UML in der Hochschullehre: Eine kritische Reflexion

Jürgen Anke, Hochschule für Telekommunikation Leipzig

Stefan Bente, Technische Hochschule Köln

anke@hft-leipzig.de, stefan.bente@th-koeln.de

Zusammenfassung

Trotz ihrer hohen Bekanntheit und Verbreitung ist die Unified Modelling Language (UML) 20 Jahre nach der Vorstellung ihrer ersten Version in der Softwaretechnik nicht unumstritten. Ein wesentlicher Grund ist ihre unklare Nutzung in der Praxis die u.a. durch Ausdifferenzierung von Softwareproduktarten, agilen Entwicklungsansätzen sowie Microservices als Architekturstil mit reduzierter Komplexität beeinflusst wird. Diese Trends befördern eine Abkehr von klassischen Top-Down-Ansätzen mit sequenziellem Vorgehen, das umfangreiche Spezifikationen der Anforderungen und des Entwurfs erfordert. Für die Lehre (insbesondere auf Bachelorniveau) stellt sich daher die Frage, ob wir UML überhaupt noch brauchen und wenn ja, wofür und in welchem Umfang. Ziel dieses Beitrags ist es, empirische Belege zur Relevanz der UML zusammenzutragen und daraus Schlussfolgerungen für mögliche Kompetenzziele in der Softwaretechnikausbildung zu ziehen. Unser Ziel ist es, eine aktuelle Sicht auf die praktische Relevanz der UML zu liefern und damit die Diskussion über die Ausgestaltung der Lehre im Fach Softwaretechnik anzuregen.

Abstract

Despite its pervasiveness and popularity, the Unified Modeling Language (UML) is not uncontroversial in software engineering 20 years after the launch of its first version. A major reason is their unclear use in practice, which is influenced by the differentiation of software product types, agile development approaches and microservices as architectural style with reduced complexity. These trends move away from traditional top-down, sequential approaches that require extensive specifications of requirements and design. For teaching (especially at the Bachelor level), therefore, the question arises whether we still need UML at all and if so, for what and to what extent. The aim of this paper is to gather empirical evidence on the relevance of UML and to draw conclusions for possible competence goals in software engineering education. Our goal is to provide an up-to-date view of the practical relevance of UML to stimulate discussion about the role of UML in software engineering courses.

Einleitung und Motivation

Die Beschreibung bestehender oder geplanter Systeme ist eine wichtige Teilaufgabe im Software Engineering. Dies wird vielfach mit grafischen Modellen der Unified Modelling Language (UML) durchgeführt. Die UML ist der De-facto-Standard die grafische Modellierung von objektorientierten Systemen, um deren Analyse, Entwurf und Dokumentation zu unterstützten (Rupp, Queins, & SOPHISTen, 2012; Sommerville, 2015).

Folgerichtig ist das Erlernen der UML ein wichtiger Bestandteil in vielen Software Engineering Modulen an Hochschulen, wie es auch in den Rahmenempfehlungen der Gesellschaft für Informatik (GI) verankert ist (Gesellschaft für Informatik, 2016). Jedoch ist die UML auch in der Lehre aus unserer Erfahrung nicht unumstritten.

Aus unserer Sicht sind mögliche Gründe dafür z.B.:

- Die Relevanz der UML in der Praxis scheint zurückzugehen, da umfangreiche Spezifikationen in agilen Entwicklungsansätzen wie z.B. Scrum nicht mehr erforderlich sind.
- Aufgrund ihrer Komplexität ist die UML schwer zu erlernen (Erickson & Siau, 2007; Seiko Akayama u. a., 2013).
- Besondere Schwierigkeiten verursacht die Verbindung verschiedener Sichten auf das Softwaresystem (Boberić-Krstićev u. a., 2011).
- Klassische UML-Tools führen aufgrund ihrer Komplexität in der Lehre oft zu Herausforderungen (Liebel, Heldal, & Steghofer, 2016; Seiko Akayama u. a., 2013).
- Der Nutzen der Modellierung erschließt sich Studierenden nicht (Boberić-Krstićev u. a., 2011; Burgueño, Vallecillo, & Gogolla, 2018).

Der Eindruck einer sinkenden Bedeutung von UML wird auch anhand der Google Trends deutlich, welche die relative Häufigkeit von Suchanfragen betrachtet (Abb. 1). Hierbei wurde der Zeitraum 2004-2018 sowie die Region Deutschland gewählt. Zudem wurde mit „Themen“ statt Suchbegriffen gearbeitet, die verschiedene Formulierungen und Schreibweisen berücksichtigen.

Vor diesem Hintergrund stellt sich die Frage, ob und wie die UML in der grundständigen Software Engineering Ausbildung zu integrieren ist. Dazu haben wir uns in diesem Beitrag an folgenden Leitfragen orientiert:

- Wie und wofür wird die UML in der Praxis tatsächlich eingesetzt?
- Welche Kompetenzen sollten dafür durch die Hochschullehre entwickelt werden?

Grundlage dieser Arbeit sind empirische Studien über den Einsatz der UML anhand ihrer tatsächlichen Nutzung. Die Relevanz in der industriellen Praxis wird anhand von bestehenden Studien bewertet. Daneben haben wir eine Analyse von studentischen Projekt- und Abschlussarbeiten hinsichtlich ihrer Nutzung von grafischer Modellierung durchgeführt.

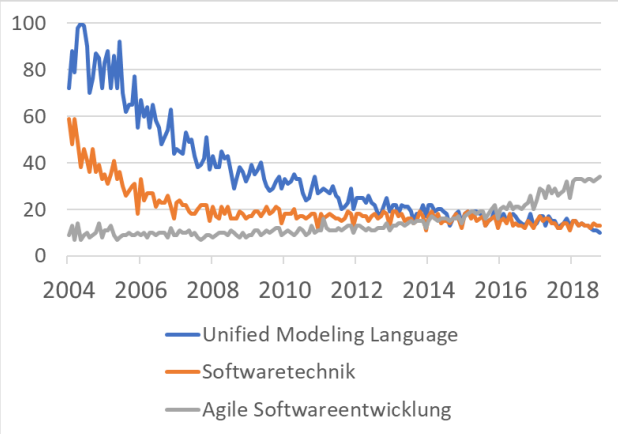

Abb. 1: Google Trends für Begriffe des Software Engineerings (2004-2018, nur Deutschland)

Dieser Beitrag ist wie folgt aufgebaut: Zunächst ordnen wir die UML historisch ein, diskutieren die Einsatzfelder sowie den Nutzen der Modellierung. Anschließend erfolgt die Analyse von Studien zur Nutzung von UML in der Praxis sowie studentischen Arbeiten. Nach der Diskussion der Ergebnisse leiten wir Schlussfolgerungen für die Rolle der UML in der Lehre ab.

UML in der Softwaretechnik

Entstehung der UML

Im Methodenkrieg der 1990er Jahre gab eine Reihe konkurrierender Ansätze für objektorientierte Methoden und Modellierungssprachen. Die erste Version der UML wurde 1997 als Vereinigung einer aus der „Unified Method“ nach Rumbaugh/Booch und dem „Object Oriented Software Engineering“ nach Jacobsen gebildet. Der Toolhersteller Rational unterstützte die UML bereits damals und trug damit zu ihrer Verbreitung bei (Rupp u. a., 2012). In den folgenden Jahren fand eine Erweiterung der UML um weitere Diagrammtypen und Konstrukte wie der Object Constraint Language (OCL) statt. Zur Standardisierung wurde die UML schließlich an die Object Management Group (OMG) übergeben. In der aktuellen Version 2.5 enthält die UML insgesamt 14 Diagrammtypen, von denen jeweils sieben zur Beschreibung der Struktur (z.B. Klassendiagramm, Verteilungsdiagramm, Paketdiagramm) und sieben zur Beschreibung des Verhaltens (z.B. Use-Case Diagramm, Aktivitätsdiagramm, Sequenzdiagramm) dienen.

Einsatzfelder und Modellarten

Modelle können beschreibend (deskriptiv) oder vorschreibend (präskriptiv) sein. Beschreibende Modelle werden einerseits für die Systemdokumentation und andererseits in der Analyse für die Beschreibung des Problemraums (Domänenmodell, Geschäftsprozesse usw.) eingesetzt. Präskriptive Modelle hingegen werden für den Entwurf und die Implementierung eingesetzt, da sie einen Bauplan („Vorschrift“) für ein zu realisierendes System beschreiben. Für die automatische Codeerzeugung auf Basis eines solchen Modells ist ein hoher Detailgrad und ein für die Zielplattform geeigneter Generator erforderlich. Wenn hingegen ein Programmierer die Implementierung manuell durchführt, kann dieser auf Basis seiner Interpretationsfähigkeiten auch mit einem weniger formalen Modell arbeiten. Tab. 1 fasst die verschiedenen Aufgaben, Modellarten und Beteiligte zusammen.

Je nachdem, wofür die UML eingesetzt wird, gibt unterschiedliche Anforderungen an die Modelle und demnach auch die Fähigkeiten der Modellersteller und -nutzer. Sommerville nennt für die Modellierung von Systemen drei verschiedene Absichten, die jeweils unterschiedliche Grade an Detaillierung und Rigorosität in der Anwendung der Modellierungssprache erfordern (Sommerville, 2015):

- *Diskussionen über ein bestehendes oder geplantes System:* Diese Modelle können unvollständig sein und die Notationen der Modellierungssprache informell verwenden.
- *Dokumentation eines bestehenden Systems:* Diese Modelle müssen ebenfalls nicht vollständig sein, wenn nur Teile des Systems dokumentiert werden. Jedoch ist Korrektheit und Genauigkeit erforderlich.
- *Codegenerierung:* Präzise Systembeschreibung als Basis für die Erzeugung der Implementation in einer modell-basierten Entwicklung.

Dies ist im Wesentlichen übereinstimmend mit der Systematisierung von Chaudon et.al., die

1. Modelle zur Analyse und Verstehen
2. Modelle zur Kommunikation
3. Modelle als Vorlage für die Implementierung
4. Modelle als Basis für die Codegenerierung

unterscheiden (Chaudron, Heijstek, & Nugroho, 2012). Die Reihenfolge dieser Nutzungsarten (von den Autoren als *modeling styles* bezeichnet) gibt ebenfalls die steigende Anforderung an Modellqualität bzw. Genauigkeit an.

Nutzen der Modellierung

Die Verwendung von UML für die Modellierung im Software Engineering soll sowohl Produktqualität als auch Entwicklungsproduktivität verbessern. Die Zwischenschritte zu diesen Wirkungen entstehen in den Kategorien Entwickler, Team, Prozess und Produkt, wie eine Studie von Chaudron u. a. (2012) ergeben hat (siehe Abb. 2). Insgesamt lässt sich sagen, dass das Ziel der Nutzung von UML im Verstehen, Beschreiben und Kommunizieren von existierenden oder geplanten Softwaresystemen in verschiedenen Lebenszyklusphasen (Analyse, Entwurf, Implementierung, Wartung) besteht. Ivar Jacobson, einer der Gründerväter von UML schreibt dazu: „Used appropriately it is a practical tool for raising the level of abstraction on software from the level of code to the level of the overall system” (Jacobson, 2009).

Die zentrale Rolle der Modellierung wird auch durch den *Guide to the Software Engineering Body of Knowledge (SWEBOK)* unterstrichen. Hier wird die UML als eine mögliche Modellierungssprache genannt und auf die Notwendigkeit verwiesen, eine für die jeweilige Aufgabenstellung geeignete grafische Notation auszuwählen (Bourque, Fairley, & IEEE Computer Society, 2014).

Phase	**Analyse**	**Entwurf**	**Implementierung**	**Dokumentation**
Original / Grundlage	Problemraum	Analysemodell	Entwurfsmodell	Implementiertes System
Modellinhalte	Analysemodelle, z.B. Domänenmodell, Geschäftsprozesse	Entwurfsmodelle, z.B. Datenstrukturen, Systemarchitektur	(Code)	Entwurfsmodell, z.B. Datenstrukturen, Systemarchitektur
Ersteller	Mensch	Mensch	Mensch	Mensch / Maschine
Nutzer	Mensch	Mensch	Mensch / Maschine	Mensch
Modellqualität	Informell, geringer Detailgrad	präzise, mittlerer Detailgrad	präzise, hohe Detaillierung	Vollständig, korrekt, genau

Tab. 1: Modellarten und ihre Eigenschaften

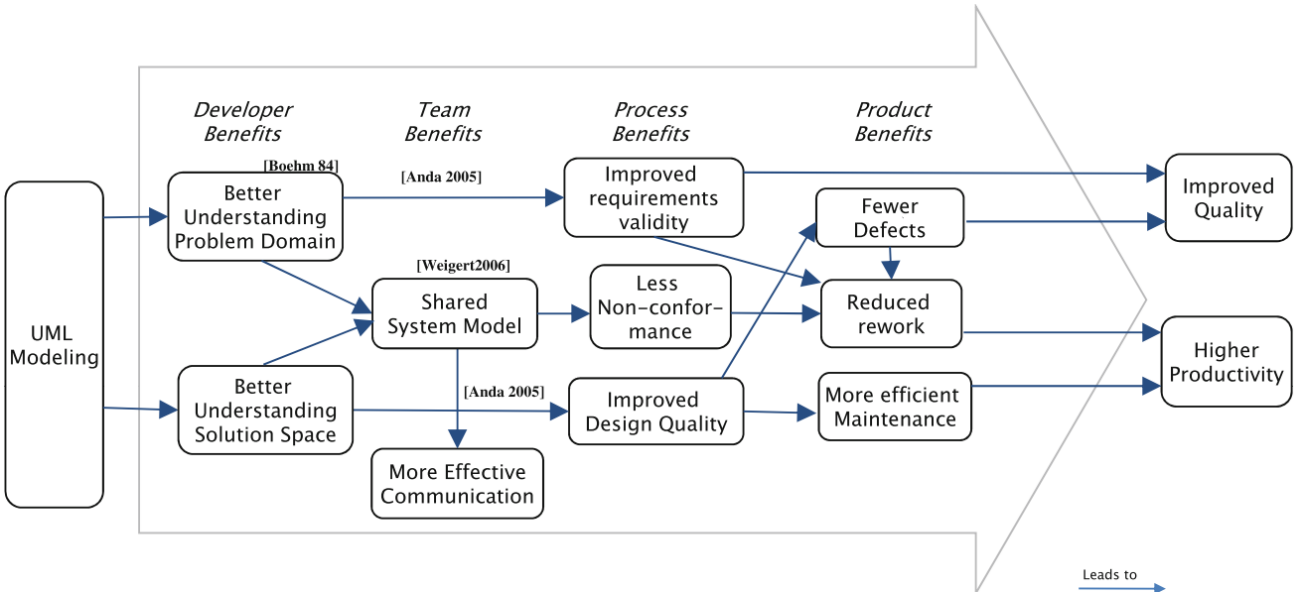

Abb. 2: Nutzen der UML Modellierung (Chaudron u. a., 2012)

UML-Nutzung durch Praktiker

Für die Beurteilung der praktischen Relevanz von UML stützen wir uns auf Studien, die den Einsatz der UML in der Praxis untersucht haben. Dabei werden einerseits die empirisch ermittelte Nutzung von UML-Modellelementen und andererseits die Verwendungszwecke untersucht.

Häufigkeit von Modellelementen der UML

Die Frage nach der Nutzung der UML anhand der tatsächlichen Verwendung von Diagrammtypen und Modellelementen wurde in den letzten 15 Jahren mittels verschiedener Methoden untersucht. Die Ergebnisse und die zugrundeliegende Methode werden nachfolgend kurz vorgestellt.

1) Delphistudie (Erickson & Siau, 2007)
   - Enterprise Systeme: 1. Klassen-, 2. Use Case-, 3. Sequenz-, 4. Aktivitätsdiagramme
   - Echtzeitsysteme: 1. Klassen-, 2. Zustands-, 3. Sequenz-, 4. Use Case-Diagramme
   - Web Applikationen: 1. Klassen-, 2. Use Case-, 3. Sequenz-, 4. Zustandsdiagramme
2) Befragung, N=284 (Dobing & Parsons, 2010)
   - 73% der Befragten nutzten in mind. 2/3 ihrer Projekte Klassendiagramme, gefolgt von Use Case (51%) und Sequenzdiagrammen (50%)
   - Klassen- und Sequenzdiagrammen wurden für die Aufgaben *Implementierung*, *Dokumentation*, *Erklären/Verständnis im Team* am häufigsten ein mindestens begrenzter Nutzen zugeschrieben.
   - Nur für *Validierung von Anforderungen mit dem Kunden* wurde der Nutzen bei Aktivitäts- und Use Case-Diagrammen am höchsten eingeschätzt.
   - Der Einbezug von Kunden wurde anhand ihrer Mitwirkung beim *Entwickeln*, *Prüfen* und *Freigeben* von Modellen bewertet. In allen Fällen wird das Use Case Diagramm am häufigsten genannt, gefolgt vom Aktivitätsdiagramm.
3) Analyse von Lehrbüchern, UML Tutorials, Hochschulkursen (Gianna Reggio, Maurizio Leotta, Filippo Ricca, & Diego Clerissi, 2013)
   - In Hochschulkursen sind Klassen-, Sequenz-, Aktivitäts-, Use Case- und Zustandsdiagramme besonders häufig.
   - In UML Tutorials zusätzlich dazu Komponenten- und Verteilungsdiagramme.
   - Die am seltensten genutzte Diagrammtypen sind Timing-, Profil-, Interaktionsübersichts- und Kompositionsstrukturdiagramme.
4) Quantitative Analyse von Open UML Modellen, N=121 (Langer, Mayerhofer, Wimmer, & Kappel, 2014)
   - Häufigste Sprachelemente: Class (100%), Use Case (47%), Interactions (39%)
   - Häufigste Elemente in Interaktionsdiagrammen: Message, Lifeline
   - Profile werden häufig für Robustness-Diagramme u. Datenbankschemata genutzt
   - Klassen sind die am häufigsten mit Stereotypen versehenen Modellelemente.

5) Experteninterviews, N=50 (Petre, 2013)
   - 35 von 50 Befragten nutzen UML gar nicht, v.a. wegen des Fokus auf die Softwarearchitektur und nicht auf das ganze System (fehlender Kontext), dem hohen Aufwand zum Verstehen der UML (unnötige Komplexität) sowie Problemen von Synchronisierung bzw. Inkonsistenzen zwischen Modell und Code bzw. Modellsichten
   - 11 von 50 Befragten nutzen UML nur selektiv, besonders in frühen Phasen zur Bewertung von Ideen, auch im Team. Sie weniger verwendeten UML meist in kleineren Modellen, adaptierten die Notation an ihre Bedürfnisse und beendeten die Verwendung, wenn UML in der jeweiligen Projektphase nicht mehr gebraucht wurde.
   - 3 der 50 Befragten nutzten UML für die automatische Codeerzeugung. Die entsprechenden Einsatzfälle waren Software für eingebettete System oder Produktlinien.

Einsatz von konzeptioneller Modellierung

Bei diesem Aspekt geht es um die Erhebung von Einsatzfällen für Modellierung, z.B. in Anhängigkeit von Unternehmensgrößen, Modellierungserfahrung, Projekttyp oder Entwicklungsphase. Die hier betrachtete konzeptionelle Modellierung umfasst auch andere Sprachen, nicht nur die UML. Wie auch schon bei den Studien zur Häufigkeit von Modellelementen werden hierfür unterschiedliche Forschungsmethoden eingesetzt.

1) Befragung zur Modellierung, N=312 (Davies, Green, Rosemann, Indulska, & Gallo, 2006)
   - Die häufigsten regelmäßig eingesetzten Modellierungstechniken sind ER-Diagramm (42%), Datenflussdiagramm (34%) und Flowcharts (29%). UML ist mit 21% auf Platz 5.
   - Die meistgenutzten Tools sind MS Visio (44%) und Rational Rose (11%).
   - Unternehmensgröße: Die UML wird bei mittleren Unternehmen (100-1000 Mitarbeiter) häufiger eingesetzt als bei kleineren oder größeren Unternehmen.
   - Modellierungserfahrung: Mitarbeiter mit 4-10 Jahren Erfahrung nutzen häufiger konzeptionelle Modelle als Kollegen, die kürzere oder längere Erfahrung haben.
   - Die häufigsten Einsatzzwecke für konzeptionelle Modellierung sind (1) Datenbankdesign/–management, (2) Geschäftsprozessdokumentation, (3) Interne Prozessverbesserung, (4) Softwareentwicklung, (5) Verbesserung kollaborativer Prozesse.
2) Analyse der Repositories von Open Source Projekten, N=10 (Osman & Chaudron, 2013)
   - Das Verhältnis von modellierten Klassen und implementierten Klassen lag zwischen 4% im größten Projekt (~900 Klassen) und 40% im kleinsten Projekt (<60 Klassen)
   - Nur wenige Projekte aktualisierten ihre initialen Modelle, meist wurden bei neuen Features neue Modelle hinzugefügt.
3) Befragung zum Einsatz von Entwurfsmodellen, N=3785 (Gorschek, Tempero, & Angelis, 2014)
   - Entwurfsmodelle werden in der Praxis nicht häufig verwendet.
   - Modelle werden eher informell und ohne Toolsupport erstellt.
   - Die Notation ist tendenziell nicht UML.
   - Die Nutzung von Modellen nimmt mit Grad an Erfahrung und Qualifikation ab
   - Modelle werden eher zur Kommunikation und Zusammenarbeit genutzt und auf Whiteboard oder Papier gezeichnet und werden nach Erstellung selten aktualisiert.
4) Befragung zum Einsatz von Skizzen und Diagrammen im Software Engineering, N=394 (Baltes & Diehl, 2014)
   - 48% aller Skizzen enthalten einige UML-Elemente, 9% ausschließlich UML.
   - Die Hälfte der Skizzen wurde von einer einzelnen Person angefertigt, 28% von zwei und 15% von drei Personen.
   - 58% aller Skizzen wurde analog (Papier, Whiteboard) angefertigt, 40% digital.
   - Digitale Skizzen werden im Vergleich zu analogen Skizzen zu einem höheren Teil (77%) überarbeitet und archiviert (94%).
   - Wesentliche Einsatzzwecke sind Entwerfen (75%), Erklären (65%), Verstehen (56%) sowie Anforderungen analysieren (45%).

Zwischenfazit und Diskussion

Da einige Studien z.T. schon über 10 Jahre alt sind, können die Ergebnisse der Studien nur mit Vorsicht auf die heutige Verwendung extrapoliert werden. Dennoch lassen sich gewisse Tendenzen bei der Nutzung von UML (bzw. konzeptionellen Modellierungsansätzen insgesamt) in der Praxis der Softwareentwicklung durchaus ausmachen.

Tendenzen in der UML-Nutzung

Die UML hat gemäß den ausgewerteten Studien in der Praxis eine sehr hohe Bekanntheit und auch eine hohe Nutzung. Allerdings werden in der Praxis die einzelnen Diagrammtypen unterschiedlich häufig verwendet. Insbesondere Klassen-, Aktivitäts-, Use-Case- und Sequenzdiagramme sind verbreitet.

Die Verwendungszwecke sind oft in frühen Phasen der Systementwicklung, z.B. zur Unterstützung der Analyse. Dabei spielt Modellierung besonders für die Kommunikation und Zusammenarbeit zwischen den Beteiligten eine wichtige Rolle.

Modelle sind häufig informell oder werden mit informellen Notationen gemischt. Alternativen zur Modellierung mit UML sind weniger formelle Modellierungsansätze wie das C4-Architekturmodell (Brown, 2018), textuelle Domain Specific Languages (DSL) oder auch Ad-Hoc-Notationen.

Als Medien werden neben Papier und Whiteboard auch verschiedene Tools eingesetzt, wobei diese nicht notwendigerweise klassische UML-Modellierungstools sind. Wenn ein konsistentes Gesamtmodell mit vielen untereinander verbundenen Sichten weniger wichtig ist, kann auf ein Model Repository verzichtet werden. In diesen Fällen können auch Zeichentools wie Visio oder eines der zahlreichen auf Kollaboration ausgelegten webbasierten Zeichenwerkzeuge für die UML-Modellierung genutzt werden.

Die Erzeugung von Code aus UML-Modellen spielt in der Praxis eine sehr untergeordnete Rolle. Der Ansatz der modellgetriebenen Architektur (MDA) mit umfangreicher Codeerzeugung aus detaillierten UML-Modellen hat sich im Wesentlichen nur in für langlebige Systeme in gut verstandenen Domänen etabliert (Sommerville, 2015).

Gründe für die veränderte Nutzung von UML

Trotz ihrer Verbreitung wird die UML in der Praxis in sehr unterschiedlichem Maße eingesetzt. Hierfür lassen sich eine Reihe von Gründen identifizieren.

Die zunehmende Diversifizierung von Softwareproduktarten schränkt der Nutzen einer einheitlichen Modellierungssprache ein. Die Diversität der Softwarelandschaft insgesamt nimmt zu. Beispiele sind die weite Verbreitung von Webapplikationen und mobilen Apps oder das Nebeneinander von Endkundenorientierten E-Commerce-Applikationen neben klassischen betrieblichen Informationssystemen oder sicherheitskritischen Echtzeitsystemen. Diese verschiedenen Softwareprodukte unterscheiden sich sowohl in ihren technischen Plattformen wie auch in ihren typischen Projektkonstellationen stark voneinander. Dies betrifft u.a. die Stabilität der Anforderungen, die Größe des Projektteams sowie den formalem Projektrahmen, was wiederum den Nutzen der Modellierung beeinflusst.

Agile Entwicklung braucht weniger UML. Agile Methoden basieren auf iterativer und inkrementeller Entwicklung, in denen die Phasen Analyse, Entwurf, Implementierung, Test und Dokumentation in kurzen Zyklen wiederholt werden. Dadurch sinkt die Notwendigkeit einer umfangreichen Spezifikation des gesamten Systems bzw. großer Systemteile, wofür in einem klassischen Wasserfall-Vorgehen häufig UML eingesetzt wird.

In der agilen Welt wird die Rolle des Software-Architekten, die klassischerweise UML als wesentliches Spezifikations- und Kommunikationsinstrument verwendet, mit einer gewissen Skepsis betrachtet. Ein Beleg ist etwa das Prinzip 11 des Agilen Manifests: "The best architectures (…) and designs emerge from self-organizing teams", (Beck u. a., 2001), kritische Aussagen zu „klassischer" Softwarearchitektur finden sich darüber hinaus an vielen Stellen in der agilen Literatur (z.B. Coplien & Bjørnvig, 2010, S. 85). Diese Grundhaltung trägt dazu bei, dass formale Modellierung in agilen Projekten weniger verbreitet ist, auch wenn seit mehreren Jahren eine Reihe von Frameworks existieren, die agiles Vorgehen mit Architekturplanung verbinden (siehe z.B. Cockburn, 2006; Larman & Vodde, 2009; Leffingwell, 2007).

Microservices haben keine zentralen Modelle mehr. Das Aufkommen des Microservice-Architekturstils als Alternative zur monolithischen oder service-orientierten Architektur ist eng verbunden mit der agilen Vorgehensweise. Microservices sind so weit wie möglich lose gekoppelt, unter Inkaufnahme von Daten-Redundanzen. Eine klassische Top-Down-Modellierung von Applikations-, Daten- und Technologiearchitektur findet hier nicht mehr statt (oder in einer wesentlich abstrakteren, Prinzipien-orientierten Weise, die nicht unbedingt mehr eine formale Modellierung erfordert).

Die Services selbst sind eher klein und interagieren mit ihrer Umgebung häufig durch REST- und Eventschnittstellen, die sich ebenfalls gut ohne eine graphische Modellierung spezifizieren lassen. Diese maximal autarken Softwareeinheiten ermöglichen eine idealtypische Anwendung des agilen Ansatzes selbstorganisierter, autarker und im Wesentlichen auf mündliche (oder zumindest informelle) Kommunikation ausgerichtete Entwicklungsteams. Eine Modellierungssprache wie UML, die inhärent von einem zentralen Gesamtmodell mit vielen komplementären Sichten ausgeht, stiftet in diesem Kontext nur geringen Nutzen.

Domain Driven Design modelliert fachlich und weniger komplex. Domain Driven Design (Evans, 2003) ist der dem Microservice-Stil zugrundeliegende verbundene Entwurfsansatz. Der Designfokus liegt hier auf dem Verständnis und der Modellierung der Domäne, also der Fachlichkeit einer Anwendung. Die Gestaltung der Applikations- und Technologiearchitektur rückt in Hintergrund, gemäß der Überzeugung, dass diese nachgeordnet und dezentral gestaltet wird. Domain Driven Design nutzt zur Analyse nicht-graphische Begrifflichkeiten wie die Ubiquitous Language oder informelle Modellierungsformen wie Context Maps (Fowler, 2014), die nicht standardisiert sind und sich höchstens (im Fall der Context Map) grob an UML-Klassendiagramme anlehnen.

UML-Nutzung durch Studierende

Wir haben anhand von 47 Abschlussarbeiten und Projektberichten in den Studiengängen Informatik, Medien- und Wirtschaftsinformatik (Bachelor und Master) der TH Köln untersucht, ob und welche Modellierungsansätze Studierende verwendet haben. Es wurden nur Arbeiten betrachtet, in denen die Konzeption und/oder Erstellung eines Softwaresystems mindestens einen Teilschwerpunkt darstellte. Die Arbeiten durften nicht älter als fünf Jahre sein, um ein aktuelles Bild zu gewinnen.

Weiterhin durfte es keinerlei Vorgaben seitens der Betreuer geben, ob (und wenn ja welche) Modellierungsansätze zu verwenden waren[1]. Da die betrachteten Arbeiten also UML nicht explizit erfordern, lässt sich aus den Ergebnissen eine Tendenz ableiten, inwiefern Studierende UML (oder konkurrierende Ansätze) als Mehrwert bei der Spezifikation und Dokumentation empfinden.

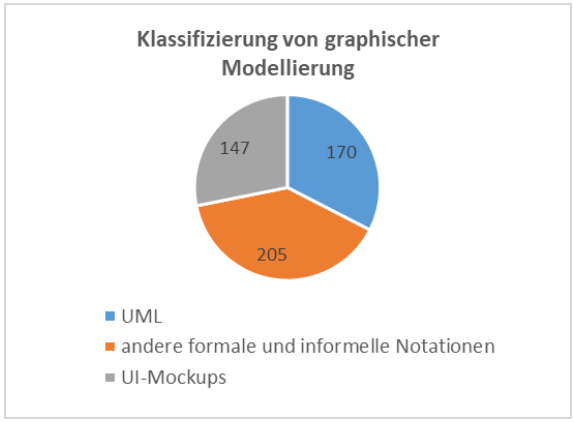

Abb. 3: Modellierungsnutzung durch Studierende

Wie Abb. 3 zeigt, verwenden weniger als die Hälfte der graphischen Modellierungen in den untersuchten Arbeiten UML. In knapp der Hälfte der Arbeiten werden zudem UI-Mockups genutzt, die im Folgenden jedoch nicht weiter betrachtet werden.

In den analysierten Arbeiten spielen nur Klassen-, Komponenten-, Aktivitäts-, Use-Case- und Sequenzdiagramme eine nennenswerte Rolle (siehe Abb. 4). In jeweils einer Arbeit kommt auch ein Objekt- und Deployment-Diagramm zum Einsatz. Das mag daran liegen, dass diese Diagrammtypen gerade diejenigen sind, die in den Softwaretechnik-Vorlesungen in Informatik, Medien- und Wirtschaftsinformatik vermittelt werden (nur Zustandsdiagramme fallen aus der Reihe, die zwar gelehrt, aber in den Arbeiten nicht auftreten). Davon abgesehen bestätigen sich die Ergebnisse der oben zitierten Studien zum UML-Einsatz: Eine Teilmenge der UML-Diagramme wird in der Praxis in nennenswertem Umfang verwendet.

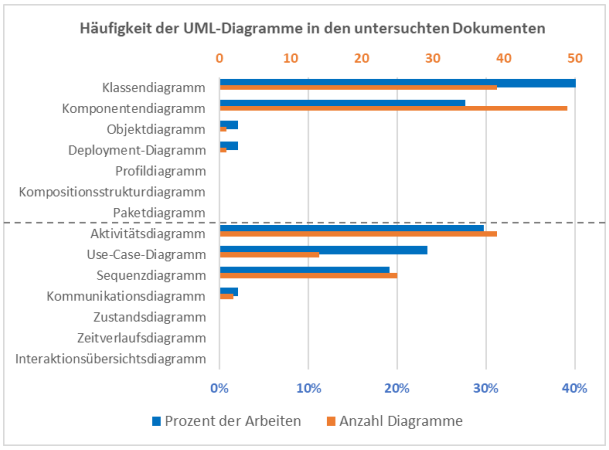

Abb. 4: Häufigkeit von UML-Diagrammen

Eine weitere Erkenntnis lässt sich aus Abb. 4 ableiten: Komponentendiagramme treten in knapp 30% der Arbeiten auf, dort gibt es davon dann insgesamt 49 Instanzen. Klassen- und Use-Case-Diagramme werden in ähnlich vielen Arbeiten verwendet, aber mit weniger Instanzen. Wenn also eine Arbeit Komponentendiagramme enthält, dann in der Regel mehrere; Use-Case-Diagramme hingegen treten fast überall maximal einmal auf, bei Klassendiagrammen ist es ähnlich.

Einsatzkontexte graphischer Modellierung

Ein wesentliches Ziel der Untersuchung war es, die Verwendung von UML-Diagrammen mit der von Nicht-UML-Notationen zu vergleichen. Hierfür wurden zwölf Einsatzkontexte für graphische Modellierung definiert, in denen neben UML auch andere formale Notationen sowie informelle Darstellungen in den untersuchten Arbeiten auftraten. Diese sind in Tab. 2 übersichtsweise dargestellt.

Die Häufigkeit des Auftretens der verschiedenen Einsatzkontexte (unabhängig davon, ob UML oder eine alternative Darstellung gewählt wurde) ist in Abb. 5 dargestellt. Demnach wird eine Komponentenstruktur in über der Hälfte der Arbeiten graphisch modelliert, technisch-algorithmische Abläufe, eine Klassenstruktur der Implementierung sowie eine Übersicht der Anwendungsfälle sind ebenfalls oft anzutreffen.

[1] Dadurch schieden z.B. Praktikumsberichte in Veranstaltungen aus, bei denen die Beschäftigung mit UML-Modellierung ein explizites Lernziel war. Die untersuchten Arbeiten wurden von neun verschiedenen Professoren betreut. Damit dürfte sich auch der Einfluss von dem Bemühen der Studierenden, ihrem Betreuer in dessen persönlichen Modellierungsvorlieben zu gefallen, über die Gesamtheit der Ergebnisse verwischen.

Einsatzkontext	**UML-Diagramm**	**Andere formale Notation**	**Informelle Variante**
Anwendungsfälle	Use Case	ArchiMate Business Services	Verknüpfung von Aktoren und Anwendungsfällen
Nutzungsszenario / Geschäftsprozess	Aktivitätsdiagramm	BPMN, EPK	Informelles Flussdiagramm
Fachliche Zustände von Geschäftsobjekten	Zustandsdiagramm	-	Informeller Graph
Fachliches Datenmodell	Klassendiagramm	-	Informeller Graph
Technischer Prozess / algorithmischer Ablauf	Aktivitätsdiagramm	BPMN, EPK	Informelles Flussdiagramm
Technische Zustandsfolge (State Machine)	Zustandsdiagramm	-	Informeller Graph
Technisches Kommunikationsszenario	Sequenz- oder Kommunikationsdiagramm	-	Mit Pfeilen und Sequenz-Zahlen annotierte Strukturen
Logisches Datenmodell / Domänenmodell	Klassendiagramm	ER-Diagramm	Informeller Graph
Physisches Datenmodell	Klassendiagramm	ER-Diagramm	Informeller Graph
Klassenstruktur der Implementierung	Klassendiagramm	-	Informeller Graph, teilw. aus Technniksymbolen
Komponentenstruktur	Komponentendiagramm	-	Informelle Block- oder Symbolgraphik
Verteilungssicht	Deploymentdiagramm	-	Informelle Block- oder Symbolgraphik

Tab. 2: Untersuchte Einsatzkontexte graphischer Modellierung

Allerdings ist es etwas ernüchternd, dass im Median jede untersuchte Arbeit nur jeweils zwei der genannten Einsatzkontexte graphisch modelliert, mit jeweils vier Diagrammen. Abb. 6 zeigt die Häufigkeit der Wertepaare „(Anzahl Diagramme, Anzahl Einsatzkontexte)“ als Blasendiagramm (je häufiger ein Wertepaar auftritt, desto größer die Blase). Bei Diagrammen liegt der Mittelwert mit ca. acht Diagrammen je Arbeit deutlich über dem Median. Einzelne Arbeiten verwenden also graphische Modellierung deutlich größerem Ausmaß als der Rest.

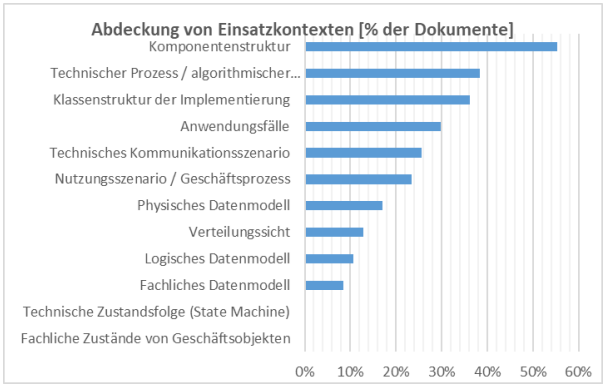

Abb. 5: Abdeckung der Einsatzkontexte

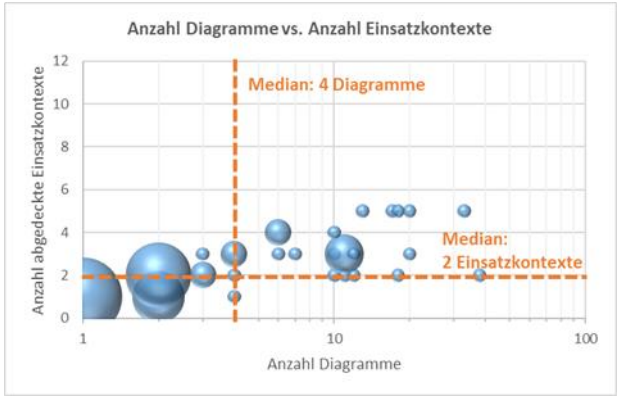

Abb. 6: Anzahl Diagramme vs. Einsatzkontexte

Verwendung von UML im Vergleich mit anderen Notationen

UML-Klassendiagramme stellen in gleich vier von zwölf Einsatzkontexten eine der Alternativen dar (siehe Abb. 7). Am häufigsten wird die Klassenstruktur der Implementierung dargestellt, hier trifft man hauptsächlich auf UML-Klassendiagramme. Das physische Datenmodell wurde in den untersuchten Arbeiten hingegen ausschließlich als ER-Diagramm modelliert. Logisches und fachliches Datenmodell werden nur selten modelliert.

UML-Aktivitätsdiagramme sind in zwei der zwölf untersuchten Einsatzkontexte von Belang. In beiden tritt eine UML-Modellierung gleichberechtigt neben anderen Darstellungsformen auf (siehe Abb. 8). Bei technischen Prozessen und algorithmischen

Abläufen trifft man häufig eine informelle Flussdiagramm-Notation an, während Geschäftsprozesse oder Nutzungsszenarien häufig als BPMN (und gelegentlich auch als EPK) modelliert waren.

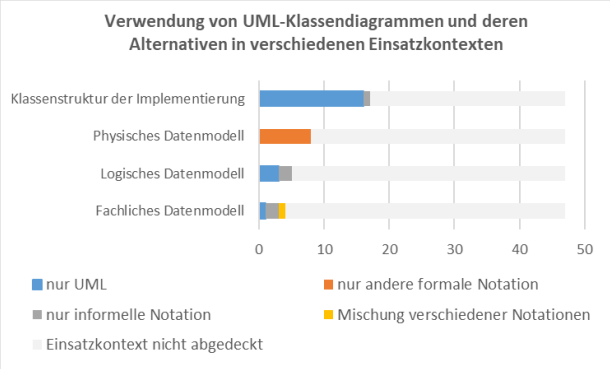

Abb. 7: Einsatzkontexte mit UML-Klassendiagrammen

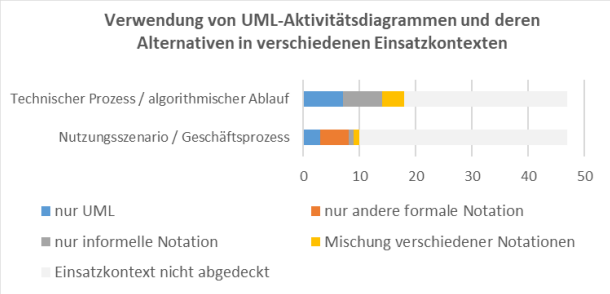

Abb. 8: Einsatzkontexte mit UML-Aktivitätsdiagrammen

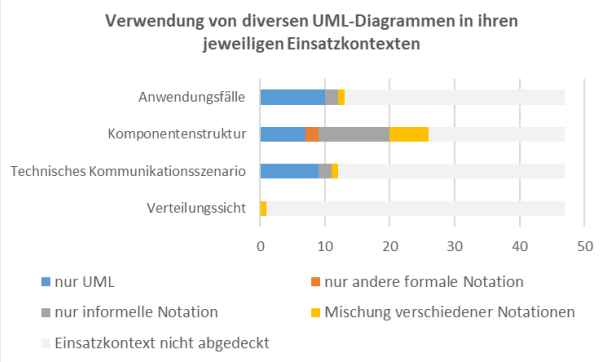

Abb. 9: Verwendung von weiteren UML-Diagrammen in ihren jeweiligen Einsatzkontexten

In Abb. 9 sind die Einsatzkontexte dargestellt, in denen UML-Use-Case-, Komponenten-, Sequenz-, Deployment- und Kommunikationsdiagramme eine Rolle spielen. Komponentendiagramme haben in einer informellen Darstellung mit gestapelten oder geschachtelten Blöcken eine große Konkurrenz. Anwendungsfälle werden hingegen in der Regel als Use-Case-Diagramme modelliert. Gleiches gilt für Kommunikationsszenarien, hier dominieren Sequenz- und Kommunikationsdiagramme.

Zwischenfazit und Diskussion

Auch wenn geplant ist, die Untersuchung noch weiterzuführen und ihre Datenbasis zu verbreitern, lassen sich schon einige Erkenntnisse festhalten.

Graphische Modellierung wird benötigt. Graphische Modellierung hat ihren festen Platz in der Dokumentation studentischer Softwarekonzeption und -Implementation. Allerdings streuen die untersuchten Arbeiten stark in Umfang und Konsistenz der verwendeten Modellierungsansätze. Möglicherweise liegt dies einfach an der natürlichen Qualitätsverteilung der untersuchten Arbeiten.

Nur ein Subset der UML-Spezifikation ist für studentische Arbeiten relevant. Die Untersuchungen zeigen, dass in den Arbeiten im Wesentlichen diejenigen Diagramme verwendet werden, auf die sich auch die Lehre an der TH Köln beschränkt (und die auch in den Praxisstudien als die dominierenden Typen festgestellt werden). Ob die Studierenden einfach das einsetzen, was sie im Studium kennengelernt haben, oder aber diese Diagrammtypen anscheinend ihren Zweck gut erfüllen, lässt sich aus dieser Untersuchung nicht klären. Die Tatsache, dass so viele andere formale und informelle Notationen eingesetzt werden, spricht doch für eine „Freiwilligkeit". Sprich: Es ist zu vermuten, dass die Studierenden nur Diagrammtypen verwenden, die sie für sinnvoll erachten und gut verstehen.

Informelle Notationen haben ihren Platz. Es fällt auf, dass informelle Notationen oft eingesetzt werden, in denen UML möglicherweise eine Überformalisierung darstellt (Komponentenstrukturen, technisch-algorithmische Abläufe). Dort, wo die UML-Diagramme jedoch entweder intuitiv klar (Use-Case-Diagramme) und/oder kompakt und ausdrucksstark (Klassen-, Sequenz-Diagramme) sind, kommen informelle Alternativen selten vor.

Andere formale Notationen haben ebenfalls ihren Platz. Bei Geschäftsprozessen und physischem Datenmodell dominieren andere Notationen als UML. Möglicherweise liegt dies an der Struktur der Informatik-Ausbildung an der TH Köln, die in den entsprechenden Fächern (Datenbanken, Informationsmanagement) die Standards „ER" und „BPMN" lehrt. Vielleicht sind diese Standards aber einfach tatsächlich verbreiteter als ihre UML-Gegenstücke.

Das Potential graphischer Modellierung für fachliche Aspekte wird nicht ausgeschöpft. Betrachtet man die Ergebnisse in Abb. 5, so sind die drei am häufigsten abgedeckten Einsatzkontexte graphischer Modellierung allesamt technischer Natur. Die fachlich geprägten Kontexte „Anwendungsfälle", „Nutzungsszenario / Geschäftsprozess" und „fachliches Datenmodell" stehen auf den Plätzen 4, 6 und 10 (von 12). Klassendiagramme werden überwiegend für Code-Dokumentation verwendet (Abb. 7). Zu-

standsfolgen von Geschäftsobjekten (die ein starkes Werkzeug fachlicher Modellierung sind) wurden in den untersuchten Arbeiten gar nicht gefunden. Dies alles zeigt ein deutlich technisches Übergewicht der graphischen Modellierung (ob in UML oder in anderer Form). Auf das Potential fachlicher Modellierung sollte also in der Lehre noch stärker hingewiesen werden.

UML in der Lehre – belassen, verändern, abschaffen?

Die Beschreibung von Software mithilfe von Modellen ist eine essentielle Aufgabe und gehört daher zum Kernbestandteil der Softwaretechnikausbildung. Bereits von 10 Jahren stellte Glinz dazu ein Thesenpapier über die Rolle der Modellierung in der Lehre vor (Glinz, 2008), das jedoch nicht spezifisch auf UML bezogen ist.

Die dargestellten empirischen Untersuchungen zeigen, dass die UML in der Praxis eher selektiv eingesetzt wird. Eine eigene schlaglichtartige Analyse von studentischen Arbeiten bestätigt diesen Eindruck auch für diejenigen Informatikern, die gerade ins Berufsleben eintreten.

Welcher Schluss ist daraus für die zukünftige Ausgestaltung und Weiterentwicklung der Softwaretechnik-Lehre zu ziehen? Ist die UML-Ausbildung in ihrer jetzigen Form noch zeitgemäß? Sollte sie verändert werden? Oder wäre es sinnvoll, sie ganz aus den Curricula von Informatik-Studiengängen zu verbannen? Diese Fragen kann nur durch eine Diskussion in der Fachcommunity beantwortet werden. Dazu möchten wir nicht nur die möglichen Alternativen aufzeigen, sondern auch selbst Stellung beziehen.

Die Erkenntnisse und Schlussfolgerungen aus Recherche und eigenen Erhebungen legen unserer Meinung nach nahe, dass eine UML-Ausbildung in der Form, wie sie vor 15 Jahren angemessen war, heute nicht mehr in die Zeit passt. Die Gründe hierfür haben wir unter „Zwischenfazit und Diskussion“ bei UML-Nutzung durch Praktiker und durch Studierende aufgezeigt. Ein reines „Belassen“ ist unserer Ansicht nach also keine sinnvolle Option.

Ebenso wenig erscheint es angebracht, UML komplett aus den Curricula zu entfernen. Unsere Erkenntnisse legen nicht den Schluss nahe, dass UML für die Softwaretechnik nicht mehr relevant ist.

Damit erscheint eine Anpassung des jetzigen Lehrkonzepts der sinnvollste Ansatz zu sein. Aus den obigen Analysen und Erkenntnissen lassen sich durchaus Bereiche identifizieren, in denen eine Neujustierung der Lehre sinnvoll ist. Um eine Fachdiskussion zum Thema anzuregen, haben wir unsere Vorschläge als Thesen strukturiert.

These 1. UML-Kompetenz sollte in der Softwaretechnikausbildung weiterhin vermittelt werden.

Die hohe Verbreitung von UML und ihre Nutzung als mittlere Abstraktionsebene zwischen fachlichem Problem und Implementierung lassen ihre Behandlung in der Lehre weiterhin sinnvoll erscheinen.

Die Aktivität der modellbildenden Abstraktion ist eine grundlegende Querschnittskompetenz in der Informatik (Engels, Hausmann, Lohmann, & Sauer, 2006). Vor diesem Hintergrund dient die Beschäftigung mit UML grundsätzlich dem Erlernen dieser Kompetenz – ganz unabhängig davon, ob man die konkreten Modellelemente auch tatsächlich später in seiner beruflichen Praxis verwendet.

Darüber hinaus scheinen einige UML-Diagrammtypen intuitiv verständlich und gut anwendbar zu sein, oder zumindest als bewusste oder unbewusste Vorlagen für eine informelle Modellierung zu dienen. Sie können daher als eine Art von informatischer Allgemeinbildung angesehen werden. (siehe auch These 2.)

These 2. Die Lehre sollte sich auf ausgewählte UML-Modellelemente fokussieren.

Die dargestellten Studienergebnisse legen nahe, dass ein Anspruch an Studierende, die UML-Sprachelemente möglichst umfänglich zu beherrschen, nicht sinnvoll ist. Insbesondere die hohe Komplexität und der hohe Formalisierungsgrad der UML erscheint nicht mehr zeitgemäß, da insbesondere die Maschinenlesbarkeit von UML Modellen für Codegenerierung bzw. direkter Ausführung („Executable UML“) sich nicht in der Breite durchsetzen konnten.

Wie die empirischen Untersuchungen zeigen, werden UML-Modelle vielfach informell zur Kommunikation und Zusammenarbeit eingesetzt. In diesen Situationen werden nur wenige Sprachelemente benötigt. Einige Diagrammtypen scheinen intuitiv verständlich und gut merkbar zu sein. Sowohl in der Praxis wie auch in der Auswertung studentischer Arbeiten werden diese Elemente mehr oder weniger nahe am definierten Standard verwendet. Dies trifft insbesondere auf Klassen-, Use-Case und Sequenz-Diagramme zu.

Andere UML-Diagrammtypen findet man häufig auch in einer informellen, „verballhornten“ Form, die sich aber durchaus mehr oder weniger deutlich an den UML-Sprachelementen orientiert. Beispiele hierfür sind Komponenten-, Deployment-, Aktivitäts-, Zustands- und Kommunikationsdiagramme. Diese Tatsache allein ist unserer Ansicht nach Grund genug, Kompetenzen in den „originalen“ UML-Diagrammtypen in der Lehre zu vermitteln.

Eine größere Bedeutung hat die UML nach wie vor für die Analyse, z.B. für Domänen- oder Geschäfts-

prozessmodelle. Da in diesen Fällen sowohl Modellersteller als auch -nutzer Menschen sind, ist ein geringeres Maß an Formalität ausreichend.

Dafür wird der durch die hohe Ausdrucksstärke bedingte große Umfang an Modellelementen jedoch nicht benötigt. Dies sorgt zum einen für Konkurrenz durch informelle Modellierungsansätze, für eine hohe Hürde beim Erlernen der UML sowie auch mangelhafte Einsicht in die Notwendigkeit der Sprachbeherrschung bei den Studierenden.

Konkrete Kandidaten für eine Verschlankung in der Lehre sind nach unserer Ansicht die Vielzahl von Beziehungstypen in Komponentendiagrammen, detaillierte Spezifikation von Extension Points in Use-Case-Diagrammen oder komplexe Entry- und Exit- Aktionen in Zustandsdiagrammen.

These 3. Die Lehre sollte sich auf die Beherrschung von Einsatzkontexten anstatt von UML-Diagrammtypen fokussieren.

Die GI gibt für die Informatikausbildung an Hochschulen folgende inhaltliche Empfehlungen mit Bezug zur UML (Gesellschaft für Informatik, 2016):

- *Stufe 1 (Verstehen):* „Verschiedene Notationen wie z.B. UML für die Modellierung von Softwaresysteme erläutern."
- *Stufe 2 (Anwenden):* „Standardsituationen im Bereich der Modellierung (…) umsetzen. Die Begriffswelt des Anwenders durch geeignete Vorgehensweisen erfassen und zu einer fachlichen Terminologie im Projekt verdichten."
- *Stufe 3 (Analysieren):* „Die Eignung eines Vorgehensmodells, einer Notation oder einer Methode für ein klassifiziertes Softwaresystem oder eine klassifizierte Aufgabe einschätzen."

In dieser Systematik ist 2 (Anwenden) die praxisrelevanteste Lernstufe. Allerdings wird in realen Projekten nicht die Kompetenz „Entwerfe ein UML-Klassendiagramm" gefordert, sondern „Entwerfe ein fachliches oder logisches Datenmodell". Daher plädieren wir dafür, in der Formulierung von Lernzielen den Fokus von UML-Diagrammtypen hin zu Einsatzkontexten graphischer Modellierung zu lenken. In Tab. 2 haben wir dazu einen Vorschlag von zwölf Einsatzkontexten gemacht.

These 4. Die Fähigkeit zur Auswahl von Modellierungsansätzen sollte gestärkt werden.

Neben der UML haben sich diverse andere mehr oder weniger formale Modellierungsansätze etabliert. Informelle Modellierungsansätze sowie formale Nicht-UML-Modellierungssprachen sollten daher vergleichend in die Lehre einfließen. Studierende sollten die Kompetenz erwerben, geeignete Modellierungsansätze, Diagrammtypen und den angemessenen Detailgrad im Modell in Abhängigkeit von der Aufgabenstellung auszuwählen. Dies ist konform mit Stufe 3 der o.g. GI-Empfehlung.

These 5. Die Modellierungskompetenz sollte auf das Absolventenprofil abgestimmt sein.

Sinnvoll wäre aus unserer Sicht eine Herleitung der angestrebten Modellierungskompetenz anhand des gewünschten Kompetenzprofils pro Studienfach und bestehenden Vorkenntnissen. Bei reinen Informatikstudiengängen ist in der Regel das Berufsbild des Softwarearchitekten Teil des Absolventenprofils, was einen bewussten Umgang mit Metaebenen und Abstraktion voraussetzt. Im Gegensatz dazu steht bei angehenden Medien- oder Wirtschaftsinformatikern eher die Anwendung im Vordergrund. Verschiedene technische Studiengänge sind z.B. mit der Entwicklung von Echtzeitsystemen befasst, in denen formale Verifikation und Codegenerierung relevant sind.

Zudem spielt Ausrichtung des Studiengangs auf eine Zieldomäne eine wichtige Rolle. Dabei sollten die verschiedenen Schwerpunkte in Softwareprodukten (Web vs. Backend, E-Commerce / Social Media vs. betriebliche Anwendungssysteme) sowie Projektmanagementansätze (agil vs. phasenorientiert) als Leitlinie dienen. In weiterführenden Modulen zum Thema Softwareproduktlinien oder Softwarewartung können besondere Einsatzbereiche der Modellierung erschlossen werden.

These 6. Die UML-Ausbildung sollte die Modellierung von Fachlichkeit stärker fokussieren.

Die Lehre sollte die frühen Phasen der Modellbearbeitung stärker in den Fokus nehmen. UML hat große Stärken in der Beschreibung einer fachlichen Domäne in Form eines fachlichen Datenmodells sowie von geschäftlichen Abläufen als Aktivitätsdiagrammen. Transaktionale Vorgänge lassen sich sehr gut als Zustandsdiagramme eines Geschäftsobjekts beschreiben.

Unsere Auswertung der studentischen Arbeiten hat im Gegensatz dazu gezeigt, dass UML eher für die technische Beschreibung aus Entwickler- statt aus Nutzersicht eingesetzt wird (siehe z.B. Abb. 7).

These 7. Die Lehre sollte nicht Tools, sondern Kommunikation und Interaktivität in den Fokus stellen.

Softwareentwicklung wird immer stärker durch Kommunikation und Zusammenarbeit geprägt. Damit sollte die Ausbildung die interaktive Nutzung von UML stärker einüben, z.B. in der Analyse- und frühen Entwurfsphase, in der in der Praxis oft auf Papier oder am Whiteboard gearbeitet wird.

Für die praktische Umsetzung ergeben sich daraus interessante Fragestellungen in Bezug auf die konkrete didaktische Umsetzung sowie die damit verbundene geeignete Toolunterstützung.

Zusammenfassung und Ausblick

Die Bedeutung von UML für die Softwaretechnik ist in Veränderung begriffen. Die Ursachen dafür liegen in Trends, die umfangreiche Modellierungen nicht mehr erforderlich machen. Dazu gehören agile Entwicklungsansätze, Microservicearchitekturen sowie das weitgehende Scheitern des MDA-Ansatzes, der detaillierte und formale Modelle erfordert. Die Aufmerksamkeit, die diesen Themen derzeit geschenkt wird, darf nicht darüber hinwegtäuschen, dass in vielen Großprojekten oder für sicherheitskritische Produkte die UML nach wie vor eine sehr wichtige Rolle spielt.

Aus unserer Sicht ist es notwendig, die Stellung der UML in der Lehre kritisch zu hinterfragen. Wir haben in diesem Beitrag dazu empirische Belege über die Nutzung der UML zusammengetragen. Diese Bestandsaufnahme diente als Grundlage für die Ableitung von Konsequenzen für die Ausgestaltung der Lehre. Naturgemäß kann dies keine abschließende Bewertung sein, sondern soll entsprechend unserer Zielsetzung Impulse zu diesem Thema liefern.

Für weiterführende Arbeiten erscheint neben einer Verbreiterung und Aktualisierung der empirischen Basis auch eine Ausdifferenzierung hinsichtlich verschiedener Studienrichtungen und -niveaus sinnvoll. Es muss beobachtet werden, wie sich die Nutzung der UML in der Praxis weiterentwickelt. Dabei sind Situationen zu betrachtet, in denen Modelle einen Mehrwert leisten. Wer ist Modellersteller, wer -nutzer? Wie umfangreich und präzise müssen die Modelle sein? Müssen aufkommende Programmierparadigmen wie funktionale und deklarative Programmierung berücksichtigt werden, da UML unmittelbar mit OOP zusammenhängt? Wie kann Fachlichkeit stärker in den Fokus genommen werden? Wie kann der identifizierte Kompetenzbedarf in praxisorientierten Lehrkonzepten umgesetzt werden? Wir hoffen, mit diesem Beitrag die Diskussion dazu ein Stück weit angeregt zu haben.

Literaturangaben

Baltes, S., & Diehl, S. (2014). Sketches and diagrams in practice. In S.-C. Cheung, A. Orso, & M.-A. D. Storey (Hrsg.), *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014* (S. 530–541). ACM. https://doi.org/10.1145/2635868.2635891

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for Agile Software Development. Abgerufen 30. November 2018, von http://agilemanifesto.org/

Boberić-Krstićev, D., Tešendić, D., Simos, T. E., Psihoyios, G., Tsitouras, C., & Anastassi, Z. (2011). Teaching Object-Oriented Modelling Using UML. In *Numerical Analysis and Applied Mathematics ICNAAM 2011* (S. 810–812). AIP. https://doi.org/10.1063/1.3636856

Bourque, P., Fairley, R. E., & IEEE Computer Society. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3. Aufl.). IEEE Computer Society Press.

Brown, S. (2018). The C4 model for software architecture. Abgerufen 21. Oktober 2018, von https://c4model.com/

Burgueño, L., Vallecillo, A., & Gogolla, M. (2018). Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, *28*(1), 23–41. https://doi.org/10.1080/08993408.2018.1462000

Chaudron, M. R. V., Heijstek, W., & Nugroho, A. (2012). How effective is UML modeling ? *Software & Systems Modeling*, *11*(4), 571–580. https://doi.org/10.1007/s10270-012-0278-4

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game* (0002 Aufl.). Upper Saddle River, NJ: Addison Wesley Pub Co Inc.

Coplien, J. O., & Bjørnvig, G. (2010). *Lean architecture for Agile software development*. Hoboken, N.J.: Wiley.

Davies, I., Green, P., Rosemann, M., Indulska, M., & Gallo, S. (2006). How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, *58*(3), 358–380. https://doi.org/10.1016/j.datak.2005.07.007

Dobing, B., & Parsons, J. (2010). Dimensions of UML Diagram Use: Practitioner Survey and Research Agenda. *Principle Advancements in Database Management Technologies: New Applications and Frameworks*, 271–290. https://doi.org/10.4018/978-1-60566-904-5.ch013

Engels, G., Hausmann, J. H., Lohmann, M., & Sauer, S. (2006). Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In J.-M. Bruel (Hrsg.), *Satellite events at the MoDELS 2005 conference* (Bd. 3844, S. 306–319). Berlin: Springer. https://doi.org/10.1007/11663430_

Erickson, J., & Siau, K. (2007). *Can UML Be Simplified? Practitioner Use of UML in Separate Domains*.

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software* (1 edition). Boston: Addison-Wesley Professional.

Gesellschaft für Informatik. (2016). *Empfehlungen für Bachelor- und Masterprogramme im Studienfach*

Informatik an Hochschulen (GI-Empfehlungen). Abgerufen von https://gi.de/fileadmin/GI/Hauptseite/Aktuelles/Meldungen/2016/GI-Empfehlungen_Bachelor-Master-Informatik2016.pdf

Gianna Reggio, Maurizio Leotta, Filippo Ricca, & Diego Clerissi. (2013). What are the used UML diagrams? A Preliminary Survey. In *EESS-MOD@MoDELS*.

Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik-Spektrum*, *31*(5), 425–434. https://doi.org/10.1007/s00287-008-0273-x

Gorschek, T., Tempero, E., & Angelis, L. (2014). On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software*, *95*, 176–193. https://doi.org/10.1016/j.jss.2014.03.082

Jacobson, I. (2009). Taking the temperature of UML. Abgerufen 1. November 2018, von http://blog.ivarjacobson.com/taking-the-temperature-of-uml/

Langer, P., Mayerhofer, T., Wimmer, M., & Kappel, G. (2014). On the usage of UML: initial results of analyzing open UML models. In H.-G. Fill (Hrsg.), *Modellierung 2014*. Bonn: Köllen.

Larman, C., & Vodde, B. (2009). *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Upper Saddle River, NJ: Addison-Wesley.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises*. Upper Saddle River, NJ: Pearson Education.

Liebel, G., Heldal, R., & Steghofer, J.-P. (2016). Impact of the Use of Industrial Modelling Tools on Modelling Education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)* (S. 18–27). IEEE. https://doi.org/10.1109/CSEET.2016.18

Osman, H., & Chaudron, M. R. V. (2013). UML Usage in Open Source Software Development: A Field Study. Gehalten auf der EESS-MOD@MoDELS 2013.

Petre, M. (2013). UML in Practice. In *Proceedings of the 2013 International Conference on Software Engineering* (S. 722–731). Piscataway, NJ, USA: IEEE Press. Abgerufen von http://dl.acm.org/citation.cfm?id=2486788.2486883

Rupp, C., Queins, S., & SOPHISTen, die. (2012). *UML 2 glasklar: Praxiswissen für die UML-Modellierung* (4. Aufl.). München: Carl Hanser Verlag.

Seiko Akayama, Birgit Demuth, Timothy Lethbridge, Marion Scholz, Perdita Stevens, & Dave R. Stikkolorum. (2013). Tool Use in Software Modelling Education. In *EduSymp@MoDELS*.

Sommerville, I. (2015). *Software Engineering, Global Edition* (10th edition). Boston: Pearson.