt werden konnte, wird das Zeitkriterium erfüllt. Zusätzlich lässt sich feststellen, dass es einige Indikatoren dafür gibt, dass das Konzept auch in Zukunft eine wesentliche Rolle in der Entwicklung eingebetteter Systeme spielen wird. Das Zeitkriterium ist deswegen vollständig erfüllt.

Sinnkriterium (2) Rekonfigurierbare Architekturen sind das Mittel der Wahl, um frühe Hardware-Prototypen zu erstellen. Dies schließt auch die Arbeitsschritte ein, in denen versucht wird, Systemcharakteristika des finalen Systems abzuschätzen. Erst wenn Funktionalität und die Erfüllung der Rahmenbedingungen des Entwurfes zufriedenstellend sind, wird dieser auf nicht-rekonfigurierbare Hardware portiert. Von praktischer Bedeutung ist die Eigenschaft rekonfigurierbarer Systeme, mehr Funktionalität in einem System zu integrieren, als eigentlich aktuell benötigt wird. Je nach Anwendungskontext wechselt das System die Konfiguration der Hardwareblöcke und aktiviert die entsprechend notwendigen Funktionen. Um den Zeitaufwand für diese Rekonfiguration gering zu halten, haben sich unterschiedliche Ansätze zur Rekonfiguration etabliert. Hier sind Techniken wie die *partielle Rekonfiguration* oder die *Multikontext-Konfiguration* zu nennen [HD10]. Als Beispiel kann ein GSM-Modul dienen, welches nur die Hardwarekonfiguration aktiviert, die für den aktuellen Aufenthaltsort bzw. dessen Frequenzband erforderlich ist. Die Laufzeitrekonfiguration ermöglicht neben dem Beheben von Fehlern auch die dynamische Anpassung der Hardware an spontane Anforderungen der Umgebung und enthält

damit ein signifikantes Optimierungspotential. In diesem Zusammenhang nennen Compton und Hauck Verschlüsselungs- oder Entschlüsselungskomponenten auf Basis eines rekonfigurierbaren Systems, das speziell für einen gegebenen Schlüssel optimiert wurde [CH02]. Trotzdem behält das System weiterhin die Funktionalität, auch andere Schlüssel zu nutzen, dann aber deutlich ineffizienter (bis zu einer weiteren Rekonfiguration). FPGAs werden zudem in Anwendungsbereichen eingesetzt, die in sehr großem Stil parallele Bearbeitung benötigen [Loc+07]. Weitere Anwendungsbereiche, die im Kontext des Sinnkriteriums interessant sind, können in [Tod+05] nachgelesen werden.

Rekonfigurierbare Systeme haben zusammengefasst für einige Nischenanwendungen, insbesondere aber für das Prototyping eingebetteter Systeme, eine signifikante Bedeutung. In der zitierten UBM-Studie zeigt sich der Trend, dass FPGAs nicht mehr in dem Umfang für Projekte des Anwendungsbereiches genutzt werden, wie dies noch vor einigen Jahren der Fall war. Das Sinnkriterium wird aus diesem Grund mit Einschränkung erfüllt.

Varianzkriterium (3) Rekonfigurierbare Architekturen beinhalten viele Bezüge zu anderen Themengebieten. Zu nennen sind hier deswegen die Idee der Parallelität, Berechnungsmodelle bzw. „Models of Computation“ oder „Virtual Prototyping“. Die Verknüpfung zum Themengebiet der Parallelität findet sich in der Möglichkeit, die Struktur- oder Verhaltensbeschreibung mit nebenläufigen Komponenten und Prozessen zu gestalten. An dieser Stelle zeigt sich ebenso der Bezug zu Berechnungsmodellen. Dem Entwickler ist es überlassen, in welcher Weise er das System spezifiziert. Zum einen können die LUTs und Flip-Flops im FPGA direkt durch die Registertransferebene beschrieben werden, zum anderen aber auch durch eine abstraktere Verhaltensbeschreibung, die Konstrukte wie Schleifen oder Verzweigungen nutzt. Bei letzterer übernimmt das Synthesewerkzeug das Mapping auf die entsprechende Zielplattform. Verknüpfungen zu Middlewares und Betriebssystemen sind sichtbar, wenn man die Laufzeitumgebung, die zur dynamischen (Re-)Konfiguration von Hardwareverbindungen dient, als abstrakte Vermittlungsschicht auffasst. Die Laufzeitumgebung ist dann ein Hardware-/Softwaresystem, welches die Neustrukturierung des restlichen Systems nach externen Anforderungen hin durchführt. Um ein rekonfigurierbares System umzusetzen, kann neben Verilog und VHDL auch SystemC benutzt werden. SystemC ist keine eigene Sprache, sondern eine Programmbibliothek, die C unter anderem um Funktionen wie Ports und eine vierwertige Logik erweitert. Hardwareseitig können CPLDs („Complex Programmable Logic Devices“) von FPGAs unterschieden werden. Allerdings ändert sich durch diese alternative Umsetzung nichts an der grundlegenden Idee der Rekonfiguration. Alle genannten Themengebiete und Konzepte sind als verwandt, aber nicht als synonym einzustufen. Der Aspekt der Rekonfiguration nach Beendigung des Fertigungsprozesses findet sich in keiner anderen analysierten Idee wieder und stellt, wie im Horizontal- und Sinnkriterium beschrieben, das zentrale Merkmal der Idee dar. Das Varianzkriterium ist deswegen erfüllt.

A.22. Synthese, Analyse und Verifikation nicht-funktionaler Anforderungen

Nicht-funktionale Anforderungen und ihre Auswirkungen im Entwicklungsprozess sind bereits in den vorherigen Ideenanalysen sichtbar geworden. Nicht nur die Definition des Begriffes, sondern auch dessen Interpretation ist schwierig [Gli05]. Eine nicht-funktionale Anforderung (bspw. Sicherheit des Systems im Kontext von Fremdzugriffen) kann im Entwicklungsprozess unter Umständen durch wenige Änderungen zu einer funktionalen Anforderung werden [Gli07].

Während viele der bisher analysierten Ideen einen methodischen Charakter hatten, wird dieser bei der vorliegenden Idee auch durch eine Sichtweisen-Komponente ergänzt. Synthese, Analyse und Verifikation als Methoden treffen auf das abstrakte Konstrukt nicht-funktionaler Anforderungen. Die folgende Analyse untersucht die Methoden zum Umgang mit nicht-funktionalen Anforderungen, wie auch die damit verbundene, prinzipielle, Problemstellung diese zu definieren.

Ein typisches englischsprachiges Merkmal nicht-funktionaler Anforderungen ist die Endung auf *-ness,-ities,-ilities* und *-ty.* „Reliability“, „Safety“, „Performance“, „Dependability“ oder „Security“ sind entsprechende nicht-funktionale Anforderungen.

Horizontalkriterium (3) Entwickler beschäftigen sich im zweiten Schritt des Entwicklungsprozesses, dem „Requirements Analysis Process“, sowie auch im Design mit der Erkennung und Modellierung funktionaler und nicht-funktionaler Anforderungen. Attribute wie Leistung und Sicherheit finden sich in der spezifischen Systemumsetzung, also der klassischen Implementierungsphase. Nicht-funktionale Anforderungen sind ebenso in den Entwicklungsschritten der Integration und in den darauffolgenden Entwicklungsphasen bis hin zur Wartung zu berücksichtigen, da sie wesentlicher Bestandteil der Systemzielstellung sind, die bis zum Ende des Systemlebenszyklus erfüllt sein müssen. Die Erhebung von Anforderungen im Allgemeinen hat eine eigene Fachrichtung innerhalb der Disziplin hervorgerufen: das „Requirements-Engineering“. Die Tragweite für das Anwendungsgebiet ist damit gegeben.

Fortbildungskriterium (2) Das Verständnis über die Wichtigkeit und die Auswirkungen nicht erfüllter nicht-funtkionaler Anforderungen ist essentiell für viele Bereiche der Entwicklung eingebetteter Systeme (bspw. „Reliable Design“). Die Schwierigkeit, diese Art von Anforderungen zu erkennen, zu verknüpfen und in der Entwicklung umzusetzen, ist unabhängig vom Verständnis über deren Notwendigkeit. Die folgende differenzierte Niveaustufung (mit Referenz auf den QDH) ist deswegen empfehlenswert:

- Das Verständnis über die Auswirkung nicht-funktionaler Anforderungen kann auf Bachelorniveau anhand von Beispielen verdeutlicht werden. Darauf aufbauend kann ebenfalls das Erarbeiten und Strukturieren von nicht-funktionalen

Anforderungen im Zusammenhang mit dem Syntheseschritt als praktische Aufgabe gestellt werden.

- Die Umsetzung (Kontext: *Anwenden* aus QDH) sowie die Analyse von Systemen auf ihre nicht-funktionalen Eigenschaften ist auf Masterniveau einzuordnen. Wie bei anderen Ideen auch kann durch eine besonders starke Unterstützung der Studierenden durch die Betreuer ein Teil der genannten Kompetenzen auch auf Bachelorniveau umgesetzt werden. In diesen Fällen ist die Nutzung von Lernumgebungen besonders sinnvoll, da sie es den Studierenden ermöglichen, ihre Aufmerksamkeit ganz auf den Aufgabenaspekt zu richten und technische Rahmenbedingungen auszublenden.

  Das eigenständige, also freie, Entwerfen eines Systems in Abhängigkeit von vorher ermittelten Anforderungen kann in Praktika stufenweise geübt werden. Auf Masterniveau lassen sich fortgeschrittene Konzepte wie die „Design-Space-Exploration“ (siehe Abschnitt A.12) oder die Controller-Synthese (siehe Abschnitt A.3) in den praktischen Ablauf eingliedern.

Macaulay und Mylopoulos haben in einem ihrer Artikel zum Themengebiet eine Umfrage unter Wissenschaftlern und Entwicklern durchgeführt, um Anforderungen an Studierende hinsichtlich ihrer Fähigkeiten und Fertigkeiten zur Bestimmung nicht-funktionaler Anforderungen zu ermitteln [MM95]. Sie kommen zu dem Schluss, dass sich deklarative und prozedurale Wissenselemente gut aufteilen lassen. Sie benennen fünf grundlegende Aspekte nicht-funktionaler Anforderungen, die sich in 14 Tätigkeitsbeschreibungen strukturieren lassen. Diese Untersuchung kann als Basis für eine Umsetzung auf Bachelorniveau verwendet werden. Für fortgeschrittene Kompetenzen unterstreichen Macaulay und Mylopoulos die Notwendigkeit praktischer Erfahrungen [MM95].

Zeitkriterium (3) Roman erläuterte 1985 bereits die prinzipiellen Schwierigkeiten im Umgang mit nicht-funktionalen Anforderungen [Rom85]. Neben der komplizierten Spezifikation geht er auch auf die Testbarkeit von nicht-funktionalen Anforderungen ein. Kaindl beschäftigte sich wenige Jahre später mit dem Vorgehen zur Erarbeitung nicht-funktionaler Anforderungen [Kai93]. Er veröffentlichte 1993 einen Ansatz, der als Zwischenschritt zwischen verbaler und formaler Anforderungsspezifikation gesehen werden kann. Eine wissenschaftliche Arbeit, in der die Begriffe *Verlässlich* und *Zuverlässig* voneinander unterschieden werden, hat Burns veröffentlicht [BL91]. Die Suche nach früheren wissenschaftlichen Arbeiten zum Themenbereich wird dadurch erschwert, dass der Begriff als solcher noch nicht weitläufig geprägt war. Alle genannten Publikationen belegen jedoch die breite Beobachtbarkeit des Themenkomplexes in der vom Zeitkriterium geforderten Zeitspanne.

Sinnkriterium (3) Nicht-funktionale Anforderungen sind in jeder Entwicklung eingebetteter Systeme wichtig. Je nach Anwendungsgebiet und Einsatzzweck ist ihre Gewichtung jedoch unterschiedlich. Die zentrale Rolle von nicht-funktionalen Anforderungen lässt sich daran ablesen, dass viele der analysierten Ideen Aufgaben-

und Problemstellungen dieser Domäne lösen müssen. Je nach Definitionsgrundlage werden nicht-funktionale Anforderungen von Systemen als der Ursprung jeglicher Entscheidungen im Entwicklungsprozess gesehen [LS95]. Das Verständnis von Erhebung, Beurteilung und Umsetzung dieser Attribute ist deswegen im praktischen wie im wissenschaftlichen Umfeld wichtig und das Sinnkriterium dadurch vollständig erfüllt.

Varianzkriterium (3) Auf die weitreichenden Verknüpfungen zu bereits analysierten Ideen wurde im Kontext des Sinn- und des Horizontalkriteriums hingewiesen. Ähnlich zum Analysevorgehen von Schwill kann diese Idee als Masteridee gesehen werden, da sie sich übergeordnet zu vielen anderen Ideen einsortieren lässt. Sie besitzt dementsprechend Berührungspunkte mit vielen weiteren Ideen. Da sie zwischen diesen eine Verbindung herstellt, verfügt sie über eine besonders wertvolle fachdidaktische Eigenschaft, die zum einen in keiner anderen analysierten Idee vorhanden ist und sich zum anderen gut für die Einführung in den Themenbereich „eingebettete Systeme“ eignet.

B. Beispielprojekt: „Spiel des Lebens“

Die folgenden Erläuterungen sollen dabei helfen, das in Abschnitt 6.4 dargestellte Beispielprojekt des Spiel des Lebens auf Umsetzungsebene zu verstehen. Für das Verständnis auf Implementierungsniveau werden die im genannten Abschnitt erläuterten Konzepte vorausgesetzt. Die aktuelle Version des Projektes ist auf dem Webserver der Universität Siegen hinterlegt: `http://www.eti.uni-siegen.de/imt_mse/forschung/projekte/gameoflife/gameoflife-final.zip?lang=de`

Rahmenbedingungen

Das Design der Komponente wurde ausschließlich in der Xilinx ISE-Suite (Edition Webpack) entworfen und kann mit diesem Werkzeug auch direkt importiert werden. Das Projekt kann auf verschiedenen FPGAs mit ausreichenden Hardwareressourcen umgesetzt werden und enthält aus diesem Grund kein User-Constraint-File, in dem typischerweise die PIN-Konfiguration der Hardware vorgenommen wird. Im beigefügten Xilinx ISE-Projekt ist ein FPGA vom Typ „Spartan-3A“ vorkonfiguriert. Die Zellmatrix für das Spiel des Lebens ist durch ein 30 x 30-Spielfeld realisiert. Eine Erweiterung der Matrixgröße ist problemlos möglich, solange der FPGA über genügend Block-RAM bzw. CLBs verfügt. Alternativ ist die Ansteuerung eines externen Speicherbausteins möglich. Die Anfangskonfiguration der Zellmatrix wurde so gewählt, dass die Zellen an den Rändern des Spielfeldes „lebendig“ sind. Alle anderen Zellen sind anfangs „tot“.

Umsetzung

Die Umsetzung der Hauptkomponente ist in drei Unterkomponenten *Spielelogik*, *Spielecontroller* und *Speichercontroller* aufgeteilt. Die Spielelogik beinhaltet den Algorithmus zur Bewertung des Zellstatus im nächsten Zeitabschnitt. Der Spielecontroller iteriert zeilenweise durch das Spielfeld und bestimmt zu jeder Zelle die Nachbarn. Die Informationen über die neun davon betroffenen Zellen (Hauptzelle und ihre acht Nachbarn) wird an die Eingänge der Spielelogik angelegt. Wenn die Spielelogik den Zellenstatus bestimmt hat, sendet sie ihn an den Spielecontroller zum Rückschreiben in die Zellmatrix. Für das Laden und Speichern von Zellwerten ist der Speichercontroller zuständig, der direkt mit dem Spielecontroller verbunden ist. Letzterer verfügt über zwei Adressleitungen für jede Matrix von

Zellinformationen. Von diesen Matrizen sind insgesamt zwei notwendig. Die erste beinhaltet den Zellstatus im aktuellen, die andere die des nächsten Zeitabschnittes. Ist eine Matrix vollständig durch die Spielelogik abgearbeitet worden, tauscht der Spielecontroller beide Matrizen. Dafür verwendet er den „Select“-Eingang am Speichercontroller, der wie ein Multiplexer Daten- und Adressleitungen zwischen verschiedenen RAM-Bausteinen schaltet. Die Anfangskonfiguration der Zellmatrix kann direkt in den RAM-Bausteinen angegeben werden.

Für eine Visualisierung (bspw. per VGA) wurden alle Komponenten bereits vorbereitet. Insbesondere verfügt der Speichercontroller über weitere Adresseingänge, mit denen ein Zellwert von einer Visualisierungskomponente unabhängig gelesen werden kann. Die Schaltung besitzt zudem alle für die Ansteuerung eines VGA-Controllers nötigen Daten- und Adressleitungen (bspw. vertikale und horizontale Synchronisation). Digilent Inc. bietet auf der Webseite zum Spartan-3 Board ein in VHDL implementiertes Referenzdesign des VGA-Controllers zum Herunterladen an [Inc14] .

Die Beobachtung des Zellverhaltens kann kontinuierlich oder schrittweise durch die Ausführung der beigelegten Testbenches simuliert werden. Auch ohne den Anschluss einer Visualisierungskomponente kann das Zellverhalten in der Simulationsumgebung iSim von Xilinx beobachtet werden. Für die vorgegebene Anfangsbelegung der Matrix lassen sich so auch in dieser Visualisierungsvariante stabile Zellkombinationen und oszillierende Zellen erkennen und die schrittweise Auswertung der Logikeinheit nachvollziehen.

Parallelisierung und Optimierung

Wie bereits in Abschnitt 6.4 erläutert, kann das vorgestellte Design in der Veranstaltung zum Varianzkriterium hinsichtlich Parallelisierungs- und Optimierungspotential untersucht werden. Da die Spielelogik in einer einzelnen Komponente implementiert ist, ist der einfachste Ansatz zur schnelleren Bestimmung des Zellstatus die Vervielfachung dieser Komponente. Verschiedene Parallelisierungsgrade von einer Komponente pro Zeile bis zu einer Komponente pro Zelle sind je nach Ressourcenverfügbarkeit auf der Zielplattform möglich. In jedem Fall muss die notwendige Anpassung des Spielecontrollers berücksichtigt werden. Zum einen sind entsprechend der Anzahl der Logikkomponenten Ein- und Ausgänge bereit zu stellen. Zum anderen kann das Interfacing zum Speichercontroller auf verschiedene Arten verbessert werden. Entweder der Spielecontroller ruft mehrere Zellinformationen sequentiell ab, bevor diese an die Spielelogik weitergegeben werden, oder die Schnittstelle zum Speichercontroller wird ebenfalls parallel umgesetzt. Die parallele Umsetzung wird voraussichtlich aus sehr vielen Logikblöcken bestehen und kann deswegen alternativ auch für jede Zeile oder Spalte der Zellmatrix mittels eines eigenen Speichercontrollers durchgeführt werden.

Das Optimierungspotential hinsichtlich der Ressourcennutzung des FPGAs kann in einer Lehrveranstaltung insbesondere durch die Überarbeitung der Spielelogik

erfolgen. Die bisherige VHDL-Beschreibung orientiert sich an einer einfach verständlichen imperativen Schreibweise des Algorithmus. Eine alternative Beschreibung in VHDL ist beispielsweise durch die Nutzung von „Do-Not-Care“-Termen möglich. Die Funktionalität des Algorithmus kann damit auch auf Register-Transfer-Ebene beschrieben werden, die unter Umständen eine effizientere Synthetisierung ermöglicht.

Literatur

[Abi+10] H.Z. Abidin u. a. „Incorporating VHDL in teaching combinational logic circuit“. In: *Engineering Education (ICEED), 2010 2nd International Congress on*. Dez. 2010, S. 225–228.

[ABP11] G. Ansaloni, P. Bonzini und L. Pozzi. „EGRA: A Coarse Grained Reconfigurable Architectural Template“. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 19.6 (Juni 2011), S. 1062–1074.

[ACM04] ACM/IEEE. *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering: A Report in the Computing Curricula Series*. Techn. Ber. The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery, 2004.

[AD94] T. Aldemir und North Atlantic Treaty Organization. Scientific Affairs Division. *Reliability and Safety Assessment of Dynamic Process Systems*. NATO ASI ser. Ser. F. Springer, 1994.

[Ada01] Marian Adamski. „A rigorous design methodology for reprogrammable logic controllers“. In: *The International Workshop on Discrete-Event System Design, DESDes*. Bd. 1. 2001.

[AG93] JR Armstrong und FG Gray. „Teaching VHDL At The Undergraduate And Graduate Levels“. In: *Proc. VIUF Conference VHDL Boot Camp, San Jose, CA*. 1993, S. 213–216.

[Agh85] Gul Abdulnabi Agha. *Actors: a model of concurrent computation in distributed systems*. Techn. Ber. Artificial Intelligence Laboratory, 1985.

[AH01] Luca de Alfaro und Thomas Henzinger. „Interface Theories for Component-Based Design“. In: *Embedded Software*. Hrsg. von Thomas Henzinger und Christoph M. Kirsch. Bd. 2211. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, S. 148–165.

[Ahm95] Ishfaq Ahmad. „A massively parallel fault-tolerant architecture for time-critical computing.“ In: *The Journal of Supercomputing* 9.1-2 (1995), S. 135–162.

[AK01] Lorin W. Anderson und David R. Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. Hrsg. von Lorin W. Anderson und David R. Krathwohl. 2. Aufl. New York: Allyn & Bacon, Dez. 2001.

[AMO05] Péter Arató, Zoltán Ádám Mann und András Orbán. „Extending component-based design with hardware components“. In: *Science of Computer Programming* 56.1-2 (2005). New Software Composition Concepts, S. 23–39.

[AP04] Jason M. Aughenbaugh und Christiaan J.J. Paredis. „The Role and Limitations of Modeling and Simulation in Systems Design“. In: *2004 ASME International Mechanical Engineering Congress and RD&D Expo*. 2004, IMECE2004–5981.

[Are01] S. Areibi. „A first course in digital design using VHDL and programmable logic“. In: *Frontiers in Education Conference, 2001. 31st Annual*. Bd. 1. 2001, pages.

[ARS96] John R Anderson, Lynne M Reder und Herbert A Simon. „Situated learning and education“. In: *Educational researcher* 25.4 (1996), S. 5–11.

[Avi+04] A. Avizienis u. a. „Basic concepts and taxonomy of dependable and secure computing“. In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (Jan. 2004), S. 11–33.

[Bag+10] Alessandra Bagnato u. a. „MADES: Embedded systems engineering approach in the avionics domain“. In: *Proceedings of the First Workshop on Hands-on Platforms and Tools for Model-Based Engineering of Embedded Systems, HoPES*. 2010.

[Bai+04] Christel Baier u. a. „Controller synthesis for probabilistic systems“. In: *Exploring New Frontiers of Theoretical Informatics*. Springer, 2004, S. 493–506.

[Bak+10] Sidi Bakhkat u. a. *Eingebettete Systeme - Ein strategisches Wachstumsfeld für Deutschland*. Techn. Ber. Bundesverband für Informationswirtschaft (BITKOM), 2010.

[Bas+10] Twan Basten u. a. „Model-driven design-space exploration for embedded systems: the octopus toolset“. In: *Leveraging Applications of Formal Methods, Verification, and Validation*. Springer, 2010, S. 90–105.

[BB04] M. Basso und G. Bagni. „ARTIST: a real-time interactive Simulink-based telelab“. In: *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*. Sep. 2004, S. 196–201.

[BCE56] Benjamin Samuel Bloom, Committee of College und University Examiners. *Taxonomy of educational objectives*. Bd. 1. David McKay New York, 1956.

[BD02] Luca Benini und Giovani De Micheli. „Networks on chips: a new SoC paradigm“. In: *Computer* 35.1 (Jan. 2002), S. 70–78.

[BD04] Luca Benini und Giovani De Micheli. „Networks on chips: A new paradigm for component-based MPSoC design“. In: *Proc. MPSoC* (2004).

[BGP07] Christian Bunse, Hans-Gerhard Gross und Christian Peper. „Applying a Model-based Approach for Embedded System Development“. In: *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*. EUROMICRO '07. Washington, DC, USA: IEEE Computer Society, 2007, S. 121–128.

[BGV96] S. Bhatia, Tushar Gheewala und P. Varma. „A unifying methodology for intellectual property and custom logic testing“. In: *Test Conference, 1996. Proceedings., International*. Okt. 1996, S. 639–648.

[Big96] John Biggs. „Enhancing teaching through constructive alignment“. In: *Higher education* 32.3 (1996), S. 347–364.

[Bil+07] Bundesministerium für Bildung und Forschung (BMBF) u. a. *IKT 2020. Forschung für Innovationen*. Techn. Ber. Bonn: Bundesministerium für Bildung und Forschung, 2007.

[BJ13] Steffen Büchner und Steffen Jaschke. „Preparation for embedded systems laboratories the virtual workspace approach“. In: *Global Engineering Education Conference (EDUCON), 2013 IEEE*. IEEE, 2013, S. 171–175.

[BJS13] Steffen Büchner, Steffen Jaschke und Sigrid Schubert. „Enhancing the Comparability between Didactic Research on Embedded Systems“. In: *Embedded System Week 2013*. Montreal, Okt. 2013.

[BKR10] N Bencheva, N Kostadinov und Y Ruseva. „On Teaching Hardware/Software Co-design using FPGA“. In: *Electronics and Electrical Engineering.–Kaunas: Technologija* 6 (2010), S. 102.

[BL91] A. Burns und A. M. Lister. „A Framework for Building Dependable Systems“. In: *The Computer Journal* 34.2 (1991), S. 173–181.

[Blu+91] Phyllis C Blumenfeld u. a. „Motivating project-based learning: Sustaining the doing, supporting the learning“. In: *Educational psychologist* 26.3-4 (1991), S. 369–398.

[BM06] Tobias Bjerregaard und Shankar Mahadevan. „A Survey of Research and Practices of Network-on-chip“. In: *ACM Comput. Surv.* 38.1 (Juni 2006).

[BMA05] B. Bailey, G. Martin und T. Anderson. *Taxonomies for the Development and Verification of Digital Systems*. Springer, 2005.

[BN91] Earl C Butterfield und Gregory D Nelson. „Promoting positive transfer of different types“. In: *Cognition and Instruction* 8.1 (1991), S. 69–102.

[Boe10] Barry. Boehm. *Systems 2020. Strategic Initiative*. Techn. Ber. Systems Engineering Research Center, 2010.

[Boh09] M. Bohr. „The new era of scaling in an SoC world“. In: *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*. Feb. 2009, S. 23–28.

[Bow08] Matt Bower. „A taxonomy of task types in computing“. In: *ACM SIGCSE Bulletin*. Bd. 40. 3. ACM. 2008, S. 281–285.

[Bra+08] Jörg Brakensiek u. a. „Virtualization As an Enabler for Security in Mobile Devices“. In: *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*. IIES '08. Glasgow, Scotland: ACM, 2008, S. 17–22.

[Bra08] Claus Brabrand. „Constructive alignment for teaching model-based design for concurrency“. In: *Transactions on petri nets and other models of concurrency I*. Springer, 2008, S. 1–18.

[Bri+05] Ed Brinksma u. a. *A Modelling Method for Embedded Systems*. Enschede, The Netherlands, 2005.

[Bro96] Alan W. Brown, Hrsg. *Component-Based Software Engineering*. John Wiley & Sons, Sep. 1996.

[Bru60] Jerome Bruner. *The process of education*. Harvard University Press, 1960.

[Bru80] Jerome Bruner. *Der Prozeß der Erziehung*. 5. Aufl. 1. dt. Aufl. 1970. Pädagogischer Verlag Schwann, 1980.

[BS02] Torsten Brinda und Sigrid Schubert. „Didactic system for object-oriented modelling“. In: *Networking the Learner*. Springer, 2002, S. 473–482.

[BS14] Steffen Büchner und Sigrid Schubert. „Criteria-based Research on Fundamentals of Embedded System Development in Higher Education“. In: *Proceedings of the International Conference on Embedded Systems and Applications (ESA 2014)* (2014).

[BS73] Mario R. Barbacci und Daniel P. Siewiorek. „Automated exploration of the design space for register transfer (RT) systems“. In: *SIGARCH Comput. Archit. News* 2.4 (Dez. 1973), S. 101–106.

[BS93] Jonathan Bowen und Victoria Stavridou. „Safety-critical systems, formal methods and standards“. In: *Software Engineering Journal* 8.4 (1993), S. 189–209.

[BST10] Karsten Berns, Bernd Schürmann und Mario Trapp. *Eingebettete Systeme - Systemgrundlagen und Entwicklung eingebetteter Software*. Vieweg, 2010, S. I–X, 1–270.

[BT10] John Biggs und Catherine Tang. „Applying constructive alignment to outcomes-based teaching and learning“. In: *Training Material for "Quality Teaching for Learning in Higher Education"Workshop for Master Trainers, Ministry of Higher Education, Kuala Lumpur*. 2010, S. 23–25.

[Büc13] Steffen Büchner. „Teaching Embedded Systems in Higher Education by Visualizing Component Relations“. In: *X World Conference on Computers in Education* (Juli 2013).

[Büc14] Steffen Büchner. „Empirical and Normative Research on Fundamental Ideas of Embedded System Development“. In: *Proceedings of Key Competencies in Informatics and ICT 2014 - in print* (2014).

[But06] Giorgio Buttazzo. „Research trends in real-time computing for embedded systems“. In: *ACM SIGBED Review* 3.3 (2006), S. 1–10.

[But95] M. Butts. „Future directions of dynamically reprogrammable systems“. In: *Custom Integrated Circuits Conference, 1995., Proceedings of the IEEE 1995*. Mai 1995, S. 487–494.

[Cas+05] P Caspi u. a. „Guidelines for a graduate curriculum on embedded software and systems“. In: *ACM Trans. Embed. Comput. Syst.* 4.3 (Aug. 2005), S. 587–611.

[Cas+08] Lillian Cassel u. a. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. Techn. Ber. New York, NY, USA, 2008.

[CAS11] Octavian Cheng, Waleed Abdulla und Zoran Salcic. „Hardware–software codesign of automatic speech recognition system for embedded real-time applications“. In: *Industrial Electronics, IEEE Transactions on* 58.3 (2011), S. 850–859.

[Cat+09] Vincenzo Catania u. a. „An Effective Methodology to Multi-objective Design of Application Domain-specific Embedded Architectures“. In: *Digital System Design, Architectures, Methods and Tools, 2009. DSD'09. 12th Euromicro Conference on*. IEEE. 2009, S. 643–650.

[Ces+02] W.O. Cesario u. a. „Multiprocessor SoC platforms: a component-based design approach“. In: *Design Test of Computers, IEEE* 19.6 (Nov. 2002), S. 52–63.

[CF75] Mario von Cranach und Hans-Georg Frenz. „Systematische Beobachtung“. In: *Sozialpsychologie, 1. Halbband: Theorien und Methoden*. 1975.

[CH00] Katherine Compton und Scott Hauck. „An introduction to reconfigurable computing“. In: *IEEE Computer* (2000).

[CH02] Katherine Compton und Scott Hauck. „Reconfigurable Computing: A Survey of Systems and Software“. In: *ACM Comput. Surv.* 34.2 (Juni 2002), S. 171–210.

[Cha+00] Kuo-En Chang u. a. „A programming learning system for beginners-A completion strategy approach“. In: *Education, IEEE Transactions on* 43.2 (2000), S. 211–220.

[Cha84] Daniel M Chapiro. „Globally-asynchronous locally-synchronous systems“. In: *Ph. D. Thesis* 1 (1984), S. 50.

[Cha99] H. Chang. *Surviving the SOC Revolution: A Guide to Platform-Based Design*. Springer, 1999.

[Che+11] Ronald S Cheung u. a. „Challenge based learning in cybersecurity education“. In: *Proceedings of the 2011 International Conference on Security & Management*. Bd. 1. 2011.

[CM81] Bernard Chazelle und Louis Monier. „A model of computation for VLSI with related complexity results“. In: *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. STOC '81. Milwaukee, Wisconsin, USA: ACM, 1981, S. 318–325.

[COB95] Pai Chou, Ross B. Ortega und Gaetano Borriello. „Interface Co-synthesis Techniques for Embedded Systems“. In: *Proceedings of the 1995 IEEE/ACM International Conference on Computer-aided Design*. ICCAD '95. San Jose, California, USA: IEEE Computer Society, 1995, S. 280–287.

[Com12] Association for Computing Machinery (ACM). *The ACM Computing Classification System (CCS)*. 2012. URL: `http://dl.acm.org/ccs.cfm` (besucht am 11. 11. 2014).

[Com14] Association for Computing Machinery (ACM). *ACM Transactions on Embedded Computing*. 2014. URL: `http://acmtecs.acm.org/` (besucht am 11. 11. 2014).

[CR10] Albert Cohen und Erven Rohou. „Processor Virtualization and Split Compilation for Heterogeneous Multicore Embedded Systems“. In: *Proceedings of the 47th Design Automation Conference*. DAC '10. Anaheim, California: ACM, 2010, S. 102–107.

[CRM10] A. Crespo, I. Ripoll und M. Masmano. „Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach“. In: *Dependable Computing Conference (EDCC), 2010 European*. Apr. 2010, S. 67–72.

[CW96] Edmund M Clarke und Jeannette M Wing. „Formal methods: State of the art and future directions“. In: *ACM Computing Surveys (CSUR)* 28.4 (1996), S. 626–643.

[Cyb13] The Steering Group for the Cybersecurity research: a vision for the UK. *Cybersecurity research - a vision for the UK*. Techn. Ber. THE ROYAL SOCIETY, Nov. 2013.

[Dam+09] Werner Damm u. a. *Nationale Roadmap Embedded Systems*. Techn. Ber. Zentralverband Elektrotechnik- und Elektronikindustrie e.V. Kompetenzzentrum Embedded Software & Systems, Dez. 2009, S. 65–137.

[De 99] H. De Man. „System-on-chip design: impact on education and research“. In: *Design Test of Computers, IEEE* 16.3 (1999), S. 11–19.

[DeH+04] André DeHon u. a. „Design patterns for reconfigurable computing“. In: *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*. IEEE. 2004, S. 13–23.

[Den03] Peter J Denning. „Great principles of computing“. In: *Communications of the ACM* 46.11 (2003), S. 15–20.

[Den07] Peter J. Denning. „Computing is a natural science“. In: *Communications of the ACM* 50.7 (Juni 2007), S. 13–18.

[Det93] D. Detterman. „The case for prosecution: Transfer as an epiphenomenon“. In: *Transfer on Trial: Intelligence, Cognition and Instruction*. Hrsg. von Douglas K. Detterman und Robert J. Sternberg. Norwood, NJ: Ablex Publishing, 1993.

[Deu12] Theresa Deutscher. „Grundideen der Mathematik“. In: *Arithmetische und geometrische Fähigkeiten von Schulanfängern*. Springer, 2012, S. 75–97.

[DG97] G De Michell und Rajesh K Gupta. „Hardware/software co-design“. In: *Proceedings of the IEEE* 85.3 (1997), S. 349–365.

[DH94] Joseph G D'Ambrosio und Xiaobo Sharon Hu. „Configuration-level hardware/software partitioning for real-time embedded systems“. In: *Hardware/Software Codesign, 1994., Proceedings of the Third International Workshop on*. IEEE. 1994, S. 34–41.

[DM13] C.E. Dickerson und D. Mavris. „A Brief History of Models and Model Based Systems Engineering and the Case for Relational Orientation“. In: *Systems Journal, IEEE* 7.4 (Dez. 2013), S. 581–592.

[DM98] Leonardo Dagum und Ramesh Menon. „OpenMP: an industry standard API for shared-memory programming“. In: *Computational Science & Engineering, IEEE* 5.1 (1998), S. 46–55.

[Dor+05] Maximillian Dornseif u. a. „Teaching data security at university degree level“. In: *Proceedings of the WISE04*. 2005.

[Dor95] Richard C Dorf. *Modern control systems*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[DS13] Yehudit Judy Dori und Irit Sasson. „A three-attribute transfer skills framework–part I: establishing the model and its relation to chemical education“. In: *Chemistry Education Research and Practice* 14.4 (2013), S. 363–375.

[Duc05] R.J. Duckworth. „Embedded system design with FPGA using HDL (lessons learned and pitfalls to be avoided)“. In: *Microelectronic Systems Education, 2005. (MSE '05). Proceedings. 2005 IEEE International Conference on*. Juni 2005, S. 35–36.

[Eck+06] Anna Eckerdal u. a. „Putting threshold concepts into context in computer science education“. In: *ACM SIGCSE Bulletin* 38.3 (2006), S. 103–107.

[Edw+01] Stephen Edwards u. a. „Design of embedded systems: Formal models, validation, and synthesis“. In: *Readings in hardware/software co-design* (2001), S. 86.

[EI14a] Institute of Electrical und Electronics Engineers (IEEE). *IEEE Taxonomy, Version 1.0*. 2014. URL: `https://www.ieee.org/documents/taxonomy_v101.pdf` (besucht am 11. 11. 2014).

[EI14b] Institute of Electrical und Electronics Engineers (IEEE). *IEEE Thesaurus*. 2014. URL: `http://www.ieee.org/publications_standards/publications/services/thesaurus_access_page.html` (besucht am 11. 11. 2014).

[Ern98] Rolf Ernst. „Codesign of embedded systems: Status and trends“. In: *Design & Test of Computers, IEEE* 15.2 (1998), S. 45–54.

[Est+02] Deborah Estrin u. a. „Connecting the Physical World with Pervasive Networks.“ In: *IEEE Pervasive Computing* 1.1 (2002), S. 59–69.

[EW12] W. Edelmann und S. Wittmann. *Lernpsychologie: Mit Online-Materialien*. Beltz, 2012.

[Fay+07] D. Fay u. a. „Teaching Fault Tolerant FPGA Design for Aerospace Applications“. In: *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*. Juni 2007, S. 61–62.

[Fey13] Prof. Dr. Dietmar Fey. *Vorlesung Rechnerarchitektur, Universität Erlangen-Nürnberg*. 2013. URL: `http://www3.informatik.uni-erlangen.de/Lehre/RA/WS2013/` (besucht am 11. 11. 2014).

[Fil+98] E. Filippi u. a. „Intellectual property re-use in embedded system co-design: an industrial case study“. In: *System Synthesis, 1998. Proceedings. 11th International Symposium on*. Dez. 1998, S. 37–42.

[Fin13] L.D. Fink. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses*. Jossey-Bass Higher and Adult Education Series. Wiley, 2013.

[Fin96] K. Finney. „Mathematical notation in formal specification: too difficult for the masses?“ In: *Software Engineering, IEEE Transactions on* 22.2 (Feb. 1996), S. 158–159.

[Fis+04] Gerhard Fischer u. a. „Meta-design: a manifesto for end-user development“. In: *Communications of the ACM* 47.9 (2004), S. 33–37.

[Ful+07] Ursula Fuller u. a. „Developing a Computer Science-specific Learning Taxonomy“. In: *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*. New York, NY, USA, 2007.

[Gab+04] Edgar Gabriel u. a. „Open MPI: Goals, concept, and design of a next generation MPI implementation“. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2004, S. 97–104.

[Gaj+00a] Daniel Gajski u. a. „Embedded Tutorial: Essential Issues for IP Reuse“. In: *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*. ASP-DAC '00. Yokohama, Japan: ACM, 2000, S. 37–42.

[Gaj+00b] Daniel Gajski u. a. *SpecC: Specification Language and Methodology*. 1. Aufl. Springer, 1. März 2000.

[Gar05] Stuart Garner. „The CLOZE procedure and the learning of programming“. In: *International Conference on Learning, Granada, Spain*. 2005.

[GBM05] Yee M. Goh, Julian Booker und Chris McMahon. „A Comparison of Methods in Probabilistic Design Based on Computational and Modelling Issues“. In: *Advances in Integrated Design and Manufacturing in Mechanical Engineering*. Hrsg. von Alan Bramley u. a. Berlin/Heidelberg: Springer-Verlag, 2005. Kap. 9, S. 109–122.

[GGA95] Christopher James Garrett, Sergio B Guarro und George E Apostolakis. „The dynamic flowgraph methodology for assessing the dependability of embedded software systems“. In: *Systems, Man and Cybernetics, IEEE Transactions on* 25.5 (1995), S. 824–840.

[Gib09] J Paul Gibson. „Weaving a formal methods education with problem-based learning“. In: *Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2009, S. 460–472.

[Gil97] Ronald J Gillespie. „The great ideas of chemistry“. In: *Journal of Chemical Education* 74.7 (1997), S. 862.

[GK83] Daniel Gajski und R.H. Kuhn. „Guest Editors' Introduction: New VLSI Tools“. In: *Computer* 16.12 (Dez. 1983), S. 11–14.

[Gli05] Martin Glinz. „Rethinking the notion of non-functional requirements“. In: *Proc. Third World Congress for Software Quality*. Bd. 2. 2005, S. 55–64.

[Gli07] Martin Glinz. „On non-functional requirements“. In: *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. IEEE. 2007, S. 21–26.

[GLV13] Marisol Garcia Valls, Iago Rodriguez López und L Fernández Villar. „iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems“. In: *Industrial Informatics, IEEE Transactions on* 9.1 (2013), S. 228–236.

[Gom+07] L. Gomes u. a. „From Petri net models to VHDL implementation of digital controllers“. In: *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*. Nov. 2007, S. 94–99.

[Guc10] Steven A Guccione. „Reconfigurable Computing Systems“. In: *The future of reconfigurable systems*. 2010.

[GV08] D.R. Garrison und N.D. Vaughan. *Blended Learning in Higher Education: Framework, Principles, and Guidelines*. The Jossey-Bass higher and adult education series. Wiley, 2008.

[Ha+07] Soonhoi Ha u. a. „PeaCE: A hardware-software codesign environment for multimedia embedded systems“. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 12.3 (2007), S. 24.

[Had13] Abraham Haddad. *Course on Probabilistic Systems and Random Signals, Northwestern University*. 2013. URL: `http://eecs.northwestern.edu/eecs-495-probabilistic-systems-and-random-signals` (besucht am 11. 11. 2014).

[Han+01] Uwe Hansmann u. a. *Pervasive computing handbook*. Bd. 3. Springer Heidelberg, 2001.

[Har+01] S. Hareland u. a. „Impact of CMOS process scaling and SOI on the soft error rates of logic processes“. In: *VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on*. Juni 2001, S. 73–74.

[Hau98] S. Hauck. „The roles of FPGAs in reprogrammable systems“. In: *Proceedings of the IEEE* 86.4 (Apr. 1998), S. 615–638.

[HB13] J. Henkel und L. Bauer. *Lehrveranstaltung Rekonfigurierbare und adaptive Systeme*. 2013. URL: `http://www.informatik.kit.edu/vorles.php?lv=24662` (besucht am 11. 11. 2014).

[HBH06] Tyson Hall, Jared Bruckner und Richard Halterman. „A Novel Approach to an Embedded Systems Curriculum“. In: *Education* 130.2 (2006), S. 184–194.

[HD10] Scott Hauck und Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation*. Morgan Kaufmann, 2010.

[Hei08] Gernot Heiser. „The Role of Virtualization in Embedded Systems“. In: *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*. IIES '08. Glasgow, Scotland: ACM, 2008, S. 11–16.

[Hen+14] Jörg Henkel u. a. „Multi-layer dependability: From microarchitecture to application level“. In: *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*. ACM. 2014, S. 1–6.

[Hen08] Thomas Henzinger. „Two challenges in embedded systems design: Predictability and robustness“. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366.1881 (2008), S. 3727–3736.

[Hen09] H. Henderson. *Encyclopedia of Computer Science and Technology*. Facts on File Science Library. Facts On File, Incorporated, 2009.

[Her07] Michael Herzceg. *Einführung in die Medieninformatik*. Oldenbourg Verlag, 2007.

[HKB05] Hochschulrektorenkonferenz, Kultusministerkonferenz und Bundesministerium für Bildung und Forschung. *Qualifikationsrahmen für Deutsche Hochschulabschlüsse*. Hrsg. von Kultusministerkonferenz. Nationaler Qualifikationsrahmen. 21. Apr. 2005.

[HKM01] A Hoffman, Tim Kogel und Heinrich Meyr. „A framework for fast hardware-software co-simulation“. In: *Proceedings of the conference on Design, automation and test in Europe*. IEEE Press. 2001, S. 760–765.

[HLR94] Nicolas Halbwachs, Fabienne Lagnier und Pascal Raymond. „Synchronous observers and the verification of reactive systems“. In: *Algebraic Methodology and Software Technology (AMAST 93)*. 1994.

[HM03] Kevin Hammond und Greg Michaelson. „Hume: a domain-specific language for real-time embedded systems“. In: *Generative Programming and Component Engineering*. Springer. 2003, S. 37–56.

[HN09] Lene Hansen und Helen Nissenbaum. „Digital Disaster, Cyber Security, and the Copenhagen School“. In: *International Studies Quarterly* 53.4 (2009), S. 1155–1175.

[HNM95] Carl Hein, David Nasoff und Lockheed Martin. „VHDL-based performance modeling and virtual prototyping“. In: *Proceedings 2nd Annual RASSP conference*. 1995, S. 87–94.

[HNR06] W. Hartmann, M. Näf und R. Reichert. *Informatikunterricht planen und durchführen*. EXamen. press Series. Springer, 2006.

[Hoc+11] Christian Hochberger u. a. *Curriculum Technische Informatik in Bachelor- und Masterstudiengängen Informatik*. Gesellschaft für Informatik e.V., März 2011.

[Hoc+13] Benny Hockner u. a. „Design space exploration for cyber physical system design using constraint solving“. In: *Specification Design Languages (FDL), 2013 Forum on*. Sep. 2013, S. 1–4.

[HS06] Thomas Henzinger und Joseph Sifakis. „The embedded systems design challenge“. In: *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. Springer. Aug. 2006.

[HS07] Thomas Henzinger und Joseph Sifakis. „The Discipline of Embedded Systems Design“. In: *Computer* 40.10 (2007), S. 32–40.

[HS13] Clyde Freeman Herreid und Nancy A Schiller. „Case studies and the flipped classroom“. In: *Journal of College Science Teaching* 42.5 (2013), S. 62–66.

[HS99] Ronald M Harden und N Stamper. „What is a spiral curriculum?“ In: *Medical Teacher* 21.2 (1999), S. 141–143.

[Hsu08] T.R. Hsu. *MEMS & Microsystems: Design, Manufacture, and Nanoscale Engineering*. Wiley, 2008.

[Hua+12] Jia Huang u. a. „Towards fault-tolerant embedded systems with imperfect fault detection“. In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. 2012, S. 188–196.

[Hua+97] Tsai Chi Huang u. a. „The teaching of VHDL in computer architecture“. In: *Microelectronics Systems Education, IEEE International Conference on*. IEEE Computer Society. 1997, S. 0133–0133.

[Hud96] Paul Hudak. „Building Domain-specific Embedded Languages“. In: *ACM Comput. Surv.* 28.4es (Dez. 1996).

[Hum06] L. Humbert. *Didaktik Der Informatik: Mit Praxiserprobtem Unterrichtsmaterial*. Leitfäden der Informatik. Teubner, 2006.

[IKF06] Roger Kreutz Immich, Diego Luis Kreutz und Antônio Augusto Fröhlich. „Resource Management for Embedded Systems“. In: *Factory Communication Systems, 2006 IEEE International Workshop on*. IEEE. 2006, S. 91–94.

[Inc06] Aberdeen Group Inc. *The Simulation Driven Design Benchmark Report*. Techn. Ber. Aberdeen Group Inc., 2006.

[Inc14] Digilent Inc. *VGA controller reference design - Spartan-3 Board*. 2014. URL: `https://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD` (besucht am 13. 11. 2014).

[Inf05] Gesellschaft für Informatik e.V. *Empfehlungen der Gesellschaft für Informatik e.V. für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen*. 2005.

[ISO08] ISO. *Systems and software engineering - System life cycle processes*. ISO 15288:2008. Geneva, Switzerland: International Organization for Standardization, 2008.

[Jan04] Axel Jantsch. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2004.

[Jas+11] Steffen Jaschke u. a. „Competence research: teaching embedded micro/nano systems“. In: *Proceedings of the 6th Workshop on Embedded Systems Education*. ACM. 2011, S. 17–24.

[Jas+12] Steffen Jaschke u. a. „Competence Oriented Embedded Systems Course for Computer Science Students“. In: *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*. WESE '12. Tampere, Finland: ACM, 2012, 6:1–6:7.

[JBS13] Steffen Jaschke, Steffen Büchner und Sigrid Schubert. „Explorative Learning and Visualization Environment for Teaching Embedded Systems in Higher Education“. In: *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*. 2013.

[JC05] David Jeff Jackson und Paul Caspi. „Embedded systems education: future directions, initiatives, and cooperation“. In: *ACM SIGBED Review* 2.4 (2005), S. 1–4.

[JCL11] Jeff C Jensen, Danica H Chang und Edward A Lee. „A model-based design methodology for cyber-physical systems“. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE. 2011, S. 1666–1671.

[JD93] Mu Der Jeng und F. DiCesare. „A review of synthesis techniques for Petri nets with applications to automated manufacturing systems“. In: *Systems, Man and Cybernetics, IEEE Transactions on* 23.1 (Jan. 1993), S. 301–312.

[JLT85] E Douglas Jensen, C Douglas Locke und Hideyuki Tokuda. „A Time-Driven Scheduling Model for Real-Time Operating Systems.“ In: *RTSS*. Bd. 85. 1985, S. 112–122.

[Jon11] Tim Jones. *Virtualization for embedded systems*. 2011. URL: `http://www.ibm.com/developerworks/library/l-embedded-virtualization/` (besucht am 11. 11. 2014).

[JS05] Axel Jantsch und Ingo Sander. „Models of computation and languages for embedded system design“. In: *Computers and Digital Techniques, IEE Proceedings-*. Bd. 152. 2. IET. 2005, S. 114–129.

[JW05] Ahmed A JERRAYA und Wayne WOLF. „Hardware/software interface codesign for embedded systems“. In: *Computer* 38.2 (2005), S. 63–69.

[Kai93] Hermann Kaindl. „The Missing Link in Requirements Engineering“. In: *SIGSOFT Softw. Eng. Notes* 18.2 (Apr. 1993), S. 30–39.

[KB09] Frank R. Kesel und Ruben Bartholoma. *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs : Einführung mit VHDL und SystemC*. Oldenbourg, 2009.

[Kel76] Robert M Keller. „Formal verification of parallel programs“. In: *Communications of the ACM* 19.7 (1976), S. 371–384.

[KG82] SY Kung und RJ Gal-Ezer. „Synchronous versus asynchronous computation in very large scale integrated (VLSI) array processors“. In: *1982 Technical Symposium East*. International Society for Optics und Photonics. 1982, S. 53–65.

[Kim+08] Jong-Kook Kim u. a. „Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling“. In: *Parallel and Distributed Systems, IEEE Transactions on* 19.11 (2008), S. 1445–1457.

[Kin93] Alison King. „From sage on the stage to guide on the side“. In: *College teaching* 41.1 (1993), S. 30–35.

[KJS11] Eunsuk Kang, Ethan Jackson und Wolfram Schulte. „An approach for effective design space exploration“. In: *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*. Springer, 2011, S. 33–54.

[Kle+12] Bruno Kleinert u. a. „Approaches to teaching future computation technologies and nanosystems“. In: *Proceedings of IFIP-Conference. Addressing educational challenges: the role of ICT* (Juli 2012).

[Kli+04] Eckhard Klieme u. a. „Was sind Kompetenzen und wie lassen sie sich messen“. In: *Pädagogik* 56.6 (2004), S. 10–13.

[Koc+04] Paul Kocher u. a. „Security as a new dimension in embedded system design“. In: *Proceedings of the 41st annual Design Automation Conference*. ACM. 2004, S. 753–760.

[Koo07] Philip Koopman. „Reliability, safety, and security in everyday embedded systems“. In: *Dependable Computing*. Springer, 2007, S. 1–2.

[Koo96] Philip Koopman. „Embedded System Design Issues (The Rest of the Story).“ In: *ICCD*. IEEE Computer Society, 1996, S. 310–.

[Kop+89] Hermann Kopetz u. a. „Distributed fault-tolerant real-time systems: The Mars approach“. In: *Micro, IEEE* 9.1 (1989), S. 25–40.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2nd. New York, USA: Springer Science+ Business Media, 2011.

[Kop98] H Kopetz. „Component-based design of large distributed real-time systems“. In: *Control Engineering Practice* 6.1 (1998), S. 53–60.

[Kra02] David R Krathwohl. „A revision of Bloom's taxonomy: An overview“. In: *Theory into practice* 41.4 (2002), S. 212–218.

[Kum+02] S. Kumar u. a. „A network on chip architecture and design methodology“. In: *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*. 2002, S. 105–112.

[Lap85] Jean-Claude Laprie. „Dependable computing and fault-tolerance“. In: *Digest of Papers FTCS-15* (1985), S. 2–11.

[Lee06] Edward A. Lee. *Cyber-Physical Systems - Are Computing Foundations Adequate?* Techn. Ber. Austin, TX: UC Berkeley EECS Dept., Okt. 2006.

[Lee08] Edward A Lee. „Cyber physical systems: Design challenges“. In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE. 2008, S. 363–369.

[Lev91] Nancy G. Leveson. „Software Safety in Embedded Computer Systems“. In: *Commun. ACM* 34.2 (Feb. 1991), S. 34–46.

[LG02] Hong Liu und David P. Gluch. „A Proposal for Introducing Model Checking into an Undergraduate Software Engineering Curriculum“. In: *J. Comput. Sci. Coll.* 18.2 (Dez. 2002), S. 259–270.

[Lie+06] Henry Lieberman u. a. *End-user development: An emerging paradigm*. Springer, 2006.

[Lif14] LifeWiki. *The Wiki for Conways's Game of Life*. 2014. URL: `http://www.conwaylife.com/wiki/Main_Page` (besucht am 11. 11. 2014).

[LL09] Hollylynne S. Lee und J. Todd Lee. „Reasoning about Probabilistic Phenomena: Lessons Learned and Applied in Software Design.“ In: *Technology Innovations in Statistics Education*. 2009.

[LLS07] I. Lee, J.Y.T. Leung und S.H. Son. *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC Computer and Information Science Series. Taylor & Francis, 2007.

[Loc+07] John W Lockwood u. a. „NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing“. In: *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*. IEEE. 2007, S. 160–161.

[Loh11] HMF Lohstroh. „Theory and Application of Multithreading The Actor Model“. In: (2011).

[LPS01] Bev Littlewood, Peter Popov und Lorenzo Strigini. „Modeling software design diversity: a review“. In: *ACM Computing Surveys (CSUR)* 33.2 (2001), S. 177–208.

[LS03] Greg Linden und Deepak Somaya. „System-on-a-chip integration in the semiconductor industry: industry structure and firm strategies“. In: *Industrial and Corporate Change* 12.3 (2003), S. 545–576.

[LS11a] Edward A Lee und Alberto Sangiovanni-Vincentelli. „Component-based design for the future“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE. 2011, S. 1–5.

[LS11b] Edward A Lee und Sanjit A Seshia. *Introduction to embedded systems*. First. UC Berkeley, 2011.

[LS95] Dieter Landes und Rudi Studer. *The treatment of non-functional requirements in MIKE*. Springer, 1995.

[LSS99] Luciano Lavagno, Alberto Sangiovanni-Vincentelli und Ellen Sentovich. „Models of computation for embedded system design“. In: *System-Level Synthesis*. Springer, 1999, S. 45–102.

[LT09] P. Liggesmeyer und M. Trapp. „Trends in Embedded Software Engineering“. In: *Software, IEEE* 26.3 (Mai 2009), S. 19–25.

[Luk+14] Marcin Lukowiak u. a. „Cybersecurity Education: Bridging the Gap Between Hardware and Software Domains“. In: *Trans. Comput. Educ.* 14.1 (März 2014), 2:1–2:20.

[Lyu+96] Michael R Lyu u. a. *Handbook of software reliability engineering*. Bd. 222. IEEE computer society press CA, 1996.

[Mad+12] A. Madanayake u. a. „Teaching freshmen VHDL-based digital design“. In: *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. Mai 2012, S. 2701–2704.

[Mad12] Azad M. Madni. „Adaptable platform-based engineering: Key enablers and outlook for the future“. In: *Systems Engineering* 15.1 (2012), S. 95–107.

[Man13] Peter Mandl. „Betriebssystemvirtualisierung“. In: *Grundkurs Betriebssysteme*. Springer, 2013, S. 289–310.

[Mar11] Peter Marwedel. *Embedded Systems Design - Embedded Systems Foundations of Cyber-Physical Systems*. 2nd Edition. ISBN 978-94-007-0256-1. Springer, 2011.

[MC06] Alice Miller und Quintin Cutts. „The Use of an Electronic Voting System in a Formal Methods course“. In: *Formal Methods in the Teaching Lab*. Formal Methods Europe Subgroup on Education. 2006, S. 3–8.

[McC97] Robert McCormick. „Conceptual and procedural knowledge“. In: *International journal of technology and design education* 7.1-2 (1997), S. 141–159.

[ME14] P. Marwedel und M. Engel. „Flipped classroom teaching for a cyber-physical system course - An adequate presence-based learning approach in the internet age“. In: *Microelectronics Education (EWME), 10th European Workshop on*. Mai 2014, S. 11–15.

[ME95] Vijay K Madisetti und Thomas W Egolf. „Virtual prototyping of embedded microcontroller-based DSP systems“. In: *Micro, IEEE* 15.5 (1995), S. 9–21.

[MGA94] C.T. Muthukumar, SergioB. Guarro und GeorgeE. Apostolakis. „Dependability Analysis of Embedded Software Systems“. English. In: *Reliability and Safety Assessment of Dynamic Process Systems*. Hrsg. von Tunc Aldemir u. a. Bd. 120. NATO ASI Series. Springer Berlin Heidelberg, 1994, S. 59–77.

[MKB03] Cathleen McGrath, David Krackhardt und Jim Blythe. „Visualizing complexity in networks: Seeing both the forest and the trees“. In: *Connections* 25.1 (2003), S. 37–47.

[ML03] Jan Meyer und Ray Land. *Threshold concepts and troublesome knowledge: linkages to ways of thinking and practising within the disciplines*. University of Edinburgh, 2003.

[ML05] Jan H.F. Meyer und Ray Land. „Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning“. English. In: *Higher Education* 49.3 (2005), S. 373–388.

[MM95] Linda Macaulay und John Mylopoulos. „Requirements Engineering: an educational dilemma“. In: *Automated Software Engineering* 2.4 (1995), S. 343–351.

[Mod03] Eckart Modrow. „Fundamentale Ideen der theoretischen Informatik.“ In: *INFOS*. 2003, S. 189–200.

[Mod06] Eckart Modrow. „Zur Ordnungswirkung fundamentaler Ideen der Informatik am Beispiel der theoretischen Schulinformatik“. In: *Informatica Didactica* 6 (2006).

[MP14] Daniel Manson und Ronald Pike. „The case for depth in cybersecurity education“. In: *ACM Inroads* 5.1 (2014), S. 47–52.

[MSC13] G.S. Mason, T.R. Shuman und K.E. Cook. „Comparing the Effectiveness of an Inverted Classroom to a Traditional Classroom in an Upper-Division Engineering Course“. In: *Education, IEEE Transactions on* 56.4 (Nov. 2013), S. 430–435.

[Müh+11] Nina Mühleis u. a. „A Co-simulation Approach for Control Performance Analysis During Design Space Exploration of Cyber-physical Systems“. In: *SIGBED Rev.* 8.2 (Juni 2011), S. 23–26.

[Mur89] T. Murata. „Petri nets: Properties, analysis and applications“. In: *Proceedings of the IEEE* 77.4 (Apr. 1989), S. 541–580.

[MYT93] K. Mori, H. Yamada und S. Takizawa. „System on Chip Age“. In: *VLSI Technology, Systems, and Applications, 1993. Proceedings of Technical Papers. 1993 International Symposium on*. 1993, K15–K20.

[NAI04] V.P Nelson, ACM und IEEE. „Computer Engineering 2004“. In: 0229748 (2004).

[Neu93] John von Neumann. „First Draft of a Report on the EDVAC“. In: *IEEE Ann. Hist. Comput.* 15.4 (Okt. 1993), S. 27–75.

[NF02] Keith V Nesbitt und Carsten Friedrich. „Applying gestalt principles to animated visualizations of network data“. In: *Information Visualisation, 2002. Proceedings. Sixth International Conference on*. IEEE. 2002, S. 737–743.

[Nie90] Jürg Nievergelt. „Computer Science for Teachers: A Quest for Classics and How to Present Them“. In: *Proceedings of the 3rd International Conference on Computer Assisted Learning*. ICCAL '90. London, UK, UK: Springer-Verlag, 1990, S. 2–15.

[Nie98] R. Niemann. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems*. Springer, 1998.

[NS92] Geoffrey R Norman und Henk G Schmidt. „The psychological basis of problem-based learning: a review of the evidence“. In: *Academic medicine* 67.9 (1992), S. 557–65.

[NS95] Klara Nahrstedt und Ralf Steinmetz. „Resource management in networked multimedia systems“. In: *IEEE Computer* 28.5 (1995), S. 52–63.

[Nur03] Paul Nurse. „The great ideas of biology“. In: *Clinical medicine* 3.6 (2003), S. 560–568.

[NX06] V. Narayanan und Yuan Xie. „Reliability concerns in embedded system designs“. In: *Computer* 39.1 (Jan. 2006), S. 118–120.

[Ost92] Jonathan S Ostroff. „Formal methods for the specification and design of real-time safety critical systems“. In: *Journal of Systems and Software* 18.1 (1992), S. 33–60.

[Pal+09] Krishna V. Palem u. a. „Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects.“ In: *CASES*. Hrsg. von Jörg Henkel und Sri Parameswaran. ACM, 26. Okt. 2009, S. 1–10.

[Pal+13] Herbert Palm u. a. „A Novel Approach on Virtual Systems Prototyping Based on a Validated, Hierarchical, Modular Library“. In: *Embedded World, Nuremberg, Germany* (2013).

[PBC95] F. Pichon, S. Blanc und B. Candaele. „Mixed-signal Modelling in VHDL for System-on-chip Applications“. In: *Proceedings of the 1995 European Conference on Design and Test*. EDTC '95. Washington, DC, USA: IEEE Computer Society, 1995, S. 218–222.

[Ped+12] Sol Pedre u. a. „A co-design methodology for processor-centric embedded systems with hardware acceleration using FPGA“. In: *Programmable Logic (SPL), 2012 VIII Southern Conference on*. IEEE. 2012, S. 1–8.

[PEP06] Andy D Pimentel, Cagkan Erbas und Simon Polstra. „A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels“. In: *IEEE Trans. Comput.* 55.2 (2006), S. 99–112.

[Pim+08] Andy D Pimentel u. a. „Calibration of abstract performance models for system-level design space exploration“. In: *Journal of Signal Processing Systems* 50.2 (2008), S. 99–114.

[Por13] Dr. Ing. Mario Porrmann. *Modul Rekonfigurierbare und parallele Rechnersysteme*. 2013. URL: `https://ekvv.uni-bielefeld.de/sinfo/publ/modul/33005813` (besucht am 11. 11. 2014).

[PV09] Arno Pasternak und Jan Vahrenhold. „Rote Fäden und Kontextorientierung im Informatikunterricht.“ In: *INFOS*. 2009, S. 280.

[Raj+10] Ragunathan (Raj) Rajkumar u. a. „Cyber-physical systems: the next computing revolution“. In: *Proceedings of the 47th Design Automation Conference*. DAC '10. Anaheim, California: ACM, 2010, S. 731–736.

[Ran94] E. Randon. „Basic concepts of HDLs: VHDL“. In: *Industrial Electronics, 1994. Symposium Proceedings, ISIE '94., 1994 IEEE International Symposium on*. Mai 1994, S. 42–46.

[RB13] Wolfgang Rosenstiel und Oliver Bringmann. *Forschungsbericht 2011-2013*. Techn. Ber. Systementwurf in der Mikroelektronik -FZI und Technische Informatik/Eingebettete Systeme Universität Tübingen, 2013.

[RCT07] Jorge G Ruiz, Chris Candler und Thomas A Teasdale. „Peer reviewing e-learning: opportunities, challenges, and solutions“. In: *Academic Medicine* 82.5 (2007), S. 503–507.

[RE92] M. Rupprecht und C. Engel. „Concurrent execution of communication protocols in high speed networks“. In: *Communications, 1992. ICC '92, Conference record, SUPERCOMM/ICC '92, Discovering a New World of Communications., IEEE International Conference on*. Juni 1992, 164–168 vol.1.

[RJS08] Kenneth G. Ricks, David Jeff Jackson und William A. Stapleton. „An Embedded Systems Curriculum Based on the IEEE/ACM Model Curriculum.“ In: *IEEE Trans. Education* 51.2 (2008), S. 262–270.

[Rom85] G. Roman. „A taxonomy of current issues in requirements engineering“. In: *Computer* 18.4 (Apr. 1985), S. 14–23.

[Rov+08] D.T. Rover u. a. „Reflections on Teaching and Learning in an Advanced Undergraduate Course in Embedded Systems“. In: *Education, IEEE Transactions on* 51.3 (Aug. 2008), S. 400–412.

[RR09] Janet Rountree und Nathan Rountree. „Issues regarding threshold concepts in computer science“. In: *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*. Australian Computer Society, Inc. 2009, S. 139–146.

[RR13] R. Rodriguez-Ponce und J. Rodriguez-Resendiz. „Integrating VHDL into an undergraduate digital design course“. In: *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*. Aug. 2013, S. 172–177.

[RRR03] Anthony Robins, Janet Rountree und Nathan Rountree. „Learning and teaching programming: A review and discussion“. In: *Computer Science Education* 13.2 (2003), S. 137–172.

[RS96] Horst Friedrich Rösler und Heinz Schmidkunz. „Die didaktische Reduktion - eine Bestandsaufnahme“. In: *NiU-Chemie 7* 34 (1996).

[Rum08] Bernhard Rumpler. „Design Comprehension of Embedded Real-Time Systems“. 2008.

[Rus81] John M Rushby. „Design and verification of secure systems“. In: *ACM SIGOPS Operating Systems Review*. Bd. 15. 5. ACM. 1981, S. 12–21.

[Rus93] John Rushby. *Formal methods and the certification of critical systems*. Citeseer, 1993.

[Sal+06] R. Saleh u. a. „System-on-Chip: Reuse and Integration“. In: *Proceedings of the IEEE* 94.6 (Juni 2006), S. 1050–1069.

[San+04] Alberto Sangiovanni-Vincentelli u. a. „Benefits and challenges for platform-based design“. In: *Proceedings of the 41st annual Design Automation Conference*. DAC '04. San Diego, CA, USA: ACM, 2004, S. 409–414.

[Sav97] John E. Savage. *Models of Computation: Exploring the Power of Computing*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[SB95] Mani B Srivastava und Robert W Brodersen. „SIERA: A unified framework for rapid-prototyping of system-level hardware and software“. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 14.6 (1995), S. 676–693.

[SBF10] Sigrid Schubert, Rainer Brück und Dietmar Fey. *Projektantrag - Kompetenzentwicklung mit eingebetteten Mikro- und Nanosystemen*. Techn. Ber. 2010.

[Sch+02] Bernhard Schätz u. a. „Model-based development of embedded systems“. In: *Advances in Object-Oriented Information Systems*. Springer, 2002, S. 298–311.

[Sch+12a] Andre Schäfer u. a. „Competence model for embedded micro- and nanosystems“. In: *Global Engineering Education Conference (EDUCON), 2012 IEEE*. Apr. 2012, S. 1–6.

[Sch+12b] Andre Schäfer u. a. „Competence model for embedded micro-and nanosystems“. In: *Global Engineering Education Conference (EDUCON), 2012 IEEE*. IEEE. 2012, S. 1–6.

[Sch+12c] Andre Schäfer u. a. „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*. New York, NY, USA, 2012.

[Sch08] Patrick Schaumont. „Hardware/software co-design is a starting point in embedded systems architecture education“. In: *Proc. WESE* (2008).

[Sch10] Patrick Schaumont. *A Practical Introduction to Hardware/Software Codesign*. Springer, 2010, S. I–XVIII, 1–396.

[Sch93] Andreas Schwill. „Fundamentale Ideen der Informatik“. In: *Zentralblatt für Didaktik der Mathematik* 25.1 (1993), S. 20–31.

[SCS87] G. Saucier, M. Crastes de Paulet und P. Sicard. „ASYL: A Rule-Based System for Controller Synthesis“. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 6.6 (Nov. 1987), S. 1088–1097.

[SDP12a] Alberto Sangiovanni-Vincentelli, Werner Damm und Roberto Passerone. „Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems.“ In: *Eur. J. Control* 18.3 (2012), S. 217–238.

[SDP12b] Alberto Sangiovanni-Vincentelli, Werner Damm und Roberto Passerone. „Taming dr. frankenstein: Contract-based design for cyber-physical systems*“. In: *European journal of control* 18.3 (2012), S. 217–238.

[SFC85] Connie U. Smith, Geoffrey A. Frank und John L. Cuadrado. „An architecture design and assessment system for software/hardware codesign“. In: *Proceedings of the 22nd ACM/IEEE Design Automation Conference*. DAC ’85. Las Vegas, Nevada, USA: IEEE Press, 1985, S. 417–424.

[Sha07] John Shalf. „The new landscape of parallel computer architecture“. In: *Journal of Physics: Conference Series*. Bd. 78. 1. IOP Publishing. 2007, S. 012066.

[Sif11] Joseph Sifakis. „A vision for computer science - the system perspective“. In: *Central European Journal of Computer Science* 1.1 (März 2011), S. 108–116.

[Sig+08] Kamana Sigdel u. a. *System-level design space exploration of reconfigurable architectures*. Springer-Verlag Berlin Heidelberg, 2008.

[Sim+06] Timothy W Simpson u. a. „Platform-based design and development: current trends and needs in industry“. In: *ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers. 2006, S. 801–810.

[Sin+01] Rajarishi Sinha u. a. „Modeling and simulation methods for design of engineering systems“. In: *Journal of Computing and Information Science in Engineering* 1.1 (2001), S. 84–91.

[SJ95] Santhanam Srinivasan und Niraj K Jha. „Hardware-software co-synthesis of fault-tolerant real-time distributed embedded systems“. In: *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC’95., European*. IEEE. 1995, S. 334–339.

[SL94] Jeffrey L Stein und Loucas S Louca. „A component-based modeling approach for system design: Theory and implementation“. In: *Ann Arbor* 1001 (1994), S. 48109–2125.

[SLS00] Marco Sgroi, Luciano Lavagno und Alberto Sangiovanni-Vincentelli. „Formal models for embedded system design“. In: *Design & Test of Computers, IEEE* 17.2 (2000), S. 14–27.

[SM01] Alberto Sangiovanni-Vincentelli und Grant Martin. „Platform-Based Design and Software Design Methodology for Embedded Systems“. In: *IEEE Des. Test* 18.6 (Nov. 2001), S. 23–33.

[Smi08] Michael John Sebastian Smith. *Application-Specific Integrated Circuits*. 1st. Addison-Wesley Professional, 2008.

[SPF07] Paul F Smith, Sameer M Prabhu und Jonathon Friedman. *Best practices for establishing a model-based design culture*. Techn. Ber. SAE Technical Paper, 2007.

[Spi90] J. Michael Spivey. „Specifying a Real-Time Kernel.“ In: *IEEE Software* 7.5 (1990), S. 21–28.

[SQJ11] John Shalf, Dan Quinlan und Curtis Janssen. „Rethinking hardware-software codesign for exascale systems“. In: *Computer* 44.11 (2011), S. 22–30.

[SR13] Mehran Sahami und Steve Roach. *Computer Science Curricula 2013*. Techn. Ber. The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society, 2013.

[SS05] Valery Sklyarov und Iouliia Skliarova. „Teaching reconfigurable systems: methods, tools, tutorials, and projects“. In: *Education, IEEE Transactions on* 48.2 (2005), S. 290–300.

[SS11] Sigrid Schubert und Andreas Schwill. *Didaktik Der Informatik*. Spektrum Akademischer Verlag GmbH, 2011.

[ST08] Guido Sandmann und Richard Thompson. *Development of AUTOSAR software components within model-based design*. Techn. Ber. SAE Technical Paper, 2008.

[ST10] National Institute of Standards und Technology. *National Initiative for Cybersecurity Education*. 2010. URL: `http://csrc.nist.gov/nice/` (besucht am 11. 11. 2014).

[Sta14] International Organization for Standardization. *Standards Catalogue*. 2014. URL: `http://www.iso.org/iso/home/store/catalogue_ics.htm` (besucht am 11. 11. 2014).

[Suh98] Nam P Suh. „Axiomatic design theory for systems“. In: *Research in engineering design* 10.4 (1998), S. 189–209.

[SWP04] C. Steiger, H. Walder und M. Platzner. „Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks“. In: *Computers, IEEE Transactions on* 53.11 (Nov. 2004), S. 1393–1407.

[Szt+05] Janos Sztipanovits u. a. „Introducing Embedded Software and Systems Education and Advanced Learning Technology in an Engineering Curriculum“. In: *ACM Trans. Embed. Comput. Syst.* 4.3 (Aug. 2005), S. 549–568.

[Szy02] Clemens Szyperski. *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[Tan92] Andrew Tanenbaum. *Modern operating systems*. Bd. 2. Prentice hall Englewood Cliffs, 1992.

[TB01] Russell Tessier und Wayne Burleson. „Reconfigurable computing for digital signal processing: A survey“. In: *Journal of VLSI signal processing systems for signal, image and video technology* 28.1-2 (2001), S. 7–27.

[TC91] The International Telegraph und Telephone Consultative Comittee (CCITT). *Security Architectures for Open Systems Interconnection for CCITT Applications - Recommendation X.800*. Techn. Ber. International telecommunication Union (ITU), 1991.

[Tec13] UBM Tech. *2013 Embedded Market Study*. 2013. URL: `http://e.ubmelectronics.com/2013EmbeddedStudy/` (besucht am 19. 11. 2014).

[Tei12] Jürgen Teich. „Hardware/Software Codesign: The Past, the Present, and Predicting the Future“. In: *Proceedings of the IEEE* 100.Centennial-Issue (2012), S. 1411–1430.

[TMW94] V. Tiwari, S. Malik und A. Wolfe. „Power analysis of embedded software: a first step towards software power minimization“. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 2.4 (Dez. 1994), S. 437–445.

[Tod+05] Timothy J Todman u. a. „Reconfigurable computing: architectures and design methods“. In: *IEE Proceedings-Computers and Digital Techniques* 152.2 (2005), S. 193–207.

[TPB14] David H. Tobey, Portia Pusey und Diana L. Burley. „Engaging Learners in Cybersecurity Careers: Lessons from the Launch of the National Cyber League“. In: *ACM Inroads* 5.1 (März 2014), S. 53–56.

[Vaa06] FW Vaandrager. „Does it pay off? model-based verification and validation of embedded systems“. In: *PROGRESS White papers* (2006), S. 43–66.

[Vog88] Udo Voges. *Software diversity in computerized control systems*. Springer, 1988.

[Vol78] Hans-Joachim Vollrath. „Rettet die Ideen!“ In: *Der Mathematische Naturwissenschaftliche Unterricht* 31 (1978), S. 449–455.

[VWW02] Monika Vetterling, Guido Wimmel und Alexander Wisspeintner. „Secure systems development based on the common criteria: the PalME project“. In: *ACM SIGSOFT Software Engineering Notes* 27.6 (2002), S. 129–138.

[Wal14] Prof. Dr. Klaus Waldschmidt. *Vorlesung Rechnertechnologie, Universität Frankfurt*. 2014. URL: `http://www.ti.cs.uni-frankfurt.de/lehre/ss14/rechnertechnologie/` (besucht am 11. 11. 2014).

[Wan02] G Gary Wang. „Definition and review of virtual prototyping“. In: *Journal of Computing and Information Science in engineering* 2.3 (2002), S. 232–236.

[Wan11] Xiaofang Wang. „Using FPGA-based configurable processors in teaching hardware/software co-design of embedded multiprocessor systems“. In: *Microelectronic Systems Education (MSE), 2011 IEEE International Conference on*. IEEE. 2011, S. 114–117.

[Wed13] Joachim Wedekind. „MOOCS–eine Herausforderung für die Hochschulen“. In: *Hochschuldidaktik im Zeichen von Heterogenität und Vielfalt, Hrsg: Reinmann, G. ua* (2013), S. 45–60.

[Wei11] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann, 2011.

[Wei91] Mark Weiser. „The computer for the 21st century“. In: *Scientific American* 265.3 (1991), S. 94–104.

[Win00] Jeannette M Wing. „Invited talk: Weaving formal methods into the undergraduate computer science curriculum“. In: *Algebraic Methodology and Software Technology*. Springer, 2000, S. 2–7.

[WKA03] R.D. Williams, R.H. Klenke und J.H. Aylor. „Teaching computer design using virtual prototyping“. In: *Education, IEEE Transactions on* 46.2 (Mai 2003), S. 296–301.

[WM00] W. Wolf und J. Madsen. „Embedded systems education for the future“. In: *Proceedings of the IEEE* 88.1 (2000), S. 23–30.

[Wol03] Wayne Wolf. „A Decade of Hardware/Software Codesign“. In: *Computer* 36.4 (Apr. 2003), S. 38–43.

[Wol94] Wayne H Wolf. „Hardware-software co-design of embedded systems [and prolog]“. In: *Proceedings of the IEEE* 82.7 (1994), S. 967–989.

[WRS95] F. Wang, K. Ramamritham und J.A. Stankovic. „Determining redundancy levels for fault tolerant real-time systems“. In: *Computers, IEEE Transactions on* 44.2 (Feb. 1995), S. 292–301.

[Wym93] A Wayne Wymore. *Model-based systems engineering*. Bd. 3. CRC press, 1993.

[Xil14] Xilinx. *FPGA vs. ASIC Design Flow Comparison*. 2014. URL: `http://www.xilinx.com/fpga/asic.htm` (besucht am 11. 11. 2014).

[Zah03] B. Zahiri. „Structured ASICs: opportunities and challenges“. In: *Computer Design, 2003. Proceedings. 21st International Conference on*. Okt. 2003, S. 404–409.

[Zav82] P Zave. „An Operational Approach to Requirements Specification for Embedded Systems“. In: *IEEE Trans. Softw. Eng.* 8.3 (1982), S. 250–269.

[Zha+13a] Liyuan Zhang u. a. „Bridging algorithm and ESL design: Matlab/-Simulink model transformation and validation“. In: *Specification & Design Languages (FDL), 2013 Forum on*. IEEE. 2013, S. 1–8.

[Zha+13b] Xibin Zhao u. a. „An Effective Heuristic-Based Approach for Partitioning“. In: *Journal of Applied Mathematics* 2013 (2013).

[Zha03] G.Q. Zhang. „The challenges of virtual prototyping and qualification for future microelectronics“. In: *Microelectronics Reliability*. Hrsg. von A. Touboul N. Labat. Bd. 43. European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. Bordeaux, France, Okt. 2003, S. 1351–1968.

[Zhu06] Dakai Zhu. „Reliability-aware dynamic energy management in dependable embedded real-time systems“. In: *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*. IEEE. 2006, S. 397–407.

[Zip13] Prof. Dr.-Ing. Peter Zipf. *Vorlesung Rekonfigurierbare Strukturen*. 2013. URL: `http://www.uni-kassel.de/eecs/fachgebiete/digitech/lehre/master-bereich/rekonfigurierbare-strukturen.html` (besucht am 11. 11. 2014).

[ZS06] Andreas Zendler und Christian Spannagel. „Zentrale Konzepte im Informatikunterricht: eine empirische Grundlegung“. In: *Notes on Educational Informatics Section A: Concepts and Techniques* 2.1 (2006), S. 1–21.

[ZS10] Valentina Zadrija und Vlado Sruk. „Component-based specification for Multi-Processor System-on-Chip design“. In: *MELECON 2010-2010 15th IEEE Mediterranean Electrotechnical Conference*. IEEE. 2010, S. 1044–1049.

[ZSK07] Andreas Zendler, Christian Spannagel und Dieter Klaudt. „Zentrale Prozesse im Informatikunterricht: eine empirische Grundlegung“. In: *Notes on Educational Informatics Section A: Concepts and Techniques* 3.1 (2007), S. 1–19.

[ZSK11] A. Zendler, C. Spannagel und D. Klaudt. „Marrying Content and Process in Computer Science Education“. In: *Education, IEEE Transactions on* 54.3 (2011), S. 387–397.

Eingebettete Systeme stellen aufgrund ihrer gesellschaftlichen und industriellen Bedeutung ein wachsendes Anwendungsfeld mit besonders kurzlebigen Technologiezyklen dar. Diese Kurzlebigkeit erschwert die Auswahl von Kompetenzen, die für Studierende mit unterschiedlichen fachlichen Schwerpunkten ein langfristig relevantes Fundament bilden.

Der wesentliche Beitrag der Arbeit besteht in der Weiterentwicklung und theoretischen Fundierung eines Forschungsvorgehens, zur Identifikation zentraler Fachelemente und deren Organisation in eine Veranstaltungs- und Curriculumsstruktur. Die aufgedeckten mehrschichtigen Beziehungen zwischen den identifizierten Methoden und Sichtweisen geben zudem eine anwendungsgebietszentrierte Struktur der Disziplin wider, die als Grundlage zukünftiger Forschung im Fachgebiet genutzt werden kann.

Steffen Büchner studierte nach seiner Ausbildung zum staatlich geprüften Assistenten für Technische Informatik ab 2005 Angewandte Informatik an der Universität Siegen. Ab 2011 arbeitete er als wissenschaftlicher Mitarbeiter im Departement Elektrotechnik und Informatik am Lehrstuhl für Didaktik der Informatik und E-Learning
Seine Promotion schloss er 2015 mit seiner Dissertation zur Ermittlung zentraler Konzepte der Entwicklung eingebetteter Systeme ab.

ISBN 978 -3-936533-58-3

universi
UNIVERSITÄTSVERLAG SIEGEN