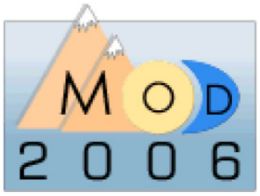
Jörg Desel, Martin Glinz (Hrsg.)

Modellierung in Lehre und Weiterbildung

Workshop auf der Modellierung 2006
Innsbruck, 23. März 2006



Technischer Bericht ifi-2006.03
Institut für Informatik, Universität Zürich
http://www.ifi.unizh.ch/techreports/TR_2006.html

Herausgeber

Jörg Desel, Prof. Dr.
Katholische Universität Eichstätt-Ingolstadt
Lehrstuhl für Angewandte Informatik
Ostenstraße 14
85072 Eichstätt
Deutschland
http://www.informatik.ku-eichstaett.de/Desel

Martin Glinz, Prof. Dr. rer. nat.
Institut für Informatik
Binzmühlestrasse 14
8050 Zürich
Schweiz
http://www.ifi.unizh.ch/~glinz

Programmkomitee

Leitung: Jörg Desel, Kath. Universität Eichstätt-Ingolstadt
Martin Glinz, Universität Zürich

Torsten Brinda, Universität Erlangen-Nürnberg
Gregor Engels, Universität Paderborn
Ulrich Frank, Universität Duisburg-Essen
Wolfgang Hesse, Universität Marburg
Peter Hubwieser, TU München
Johannes Magenheim, Universität Paderborn
Heinrich Mayr, Universität Klagenfurt
Markus Nüttgens, Universität Hamburg
Andreas Oberweis, Universität Karlsruhe
Barbara Paech, Universität Heidelberg
Klaus Pohl, Universität Duisburg-Essen
Martin Wessner, Fraunhofer IPSI Darmstadt

Inhalt Seite

Vorwort

Modellierung ist ein Querschnittsthema in der Informatik, das sich seit einigen Jahren als eigenständiges Gebiet in Forschung, Lehre und Anwendung etabliert hat. Die Tagungsreihe „Modellierung“ dient seit 1998 dem Austausch von Ansätzen und Erfahrungen im Bereich der Modellierung zwischen den verschiedenen Teildisziplinen der Informatik.

Dieser halbtägige Workshop ist dem Thema „Modellierung in Lehre und Weiterbildung“ gewidmet. Das Programmkomitee des Workshops hat aus den eingereichten Beiträgen fünf ausgewählt. Sie umfassen die ganze Spannbreite von der Ausbildung in der Schule bis zur Weiterbildung und von allgemeinen Lehreinheiten für die Softwaresystem-Modellierung bis zu speziellen Lehrkonzepten für eine bestimmte Modellierungstechnik.

Wir danken allen Autoren und dem Programmkomitee, die mit ihrem Engagement für das Zustandekommen dieses Workshops gesorgt haben. Wir danken ferner allen Personen, die uns bei der Organisation des Workshops und bei der Herstellung des Tagungsbandes unterstützt haben, insbesondere Frau Prof. Dr. Ruth Breu in Innsbruck und Frau Merlo-Schett in Zürich.

Eichstätt und Zürich, im März 2006

Jörg Desel
Martin Glinz

Vom Modellverstehen zum Modellerstellen

Lars Borner, Barbara Paech, Jürgen Rückert

AG Software Engineering
Fakultät für Mathematik und Informatik
Universität Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg
Email: {borner | paech | rueckert@informatik.uni-heidelberg.de}

Zusammenfassung: An der Universität Heidelberg wird in Lehrveranstaltungen im Bereich Software Engineering Grundlagenwissen zur Modellierung vermittelt. Hierfür wird ein dreistufiger Ansatz verwendet. In einem ersten Schritt lernen die Studierenden mögliche Modellierungstechniken verstehen. Darauf aufbauend ergänzen und verändern die Studierenden gegebene Modelle. Abschließend erstellen sie selbständig eigene Modelle. In diesem Beitrag stellen wir unser Vorgehen vor, das Verstehen, das Verwenden und das Erstellen von Modellen den Studierenden innerhalb einer Lehrveranstaltung zu vermitteln.

1. Einleitung

Die Fähigkeit, komplexe Sachverhalte abstrakt modellieren zu können, ist eine der wichtigsten Eigenschaften von Software-Entwicklern. Nur so können große Softwaresysteme überschaubar dargestellt und verständlich kommuniziert werden.

Aus diesem Grund legen die Lehrenden des Lehrstuhls Software Systeme der Universität Heidelberg sehr großen Wert auf die Vermittlung verschiedener Modellierungstechniken und grundlegender Modellierungsfertigkeiten. Diese Wissensvermittlung findet aus Zeit- und Personalmangel nicht im Rahmen einer eigenen Modellierungsveranstaltung statt, sondern in fachspezifischen Lehrveranstaltungen, wie in der Software Engineering I Vorlesung oder im Anfängerpraktikum. Diese Lehrveranstaltungen werden vorrangig für Bachelor und Master Studierende des Studienganges „Anwendungsorientierte Informatik“ angeboten. Aber auch fachfremde Studierende können an den Lehrveranstaltungen teilnehmen. Die Software Engineering I Lehrveranstaltung wird innerhalb eines Semesters durchgeführt und hat einen Umfang von sechs Semesterwochenstunden. Im Laufe des Semesters werden die wichtigsten Grundlagen aus dem Bereich der Modellierung anhand ausgewählter Techniken erläutert. Dabei wird in einem ersten Schritt ein Verständnis für die einzelnen Modellierungstechniken vermittelt. Im Anschluss daran sollen die Studierenden gegebene Modelle verwenden und erweitern. Abschließend ist es Aufgabe der Studierenden, eigene Modelle mit Hilfe der vermittelten Modellierungstechnik zu entwickeln. Für die Vermittlung des Modellierungswissens wird neben ausgewählten Beispielmodellen auch ein realistisch großes Beispielsystem verwendet. Durch diese

Beispiele können sowohl die Zusammenhänge zwischen den einzelnen Modellen erläutert als auch das Motivationsproblem (vgl. [Gli99]) gelöst werden.

Wie ein Ablauf der Software Engineering I Einführungsvorlesung mit den entsprechenden Übungen aussieht, ist in [PBR+05] erläutert.

Im Folgenden zeigen wir, wie das Modellierungswissen in drei Schritten vermittelt werden kann. Dabei werden die einzelnen Schritte *Modellverstehen*, *Modellverwenden* und *Modellerstellen* anhand von Beispielen erläutert. Danach diskutieren wir die Vor- und Nachteile unserer Vorgehensweise, unsere Erfahrungen sowie Konsequenzen für kommende Lehrveranstaltungen.

2. Umgang mit Modellen

Der Modellbegriff ist ein wesentlicher Bestandteil jeder wissenschaftlichen Fachdisziplin und spielt besonders im Bereich des Software Engineering eine tragende Rolle. Die nachfolgende Beschreibung einer möglichen Vermittlung dieses Modellbegriffs und der verschiedenen Modellierungstechniken wird von uns in allen Lehrveranstaltungen im Bereich Software Engineering praktiziert. Die Grundlagen im Bereich Modellierung werden dabei in der Grundvorlesung *Software Engineering I* und im Anfängerpraktikum (vgl. [SWE05]) vermittelt. Die späteren Lehrveranstaltungen, und zwar *Software Engineering IIa* und *IIb* und das *Fortgeschrittenen Praktikum* (vgl. [SWE05]), bauen auf diesem Grundwissen auf und erweitern es.

Um den Studierenden ein grundlegendes Verständnis von Modellen und den verschiedenen Modellierungstechniken zu geben, muss in einem ersten Schritt der Modellbegriff definiert und anhand von alltäglichen Beispielen erläutert werden. Dabei gehen wir kurz auf die drei Merkmale eines Modells, nämlich das Abbildungs-, das Verkürzungs- und das pragmatische Merkmal (vgl. [Lud02]), ein und besprechen diese anhand von bekannten Modellen, beispielsweise einem Stadtplan oder einer Modelleisenbahn. Im Anschluss daran wird auf den möglichen Einsatz von Modellen im Software Engineering eingegangen. Nachdem dieses Grundverständnis gelegt worden ist, können sich die Studierenden mit verschiedenen Modellen auseinander setzen. Dabei richtet sich die Vermittlung dieser Modelle nach folgendem Schema:

- Modellverstehen (*Modelle, Modellelemente und Modellierungstechniken kennen lernen und verstehen*)
- Modellverwenden (*Vorhandene Modelle erweitern und ergänzen*)
- Modellerstellen (*Selbstständig Modelle erstellen*)

Die drei oben beschriebenen Schritte werden innerhalb eines von den Studierenden selbst zu realisierenden Softwareprojektes durchgeführt. Dabei wird ein bereits vorhandenes Softwaresystem von den Studierenden erweitert. Hierzu müssen sie als erstes das System verstehen, um es später selbstständig erweitern zu können. Innerhalb dieser

„Kennenlernphase“ werden die verschiedenen Modelle der Software Entwicklung von den Studierenden „**verstanden**“ und auch „**verwendet**“. Bei der eigentlichen Erweiterung des Systems erhalten die Studierenden die Aufgabe, selbstständig eigene Modelle zu „**erstellen**“ (vgl. [PBR+05]).

Wie bei [DHZ+99], [LR01], [SB03] und vielen anderen Autoren wird ein Großteil der Aufgaben an der Universität Heidelberg im Team gelöst. Dies hat folgende Vorteile: es können innerhalb der Teams Synergieeffekte ausgenutzt werden, d.h. „schwächere“ Studierende können von „besseren“ lernen. Gleichzeitig werden die Softskills (vgl. [LMS+03]) der Studierenden verbessert und der Betreuungs- und Korrekturaufwand verringert. In Bezug auf Modellierung ermöglicht das Team die wichtige Erfahrung, dass verschiedene Personen typischerweise sehr unterschiedliche Modelle entwickeln und dass die Diskussion über diese Unterschiede das Verständnis für den modellierten Sachverhalt und die erstellten Modelle erhöhen kann.

Nachfolgend soll auf die einzelnen Phasen der Modellvermittlung näher eingegangen werden. Dazu werden das Ziel und die Vorgehensweise jeder Phase erläutert und mit Hilfe von ausgewählten Beispielen verdeutlicht.

2.1 Modellverstehen

In dieser Phase sollen die Studierenden in eine bestimmte Systemsicht (z.B. Aufgaben, Daten, Interaktion) eingeführt werden. Sie lernen die Sicht und eine oder mehrere Modellierungstechniken für diese Sicht kennen. Die Studierenden sollen so in die Lage versetzt werden, ein vorgegebenes Modell für diese Systemsicht zu lesen und dessen Inhalte zu verstehen. Darüber hinaus können sie anschließend die Inhalte eines Modells an Dritte kommunizieren.

Um dieses Ziel zu erreichen, werden in einem einführenden Vortrag die Sicht und die Modellierungstechniken vorgestellt. Hierbei werden sehr kleine Beispielmodelle verwendet, um die einzelnen Modellelemente verständlich zu erläutern.

Im Anschluss an den Vortrag erhalten die Studierenden die Aufgabe, vorgegebene Modelle zu verstehen und daraus entsprechende Informationen zu gewinnen. Die vorgegebenen Modelle sind deskriptive Modelle (vgl. [Lud02]) des Softwaresystems, welches später erweitert werden soll. Hierbei ist zu beachten, dass die Studierenden eine konkrete Aufgabe erhalten, die sie anhand dieser Modelle zu lösen haben. Das einfache „*schaut euch mal das Diagramm an und teilt mir mit, worum es geht*“ ist für die Studierenden zu langweilig. Darüber hinaus ist es für die Lehrenden schwierig, zu überprüfen, ob die Studierenden sich mit den vorgegebenen Modellen auseinander gesetzt haben. Daher werden konkrete Fragen zum Modell gestellt (siehe unten).

Zusätzlich wird das Motivationsproblem (vgl. [Gli99]) verringert, da die vorgegebenen Modelle das zu ändernde Softwaresystem beschreiben. Die Studierenden können also das in den Modellen repräsentierte Wissen für spätere Aufgaben weiterverwenden.

Die Verfügbarkeit des Originals, von dem das Modell abstrahiert, ist eine wichtige Voraussetzung für das Verstehen einer Modellierungstechnik. Nur wenn den Studierenden bewusst ist, wovon sie abstrahieren, können sie die Vorteile dieser Abstraktion erkennen. In unserem Fall können die Studierenden das existierende Softwaresystem ausprobieren (Benutzersicht) und den Programmquelltext einsehen (Entwicklersicht).

Als Beispiel für das Verstehen von Modellen sollen hier die Klassen- und Sequenzdiagramme genannt werden.

Um ein Klassendiagramm zu verstehen, wird den Studierenden Zugriff auf den Programmquelltext gegeben. Typische Fragen sind dann: „*Wie spiegelt sich die Beziehung, die im Klassendiagramm zwischen Klasse A und Klasse B enthalten ist, im Quelltext wider?*“, „*Wie wurden die im Klassendiagramm enthaltenen Multiplizitäten der Assoziation zwischen Klasse A und B im Quelltext realisiert?*“ oder „*Welche Entwurfsmuster sind im Klassendiagramm enthalten und wie werden sie im Programmquelltext umgesetzt?“*. Anhand dieser Fragen erkennen die Studierenden, wie die implementierten Klassen mit Hilfe des Klassendiagramms modelliert sind, wie eine Beziehung zwischen zwei Klassen im Programmquelltext abgebildet sein kann und wie Multiplizitätsvorgaben umgesetzt sind. Darüber hinaus lernen sie verschiedene Entwurfsmuster kennen. Ein Beispiel für ein verwendetes Klassendiagramm zeigt Abbildung 1. Es beschreibt einen kleinen Ausschnitt des Beispielsystems.

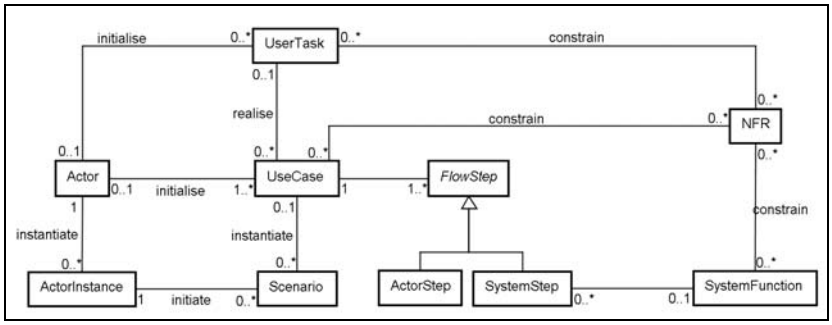


Abbildung 1: Beispiel eines verwendeten Klassendiagramms

Um ein Sequenzdiagramm zu verstehen, müssen die Studierenden ebenfalls Zugriff auf den Programmquelltext haben. Typische Fragen hierzu sind dann: „*Von welchen Klassen sind wie viele Instanzen im Sequenzdiagramm abgebildet?*“ oder „*Welche Nachrichten im Sequenzdiagramm sind Aufrufe von statischen Methoden?*“. Hierbei lernen die Studierenden zum einen, dass ein Sequenzdiagramm mehrere Instanzen einer Klasse darstellen kann und zum anderen, wie der Aufruf an eine statische Methode modelliert wird.

Die Fähigkeit, den Inhalt von Modellen an Dritte zu kommunizieren, wird durch die sehr umfangreiche Aufgabenstellung gewährleistet. Die Studierenden müssen die Aufgabe in kleinere Teilaufgaben gliedern und innerhalb des Teams verteilen. Anschließend erstellen sie gemeinsam eine Gesamtlösung, wodurch sie gezwungen sind, ihren Teil der Modelle und Lösungen an die restlichen Teammitglieder zu kommunizieren.

Diese Vorgehensweise beim Verstehen von Modellen kann auch auf Modelle der Anforderungsphase (wie z.B. Prozessmodelle) angewendet werden. Hierbei wird nicht der Programmquelltext als Referenz verwendet, sondern das vollständig lauffähige System.

2.2 Modellverwenden

Nachdem die Studierenden in der Lage sind, vorgegebene Modelle zu lesen und zu verstehen, steht in einem zweiten Schritt die Verwendung der Modelle im Mittelpunkt. Ziel dieser Phase ist es, zum einen die Studierenden dazu zu befähigen, vorgegebene Modelle selbständig zu erweitern, zu ergänzen und zu verfeinern. Zum anderen lernen sie, den Zusammenhang zwischen verschiedenen Modellen, Modellelementen und Modellierungstechniken zu erkennen und Informationen von einem Modell auf ein anderes zu übertragen.

Um dieses Ziel zu erreichen, verfolgen wir drei mögliche Vorgehensweisen: die Korrektur eines fehlerhaften Modells, die Beseitigung von Inkonsistenzen zwischen zwei Modellen bzw. die Ergänzung eines unvollständigen Modells. Diese drei Vorgehensweisen lassen sich auf deskriptive Modelle anwenden. Im ersten Fall korrigieren die Studierenden ein Modell, das Fehler enthält. Die Fehler werden durch Vergleich mit dem Original erkannt. Sie müssen gefunden und behoben werden. Im zweiten Fall ist es die Aufgabe der Studierenden zu überprüfen, ob zwei oder mehrere Modelle konsistent zueinander sind, und gefundene Inkonsistenzen gegebenenfalls zu beseitigen. Im letzten Fall erhalten die Studierenden ein Modell, in welches sie fehlende Modellelemente mit der geforderten Modellierungstechnik ergänzen. Die Vervollständigung ist auch sinnvoll für präskriptive Modelle. Dadurch ist es möglich, kritische Teilschritte beim Erstellen eines Modells einzuüben.

Die einzelnen Vorgehensweisen sollen anhand kleinerer Beispiele der Anforderungsphase verdeutlicht werden. Um den Studierenden das Modell der Use Cases nahe zu bringen, geben wir ihnen in einem ersten Schritt die Rümpfe einzelner Use Cases vor. In diesen Rümpfen können bereits der Name sowie die Vor- und Nachbedingungen vorgegeben sein. Aufgabe der Studierenden ist es jetzt, die noch fehlende Ablaufbeschreibung des Use Cases zu ergänzen. Das Wissen dazu erhalten sie bei einem deskriptiven Modell aus dem existierenden Softwaresystem oder bei einem präskriptiven Modell aus dem Lastenheft für die Erweiterung der Software. Somit lernen sie ein Modellelement des Use Cases selbstständig erstellen, ohne selbst ein komplettes Use Case Modell aufstellen zu müssen.

Ein weiteres Beispiel für das Verwenden von Modellen sind Domänendatendiagramme. Die Studierenden sollen hier anhand einer Checkliste überprüfen, ob die Beziehungen zwischen den dargestellten Entitäten des Domänendatendiagramms korrekt sind. Im

Diagramm selbst sind einige Beziehungen bzw. Multiplizitäten offensichtlich falsch. Auch hier liefert das existierende System das notwendige Wissen. Diese Fehler werden von den Studierenden aufgedeckt und korrigiert.

Im Fall der Konsistenzprüfung kann jedes Modell in sich stimmig, im Vergleich zu anderen Modellen jedoch nicht korrekt sein, z.B. durch Auslassungen. So können z.B. Domänendaten, die im Use Case erwähnt sind, im Domänendatendiagramm fehlen. Damit die Studierenden die Inkonsistenzen beseitigen können, müssen sie entscheiden, ob das Modell mit den Auslassungen oder das Modell mit den zusätzlichen Domänendaten korrekt ist. Die Überprüfung solcher Zusammenhänge und die Vereinheitlichung ist ein weiterer wesentlicher Schritt für die Verwendung von Modellen.

2.3 Modellerstellen

Ziel der letzten Phase ist es, die Studierenden dazu zu befähigen, selbständig aus den gegebenen Informationen entsprechende präskriptive Modelle zu erstellen. Hierbei steht nicht nur das „Notieren“ eines Modells im Vordergrund. Das selbständige Erstellen von Modellen ist viel komplexer als das einfache Verstehen und Verwenden von Modellen. Die Studierenden müssen daher in die Lage versetzt werden, die wichtigen Informationen zu erkennen und schrittweise abstrahiert in einem Modell festzuhalten. Es zeigt sich, „dass diese Situationen tatsächlich am besten unter den konkreten Bedingungen eines Modellierungsprozesses erfahrbar gemacht werden können“ ([Kna01] S. 134-135). Aus diesem Grund wird in unseren Lehrveranstaltungen die Phase der Modellerstellung in die Erweiterung eines bestehenden Softwaresystems eingebettet. Aufgabe der Studierenden ist es, eine Änderung in ein Softwaresystem nach Vorgaben des Lastenhefts zu integrieren. Dabei erstellen sie alle notwendigen Modelle der Anforderungsphase (z.B. Use Cases, Use Case-, Domänendaten- und Analyseklassendiagramme), der Entwurfsphase (z.B. Klassen- und Sequenzdiagramme) und der Qualitätssicherung (z.B. Testfälle, Testdaten). Dabei bauen die Modelle der späteren Entwicklungsphase auf den Modellen der früheren Phase auf. Um die Entscheidungen während des Modellierens transparent zu machen, müssen die Studierenden die wichtigsten Modellierungsoptionen, deren Vor-

und Nachteile sowie die Begründung(en) für eine Modellierungsoption dokumentieren (Rationales). Abschließend werden die geforderten Änderungen im Programmquelltext umgesetzt.

Bevor die Studierenden selbstständig ein Modell erstellen, wird in Zusammenarbeit mit den Lehrenden ein Teil des Modells erstellt. Hierbei lernen sie die wichtigen Teilschritte für das Aufstellen eines Modells kennen.

In dieser Phase ist es sehr wichtig, den Studierenden auf jedes selbst erstellte Modell aussagekräftiges Feedback zu geben. Darüber hinaus prüfen die Studierenden ihre erstellten Modelle untereinander in mehreren Inspektionsphasen. In einer Inspektionsphase werden die Modelle der vorhergehenden Entwicklungsphase von anderen, nicht an der Erstellung beteiligten Studierenden überprüft. Somit wird ihnen das Sammeln von Erkenntnissen über fremde Modelle ermöglicht und sie können von diesen Modellen auf eventuelle Missstände in den eigenen Modellen schließen. Im Anschluss an die Inspekti-

onsphase bzw. nach dem Feedback muss den Studierenden die Möglichkeit gegeben werden, die aufgedeckten Probleme in ihren Modellen zu beseitigen.

Damit wir die Studierenden bei der Erstellung der notwendigen Modelle für die Systemerweiterung unterstützen können, wählen wir einen kleinen Ausschnitt der zu realisierenden Anforderungen, z.B. einen Use Case, aus. Diese Auswahl dient im späteren Verlauf als durchgehendes Beispiel für die Modellerstellung in den verschiedenen Entwicklungsphasen. Anhand dieses Beispiels wird das Erstellen gemeinsam mit den Studierenden geübt. Dabei ist darauf zu achten, dass sich das Beispiel von den weiteren Anforderungen möglichst getrennt betrachten lässt.

3. Erfahrungen

In diesem Kapitel diskutieren wir die Erfahrungen aus unseren Lehrveranstaltungen und die daraus resultierenden Ergebnisse. Die oben beschriebene Vorgehensweise wurde in der Software Engineering I Einführungsvorlesung in den Sommersemestern 2004 und 2005 sowie im Anfängerpraktikum im Wintersemester 2005/2006 angewendet und wurde mit jeweils 10, 25 bzw. 15 Studierenden durchgeführt.

Während der Arbeit mit den Studierenden ist es sehr wichtig, dass sie für jeden ihrer Arbeitsschritte entsprechendes Feedback bekommen. Dies hilft einerseits, Fehler rechtzeitig zu beheben. Andererseits können die Studierenden auf diese Weise zur Weiterarbeit motiviert werden. Da eine Modellierungsaufgabe nie „richtig falsch“ gelöst werden kann, diskutieren wir mit den Studierenden die verschiedenen Lösungen und geben ihnen entsprechende Verbesserungsvorschläge.

Bei der Vermittlung der verschiedenen Systemsichten und Modellierungstechniken haben sich durchgängige Beispiele auf Basis eines ausreichend großen Softwaresystems bewährt. Für diese Beispiele existieren deskriptive Modelle, die die Studierenden schrittweise an Modelle und an das System heranführen. Die Beispielmodelle haben sich darüber hinaus als „Nachschlagewerk“ für die Studierenden etabliert, aus dem sie Wissen für ihre eigenen, selbst zu erstellenden Modelle gewinnen können.

Weiterhin ist eine Werkzeugunterstützung bei der Erstellung von Modellen unverzichtbar. Durch diese Unterstützung kann ein Großteil von Fehlern im Umgang mit verschiedenen Modellierungstechniken ausgeschlossen werden, z.B. durch Vorgabe von definierten Vorlagen oder der Einschränkung der möglichen Modellelemente. Bei der Auswahl sollte jedoch darauf geachtet werden, dass das Werkzeug möglichst einfach zu verstehen und zu handhaben ist, da innerhalb der Lehrveranstaltung nicht genügend Zeit für eine ausführliche Erläuterung zur Anwendung des Werkzeugs zur Verfügung steht. In unseren Lehrveranstaltungen haben sich das UML-Modellierungswerkzeug Jude (vgl. [Jud05]) und das Dokumentationswerkzeug Sysiphus (vgl. [Sys05]) bewährt.

Als Nachteil unserer Vorgehensweise ist der sehr hohe Betreuungs- und Vorbereitungsaufwand der Lehrveranstaltungen zu nennen. Zum einen bekommen alle Studierenden sowohl für die Einzel- als auch für die Teamaufgaben Feedback auf ihre Lösungen.

Hierfür wird eine große Anzahl von BetreuerInnen benötigt. Diese Betreuung wird von den MitarbeiterInnen des Lehrstuhls aber auch von Studierenden höherer Semester übernommen. Zum anderen nimmt die Anpassung der Modelle des Beispiels auf die aktuelle Version des Systems viel Zeit in Anspruch.

Ein weiterer Nachteil ist die Tatsache, dass aufgrund der Kürze der Zeit nicht alle wichtigen Aspekte der Modellierung (z.B. Metamodellierung, Modelltheorie usw.; vlg. [Gli99]) in unserer Lehrveranstaltung thematisiert werden können.

Im Laufe der verschiedenen Lehrveranstaltungen sind wir immer wieder auf das Problem gestoßen, dass das Erstellen von Modellen innerhalb eines verteilten Teams, in welchem die Teammitglieder an verschieden Orten arbeiten, sehr schwierig ist. Die aktuell eingesetzten Werkzeuge unterstützen diese Art der Entwicklung nur bedingt. Derzeit kann ein Modell nur von einer Person bearbeitet werden. An dieser Stelle würden wir uns ein Werkzeug wünschen, mit dem die gleichzeitige (parallele) und verteilte Arbeit an einem Modell möglich ist.

Trotz dieser Nachteile glauben wir, dass unser Vorgehen vom Modellverstehen zum Modellerstellen richtig ist. Natürlich könnte man in einer separaten Modellierungsveranstaltung, bei der die Studierenden schon vorher komplexe Systeme erstellt haben und mit den verschiedenen Tätigkeiten des Software Engineering vertraut sind, noch mehr Erfolge erzielen.

Danksagung

Wir möchten uns bei Anke Borner, Andrea Herrmann, Timea Illes und Dima Suliman für ihre tatkräftige Unterstützung und ihre vielen konstruktiven Hinweise im Reviewprozess bedanken.

Literaturverzeichnis

[DHZ+99] B. Demuth, H. Hußmann, S. Zschaler, L. Schmitz: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. In: Proceedings der SEUH '99: B. Dreher, C. Schulz, D. Weber-Wulff (Hrsg.): Software Engineering im Unterricht der Hochschulen, Workshop des German Chapter of the ACM und der Gesellschaft für Informatik e.V. (GI), Februar 1999 in Wiesbaden, B.G. Teubner Stuttgart Leipzig 1999, S. 21 – 30

[Gli99] M. Glinz: Modellierung im Informatikstudium. In: Proceedings der Modellierung 1999: J. Desel, K. Pohl, A. Schür (Hrsg.): Modellierung 1999, Workshop der Gesellschaft für Informatik e.V. (GI), März 1999 in Karlsruhe, B.G. Teubner Stuttgart Leipzig 1999, S. 190 – 195

[Jud05] Jude http://objectclub.esm.co.jp/Jude/, 15.02.2006

[Kna01] U. Knauer: Modellierung in der Ausbildung (von Mathematikern / Informatikern?) - Positionspapier In: Proceedings der Modellierung 2001: G. Engels, A. Oberweis, A. Zündorf (Hrsg.): Modellierung 2001, Workshop der Gesellschaft für Informatik e.V. (GI), März 2001 in Bad Lippspringe, Gesellschaft für Informatik, S. 133 - 137

[LR01] C. Lewerentz, H. Rust: Die Rolle der Reflexion in Softwarepraktika. In: Proceedings der SEUH 7: H. Lichter, M. Glinz (Hrsg.): Software Engineering im Unterricht der Hochschulen, Workshop des German Chapter of the ACM und der Gesellschaft für Informatik e.V. (GI), Februar 2001 in Zürich, dpunkt Verlag, S. 73 - 86

[LMS+03] H. Lichter, R. Melchisedech, O. Scholz, T. Wiler: Erfahrung mit einem Workshop-Seminar im Software Engineering-Unterricht. In: Proceedings der SEUH 8: J. Siedersleben, D. Weber-Wulff (Hrsg.): Software Engineering im Unterricht der Hochschulen, Workshop des German Chapter of the ACM und der Gesellschaft für Informatik e.V. (GI), Februar 2003 in Berlin, dpunkt Verlag, S. 89 – 100

[Lud02] J. Ludewig: Modelle im Software Engineering – Eine Einführung und Kritik. In: Proceedings der Modellierung 2002: M. Glinz, G. Müller-Luschnat (Hrsg.): Modellierung 2002, Workshop der Gesellschaft für Informatik e.V. (GI), März 2002 in Tutzing, Gesellschaft für Informatik, S. 7 – 22

[PBR+05] B. Paech, L. Borner, J. Rückert, A.H. Dutoit, T. Wolf: Vom Code zu den Anforderungen und wieder zurück: Software Engineering in sechs Semesterstunden. In: Proceedings der SEUH 9: K.P. Löhr, H. Lichter (Hrsg.): Software Engineering im Unterricht der Hochschulen, Workshop des German Chapter of the ACM und der Gesellschaft für Informatik e.V. (GI), Februar 2005 in Aachen, dpunkt Verlag, S. 56 - 67

[Sys05] Sysiphus http://wwwbruegge.in.tum.de/Sysiphus, 15.02.2006

[SB03] C. Stutz, A. Burghof: Software Engineering kompakt: Interne Schulungen bei sd&m. In: Proceedings der SEUH 8: J. Siedersleben, D. Weber-Wulff (Hrsg.): Software Engineering im Unterricht der Hochschulen, Workshop des German Chapter of the ACM und der Gesellschaft für Informatik e.V. (GI), Februar 2003 in Berlin, dpunkt Verlag, S. 101 - 110

[SWE05] www-swe.informatik.uni-heidelberg.de, 15.02.2006

Modellierung von Softwaresystemen - Vorlesung und Seminar im Studiengang IT-Systemtechnik

Bernhard Gröne, Peter Tabeling

Hasso-Plattner-Institut für Softwaresystemtechnik
Prof. Dr. Helmert - Straße 2-3
14482 Potsdam
{bernhard.groene,peter.tabeling}@hpi.uni-potsdam.de

Zusammenfassung: Dieser Beitrag beschreibt Lehreinheiten zur Modellierung im Studiengang IT-Systemtechnik, bestehend aus einer Vorlesung und einem ergänzenden Seminar. Die Konzentration auf große Systeme und die Nutzung von Modellen als Kommunikationsmittel sind dabei Schwerpunkte. Ausgehend von den Grundideen werden Inhalte und Struktur der Veranstaltungen sowie deren praktische Durchführung beschrieben.

1 Hintergrund

Die im Folgenden beschriebenen Lehrmodule sind Bestandteile des Studienganges „IT-Systemtechnik“ (Bachelor) am Hasso-Plattner-Institut in Potsdam. Ein Kerngedanke des Studienganges besteht in der Vermittlung von Fähigkeiten zur ingenieurmäßigen, arbeitsteiligen Erstellung komplexer Softwaresysteme [We01]. Daher sind bereits in den ersten Semestern eine Vorlesung und ein Seminar zur Modellierung von Softwaresystemen vorgesehen.

2 Die Vorlesung

Die Vorlesung „Grundlagen der Systemmodellierung“ wird als Pflichtfach im ersten und zweiten Semester gehalten [Ta05]. Dabei sind Vorlesungsinhalte und -struktur stark von zwei Leitideen geprägt:

- Betonung der Systemsicht
  Ein Leitgedanke besteht darin, Softwaresysteme nicht als umfangreiche Zusammenstellung von Programmteilen zu verstehen, sondern zunächst das eigentlich gewollte System zu betrachten, welches als dynamisches Gebilde funktionieren und mit seiner Umgebung interagieren soll. Daher werden zu Beginn Begriffe und Darstellungsmittel vermittelt, die über reine Softwaresysteme hinaus anwendbar sind und z.B. die Modellierung heterogener bzw. eingebetteter Systeme unterstützen. Daher wird auch zunächst eine Beschränkung auf bestimmte Programmiersprachen, „Paradigmen“ oder Anwendungstypen ver-

mieden. Dies erklärt auch, weshalb die Objektorientierung relativ spät behandelt wird (siehe Abschnitt 2.2).

- Modelle als Kommunikationsmittel

  Ein zweiter wichtiger Gedanke ist die Vermittlung von Modellierungstechniken zur kommunikationsfreundlichen Darstellung komplexer Systeme. Damit soll der steigenden Bedeutung von Modellen als Kommunikationsmittel in arbeitsteiligen Entwicklungsprozessen Rechnung getragen werden.

  Gerade die Entwicklung großer Systeme erfolgt oft nicht auf „der grünen Wiese“, sondern auf Basis gegebener Systeme und Systemkomponenten. Daher gilt es, auch diese zu untersuchen und das gewonnene Verständnis mittels Modellen festzuhalten und weiterzugeben. Der Analyse-Aspekt der Modellierung hat deshalb eine hohe Bedeutung in der Vorlesung und dem ergänzenden Seminar.

Neben den hier vorgestellten Veranstaltungen gibt es weitere Veranstaltungen, in denen Modellierung behandelt wird, insbesondere deren konstruktive Nutzung. (Diese Veranstaltungen liegen jedoch nicht in der Zuständigkeit der Autoren.)

2.1 Inhalte und Aufbau

Stoffauswahl und -abfolge sind von der Überzeugung geprägt, dass vor dem Vermitteln von Modellierungsnotationen eine solide begriffliche Basis gelegt werden muss. Daher werden gerade zu Beginn grundlegende Begriffe wie System, Modell und Information diskutiert. Dabei werden auch Bezüge zu formalen Sprachen und der Mathematik (diskrete Strukturen) hergestellt.

Nach der Betrachtung grundsätzlicher Systemeigenschaften (z.B. Determiniertheit, Kausalität) wird die Verhaltensmodellierung sequentieller Systeme behandelt, insbesondere das Automatenmodell. Schließlich erfolgt eine Einführung in die Modellierung nebenläufigen Verhaltens mittels Petrinetzen. Die Verhaltensmodellierung von Systemen mit großen Zustandsrepertoires erfordert die Unterscheidung von Steuer- und Operationszuständen sowie die Berücksichtigung rekursiver Abläufe – entsprechende Petrinetz-Erweiterungen werden vermittelt.

Grundtypen programmierter Systeme (prozedural, funktional, deklarativ) sowie einfache Modelle für entsprechende Abwicklersysteme werden als nächstes behandelt. Dabei wird auch eine Verbindung zu Prozessoren, virtuellen Maschinen und Betriebssystemen aufgezeigt.

Das zweite Semester ist der Modellierung komplexer Systeme gewidmet, dem Kerngedanken der Vorlesung. Eine tragende Bedeutung haben hier die Fundamental Modeling Concepts (FMC) [KT02][KW03][KGT06], mittels derer die Grundaspekte Verhalten, Wertebereich (Datenstrukturen/Entity-Relationship-Modellierung) und Aufbaustruktur vermittelt werden, sowie die Nutzung verschiedener Modelle zur Unterscheidung von Sichten, Abstraktionsebenen und Aspekten.

Es erfolgt eine Beschreibung der Grundkonzepte der objektorientierten Modellierung, wobei auch Bezüge zu abstrakten Datentypen, dem Geheimnisprinzip und der Entity-Relationship-Modellierung hergestellt werden. Davon ausgehend wird die Unified Modeling Language (UML) vorgestellt. Es werden alle Diagrammtypen behandelt, allerdings mit Schwerpunkt auf den in der Praxis vorrangig benutzten Diagrammen.

Im letzten Abschnitt der Vorlesung erfolgt eine Diskussion des Architekturbegriffs und typischer architektureller Sichten, insbesondere nach [HNS00]. Dabei wird insbesondere die Nutzung von Modellen als Kommunikationsmittel in Projekten, deren Einsatz bei der Systementwicklung und die Nutzung verschiedener Typen von Mustern behandelt.

Abschließend werden spezielle Modellierungsaspekte verteilter Systeme betrachtet, beispielsweise der Entwurf von Multitaskinglösungen mittels Petrinetz-Zerschneidung (Zerlegung eines nebenläufigen Netzes in sequentielle Teilnetze und deren Abbildung auf Tasks) oder die Darstellung von Synchronisationsmechanismen. Dabei wird auch ein Bezug zu transaktionsverarbeitenden Systemen hergestellt.

Entsprechend der zweiten Leitidee (siehe oben) stellt die Qualität von Modellen, insbesondere Modelldarstellungen, einen wichtigen Aspekt der Vorlesung dar. Er wird vor allem im zweiten Semester als Querschnittsthema behandelt. Wegen der großen Bedeutung einer guten Darstellung für die Verständlichkeit von Modellen werden u.a. konkrete Techniken zur Layout-Gestaltung, zur Verwendung von Linienstärken, Flächenaufteilung, Beschriftung oder der Anordnung von Diagrammelementen vorgestellt, bis hin zu typischen Darstellungsmustern.

2.2 Vermittlung des Stoffes

Die skizzierten Inhalte werden durch viele Beispiele untermauert. Wegen der Präsentation im Vorlesungsrahmen können dies jedoch nur kleine, tafeltaugliche Beispiele sein. Allerdings erfolgt im zweiten Semester auch ein längerer Exkurs (mehrere Vorlesungsstunden) über ein Anwendungsprojekt aus der Industrie. Auf diese Weise kann den Studenten einerseits die kombinierte Anwendung vorab erklärter Modellierungskonzepte gezeigt werden, andererseits können nur so Komplexität und Umfang von Modellierungsaufgaben aus der Praxis aufgezeigt werden.

Eine eigene - aktive - Modellierungsleistung ist zunächst im Rahmen einer flankierenden Übung zu erbringen. Mittels thematisch abgegrenzter Aufgaben kann gezielt die Anwendung einzelner Modellierungsprinzipien und -notationen geübt werden, beispielsweise die Erstellung eines Automatengraphen oder eines Klassendiagrammes, ausgehend von einer textuellen Beschreibung des Modellierungsgegenstandes.

Mittels der begleitenden Übungsaufgaben kann zwar ein Grundverständnis überprüft werden, aber keine ausreichende Erfahrung mit komplexeren Modellierungsproblemen gewonnen werden, bei denen Problemstellungen in Kombination auftreten oder eine umfangreiche Einarbeitung in ein zu modellierendes System erforderlich ist. Zur Erlangung dieser Modellierungsfertigkeiten dient daher das im Folgenden beschriebene Seminar, welches eine wichtige Ergänzung zur Vorlesung darstellt.

3 Das Seminar

Das Seminar wird optional für das vierte Fachsemester angeboten. Es ergänzt die oben beschriebene Vorlesung und bietet die Möglichkeit, das Gelernte in der praktischen Anwendung zu vertiefen. Die Teilnehmer sollen dazu komplexe technische Sachverhalte in Vortragsform und schriftlicher Ausarbeitung (Paper) vermitteln [GT05].

Mit dem Seminar soll gleichzeitig eine Vorbereitung auf die berufliche Praxis erfolgen, insbesondere die eigenständige Einarbeitung in technische Sachverhalte und deren effiziente Präsentation in Meetings und Berichten. Daher ist viel Stoff für eine begrenzte Vortragszeit bzw. ein kurzes Paper aufzubereiten, was eine kompakte Modellierung, angemessene Abstraktion und Darstellung erfordert.

3.1 Organisatorischer Rahmen und Durchführung

Im Vorfeld geben die Seminarleiter einen Themenbereich vor, beispielsweise HTTP Server oder Application Server Technologie. Einzelne Unterthemen werden zu Beginn an die Teilnehmer vergeben, wobei auch die über den Vorlesungszeitraum verteilten Termine für Vortrag bzw. Abgabe der Ausarbeitung festgelegt werden.

Jeder Teilnehmer muss zunächst die vorgegebene Dokumentation studieren und sich ggf. Information aus weiteren Quellen (z.B. Handbücher, Versuche mit dem zu modellierenden System oder Quellcode) beschaffen. Ausgehend davon sind Modelle zu erstellen und der Vortrag vorzubereiten.

Eine Woche vor Vortrag erfolgt eine Besprechung des Vortragsmaterials mit dem Betreuer. Gegenstand sind die Auswahl der Themen, die Strukturierung des Vortrags und natürlich die erstellten Systemmodelle. Oft tauchen auch offene Fragen zum beschriebenen System auf, die der Teilnehmer bis zu seinem Vortrag zu klären hat.

Nach einer ggf. erforderlichen Überarbeitung ist im Seminar der Vortrag zu halten (ca. 25 Minuten), woran sich eine intensive Diskussion mit allen Seminarteilnehmern anschließt (20 Minuten). Diese wird vom Seminarleiter in drei Teile unterteilt:

- Inhaltliche Fragen (Seminargruppe, Betreuer)

  Es können einerseits Unklarheiten bereinigt werden, andererseits kann der Seminarleiter feststellen, ob der Vortragende das Thema ausreichend tief bearbeitet und verstanden hat.
- Anmerkungen zum Aufbau des Vortrags, Darstellung, Modelle

  Dabei kann man z.B. analysieren, warum eine gewählte Darstellung oder ein Modell gut oder schlecht geeignet war, um einen Sachverhalt zu erklären. Die Seminargruppe lernt dabei, auf solche Aspekte zu achten und soll auch konstruktive Kritik üben, also Lösungsvorschläge machen.
- Anmerkungen zum Vortragsstil

Auf Basis des Vortragsmaterials und der Anmerkungen bzw. Kritikpunkte während der Diskussion ist innerhalb von zwei Wochen die Ausarbeitung zu schreiben. Diese orientiert sich bzgl. Form, Inhalt und Umfang an typischen Konferenzveröffentlichungen und kann daher auch in Englisch verfasst sein. Die Ausarbeitung soll im Wesentlichen die Struktur und den Inhalt des Vortrags wiedergeben, aber auch Anmerkungen und Fragen aus der Vortragsdiskussion berücksichtigen. Damit das zeitnah erfolgt, gilt die kurze Abgabefrist. Die Benotung der Leistung eines Seminarteilnehmers erfolgt in drei Bereichen:

- Inhaltliche Abdeckung des Themas

  Wie gut ist der Teilnehmer mit dem Thema vertraut und kann weitergehende Fragen beantworten? Erfassen die präsentierten Modelle in ausreichender Breite und Tiefe den zu beschreibenden Gegenstand?
- Aufbereitung und Vermittlung des Themas

  Sind die gezeigten Modelle leicht verständlich und gut gestaltet? Wurden sinnvolle Abstraktionen vorgenommen, die Schwerpunkte richtig gesetzt? Stimmt die inhaltliche Abfolge im Vortrag bzw. der Ausarbeitung? Wurden Notationen und Gestaltungsprinzipien beachtet?
- Vortragstechnik bzw. schriftliche Form

  Konnte der Teilnehmer in seinem Vortrag bzw. in seiner Ausarbeitung Interesse am Thema wecken und die Inhalte gut vermitteln?

3.2 Ergebnisse und Erfahrungen

Eine wesentliche Betreuungsleistung stellt die Diskussion des Vortragsmaterials eine Woche vor dem Vortrag dar. Hier gilt es, Hinweise zu geben, ohne den Teilnehmern den Raum für Eigenleistung zu nehmen. Die Erfahrung zeigt, dass gut vorbereitete Teilnehmer bereits bei dieser gemeinsamen Besprechung viel lernen.

Alle Teilnehmer mussten für ihren Vortrag eine sehr gezielte Auswahl treffen und geeignete Modelle erstellen, um das entsprechende Thema überhaupt in 25 Minuten vermitteln zu können. Der Seminarleiter benötigt hier einige Erfahrung bei der Abgrenzung der Themen, um einen angemessenen Schwierigkeitsgrad zu erreichen.

Der Zwang, bereits eine Woche vor dem eigentlichen Vortrag das Vortragsmaterial beim Betreuer vorzustellen, führte bereits im Vorfeld zu recht guten Ergebnissen. Die Qualität der Vorträge und Ausarbeitungen war daher in den meisten Fällen gut bis sehr gut. Die entstandene Sammlung von Aufsätzen über Technologie und Produkte konnten zum Teil als Technischer Bericht veröffentlicht werden [Gr04].

4 Abschließende Bemerkungen

Nach Erfahrung der Autoren erfordert eine nachhaltige Vermittlung von Modellierungswissen und -fähigkeiten einen hohen Anteil praktischer Beispiele und viel Raum für das

Sammeln eigener Erfahrungen. Dies gilt in besonderer Weise für die Erstellung von Modellen als Kommunikationsmittel, was einen Schwerpunkt der vorgestellten Lehreinheiten darstellt. Hier sind oft pragmatische Vorgehensweisen gefragt, die sich nur begrenzt als allgemeine Richtlinien formulieren bzw. lehren lassen. Eine Modellierungsvorlesung sollte daher stets mit einem Seminar - ähnlich dem oben vorgestellten - ergänzt werden. Eine ausreichende Komplexität kann durch die Integration von umfangreichen Fallbeispielen bzw. die Modellierung gegebener Systeme aus der Industrie erreicht werden.

Als Problem mag die frühe Positionierung der Lehreinheiten im Studienablauf betrachtet werden. Man mag sich fragen, ob die Studierenden nicht erst fundierte Programmierkenntnisse erlernen sollten, um zunächst den Gegenstand der Modellierung besser kennenzulernen. Für die gewählte zeitliche Platzierung spricht jedoch die frühzeitige Prägung der Studierenden, die - in der Folge - das Arbeiten mit Modellen als selbstverständlichen Zugang zu Softwaresystemen verinnerlichen und Modellierung nicht als „Soft Skill“ betrachten, die dem Schreiben großer Programme nachgeordnet ist. Die guten Erfahrungen mit den ersten Absolventenjahrgängen bestätigen diesen Ansatz.

Literaturverzeichnis

[Gr04] Gröne, Knöpfel, Kugel, Schmidt: The Apache Modelling Project. Technischer Bericht Nr. 5 des Hasso-Plattner-Instituts für Softwaresystemtechnik, Potsdam, 2004

[GT05] Bernhard Gröne, Peter Tabeling: Materialien zum Seminar Systemmodellierung. wendtstud1.hpi.uni-potsdam.de/sysmod-seminar, Stand vom November 2005

[HNS00] Hofmeister, Nord, Soni: Applied Software Architecture, Addison Wesley Longman, 2000

[KGT06] Knöpfel, Gröne, Tabeling: Fundamental Modeling Concepts - Effective Communication of IT Systems. Wiley and Sons, Februar 2006

[KT02] Keller, Tabeling et. al. Improving Knowledge Transfer at the Architectural Level - Concepts and Notations. The 2002 International Conference on Software Engineering Research and Practice, Las Vegas USA, 2002

[KW03] Frank Keller, Siegfried Wendt: FMC - An Approach Towards Architecture-Centric System Development. Proceedings of 10th IEEE Symposium and Workshops on Engineering of Computer Based Systems, Huntsville USA, 2003

[Ta05] Peter Tabeling: Softwaresysteme und ihre Modellierung - Grundlagen, Methoden und Techniken. Springer Verlag, Heidelberg, 2005

[We01] Siegfried Wendt: Softwaresystemtechnik - eine Informatik-Ingenieurdisziplin. Buchbeitrag in: „Das ist Informatik“, Springer, 2001

Collegium Logicum 2006: Fundierung der Modellierung in Lehre und Weiterbildung

Elisabeth Heinemann

Wirtschaftsinformatik I: Entwicklung von Anwendungssystemen
Technische Universität Darmstadt
Hochschulstr. 1
64289 Darmstadt
heinemann@winf.tu-darmstadt.de

Abstract: Viele Jahre schon gehört das *Collegium Logicum*, also der Logikunterricht nicht mehr zum gelehrten Fächerkanon unserer Schulen. Und als vor etwa ca. 150 Jahren die Mathematik Anstalten machte, dieses elementarste Grundlagenfach aller spezifizierenden Wissenschaften wieder aus der Versenkung hervorzuholen, vergaß man jedoch die schulische Bildung mit einer praktischen Logik anzureichern. Dabei ist es gerade sie, die uns hilft, unsere Welt sprachlich zu erfassen [Lore68]: „In der Logik handelt es sich um Operationen, die statt mit Zahlen mit Gedanken, konkret gefasst, mit sprachlichen Aussagen vorgenommen werden." Und das sprachliche Erfassen, dieses „how to specify (model) the world" wird der Mensch von morgen brauchen, um die Möglichkeiten einer unbegrenzt vernetzten Welt zu nutzen, und sich darin nicht unhaltbar zu verstricken. Deshalb ist dieses Papier ein Plädoyer für die Fundierung der Modellierung in Lehre und Weiterbildung auf der Grundlage einer praktischen Logik.

1 Einleitung

Deutschlands Hochschulen kämpfen. Sie kämpfen um international renommierte Lehrkräfte, Mittel und Reputation. Zu guter Letzt aber vor allem um qualifizierte Studenten. Die ganze Situation erinnert ein wenig an einen Gladiatorenkampf, bei dem alle auf das Urteil der Politik und anderer zuständiger Institutionen warten: Daumen hoch oder Daumen runter. Elite-Universität, Exzellenz-Cluster, Graduiertenschulen & Co. sind die Schlagworte, mit denen sich die Hochschulverantwortlichen, administrierend wie lehrend, im wahrsten Sinne des Wortes herumschlagen. Nach Außen bemüht man sich um Qualität, nach Innen kämpft man dagegen verzweifelt um den Erhalt lieb gewonnener und teils mühsam erkämpfter Privilegien, sprich (Dritt-)Mittel und Landesbedienstetenstellen. Und die Bedrohung ist durchaus real, vielleicht viel realer als den meisten „Gladiatoren" bewusst sein dürfte. Denn längst verschiebt sich der Schauplatz des Wettbewerbs um die fähigsten Studenten – Grundlage einer jeden universitären Existenz- und Exzellenzberechtigung – auf den Cyberspace [HiTu05]: „Online learning is a new social process that is beginning to act as complete substitute for both distance learning and the traditional face-to-face class." Warum sollte der hochtalentierte und motivierte Nachwuchs städte- oder sogar länderübergreifend der Bildung nachreisen, um am gewählten Studienort „nur" in ausgewählten Fächern die Besten der Besten als Lehrende offeriert zu bekommen, wenn er mit ein paar „Mausklicks" online bei der Weltelite seines Faches studieren kann? Denn die Zukunft wird *uns* verändern, wird *unseren Umgang* mit den Dingen, die uns umgeben und mit Hilfe derer wir unser tägliches Leben bewältigen, verändern.

Inzwischen geht es z.B. bei der Entwicklung von Anwendungssystemen nicht mehr nur um die Implementierung von Software, sondern in gleichem Maße um die Spezifizierung jenes Wissens, welches die Anwender erworben haben müssen, um bei der Interaktion mit Computern die Anwendungssysteme überhaupt so nutzen zu können, wie es diese vorsehen [Ort06]. Und die Systeme entwickeln sich vernetzt wie nie zuvor. Semantische Netze, Wikipedia, Google und ebay, E-Voting und E-Ticketing, etc. verlangen vom „im Netz der Netze lebenden" Menschen in immer stärkerem Maße logik- und sprachbasierte (Vor-)Kenntnisse. Diese zu vermitteln muss Aufgabe unserer Schulen sein.

2 Lösungsansatz: Modelle und Sprachen

Wir erschließen uns unsere Welt mit Sprache, indem wir diese instrumentalisieren. Dieses umfassende Spezifizieren bzw. Modellieren gilt es deshalb – bezogen auf den Spracheinsatz – von frühester Jugend an zu erlernen. Denn wenn *wir* in der Lage sind, unsere Welt auf diese Weise zu erschließen, dann kann es auch eine von Menschen gebaute und mit sprachlichen „Weltsichten" gefütterte Maschine [Hein06]. Ein Weg diesen Spagat zu bewältigen ist der Einsatz einer *logik-basierten Rationalsprache* als Lehr- bzw. Demonstrationssprache, ein Ansatz, der bereits in den 1970er Jahren von Kamlah/Lorenzen mit ihrer „Logischen Propädeutik" in einer Auflage von 30.000 Exemplaren verbreitet wurde [KaLo90]. Doch warum sollten wir auf die Rekonstruktion einer logik-basierten Rationalsprache überhaupt eingehen, lernen doch die Kinder in der Schule durchaus ihre eigene Muttersprache? Und sind wir nicht auch als Erwachsene – mit gewissem Abstand zu unserer Schulzeit – in der Lage sprachlich zu kommunizieren? Diese Fragen können wir mit einem „klaren Jein" beantworten. Unsere Kommunikationsmittel beherrschen wir, doch der Vorteil des Vertrautseins mit der eigenen Sprache wird eingeschränkt durch eine gewisse Betriebsblindheit, die uns allmählich zu Eigen geworden ist. Der Sprachforscher Steven Pinker bringt es auf den Punkt [Pink96]: „Die Umgangsprache ist – genau wie das Farbensehen oder das Gehen – eine technische Meisterleistung, die so reibungslos funktioniert, dass der Anwender ihre Beherrschung als selbstverständlich annimmt und sich der komplizierten Maschinerie, die sich hinter der unauffälligen Oberfläche verbirgt, gar nicht bewusst wird."

Wie können wir also einem unreflektierten Gebrauch der eigenen Sprache und der ihr immanenten Logik entgegenwirken? Es bietet sich die Informatik an, die unsere Lebensbereiche – privat wie beruflich – jeden Tag ein wenig mehr durchdringt, langsam aber stetig. Deren Möglichkeiten, eine Sprache zu rekonstruieren und somit für die weitere Verarbeitung durch einen Rechner modellierbar zu machen, bietet auch dem Benutzer entsprechender Anwendungssysteme die Chance, die seiner Sprache zugrunde liegenden Prozesse und Fähigkeiten zu erkennen, zu durchschauen und möglicherweise auch qualitativ besser einzusetzen [Hein06]. Um das Sprachverstehen zu fördern, ist es also offensichtlich hilfreich, die (normative) Genese einer Sprache zu verstehen, d.h. wie sie kontrolliert entsteht.

Freilich können sich Menschen in einer Rationalsprache nicht unterhalten, dazu ist sie auch nicht gedacht. Sie ist vielmehr als eine Art *Praxis-* oder *Lehrsprache* mit einer rationalen Syntax und Semantik aufzufassen, die ihren Sitz in der Pragmatik, also im bewussten, sprachgesteuerten Handeln hat und Verständnis für die empirische Sprache

wie das Deutsche oder Italienische schaffen soll [Wede03]. Denn Menschen können mittels einer Rationalsprache leichter eine methodische Rekonstruktion unserer gesprochenen Sprachen und der mit ihnen formulierten Sprachartefakte, aber auch (als Experten) die Modellierung vornehmen. Und das wiederum schafft auch Verständnis und Fähigkeit im Umgang mit den Medien des Cyber-Zeitalters [Witt73]: „Eine Sprache erfinden könnte heißen, auf Grund von Naturgesetzen (oder in Übereinstimmung mit ihnen) eine Vorrichtung zu bestimmtem Zweck erfinden; […] Ich sage hier damit etwas über die Grammatik des Worts »Sprache« aus, indem ich sie mit der des Wortes »Erfinden« in Verbindung bringe."

3 Die Orientierung

Es gilt, unsere Jugend zu einer die Möglichkeiten des weltweiten Netzes bestmöglich nutzenden Gemeinschaft heranzubilden. Denn dass die heute 6- bis 14-jährigen im Jahre 2020 andere Lebensumstände vorfinden werden als die Erwachsenen unserer Tage, wird niemanden ernsthaft bezweifeln können [Netz05]. Rationalsprachliche Aspekte und die einer allgemeinen bzw. praktischen Logik sollten hierbei nicht (nur) unbedingt als eigenes Fach, sondern viel eher als roter Faden durch die Lehrinhalte des existierenden Fächerkanons laufen. Wünschenswert wäre eine Ergänzung um Aspekte der praktischen Logik und das möglichst früh. Ziel ist es, zu verstehen, was z.B. auch in der Modellierung geschieht, wenn wir „eine Sprache in die Welt setzen". Hilfreich hierbei ist der Einsatz einer *Rationalen Grammatik* als Lehr- bzw. Demonstrationssprache. Sie ermöglicht eine sprachbildnerische Herangehensweise, die zwei wesentliche Aspekte berücksichtigt: die *Sprachkompetenz* im Sinne Chomskys und – mit Blick auf die *Praktische Logik* (oder z.B. auch auf eine Modellierungssprache wie der UML) –, methodologisch bzgl. des Vorgehens gesehen die *Sprachperformanz*. Denn offensichtlich brauchen wir für die Meisterung der Zukunft einen „neu gebildeten" Menschen, der seine Umwelt genauer als bisher spezifizieren kann! Dies ist übrigens kein völlig neuer Aspekt, denn bereits der Paläanthropologe Derek Bickerton beantwortete die Frage „Half die Grammatik dem Menschen, bessere Werkzeuge oder Geräte zu bauen?" mit einem klaren „ja" [Bick03].

Wer modelliert oder auch konstruiert, der spezifiziert die Dinge um sich herum aufgrund der bereits als Kind entwickelten Sprachbegabung. Die Wirtschaftsinformatik, die das Spezifizieren mit Recht zu einer ihrer Kernaufgaben zählen darf und daher professionell ausübt, bietet mit einer aus ihr hervorgegangenen, sprachbasierten *Spezifikationstheorie* die Grundlage für zwei Entwicklungen, die der gewünschten Stoßrichtung den richtigen Schwung geben könnten [Ortn06]:

- Sie bildet zum einen eine *Orientierung* für die neue Situation in unseren Schulen bzw. allgemein gesprochen unseren Bildungsstätten beim Erlernung der individuellen Sprachbenutzung [HiTu05] und [Hein06] und
- zum anderen liefert sie ein Modell zur Implementierung der Unterstützung dieser Sprachbenutzung mittels einer auf die Bedürfnisse und das Können des Anwenders reagierenden, quasi „mitdenkenden", dynamischen (Zwischen-) Sprache im weltweiten Netz [Ortn05].

Ersteres soll nun an einem konkreten Unterrichtsbeispiel gezeigt werden, dass sich, ebenso wie die vollständige Erläuterung einer Rationalsprache in einem bildungsrelevanten Kontext in [Hein06] findet.

4 Vertiefung und Ausblick: Ein Unterrichtsbeispiel

Im Ausblick dient ein Vorschlag zur Gestaltung einer Unterrichtsstunde zum Thema *Namensgebung und Kennzeichnung* wie sie in [Hein06] näher ausgeführt ist.

4.1 Das Thema

In unserem täglichen Leben werden wir auf unterschiedlichste Art und Weise mit *Namen* ebenso wie mit *Kennzeichnungen* konfrontiert. Eingängigstes Beispiel sind wir wohl selbst: wir haben einen Namen ohne den wir – standesamtlich festgehalten – überhaupt nicht erst auf diese Welt losgelassen werden, und es würden uns darüber hinaus bestimmt einige Kennzeichnungen einfallen, mit denen wir mehr oder weniger eindeutig identifizierbar sind.

Stellen wir uns einmal vor, wir laden alle 28 Herren Meier, Meyer, Mair, Maier und Mayer, die den Vornamen Stefan haben und im Telefonbuch der Stadt München stehen ein und bitten sie in einem Raum Platz zu nehmen. Die Anwesenden kennen die Namen ihrer Mitstreiter natürlich nicht. Nun kommen wir herein und rufen „Wer von ihnen ist Herr Stefan Meier? Ich bitte um Handzeichen.“ Die Aussprache der Namen aller anwesenden Herren ist gleich, so dass sich mit sehr großer Wahrscheinlichkeit jeder von ihnen melden wird. Obwohl ein Name vorhanden ist, war es doch offensichtlich nicht möglich, durch ihn eine eindeutige Identifizierung der gesuchten Person durchzuführen. Das liegt daran, dass es sich bei unsrem Beispiel „Stefan M{ey,ei,ai,ay}(e)r“ um keinen *echten Eigennamen* handelt, da er den Eigenschaften desselben nicht vollständig genügt. Erst wenn *Existenz* und *Eindeutigkeit* vorliegen, können wir von einem echten Eigennamen sprechen im Sinne von „Keine Entität ohne Identität“ oder anders formuliert [WOI04b]: „In der rationalen Grammatik werden Benennungen auf solche Eigennamen beschränkt, die nach Vereinbarung genau einen Gegenstand sprachlich vertreten.“

Eigennamen können unabhängig von ihrer „Echtheit“ in *sprechend* und *nichtsprechend* unterteilt werden. Im Beispiel „4711 ist ein Klassiker“ wird „4711“ für die meisten von uns einen *sprechenden Eigennamen* darstellen, bei dem wir sofort wissen, dass von dem berühmten Kölner Duftwasser die Rede ist. Wenn ich aber aussage „Der Artikel mit der Artikelnummer 4711 ist ein Klassiker“, kann hinter der Artikelnummer 4711 theoretisch alles versteckt sein, denn aus dem Kontext wird nicht klar, über welchen Gegenstand hier etwas ausgesagt wird. Es handelt sich also in diesem Fall um einen *nichtsprechenden Eigennamen*. Bezogen auf das sprachliche Handeln, also die Anwendung von Sätzen können wir mit Mittelstraß sagen [Mitt74]: „Eigennamen sind Wörter, die in Sätzen denjenigen Gegenstand vertreten, über den jeweils prädiziert werden soll; d.h. sie vertreten strenggenommen einen Gegenstand mitsamt jener auf ihn hinweisenden Geste, mit der man sich vor der Einführung von Eigennamen behilft.“

Eine *Kennzeichnung* hingegen ist so etwas Ähnliches wie eine Beschreibung. Wenn uns jemand etwas beschreibt, dann wissen wir zumeist auch, welches Objekt (Gegenstand, Mensch, et.) damit gemeint ist, ohne das es einen „echten", das heißt, einen eindeutig identifizierenden Namen hat. Zieht man ein Elementarsatzschema (vgl. z.B. [WOI04a]) als Basis heran, so ergibt sich für den Eigennamen, die Kennzeichnung und den Prädikator, als „Hauptperson" des Elementarsatzes, eine Zuordnung zu den auf einen Satz bezogenen Grundeigenschaften *bezugnehmend* und *beschreibend* (s. Abbildung 1):

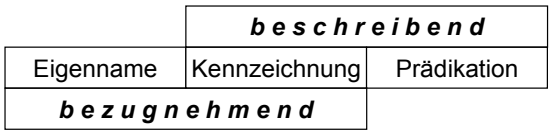


Abbildung 1: Eigenname, Kennzeichnung und Prädikation

4.2 Zielgruppe und -setzung

Der folgende Ablauf zeigt eine mögliche Übung zum Thema *Namensgebung und Kennzeichnung* auf, wie sie sowohl mit Grundschulkindern als auch mit älteren Schülern gut vorstellbar ist. Die Kinder sollen hierbei

- den Unterschied zwischen *Name*, *Kennzeichnung* und *Erklärung* verstehen,
- erkennen, wie wichtig Bezeichnungen im täglichen Leben sind bzw. auch welche Gefahren möglicherweise in einem nicht korrekten oder exakten sprachlichen Umgang mit Namen und Kennzeichnungen liegen,
- und diese Erkenntnisse auf den Umgang mit Rechnern und insbesondere den mit dem Internet übertragen.

4.3 Übungsdesign

Vorbereitungsphase: Zur Vorbereitung werden so viele Namenskarten erstellt wie Schüler am Unterricht teilnehmen. Auf den Kärtchen steht jeweils einer der folgenden Namen: Maier, Meyer, Meier, Mayer, Meir und Mair. Je nach Schülerzahl können die einzelnen Namen natürlich mehrfach auftreten. Die Namenskärtchen werden mit einem Klebestreifen unter den Stühlen der Schüler befestigt, bevor diese das Klassenzimmer betreten.

Einstieg („*Warm-up*"): Nachdem die Kinder eingetreten sind, sich gesetzt haben und die allgemeine Begrüßung erfolgt ist, gibt die Lehrerin folgende Anweisung: „Ihr findet auf der Unterseite eurer Stühle Namenskärtchen. Nehmt es euch, aber lest es so, dass kein anderer den Namen darauf lesen kann. Ich rufe euch dann einzeln auf, und wer „seinen" Namen von dem Kärtchen hört, der meldet sich bitte und ruft „Hier". Die Schüler erkennen noch keinen Sinn in der Aktion. Erst nachdem die Lehrerin irgendeinen der Namen, die phonetisch alle gleich klingen, aufruft und sich daraufhin alle Schüler melden, gibt es Verwirrung. Vermutlich werden die Kinder anfangen, die eigenen Kärtchen mit denen der Mitschüler zu vergleichen und selbsttätig „zu des Rätsels Lösung gelangen".

Erklärung (*Lehrervortrag*): Es erfolgt eine Beschreibung echter und potenzieller Kennzeichnungen und eine Erläuterung von Eigennamen. Jeder Begriff wird mit Beispiel und Gegenbeispiel eingeführt und jeweils erfragt, warum das so sinnvoll ist.

Gruppenarbeit: es werden Zweiergruppen gebildet, die zusammen folgende Arbeitsaufgaben bearbeiten: Beispiele finden für echte und potenzielle Kennzeichnungen und Klärung der Frage, was deren Nachteile bzw. Vorteile sind. Anschließend nennt jede Gruppe je zwei Beispiele vor der gesamten Klasse, welche beurteilen soll, ob diese korrekt sind oder nicht. Die Kinder üben im Team, zu einer gemeinsamen Lösung zu kommen und vertiefen zusammen ihr Verständnis des gelernten Stoffs (z.B. [Matt02]). Darüber hinaus wird die Aufmerksamkeit erhöht, wenn die Gruppen ihre Ergebnisse vorstellen und die Beurteilung der Klasse gefordert ist. Die daraus entstehende Diskussion unterstützt den Verständnisprozess.

Überdies bietet diese Einstiegsübung die Möglichkeit, auf weiterführende Aspekte der Modellierung einzugehen wie etwa das Heranziehen von (identifizierenden) Schlüsseln.

5 Literaturverzeichnis

[Bick03] Bickerton, D.: Was ist ein „Was“?. In: DER SPIEGEL, Nr. 43, S. 223-228, 2002.

[Hein06] Heinemann, E.: Sprachlogische Aspekte rekonstruierten Denkens, Redens und Handelns. Aufbau einer Wissenschaftstheorie der Wirtschaftsinformatik. Dtsch.-Univ.-Verlag, Wiesbaden, 2006.

[HiTu05] Hiltz, S. R.; Turoff, M.: Education Goes Digital: The Evolution of Online learning and the Revolution in Higher Education. In: Comm. of the ACM, 48 (2005) 10, S. 59-64.

[LaLo90] Kamlah, W.; Lorenzen, P.: Logische Propädeutik – Vorschule des vernünftigen Redens. BI-Wiss.-Verlag, 1990

[Lore68] Lorenzen, P.: “Collegium Logicum”. In: Methodisches Denken. Suhrkamp Verlag, Frankfurt/Main, 1968.

[Matt02] Mattes, W.: Methoden für den Unterricht: 75 kompakte Übersichten für Lehrende und Lernende. Schöningh, Paderborn, 2003.

[Mitt74] Mittelstraß, J.: Die Möglichkeit von Wissenschaft. Suhrkamp Verl., Frankfurt/M. 1974.

[Netz05] Netzwerkkinder: „So werden wir leben“. http://www.netzwerkkinder.at/netzwerkkinder2004/menu.php?flash=1. Abruf am 15.04.2005.

[Ortn05] Ortner, E.: Sprachbasierte Informatik. Wie man mit Wörtern die Cyber-Welt bewegt. EAGLE Verlag, Leipzig 2005.

[Ortn06] Ortner, E.: Making it explicit: Eine universelle Spezifikationstheorie für die Entwicklung von Anwendungssystemen im Internet. Veröffentlichung geplant für 2006.

[Wede03] Wedekind, H.: „Sprachbasierte Informatik und Wirtschaftsinformatik". In: WIRTSCHAFTSINFORMATIK, 45(2), S. 251-252, 2003.

[WOI04a] Wedekind, H.; Ortner, E.; Inhetveen, R.: „Informatik als Grundbildung – Teil II: Bildung von Elementarsätzen“. In: Informatik-Spektrum, Juni 27(3), S. 265-272.

[WOI04b] Wedekind, H.; Ortner, E.; Inhetveen, R.: „Informatik als Grundbildung – Teil V: Namengebung und Kennzeichnung“. In: Informatik-Spektrum, 27(6), S. 551-556, 2004.

[Witt73] Wittgenstein, L.: Philosophische Grammatik. Suhrkamp Verlag, Frankfurt/Main, 1973.

[Witt84] Wittgenstein, L.: Tractatus logico-philosophicus, Tagebücher 1914-1916, Philosophische Untersuchungen. Suhrkamp Verlag, Frankfurt/Main, 1984.

An Execution Model for Teaching Story Diagrams

Ira Diethelm, Leif Geiger, Christian Schneider, Albert Zündorf

University Kassel

Abstract: In teaching a programming language, one frequently uses some machine model illustrating the execution of a program. In case of a usual object oriented programming language this machine model might consist of objects stored in a heap, a procedure call stack, local variables, a program counter, etc. Due to our opinion, this level of abstraction is too low to really leverage the learning of an object oriented programming language. On the other hand, the usual object oriented modelling metaphor of objects that exchange messages is not detailed enough for learning programming. In the Fujaba project, we use story diagrams, a new graphical programming language that combines UML activity diagrams and UML object diagrams with the semantics of graph transformations. To facilitate the teaching of this new language we developed a new execution model on an appropriate level of abstraction. This execution might also be helpful for learning usual object oriented programming languages.

1 Introduction

For the illustration of the execution of usual imperative programming languages one frequently employs an abstract machine model consisting of a heap, a procedure call stack with frames for parameters and local variables and a program counter, etc. To illustrate a walk through, e.g. at the black board, one may also use some trace table that shows variables and their current values. In object oriented programming, the heap content is usually depicted as UML object diagram. Attribute values are shown within the boxes and links represent pointers. These common abstract machine models are inspired by real machines and by compiler construction techniques. They model the execution on the level of abstraction induced by current programming languages.

On the model level of abstraction one frequently uses object diagrams to depict the current system state. Execution uses the metaphor of message sending between objects. The receiving object executes the corresponding request and sends back some result. Unfortunately, this execution model lacks appropriate means for the illustration of recursive method calls, for the representation of trace tables for local variables or for program counters. In teaching Fujaba graph transformations [Fu02], we additionally have the problem to illustrate the graph pattern matching process.

To facilitate the teaching of object oriented modelling, of model execution and of story diagrams, we therefore developed a more detailed execution model for object oriented models and for story diagrams. This model elaborates the objects send messages metaphor

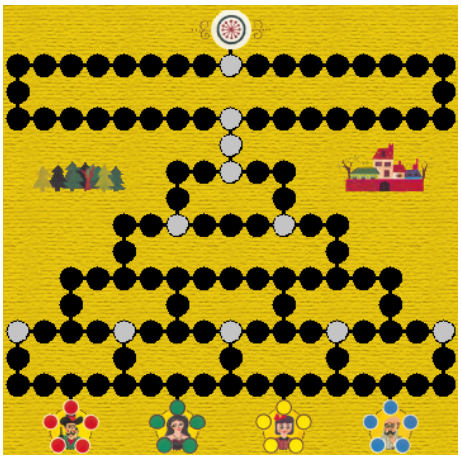


Figure 1: The Malefiz board

and adds features illustrating recursion, program counters and multi threading.

Our work has been inspired by [STSN02]. However, our approach does not employ class diagram elements, actively. This paper is based on our work on story driven modeling [DGZ04b, DGZ05] where our modelling process is elaborated. [DGZ05b, DGZ05d] report on our general teaching approach for object oriented modelling and programming and for story diagrams. [DGZ05b, DGZ05d] also contain first elements of our execution mechanism, especially they introduce our use of post-its. Some simple post-it animations may also be found in [Fu04]. This paper elaborates these ideas and adds an explicit execution mechanism to our teaching approach.

2 Running Example

As a running example for this paper we use a simple model of the board game Malefiz, cf. Figure 1. The left of Figure 2 shows a UML object diagram modelling some fraction of a game situation where player `ira` has thrown a `3`. In the depicted situation, player `ira` has just decided to move her stone `stone1` from field `f1` to field `f11`. We assume, that player `ira` has clicked some items at the graphical user interface that now result in the invocation of method `move` with parameter `f11` on object `stone1`. This is depicted by the corresponding collaboration message in Figure 2.

The `move` method has to check whether the stone is able to reach the desired field with the given die value and then it actually moves the stone. This behavior is specified within the two story diagrams on the right of Figure 2. In the following, we will use our new story diagram execution model to explain how method `move` and the helper method `isReachable` work.

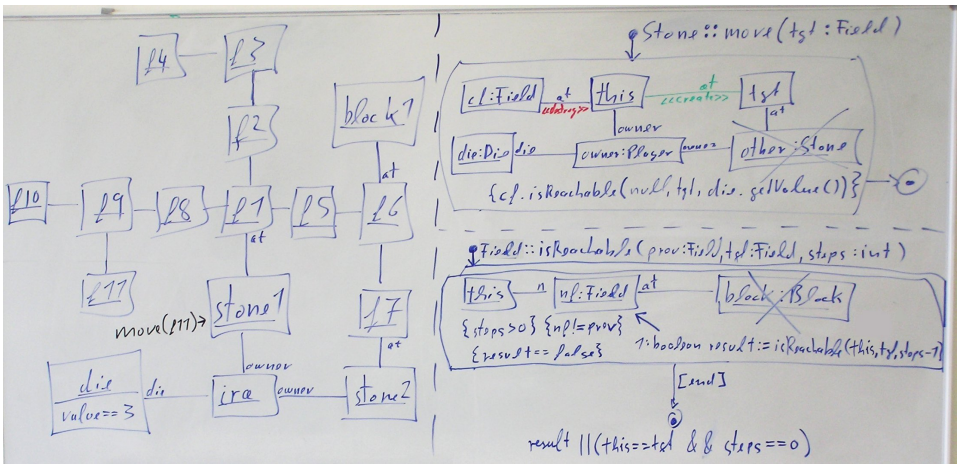


Figure 2: Object Diagram and Story Diagrams for a cutout of a Malefiz game

3 The Story Diagram Execution Model - Stem

Our STory diagram Execution Model `Stem`[1] relies on the usual object model. Thus, the (whole) system state is represented by a set of objects, cf. the left of Figure 2. Objects may have attributes with current values. Objects may refer to each other using links.[2]

In principle, Stem sticks to the metaphor of objects collaborating via message exchange. However, due to our experiences, for many students this execution metaphor is not detailed enough. If one tries to model some situation just with objects that exchange messages, he[3] frequently is misguided by his overview of the whole object structure. Due to this overview, for the students it is quite obvious that the `move` invocation in Figure 2 is valid and that all that has to be done is to create an `at` link between `stone1` and field `f11` (and to delete the `at` link between `stone1` and `f1`). They do not recognize, that the validity check is a quite complex operation. Thus, due to their overview, the students frequently fail to identify the complexity of certain execution steps. In turn, the students frequently come up with models that are inappropriate for the implementation of the desired functionality.

In order to force our students to change their perspective from the overview of the whole model to the restricted view of a program execution mechanism, this paper extends the usual ”objects exchange messages” metaphor. We assume, that each object is inhabited by some oblivious little guy. This little guy is very lazy i.e. on default he does nothing. He becomes active only when he receives an explicit command, e.g. the `move(f11)` command. For each possible command, the object guy has a tray with instruction sheets (i.e. story diagrams). If he receives a command, he takes one instruction sheet from the

[1] Stem is just easier to pronounce than Sdem

[2] Note, this model may easily incorporate aspects of graphical user interfaces or terminals. The user interface and its components are just modelled as additional objects belonging to the overall set of objects.

[3] This holds similarly for both genders. This paper lazily uses only the male gender. The reader might equally replace he and his with she and her, respectively.

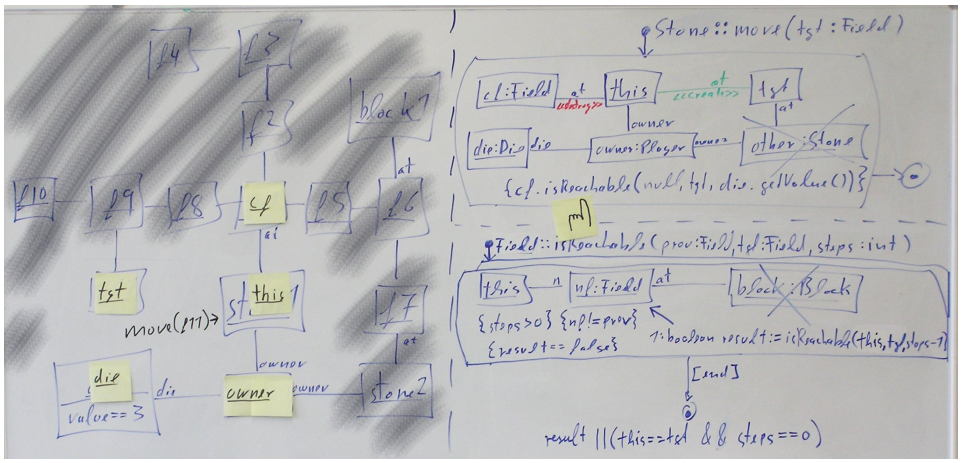


Figure 3: Object Diagram and Story Diagrams for a cutout of a Malefiz game

corresponding tray and starts to execute it. In our case the `stone1` guy would start to execute method `move`, cf. Figure. 2.

The object guy is totally oblivious and does not recall his surrounding from previous command executions. All he knows is that from his point of view the object he is living in is named `this`. In addition, he might get knowledge via parameters passed with the command invocation, e.g. the `tgt` object passed in the `move(f11)` message. All the remaining objects are hidden to him in some fog of oblivion. In the lecture, this fog of oblivion might be added to the slide or to the white board using some appropriate shading, cf. Figure 3. In our example, the `stone1` guy now has to execute the graph transformation shown in the activity box in the upper right of Figure2. A Fujaba graph transformation is executed by matching its object structure to the runtime object structure. Thus, the object guy has to identify a `cf`, a `this`, a `tgt`, a `die`, and an `owner` object. As already mentioned, the object guy names the object he is living in `this`. To illustrate this view, we use correspondingly labeled post-its, cf. Figure 3. In addition, in the command invocation, object `f11` has been passed as actual parameter for the formal parameter `tgt`. Thus, for the object guy object `f11` has name `tgt`. In the graph transformation, the object attached to the `this` object via an `at` link has name `cf`. Thus object `f1` gets the name `cf`. Accordingly, `ira` becomes the `owner` object and the `die` object is named `die`, again, cf. Figure 3.

Note, our execution metaphor does not employ variables. Using aliases instead of variables turned out to be simpler for our students. Since our execution metaphor does not employ an abstract machine with a procedure call stack where variables may be seen as names for certain memory cells, we just do not need that term. Accordingly, parameter passing is introduced as an aliasing mechanism. Similarly, we introduce local variables as some kind of names that refer to certain values that may change during the execution. The computer science expert might correctly think that this exactly characterizes a variable. However, our students get acquainted to our name metaphor significantly faster than they

get acquainted to the term variable in earlier teaching approaches.

As already mentioned, our execution metaphor does not use an explicit procedure call stack. However, for (reentrant) method invocations we need to provide new name spaces. In addition, once a method invocation has been completed, we have to continue the execution after the point of its invocation. We achieve this, by extending the message exchange metaphor, again. If an object guy has to invoke some method, he prepares a corresponding command letter with two addresses, one for the receiver of the message and one for the sender and then he transmits this letter. At this point in time, our execution metaphor has to deal with concurrency. In case of a single thread, execution has to continue at the receiver of the current messages. On method return, execution continues at the invocation point. At the first glance this contradicts to our execution metaphor where each object is inhabited by its own object guy and where all these object guys could work in parallel. In addition, we want to use our execution metaphor also for the discussion of multi threaded executions. Thus, our execution metaphor needed representees for threads. Therefore, we introduced so-called postmen. Each command letter is handed to a postman. This postman travels to the receiving object, enters it and hands the message to the inhabitant. Then, the postman waits for the reply. If the inhabitant wants to send a subsequent message, this is again done by the same postman. If a command execution completes, the corresponding object guy may write a result on the corresponding letter and then the postman brings this letter back to its sender. The sender guy is still waiting for the reply with his finger still pointing to the method invocation on his current instruction sheet. Thus, in our metaphor, the chain of letters forms the procedure call stack. Each object guy uses his own set of names (or post-its), this introduces name spaces. The pointing fingers of the participating object guys represent the program counters. Note, if the postman reenters an already active object, we have to extend our metaphor, again. In this case, we assume that each object is inhabited by a family of object guys and the new command will be executed by another lazy inhabitant (with another name space and pointing finger) while the already active object guy continues to wait for the expected reply with his finger pointing to the current instruction.

In our example, the object guy executing method `move` has now to check the textual constraint cf.isReachable(). This is a (recursive) method invocation, thus he prepares a corresponding letter and sends the postman to object `cf`. Note, the postman takes care of parameter passing. This means, if the `stone1` guy uses his local names `cf` and `tgt`, the postman knows which objects are meant and when he reaches the receiver object, he tells the receiving guy, which object belongs to which formal parameter. In our example, the guy living in object `cf` (or originally `f1`) names his object $\texttt{this}_2$. Note, in order not to mix-up the name spaces of the `stone1` guy and the `cf`/`f1` guy, one may either use different colors for the post-its as in Figure 4 or one may use indices for each nesting level. Accordingly, the `f11`/yellow `tgt` object becomes the green `tgt` object. For the green parameter `steps`, we use a post-it assigning this name to the value `3`. In addition, we use a green `result=false` post-it to illustrate the initialization of the corresponding flag.

Now the `f1`/yellow `cf`/green `this` guy executes the graph transformation in the body of method `isReachable`. Thus, he looks for `nf` objects reached via `n` links. In our

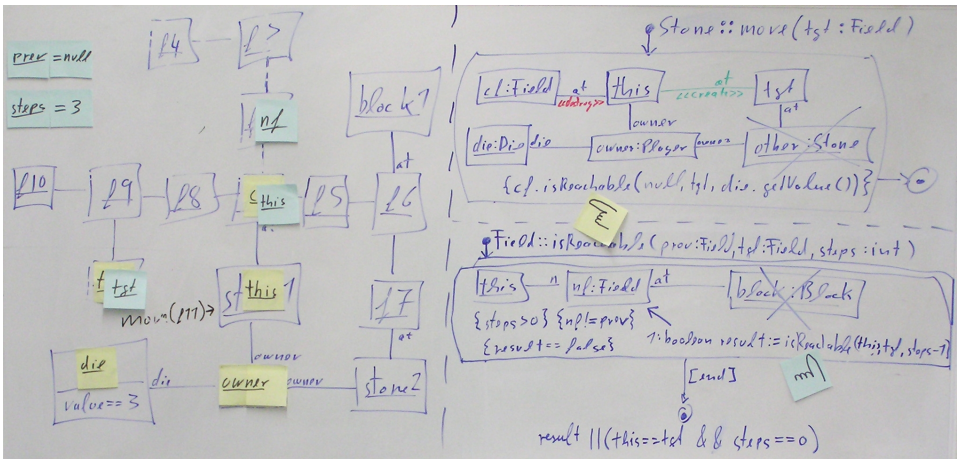


Figure 4: Illustration of a recursive call chain

example there are three possible candidates `f2`, `f5`, and `f8`. As first try, the green `this` guy may assign green `nf` to object `f2`. In this case, the simple constraints are satisfied and since `f2`/green `nf` has no blocker[4] the depicted collaboration message `1:` is issued. This implies a call to method `isReachable` on object `f2`/green `nf`. Thus, the postman is send forward. The inhabitant of object `f2`/green `nf` names his object red `this` and so on. Recursion stops when the postman reaches `f4`. At `f4` our object structure has no more candidate for `nf` and blue `steps` equals `0`. The return expression evaluates to `false`. This reply is written on the current command letter and the postman walks back the call chain until he reaches `f1`/`cf`/green `this`. There the green `this` guy is still waiting for the reply with his finger on the instruction sheet. Note that the activity in the body of method `isReachable` is a so called for-each activity. Such activities iterate through all possible name assignments. Thus, when the postman comes back to green `this` with a negative reply, green `this` just assigns name `nf` to the next candidate. As soon as objects `f8`, `f9`, and `f11` are visited, the return expression evaluates to `true`. Thus, the postman carries this positive result back to object `stone1`/`this`, where the first object guy of our execution still sits with his finger on the textual constraint. Since the reply to the method call is `true`, the constraint holds and the actions are executed, i.e. the `at` links are modified as desired. Then the postman walks back to the user.

4 Summary

As one might notice, checking the validity of the user command is a quite complex operation. From their overview perspective, students tend to ignore this kind of complexity and thus they tend to produce insufficient models. Our object guy metaphor forces our

[4] A crossed out object represents the constraint that it must not be possible to find a match for this object.

students to change their perspective and to deal with this kind of problems. The fog of oblivion, helps them to identify objects necessary for the execution of some step. This is a starting point for the implementation of that step. Once they come up with story diagrams, the post-its enable them to validate their behavior specification. Since we have introduced these didactic aids, we observe that our students deal much better with their assignments. In addition, the postmen metaphor enables us to discuss concurrency problems and the use of synchronization mechansims: In case of multiple threads, multiple postmen are on duty. Each postman (and each object guy) works on his own pace. Occasionally, two postmen may visit the same object at the same time. Then each postman is served by another lazy object inhabitant. This may cause the typical concurrency problems, e.g. if two inhabitants want to modify the same object differently or if one object guy removes an object that has just been used as match by another object guy. In order to avoid these problems certain synchronization mechanisms are necessary. This may be introduced by (heavy iron) locks attached to (sections of) instruction sheets. One (heavy iron) key for such a lock is attached to (the outside) of each object. In order to execute locked instructions, the postman has to acquire the corresponding key which is returned afterwards. This ensures that a critical section is executed by at most one object guy at a time. Thus, our execution mechanism may also provide the usual metaphors for synchronization aspects.

It also becomes easy to discuss active versus passive objects and method invocation versus signal sending. In usual systems all objects are passive and the only postman comes from the user or from the graphical user interface. An active object (or a thread) employs its own postman. However, if the object guy within the active object sends his postman with a message, he freezes until the postman comes back with the result. In case of a usual method invocation, the postman stays at the receiver and waits for reply or for a subsequent letter. In case of signal sending, the postman just throws the letter in the mailbox and then he directly returns to the sender (without a reply). Accordingly, the signal remains ignored until the receiving object guy looks into his mailbox. However, this may only happen, if the current instruction sheet asks the object guy to do so and if a postman is present that forces the object guy to be active. Altogether, the postmen are the driving forces in our execution metaphor. Only if a postman enters, an object guy becomes active, either starting a new execution or continuing a former execution. Thus, the locations of the postmen represent the only points of activity within the whole system. This postmen metaphor has already successfully been applied in teaching multi-threading aspects of graphical user interfaces in an software engineering course at University Kassel. The postmen metaphor was used to explain why the graphical user interface of some programs freeze if an invoked operation needs very long time (or loops for ever). Similarly, we explored a scenario where each invoked operation runs in its own thread / employs its own postman. In that case one has to deal with the concurrency of multiple operations invoked in a short period of time. We also discussed a scenario with only two postmen, one employed by the GUI and one employed by the application. The GUI postman delivers the user commands to the postbox of a root object of the application and the application postman takes care of them one after the other. In this scenario, the GUI remains responsive even if the application postman is busy. However, the application needs not to deal with concurrency problems.

While we have first experiences, applying our metaphor to concurrent programs still needs

more evaluation. In addition, the object guy metaphor has been introduced only recently. Before that, the postmen did not only carry messages around but when they entered an object they also executed the instruction sheets. Using only postmen results in a somewhat simpler execution mechanism. In addition it emphasizes the representation of execution threads. The object guy metaphor is somewhat more complicated, especially for re-entrant programs where the postman comes back to an object containing already an active inhabitant. At this point we suddenly had to introduce a family of object guys living in each object in order to represent multiple procedure call frames and name spaces attached to the same object. However, the object guy metaphor emphasizes the discussed change of perspective from the overview of the whole object structure to the restricted view of a method execution. It is easy to imagine that a sleeping object guy knows little about his surrounding and that he forgets everything between two method executions. This is somewhat harder to imagine for a postman that has to navigate through the whole object structure while he delivers messages but that has to forget everything about the outside as soon as he enters an objects and starts to execute some instruction sheet. However, we still need more teaching experiences in order to judge finally, whether the emphasis on the change of perspective justifies the additional complexity introduced by the oblivious object guys.

References

[BRJ99] Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide; Addison Wesley, ISBN 0-201-571168-4, 1999

[DGZ04b] I. Diethelm, L. Geiger, A. Zündorf: Systematic Story Driven Modeling, a case study; SCESM Workshop 2004; ICSE 2004, Edinburgh, Scottland, May 24 28 (2004)

[DGZ05] Ira Diethelm, Leif Geiger, and Albert Zündorf: Applying Story Driven Modeling to the Paderborn Shuttle System Case Study; book chapter in S. Leue and T.J. Systä (Eds.): Scenarios, LNCS 3466, pp. 109133, 2005. Springer 2005

[DGZ05b] Ira Diethelm, Leif Geiger, Albert Zündorf: Teaching Modeling with Objects First; WCCE 2005, Cape Town, South Africa, 2005

[DGZ05d] I. Diethelm, L. Geiger, A. Zündorf: Mit Klebezettel und Augenbinde durch die Objektwelt; in Informatik und Schule (INFOS) 2005, Dresden, Germany, 2005

[Fu02] Fujaba Homepage, Universität Paderborn, http://www.fujaba.de/.

[Fu04] Fujaba Team Kassel: Story Driven Modelling with Fujaba; Powerpoint Präsentation 2004, http://www.se.eecs.uni-kassel.de/se/fileadmin/se/courses/MSE/download/fujaba/FujabaTutorialStoryDrivenModeling.ppt

[STSN02] F Steimann, U Thaden, W Siberski, W Nejdl "Animiertes UML als Medium für die Didaktik der objektorientierten Programmierung" Modellierung 2002 GI Lecture Notes in Informatics (2002).

Modellieren versus Programmieren in der beruflichen informatischen Schulbildung

Joachim Fels

IT-Koordination
Gebhard-Müller-Schule Biberach
Leipzigstr. 25
88400 Biberach
mail@fels-it.de

Abstract:

In der Abteilung IT-Koordination der Gebhard-Müller-Schule (GMS) werden Inhalte sowie Methoden des Informatikunterrichtes an berufsbildenden Schulen des Landes Baden-Württemberg entwickelt und evaluiert. Die Abteilung versteht sich deshalb als Lieferant und Testinstanz für zukünftige Lehrplaninhalte. Derzeit ist von Seiten sowohl der Lehrer als auch der Ausbildungsbetriebe ein großer Klärungsbedarf bezüglich Fragen der inhaltlichen und methodischen Ausrichtung des Unterrichtsgegenstandes „Softwareentwicklung“ festzustellen. Im Zentrum des Interesses steht die Frage, in welchem Verhältnis einerseits Modellierungs- und andererseits Programmierungswissen als relevant für den schulischen Informatikunterricht angesehen werden müssen. Damit einher geht die methodische Frage, auf welche Weise die als relevant bezeichneten Inhalte sinnvoll vermittelt werden können. Der vorliegende Beitrag will vor dem Hintergrund des derzeitigen Standes der Didaktik der Informatik die Durchführung einer empirischen Studie zur Bewertung verschiedener Möglichkeiten der Vermittlung von Modellierungs- und Programmierungswissen im Softwareentwicklungsunterricht motivieren. Im Rahmen einer solchen Studie soll insbesondere die Frage geklärt werden, inwieweit sich eine kombinierte Vermittlung von Programmier- und Modellierungswissen im Unterricht als vorteilhaft gegenüber dem herkömmlichen Modell von zeitlich getrennten Unterrichtsphasen der Programmierung und Modellierung erweist.

1 Einleitung

Betrachtet man den derzeitigen Diskussionsstand innerhalb der Schuldidaktik der Informatik, so lässt sich trotz der eher geringen Anzahl an wissenschaftlichen Publikationen eine Art Mainstream feststellen: Der Fokus des Interesses liegt zum einen auf der Herausarbeitung des Allgemeinbildungscharakters von informatischer Bildung und zum anderen auf der Betonung der Wichtigkeit der Modellierung bei der Vermittlung von Bildungsinhalten im Kontext der Softwareentwicklung. Diese Schwerpunktsetzungen sind historisch gewachsen: Die Anfangsphase der Didaktik der Informatik war stark fachwissenschaftlich und auf das Lösen von algorithmischen Problemlösungen hin ausgerichtet, die Erstellung von funktionierendem Quellcode stellte das Ziel eines erfolgreichen Unterrichts dar. Diesen problemorientierten Didaktiken mit ihrer Konzentration auf die Vermittlung von Programmierkompetenz wird in aktuellen informatikdidaktischen Publikationen abgesprochen, zur Begründung des allgemein bildenden Wertes des Informatikunterrichtes beitragen zu können: „Wenn im Informatikunterricht Programmieren vermittelt wird, dann haben die Schülerinnen und Schüler, falls der Unterricht lernwirksam ist, bestenfalls (…) programmieren gelernt." [Sc03, S. 26] Dementsprechend betonen neuere didaktische Ansätze die Wichtigkeit der Modellierung: Die Entwicklung von Quellcode wird nicht mehr als prüfungsrelevant angesehen, der Lehrer muss nicht mehr auf programmiertechnische Details wie spezielle Datenstrukturen oder bestimmte Algorithmen eingehen [Vgl.: Hu04]. Das im Unterricht erstellte Modell soll allerdings im Anschluss an seine Erstellung von den Schülern 'realisiert' und überprüft werden. Da Programmieren innerhalb dieses Ansatzes jedoch ein „Unterrichtsdilemma"[1] [Vgl.: Hu04, S. 169] darstellt, wird die sich somit eröffnende „unterrichtsmethodische Lücke" [Sc03, S. 36] beispielsweise durch den Hinweis auf die Einsatzmöglichkeiten von CASE-Tools mit automatischer Codegenerierung zu schließen versucht [Vgl.: Hu00, S. 42]. In aktuellen informatikdidaktischen Publikationen wird zu zeigen versucht, dass objektorientierte Modellierung im Gegensatz zur reinen Programmierung als allgemein bildender Wert aufgefasst werden muss. Torsten Brinda liefert einen Überblick über derartige Argumentationslinien [Vgl.: Br04, S. 10-43]. Systemorientierte Didaktiken betonen ebenfalls den hohen allgemein bildenden Stellenwert der Modellierung: Die Schüler sollen lernen, dass das Erstellen eines technischen Systems auch ein sozialer Prozess mit gesellschaftlichen Auswirkungen ist und dass die sozialen Auswirkungen eines Informatiksystems ihren Ursprung in der Analyse- und Entwurfsphase der Softwareentwicklung haben [Vgl.: Sc03, S. 40, in Anlehnung an: Ma01].

Der Modellierung kommt daher im systemorientierten Ansatz insofern eine zentrale Rolle zu, als durch sie die Einbettung der geplanten Software in ihren (zukünftigen) Kontext erfolgt: Innerhalb der systemorientierten Didaktiken zur Schulinformatik wird betont, dass technologische Bildung, hier verstanden als überblicksartiges technologisches Einordnungs- und Orientierungswissen, „ein unverzichtbarer Bestandteil der Allgemeinbildung sei" [Ebd.: S. 37, vgl: ebd.: S. 38ff]. Der Trend hin zu von konkreter Pogrammierung losgelöster Modellierungstätigkeit wird durch die Einbeziehung systemorientierter Didaktiken in das Unterrichtsgeschehen offensichtlich verstärkt.

[1] Laut Peter Hubwieser besteht das Dilemma des Programmierens darin, dass einerseits gewisse Grundkenntnisse bezüglich Programmierung und Programmiersprachen für das Verständnis von Informationssystemen unverzichtbar sind, Programmierung aber andererseits keinen allgemein bildenden Lerninhalt darstellt.

Kapitel 2 will einige der derzeit von der informatikdidaktischen Forschung offen gelassenen Fragestellungen thematisieren und für eine Beschäftigung mit dem Informatikunterricht an berufsbildenden Schulen motivieren. Kapitel 3 stellt die Konzeption einer empirischen Studie zur Überprüfung und Wertung der verschiedenen Möglichkeiten der unterrichtlichen Einbettung von Modellierungs- und Programmierungswissen im berufsbildenden Schulwesen vor.

2 Offene Fragen der Didaktik der Informatik

Evaluation und Qualitätssicherung

Franz Eberle gab 1996 eine Übersicht über offene Forschungsfragen und betonte die fehlende empirische Überprüfung im Bereich der Didaktik der Informatik: „Viele Aussagen in dieser Arbeit mit deskriptivem Charakter sind empirisch gar nicht oder mangelhaft nachgewiesen, beruhen nur auf Alltagsbeobachtungen oder basieren auf nichtrepräsentativer qualitativer Forschungsmethodik. Daraus ergibt sich eine breite Palette von Forschungsfragen, die empirisch geklärt werden sollten“ [Eb96, S. 427]. Auch Thomas [Vgl. Th02, S. 82] betont das Fehlen empirischer Studien zur Evaluation von Unterrichtssequenzen. Viele der bisher publizierten Arbeiten zur Didaktik der Informatik sind stark bildungstheoretisch orientiert und bieten lediglich ergänzend kleinere Unterrichtssequenzen zur Veranschaulichung der entwickelten Theorie an. Mittlerweile existieren zwar vereinzelt empirisch orientierte Arbeiten [Vgl.: Sc03], die sich zwischenzeitlich im Bereich der Informatik etablierten Bildungsstandards können allerdings nach wie vor als empirisch nicht ausreichend evaluiert bezeichnet werden [Vgl.: Ma05, S. 1].

Das Verhältnis von Programmierung und Modellierung

Mit der starken Fokussierung auf die Modellierung geht in den aktuellen informatikdidaktischen Publikationen die Vernachlässigung der Frage nach dem Bildungsziel und der sinnvollen methodischen Einbindung von Programmierwissen im Unterricht einher. Die Frage nach dem grundlegenden Verhältnis der Vermittlung von Modellierung und Programmierung ist durch die bislang vorliegenden informatikdidaktischen Publikationen keineswegs (abschließend) beantwortet. Vielleicht ist – im Gegensatz zur derzeitigen Hauptmeinung der Informatikdidaktik – prozedurales (Programmier-) Wissen für das Erlernen von (objektorientierter) Modellierung sogar förderlich: „The papers reviewed show that identifying objects is not an easy process, that novices need to construct a model of the procedural aspects of a solution in order to properly design objects/classes“ [RRR03, S. 154]. Schulte unternimmt zwar einen Versuch der Integration „von notwendigem Implementationswissen in den Unterricht“ [Sc03, S. 187], betont allerdings, dass damit „die Diskussion über die Rolle und möglichst angemessene Integration der Vermittlung von Implementationswissen auch mit der vorliegenden Arbeit nicht abgeschlossen (ist)“ [Ebd.: S. 187]. Eine ganze Reihe von Fragen bleiben angesichts des momentanen Diskussionsstandes der Didaktik der Informatik in Bezug auf das Verhältnis von Modellierung und Programmierung im Unterricht offen. So ist unklar und empirisch noch nicht untersucht, wie sich die unterschiedliche Integration von Implementationswissen in den Unterricht auf beispielsweise das Lernziel einer Kompetenz zur Entwicklung von Software im schulischen Bereich auswirkt.

Insbesondere wäre es interessant zu wissen, ob und wenn ja, wie sich ein vorgeschalteter (vielleicht sogar imperativer) Programmierkurs auf das nachträgliche Erlernen von (objektorientierten) Modellierungsfähigkeiten auswirkt. Im Bereich der Hochschuldidaktik wurde diese Frage zwar bereits diskutiert, für den Bereich der Schuldidaktik der Informatik existieren hierfür jedoch noch keine Forschungsarbeiten [Vgl.: Sc03, S. 191f].

Systemorientierung im Unterricht

Zukünftige Forschungen sollten darüber hinaus den Schwerpunkt der Untersuchung nicht nur – wie bislang ausnahmslos geschehen – auf den Anfangsunterricht der Programmierung und der Modellierung beziehen, sondern auch den weiteren Verlauf des Unterrichts berücksichtigen. Dies gilt insbesondere für den Versuch, den systemorientierten Ansatz im konkreten Unterricht zu realisieren: Es ist auffallend, dass, trotz der überall anzutreffenden Forderung nach Integration einer systemorientierten Didaktik der Softwareentwicklung, bislang nur äußerst wenige konkrete Unterrichtsmodelle mit expliziter Betonung einer systemorientierten Perspektive anzutreffen sind. Es wäre interessant, die Integration einer systemorientierten Perspektive im Unterricht in Verbindung mit unterschiedlichen Ansätzen der Vermittlung von Modellierungs – und Programmierwissen kritisch zu evaluieren.

Schließung der „unterrichtsmethodischen Lücke“

Ziel eines jeden Prozesses zur Entwicklung von Software sollte funktionierender Code sein [Vgl: GH05, S. 13] – eine Tatsache, die vielleicht (wieder) stärker in das Zentrum unterrichtlicher Aktivitäten gerückt und insbesondere bei der Beantwortung der Frage nach der Überbrückung der weiter oben im Rahmen der Beschreibung des informationsorientierten Ansatzes erwähnten „unterrichtsmethodischen Lücke“ zwischen Modell und lauffähigem Programm berücksichtigt werden sollte. Hierzu würde die Evaluation des Einsatzes verschiedener CASE-Tools im Unterricht sowie die Thematisierung der Frage, wie das verwendete Tool aus dem Modell den Code generiert [Vgl.: ebd.: S. 14] interessante Aufschlüsse ergeben. Wie Schulte richtig bemerkt, könnte im Unterricht direkt mit UML-Notationen, anstatt mit anderen Hilfsmitteln gearbeitet werden [Vgl: Sc03, S. 190]. Vergleichende (empirische) Untersuchungen liegen für diesen Bereich bislang allerdings ebenfalls noch nicht vor.

Softwareentwicklung in der allgemeinen und beruflichen Bildung

Ein Bereich, der von den bislang veröffentlichten Forschungsarbeiten zur Didaktik der Schulinformatik ausgeklammert wird, betrifft die schulische Ausbildung zukünftiger Software-Entwickler. Dies hängt offensichtlich mit der starken Fokussierung der derzeitigen Informatikdidaktik auf allgemein bildende Inhalte und Ziele des Unterrichts zusammen. Aus diesem Grund wird dem Endprodukt des Softwareentwicklungsprozesses, der lauffähigen Software, nur marginale Aufmerksamkeit geschenkt. Torsten Brinda etwa versucht den Allgemeinbildungscharakter von objektorientierter Modellierung herauszuarbeiten und bemerkt: „Produktentwicklung und die Ausbildung von Software-Entwicklerinnen und –entwicklern sind keine Ziele informatischer Allgemeinbildung“ [Br04, s. 122f]. Gleichwohl räumt er ein, dass diese Einschätzung für den Bereich der beruflichen Bildung relativiert werden müsste [Vgl.: ebd.: S. 123].

Die derzeitige Didaktik der Informatik untersucht zwar Lernstrategien von Lernenden, übersieht hierbei aber, dass Software-Entwickler ebenfalls Lernstrategien haben. In Bezug auf den Informatikunterricht an berufsbildenden Schulen müssen wahrscheinlich viele der bislang getroffenen Aussagen der Didaktik der Informatik relativiert werden. So fordern aktuelle – immer auf den Allgemeinbildungsgehalt von informatischer Bildung ausgerichtete – Arbeiten beispielsweise als ein Gütekriterium für im Unterricht eingesetzte CASE-Tools mit Codegenerierung die strikte Trennung von Modell und generiertem Code [Vgl. etwa: ebd., S. 125]. Die Frage stellt sich, ob dieses Gütekriterium aufrechterhalten werden kann, nach dem sich die Didaktik der Informatik (teilweise) von ihren Legitimationszwängen hinsichtlich ihres Allgemeinbildungscharakters gelöst und beispielsweise der Frage nach der unterrichtlichen Integration von Implementierungswissen zugewandt hat.

Die herkömmliche Didaktik der Informatik könnte vermutlich von einem Dialog mit den an berufsbildenden Schulen gemachten Erfahrungen im Informatikunterricht profitieren. Dies deshalb, weil der Informatikunterricht an berufsbildenden Schulen dem Legitimationsdruck hinsichtlich allgemeiner Bildungsinhalte und- ziele niemals so stark ausgesetzt war wie derjenige an allgemein bildenden Schulen. Des weiteren könnten die bislang noch relativ abstrakt anmutenden Ziele systemorientierter Informatikdidaktik im Umfeld eines auf berufliche Bildung ausgerichteten Unterrichts leicht konkretisiert und somit auch (empirisch) auf ihre Relevanz hin überprüft werden. Zu denken wäre hierbei etwa an Model-Driven-Development als Unterrichtsthema: Die vielfältigen Querbezüge zum jeweiligen Profilfach (beispielsweise BWL im kaufmännisch geprägten Bereich der beruflichen Bildung) könnten zur systemorientierten Thematisierung des Anwendungskontextes herangezogen werden, um zum Beispiel Änderungen in zugrunde liegenden Geschäftsprozessen durch den Einsatz von Informatiksystemen zu beleuchten.

3 Softwareentwicklung und berufliche Bildung

3.1 Relevante Aspekte der Softwaretechnik für berufliche informatische Bildung am Beispiel des Faches „Informationsmanagement“

Die derzeit im Rahmen beruflicher Bildung als relevant betrachteten Aspekte des Softwareentwicklungsprozesses können anhand der Lerninhalte des seit dem Schuljahr 2000/2001 an Baden-Württembergischen Wirtschaftsgymnasien eingeführten Faches „Informationsmanagement“ dargestellt werden. Ziel dieses im wöchentlichen Umfang von 4 Stunden unterrichteten und sowohl schriftlich als auch mündlich abiturablen Faches ist es, die Schüler auf Anforderungen in Beruf und Studium der Informatik vorzubereiten.[2] Grundsätzlich soll bei der Umsetzung der Lehrplaninhalte der aktuelle Stand der Fachwissenschaften berücksichtigt werden. Sämtliche Unterrichtsgegenstände sollen mit Computerunterstützung gelehrt und gelernt werden, eine „Modellierungstätigkeit auf dem Papier“ kommt daher beispielsweise nicht in Frage.

[2] Alle Aussagen zum Fach Informationsmanagement am Wirtschaftsgymnasium beziehen sich auf [KU03].

Dementsprechend müssen gemäß der Vorgabe des Lehrplans die Leistungskontrollen sowie die Abiturprüfung mit einer ganzheitlichen Aufgabenstellung unter Verwendung des Computers durchgeführt werden. Das Fach „Informationsmanagement“ soll laut Lehrplan weitestgehend verzahnt mit dem Profilfach „Volks- und Betriebswirtschaftslehre“ unterrichtet werden, daher können die vor dem Unterrichtsgegenstand „Softwareentwicklung“ vermittelten Kenntnisse zu Geschäftsprozessen als Grundlage und Ausgangspunkt für den unterrichtlichen Prozess der Softwareentwicklung angesehen werden. Ziel des Unterrichts ist, dass die Schüler kundenspezifische Anwendungen zur Unterstützung von Geschäftsprozessen planen und realisieren. Bezüglich des im Rahmen dieses Faches vermittelten Stoffes zur Softwareentwicklung schreibt der Lehrplan die Einführung in die objektorientierte Anwendungsentwicklung mit Hilfe der UML und einer geeigneten Entwicklungsumgebung vor. Innerhalb des Unterrichtsschwerpunktes „Objektorientierte Systemanalyse und –entwicklung“ ist die Modellierung auf die Modellierung von Klassen mit Hilfe der UML beschränkt, die objektorientierte Programmierung nimmt größeren Raum ein und beinhaltet etwa alle grundlegenden Programmstrukturen sowie den Umgang mit Klassenbibliotheken.

Der Lehrplan verlangt zwar objektorientierte (statische) Modellierung *und* Programmierung, dennoch besteht Uneinigkeit, auf welche Weise und zu welchem Zeitpunkt Programmierungswissen im Unterricht vermittelt werden soll. In der Praxis ist daher eine relativ große Bandbreite an verschiedenen Möglichkeiten anzutreffen: Teilweise wird dieses didaktische Problem mit vorgeschalteten (strukturierten oder objektorientierten) Programmierkursen, teilweise mit an den Unterrichtskomplex „Modellierung“ angehängten Programmiereinheiten zu lösen versucht.

Momentan steht die Forderung der Lehrer nach einer Untersuchung im Raum, wie die unterschiedlichen Ansätze zur Integration von Programmierungswissen im Unterricht zu bewerten sind und wie insbesondere die Idee der kombinierten, d.h. integrativen Vermittlung von (statischem) Modellierungs- und Programmierungswissen in der Praxis zu beurteilen ist. Diese Forderung gewinnt durch die zukünftig wünschenswerte Einbindung der dynamischen Modellierung in den Unterricht zusätzlich an Komplexität. Die Erfahrungen aus der Praxis zeigen, dass es vor allem aufgrund der Ganzheitlichkeit („vom Geschäftsprozess zum Modell zur fertigen Anwendung“) und der damit einhergehenden besseren Akzeptanz und Verständnismöglichkeit des Unterrichts sinnvoll ist, Modellierung *und* Programmierung zu unterrichten. Ein rein auf Programmierung beziehungsweise Modellierung ausgerichteter Unterricht würde von den allermeisten Lehrern abgelehnt werden.

3.2 Konzeption einer empirischen Studie zur Evaluation eines integrativen Unterrichtskonzeptes für die Softwareentwicklung

Die offenen Fragen der Didaktik der Informatik sowie die im Fach Informationsmanagement gemachten Erfahrungen belegen die theoretische sowie praktische Relevanz einer empirischen Studie zur Überprüfung der verschiedenen Möglichkeiten der Vermittlung von Modellierungs- und Programmierwissen im Unterricht. Hierzu wird derzeit an der GMS ein durchgängiges Unterrichtskonzept entwickelt, welches empirisch begleitet und evaluiert werden soll. Im Rahmen dieses Unterrichtskonzeptes soll Programmierungs- und Modellierungswissen integrativ unter Beachtung der Querbezüge zum Profilfach „Volks- und Betriebswirtschaftslehre“ vermittelt werden. Kennzeichnend für dieses Unterrichtskonzept ist einerseits die gleichrangige Betonung sowie andererseits die gleichzeitige Vermittlung von Modellierung und Programmierung im Unterricht. Des Weiteren stellt dieses Unterrichtskonzept offensichtlich einen Versuch zur Schließung der oben beschriebenen „unterrichtsmethodischen Lücke“ dar. Die Vor- und Nachteile dieses integrativen Unterrichtsansatzes sollen durch die nachfolgend umrissene, zweigeteilte und empirisch begleitete Unterrichtsstudie untersucht und kritisch bewertet werden. Insbesondere sollen Erkenntnisse über den sinnvollen Aufbau einer empirischen Langzeitstudie gewonnen werden.

- Im Rahmen eines als Querschnittstudie angelegten ersten Untersuchungsschrittes sollen ca. 10 nach unterschiedlichen methodischen Vorgehensweisen beschulte Schülergruppen der Sekundarstufe II des beruflichen Schulwesens einschließlich der jeweils unterrichtenden Lehrkräfte nach ihren Erfahrungen und motivationalen Veränderungen in der Unterrichtseinheit „Softwareentwicklung“ befragt werden. Ziel dieser Studie ist die Evaluierung des Ist-Zustandes sowie die messtechnische Aufbereitung der Stärken und Defizite der verschiedenen etablierten unterrichtlichen Vorgehensweisen (Programmierung bzw. Modellierung zuerst). Parallel zu dieser Untersuchung sollen die in Baden-Württemberg für die inhaltliche Ausgestaltung der Lehrpläne im Bereich Informatik an berufsbildenden Schulen verantwortlichen Lehrkräfte als Experten nach ihrer Einschätzung des Ist-Zustandes sowie nach denkbaren Verbesserungsvorschlägen befragt werden.

- Im zweiten Schritt der Untersuchung soll ein empirischer Vergleich zwischen den herkömmlichen Unterrichtsansätzen und der neuen, integrativen Unterrichtsmethode durch Analyse des Unterrichtsverlaufes sowie des Unterrichtserfolges durchgeführt werden. Hierzu werden 4 Schülergruppen à ca. 15 Schüler im Rahmen eines ca. 6-wöchigen Unterrichtsversuches ergebnis- sowie prozessorientiert begleitet. Zwei dieser Schülergruppen werden nach den traditionellen Methodiken (Programmierung bzw. Modellierung zuerst) unterrichtet und fungieren daher insofern auch als Kontrollgruppen, als diese beiden Ansätze die herkömmliche Unterrichtssituation widerspiegeln. Zwei Schülergruppen werden integrativ unterrichtet, wobei eine Schülergruppe über ausgeprägtes prozedurales Vorwissen verfügt.

- Inhaltlich werden die 4 Schülergruppen anhand der gleichen Unterrichtsbeispiele aus dem kaufmännisch orientierten beruflichen Schulwesen unterrichtet.

4 Zusammenfassung und Ausblick

Der vorliegende Artikel will für die Beschäftigung mit dem bislang wenig beachteten Bereich der beruflichen informatischen Schulbildung motivieren. Die „offenen Fragen der Didaktik der Informatik“ können den Weg für eine erste Beschäftigung mit der derzeit diskutierten methodisch-didaktischen Ausrichtung der Unterrichtseinheit „Softwareentwicklung“ weisen: Nötig ist eine empirisch angelegte Überprüfung und Wertung von verschiedenen Möglichkeiten der unterrichtlichen Vermittlung von Modellierungs- und Programmierungswissen. Es wird hierbei u.a. interessant sein zu sehen, inwieweit der Forderung nach der Schließung der „unterrichtsmethodischen Lücke“ im Rahmen eines integrativen Unterrichtskonzeptes nachgekommen werden kann. Zeitlich ist die Durchführung dieser Überprüfung folgendermaßen geplant: Die in Abschnitt 3.2 umrissene Konzeption einer empirischen Studie wird im Februar und März 2006 im Rahmen von Lehrerfortbildungen vorgestellt, diskutiert und gegebenenfalls abgeändert werden. Im April 2006 soll mit dem ersten Untersuchungsschritt begonnen werden. Idealerweise sollte der zweite Untersuchungsschritt mit dem Schuljahr 2005/06 abgeschlossen sein. Die Situation des Informatikunterrichts an berufsbildenden Schulen ist in der didaktischen Forschung noch nicht adäquat berücksichtigt worden. Daher besteht die Möglichkeit, dass die Ergebnisse der anvisierten Studie an manchen Stellen von mittlerweile etablierten Überzeugungen der hauptsächlich auf den allgemein bildenden Informatikunterricht ausgerichteten Didaktik der Informatik abweichen können. Deshalb ist das beschriebene Projekt als ergebnisoffen zu kennzeichnen.

Literaturverzeichnis

[Br04] Brinda, Torsten: Didaktisches System für objektorientiertes Modellieren im Informatikunterricht der Sekundarstufe II, Siegen 2004.

[Eb96] Eberle, Franz: Didaktik der Informatik bzw. einer informations- und kommunikationstechnologischen Bildung auf der Sekundarstufe II, ,Aarau 1996.

[GH05] Giese, Holger; Roques, Pascal (Eds.): Educators´ Symposium of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Jamaica 2005.

[Hu00] Hubwieser, Peter: Informatik am Gymnasium. Ein Gesamtkonzept für einen zeitgemäßen Informatikunterricht, München 2000.

[Hu04] Hubwieser, Peter: Didaktik der Informatik, Berlin 2004.

[KU03] Kultus und Unterricht. Amtsblatt des Ministeriums für Kultus, Jugend und Sport Baden-Württemberg. Bildungsplan für das berufliche Gymnasium der sechs- und dreijährigen Aufbauform. Wirtschaftswissenschaftliche Richtung. Informationsmanagement, August 2003.

[Ma01] Magenheim, Johannes: Deconstruction of Socio-technical Information Systems with Virtual Exploration Environments as a Method of Teaching Informatics, Edmedia 2001.

[Ma05] Magenheim, Johannes: Towards a Competence Model for Educational Standards of Informatics. In: WCCE 2005 - Proceedings of the 8th IFIP World Conference on Computers in Education, University of Stellenbosch, Cape Town (SA), 4-7. Juli 2005.

[Sc03] Schulte, Carsten: Lehr- Lernprozesse im Informatik-Anfangsunterricht, Paderborn 2003.

[Th02] Thomas, Marco: Informatische Modellbildung. Modellieren von Modellen als ein zentrales Element der Informatik für den allgemeinbildenden Schulunterricht, Potsdam 2002.

[RRR03] Robins, Anthony; Rountree, Janet; Rountree, Nathan: Learning and Teaching Programming: A Review and Discussion, Computer Science Education, 2003, Vol. 13, No. 2.