Die Erprobung von Z in der Lehre[1]

Jörn Mordau, Thorsten Spitta
Universität Bielefeld, Fakultät für Wirtschaftswissenschaften

Zusammenfassung: Es wird von einem Projektseminar berichtet, in dem eine standardisierte Prüfungsverwaltung in drei aufeinander folgenden Semestern mit **Z** spezifiziert wurde. Die Anwendung kann als repräsentatives aber dennoch überschaubares Beispiel für Betriebliche Informationssysteme gelten. Ausgewertet werden sowohl die Erfahrungen mit Z für die praxisnahe Problemstellung als auch die Ergebnisse in Form recht komplexer Datenmodelle. Einige Erkenntnisse aus der Projektarbeit sind in eine Neuentwicklung des Standardsystems HIS/POS der HIS GmbH eingeflossen, insbesondere bezüglich der regelbasierten Komponente des Modells, der Prüfungsordnung.

1 Problemstellung und Überblick

Z wird seit einigen Jahren als neue formale Spezifikationsmethode herausgestellt, die durch die Möglichkeit der Einbettung formaler Elemente in Prosatexte die Nachteile durchgängig formaler Methoden wie etwa VDM vermeidet [5]. Obwohl man bereits von einer '**Z**-community' sprechen kann [10], bleibt die skeptische Haltung industrieller Entwickler gegenüber formalen Methoden ungebrochen (s. auch [8]). Es werden sogar formale Methoden von Praktikern noch nicht einmal als zukunftweisend eingestuft (vgl. [9]).

Hier setzte unser *Forschungsinteresse* an, die Anwendbarkeit von **Z** in einer Lehrveranstaltung anhand eines praktischen aber noch überschaubaren realen Problems zu überprüfen. Es handelt sich um ein Praxisprojekt mit einer keineswegs mehr trivialen Anwendung, die in der Hochschule selbst dringend gebraucht wird, und zwar die Prüfungsverwaltung für eine Fakultät mit Massenbetrieb. Das dominierende *Lehrziel* des über drei Semester mit jeweils neuen Teilnehmern durchgeführten Projektseminars war, den Studenten die Erfahrung zu vermitteln, wie differenziert man eine reale Problemstellung spezifizieren muß, bevor man mit der Realisierung beginnen darf.

Der Beitrag soll folgende Fragen beantworten:

- Wie kann man Aspekte praktischer Softwareentwicklung realitätsnah in der Universität vermitteln? Die Antwort behandelt die *Problemstellung* und die *Projektorganisation* der Lehrveranstaltung.

---

[1] erschienen: In: Forbrig, P.; Riedewald, G. (Hrsg): Software Engineering im Unterricht der Hochschulen SEUH `97, Fachtagung GI Febr. 1997 in Rostock, Teubner, Stuttgart 1997, 24-33.

- Welche wissenschaftlich validen Ergebnisse kann man von Studenten erwarten? Hier geht es um die *Verwertbarkeit* der gewonnenen Erkenntnisse.
- Welche Schlußfolgerungen kann man aus dieser fallorientierten empirischen Erprobung von **Z** ableiten?

2 Die Lehrveranstaltung und ihre Projektorganisation

Die **Lehrveranstaltung** *BI-Projekt* müssen Studenten der Betriebswirtschaftslehre (BWL) in Bielefeld im Rahmen des Faches *Betriebsinformatik* im Hauptstudium besuchen. Sie setzt eine *Einführung in die Informatik* (Grundstudium) und eine Vorlesung *Software Engineering* oder *Informationsmanagement* sowie einen Programmierkurs voraus. Der Bielefelder Studiengang ist der einzige BWL-Studiengang in Deutschland, der das Fach *BI* im Hauptstudium zur Pflicht macht. Die Studenten haben eine gute mathematische Grundausbildung (3 Semester Mathematik, 4 Semester Statistik, Ökonometrie und Operations Research). Es finden jedes Semester drei bis vier parallele BI-Projekte zu verschiedenen Themen statt.

Die **Problemstellung**, die für die Erprobung von **Z** verwendet wurde, ist eine Prüfungsverwaltung für eine Fakultät, also ein Diskursbereich, der den Studenten wohl vertraut ist. Die Probleme mit einem Standardsystem wie auch die Entwicklungsmethodik, die der Spezifikation vorausgeht, sind in [14] dargelegt.

Bei der Veranstaltungsform *Projekt* arbeiten die Teilnehmer in kleinen Gruppen an einem gemeinsamen Entwicklungsdokument, das in Abschnitten parallel bearbeitbar sein muß. Das Dokument entsteht schrittweise im Rahmen der Veranstaltung, ist ständige Kommunikationsbasis für die Arbeitsgruppen und bildet zum Projektabschluß deren Ergebnis, im Fall dieses Projektes (näheres s. u.) eine Spezifikation. Daneben gibt es andere zentrale Ergebnisse, auf die im Dokument nur verwiesen wird und die mit Spezialwerkzeugen erstellt sind. Im vorliegenden Fall handelt es sich um einen Datenkatalog und ein Relationenmodell, aus dem ein Entity-Relationship-Modell generiert werden kann. Als Werkzeug wurde ERWin/SQL von Logic Works /Princeton benutzt, das sich in Großprojekten des Pentagon und bei der Firma Boeing bewährt hat.

Auf den Aspekt, daß lesbare *Dokumente* eine zentrale Kommunikationsbasis in Softwareprojekten darstellen (vgl. auch [4]), wurde großer Wert gelegt. Dies ist nicht selbstverständlich und stellt sich nicht ohne besondere Vorkehrungen ein. Eine realitätsnahe Spezifikation nimmt schnell 100, manchmal bis über 1000 Textseiten ein (ein Beispiel wird diskutiert in [13, 160]).

Das Thema wurde in Form vorstrukturierter Teilaufgaben bearbeitet, davon einige bewußt von parallelen Gruppen, um zu zeigen, daß es sich im Software Engineering nicht um schematische Lösungsverfahren handelt. Zwischenergebnisse mußten im Plenum vorgetragen werden, *nachdem* sie schriftlich dokumentiert waren.

Neben der **Organisation** *im* Projekt gab es ein Koordinationsproblem *zwischen* aufeinander folgenden Projekten. Das Thema Prüfungsverwaltung wurde in drei Schritten bearbeitet:

- WS 94/95: Fachkonzept (Arbeitsabläufe und grobes Datenmodell) [I],
- SS 95: Spezifikation I (Fakultätsanwendung) [II],
- WS 95/96: Spezifikation II (Standardanwendung) [III].

Die konsekutive Organisationsform hatte zum Ziel, daß ein Folgeprojekt ausschließlich auf der Basis eines schriftlichen Dokumentes die vorangegangenen Arbeitsschritte nachvollziehen und darauf aufbauend weitere ausführen soll. Dies entspricht bestimmten Praxissituationen in Projekten oder in der Wartung. Entwickler müssen sich mit einem vorliegenden Dokument auseinandersetzen, ohne die Ersteller desselben mündlich befragen zu können. Dies führte in beiden Folgesemestern zu einer Revision und Weiterentwicklung des vorliegenden Datenmodells. **Z** wurde in den beiden letzten Projekten verwendet.

Wir halten diese Projektorganisation für realitätsnah, da ein Problem bearbeitet wurde, bei dem Studenten Betroffene sind und genügend Kenntnisse über den Diskursbereich besitzen. Die hochschulspezifische Organisation des Projektes hat mehr Ähnlichkeiten mit Praxisprojekten als Unterschiede. Im einzelnen:

Tab. 1: Einbeziehung der Benutzer im Lehrbereich und im Fachbereich eines Betriebes.

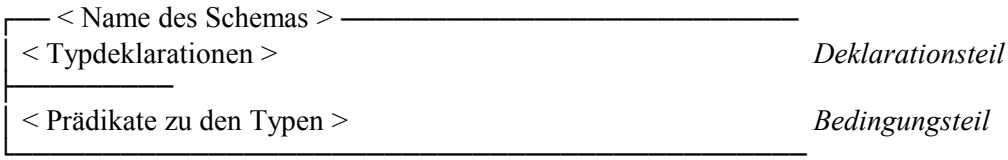
Merkmal	*Universität*	*Betriebl. Fachbereich*
zeitliche Beteiligung	nur punktuell	
Qualifikation	stark streuend	
methodische Fähigkeiten	gering	
Problemkenntnis	hoch	
Betroffenheit	hoch	
Engagement	stark streuend	
personelle Kontinuität	keine	stark
Kapazität Projektgruppe	hoch	gering
Möglichkeit konkurrierender Sichten	gut	schlecht
Betroffenheit nach Projektabschluß	sehr gering	sehr hoch

Diese Synopse kann nur gelten, wenn ein Projekt aus dem jeweiligen Erfahrungshintergrund der Benutzer zu bearbeiten ist. Der Dozent entspricht in einem Praxisprojekt der Rolle der DV-Projektmitglieder und des Projektleiters in einer Person. Hier gibt es kaum Analogien.

3 Das Sprachkonzept von Z

Z ist eine formale, auf Prädikatenlogik und Mengenlehre fußende Spezifikationssprache. Durch ein sogenanntes *Schema* werden typisierte Objekte und wahrheitswertige Aussagen über diese Objekte gekapselt. Schemata können wieder ineinander verschachtelt sein oder mit logischen Konnektoren verbunden werden.

Das „Metaschema“ von **Z** sieht folgendermaßen aus:

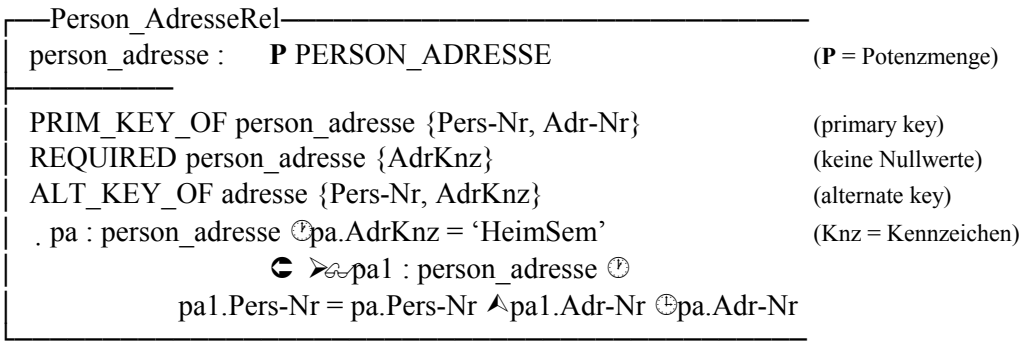


Im Gegensatz zur älteren Sprache VDM werden die zu einer Operation gehörigen Vor- und Nachbedingungen nicht explizit gekennzeichnet, sondern ergeben sich implizit aus dem Bedingungsteil des jeweiligen Schemas. Der Schwerpunkt liegt hier also nicht in der Umsetzung einer Spezifikation in Programmcode, sondern in der Erstellung einer in sich stimmigen Spezifikation. Die Gestaltung des Schemas als separate optische Einheit mittels einer Umrandung dient gerade dazu, die formalen Teile des Dokuments von den informellen, in Prosa abgefaßten Teilen abzuheben. Damit wird die prosaische Entwicklung eines Gedankenganges unterstützt (siehe hierzu auch [5, Kap. 4.10]), der schließlich in formale, nachprüfbare Ausdrücke mündet.

Die Festlegung auf wenige fundamentale Konzepte macht **Z** vielseitig verwendbar. Dies zeigt bereits ein Blick auf die inzwischen reichhaltige Literatur, in der **Z** z. B. mit objektorientierten Methoden kombiniert oder etwa um Komponenten temporaler Logik erweitert wird. In unserem Fall (betriebliches Informationssystem) interessierten wir uns für die Einbindung in ein Relationenmodell, das in [1] eingehend behandelt wird.

Beispiel:

Unter der Annahme, daß Datentypen für die Attribute und die Relationen `person` und `adresse` bereits spezifiziert sind, könnte eine Relation, die Personen mit Adressen assoziiert, folgendermaßen geschrieben werden:



Die Relation `person_adresse` drückt die Beziehung zwischen Personen und Adressen aus. Sie ist als Teilmenge aller Entitäten vom Typ `PERSON_ADRESSE` oder – anders formuliert – als Element der Potenzmenge über den Typ `PERSON_ADRESSE` anzusehen. In der Regel haben Studenten eine Semester- *und* eine Heimatanschrift. Wenn *nicht* zwischen Heimat- und Semesteradresse unterschieden wird, nimmt das Attribut `AdrKnz` den Wert `'HeimSem'` an. Es gilt nun die Bedingung, daß *genau dann*, wenn eine Person-Adreß-Kombination das Kennzeichen `'HeimSem'` trägt, diese Person nur *eine* Adresse hat. Im Umkehrschluß folgt daraus, daß bei den Kennzeichnun-

gen `'Heim'` und `'Sem'` für die betreffende Person nicht nur *eine* Adresse vorhanden ist. Man vermeidet mit dieser Festlegung, daß die Behandlung der unterschiedlichen Fälle von Person-Adreß-Beziehungen erst bei der Dateneingabe mit möglicherweise inkonsistenen Einträgen entschieden wird. Außerdem bildet die konsistente Lösung dieses Zuordnungsproblems die Voraussetzung für ein korrekt funktionierendes Anwendungsprogramm, das auf Heimat- und Semesteradressen zugreift.

Das Beispiel zeigt, wie notwendig die exakte Spezifikation von Integritätsbedingungen bereits bei scheinbar trivialen Sachverhalten ist. Man sieht, daß der Entwickler und der die Spezifikation lesende Benutzer neben recht einfachen Formalismen zur Beschreibung von Typen die Ausdrucksweise und Symbole der Prädikatenlogik beherrschen muß, um Aussagen eines Schemas interpretieren zu können.

4 Die Problemklasse Betriebliche Informationssysteme

Der als Beispiel gewählte Diskursbereich *Prüfungsverwaltung* hat nicht nur den Vorteil der Vertrautheit für die Studenten. Das Thema ist ein überschaubares aber trotzdem repräsentatives Beispiel für die Problemklasse *Betriebliche Informationssysteme*, deren Eigenschaften kurz skizziert seien:

- Datenbasis mit Stamm-, Bewegungs- und verdichteten Daten (z.B. Prüfungsordnung, Student, Prüfung, Notenverteilung),
- Mehrbenutzerfähigkeit,
- Arbeitsabläufe als Transaktionen, deren Konsistenz in der Datenbasis gesichert werden muß (sog. *Geschäftsprozesse*, z. B. Noten eintragen für 500 Klausuren),
- Verdichtung und/oder Auslagerung von Vorgängen nach einer gewissen Zeit (Prüfungen zurückliegender Semester),
- Mandantenfähigkeit (Einsatz für mehrere Fakultäten),
- Zusammensetzung aus Teilsystemen verschiedener Paradigmata:
  - *administrativ* (Immatrikulationsbüro und Prüfungsamt),
  - *regelbasiert* (Prüfungsordnung),
- Benutzeroberfläche sowohl für Massenvorgänge (z.B. Anmeldungen) als auch Einzelfälle (z.B. Stammdatenpflege) und Abfragen,
- Zugriffsschutz wegen sensibler Daten,
- Schnittstelle zu Standardsystemen wie Textverarbeitung (Bescheinigungen u.ä.).

Angesichts der großen Probleme mit der Softwareunterstützung von Prüfungsämtern in Hochschulen ist das Problem auch sehr praxisnah. Die Anwendung ist für eine Fakultät mit Massenbetrieb und entsprechend straffer Prüfungsorganisation sehr viel komplexer als einige „selbstprogrammierende“ Fakultäten zunächst glauben. Das Datenmodell der schon stark verallgemeinerten Lösung [II] besteht aus 22 Objekttypen. Die Standardlösung, die auch ein Kreditpunktesystem für studienbegleitendes Prüfen abbildet, muß mit 60 Objekttypen bereits als recht komplex bezeichnet werden [III].

Da die an mehreren Universitäten und Fakultäten installierte Standardsoftware HIS/POS (POS= Prüfungs-Organisations-System) kaum noch wartbar ist (Details s. [12]), wird seitens der dem Bund und den Ländern gehörenden HIS GmbH an einer Neuentwicklung gearbeitet. Einige Ergebnisse unserer *Spezifikation I* [II] haben bezüglich der generalisierten Abbildung einer Prüfungsordnung in dieser Neuentwicklung Verwendung gefunden. Es handelt sich hierbei um die regelbasierte Komponente, den zentralen und schwierigsten Teil des Systems.

5 Die Anwendbarkeit von Z in der Lehre

Bei der Anmeldung zur Lehrveranstaltung wußten die Studenten nur, daß ein Problem aus dem Prüfungswesen spezifiziert werden sollte. In der Vorlesung *Software Engineering*, die wenig formal ausgerichtet ist, wird **Z** nicht gelehrt. Insofern war die überwiegende Reaktion der Studenten bei der Vermittlung von **Z** zu Beginn der Veranstaltung *Abwehr* gegen die angesichts eines scheinbar einfachen Problems uneinsichtigen Formalismen. Dies wandelte sich bis zum Ende des jeweiligen Semesters bei etwa einem Viertel der Teilnehmer in eine gewisse Begeisterung, sehr komplexe Probleme nun präzise ausdrücken zu können, verbunden mit der Einsicht, daß man dies in Prosa nicht kann. Die übrigen Studenten schätzen wir so ein, daß sie das taten, was die Dozenten von ihnen verlangten, damit sie den Schein bekamen.

Die in [II] und [III] dokumentierten Ergebnisse zeigen, daß die Studenten verständliche und einigermaßen korrekte Spezifikationen in **Z** schreiben konnten. Die Entstehungsgeschichte in der Veranstaltung zeigt jedoch auch, daß der verwendete *Begriffskatalog* des Datenmodell-Tools (hier: ERWin/SQL) nicht ausreichte. Dies brachte unnötige Abstimmungsprobleme mit sich. Für die Studenten wurde so erfahrbar, welche zentrale Rolle Begriffe und zu deren Verwaltung ein leistungsfähiges Dictionary/ Repository in der Softwareentwicklung spielen (vgl. hierzu besonders die Arbeiten von Ortner, z. B. [11]). Bevor man jedoch darangeht, Werkzeuge für und um **Z** herum zu entwickeln, sollten einige wichtige methodische Fragen geklärt werden, die in der Veranstaltung offenkundig wurden. Eine gewisse Abwehr der Studenten gegen **Z** resultiert nicht aus der Lehrsituation, sondern aus der Methode selbst.

Wie alle formalen Spezifikationsmethoden, hat auch **Z** einen Vollständigkeitsanspruch, der dazu zwingt, auch triviale Sachverhalte umständlich zu spezifizieren, die informal genauso klar sind (vgl. [17]). Dies minderte die Motivation, sich formal auszudrücken. Dieser und andere Aspekte sollen im folgenden Abschnitt ausgeführt werden.

6 Die praktische Anwendbarkeit von Z

Die große Flexibilität und Offenheit von **Z** hat auch ihren Preis. Bei der Projektarbeit zeigte sich, daß es keine konkreten Vorgaben gibt, wie ein Dokument gegliedert und aufgebaut werden und in welcher Weise ein Prosatext eingebunden werden soll, um

eine verständliche und anpassungsfähige Spezifikation zu erhalten. Diesbezügliche Konventionen mußten daher während der Projektarbeit erst entwickelt und festgelegt werden. Wie Hoare in seinem Vorwort zu [2] feststellt, war auch hier das Abfassen des informellen Prosateils noch schwerer zu vermitteln und auszuüben als die Beherrschung der formalen Notationen und Konzepte.

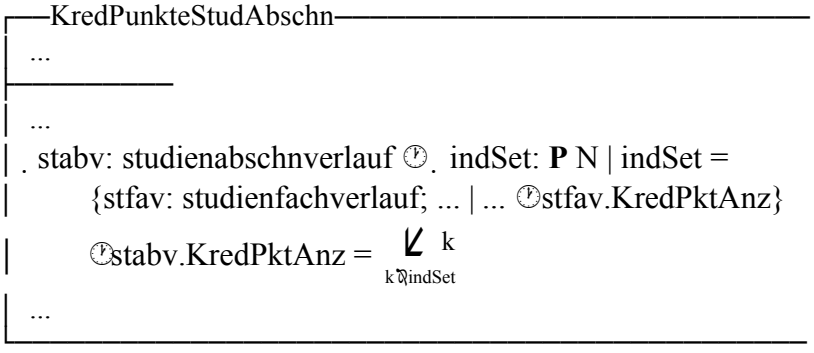
Inwieweit uns dies gelungen ist, kann der Leser über WWW insbesondere in [III] nachvollziehen. Bezüglich der Gliederung gibt es in Anlehnung an das vor ca. 10 Jahren entworfene objekttyporientierte Vorgehensmodell OBAS [13] eine Unterteilung in *Grundobjekttypen* (z.B. `Student`) und *Vorgangsobjekttypen* (z.B. `Studienverlauf`), an die jeweils *Objektfunktionen* (z. B. `anlegenStudent`) bzw. *Berechnungsfunktionen* (z. B. `generiereNoten`) gekoppelt sind. Diese Sicht eines betrieblichen Informationssystems betont den grundlegenden Unterschied zwischen Daten erzeugenden Funktionen, die persistent gespeicherte Daten produzieren, und rein temporäre Ergebnisse erzeugenden Funktionen. Natürlich sind auch andere Sichten und damit Gliederungsschemata denkbar.

In Abschnitt 3 wurde in einem kurzen Beispiel erläutert, wie mit Hilfe der Prädikatenlogik Aussagen über Zustände formuliert werden können, die über eine bloße Typdeklaration hinausgehen. Aus vorhandenen Bedingungen lassen sich weitere Bedingungen logisch ableiten, die wiederum an einer anderen Stelle der Spezifikation eine wichtige Voraussetzung bilden. Die Konsistenz der verschiedenen Bedingungen kann also formal überprüft d. h. bewiesen werden. Bei der Spezifikation eines Informationssystems lassen sich der Zustandsraum persistenter Daten und die auf sie zugreifenden Operationen einander gegenüberstellen. Die Vor- und Nachbedingungen dürfen dabei den invarianten Eigenschaften des Systems nicht widersprechen. Es läßt sich somit formal untersuchen, ob das Datenmodell den funktionalen Anforderungen genügt.

Wie von uns aufgrund anderer Erfahrungen erwartet, lernte nur eine Minderheit der Projektteilnehmer die benötigten Formalismen wirklich beherrschen. Die in der Informatik verbreitete Behauptung, formale Spezifikationen verhinderten oder verringerten Fehler, erscheint vor dem Hintergrund dieser praktischen Erfahrung kühn. Die Klage über eine mangelnde Vorbildung vieler Studenten bezüglich mathematischer und logischer Grundkonzepte ist zum Teil berechtigt, greift aber letztendlich zu kurz. Zum einen scheinen die „Champions“ unter den Formalisten zu wenig zur Kenntnis zu nehmen, was für einen durchschnittlichen Studenten oder Programmierer als leicht und eingängig vorausgesetzt werden darf (vgl. hierzu [6]), zum anderen sind die einschlägigen Bücher und Nachschlagewerke zu sehr auf die mathematische Fundierung von Grundkonzepten als auf deren praktische Umsetzung in einer umfangreichen Spezifikation ausgerichtet (dies wird besonders deutlich in den Büchern von Spivey [15] und [16].)

Es besteht ein großer Unterschied darin, einen Formalismus lesend zu verstehen oder ihn selbst aktiv jenseits der kleinen Demonstrationsbeispiele zu gestalten. Wenn der von uns ansonsten sehr geschätzte Anthony Hall feststellt: *‘The fact is that the mathematics for specification is easy.’* [7, 16], übersieht er die Problematik der adäquaten

Anwendung der Formalismen. Dabei liegen die Schwierigkeiten nicht allein in einer angemessenen Abstraktion der Realität begründet. Bereits die Suche nach der am wenigsten umständlichen Notation ist ein vom Entwickler im Einzelfall zu lösendes Problem. Dies behindert eine Verwendung von **Z** für praktische Entwicklungen. In unserem Projekt wurde z. B. lange darüber nachgedacht, wie die Aufsummierung sogenannter „Kreditpunktzahlen“ wiedergegeben werden sollte, die sich aus bestimmten Leistungen für einen Studenten ergeben. In einem konkreten Fall einigte man sich schließlich auf folgende Form:



Hier soll also zu einem bestimmten „Studienabschnittsverlauf“ *stabv* eine Kreditpunktanzahl ermittelt werden, die sich aus der Aufsummierung aller Kreditpunktzahlen der zugehörigen „Studienfachverläufe“ ergibt, welche wiederum in der Indexmenge *indSet* zusammengefaßt sind. Die Auslassungspunkte innerhalb der Mengenklammern deuten die hier weggelassenen Deklarationen an, sowie die notwendigen Bedingungen, die zwischen den deklarierten Variablen bestehen. Für den Fall einer leeren Indexmenge wurde zusätzlich die Konvention vereinbart, daß dann die Summe den Wert *0* ausmacht.

Neben einer Vielzahl von falschen Lösungsvorschlägen wie z.B.

gab es auch den Vorschlag, anstelle der Indexmenge eine Sequenz von Kreditpunktzahlen zu spezifizieren, um den Index der Sequenz für die Indizierung der Summe zu verwenden. Dies wäre eine Überspezifikation zu Lasten der oben gezeigten eleganteren und einfacheren Darstellung. Gerade die Mächtigkeit der Sprache verlangt besondere Fähigkeiten, sich in dieser Sprache geschickt auszudrücken. Selbst wenn wir die formalen Konzepte als einfach betrachten, bedeutet es noch lange nicht, daß auch die Anwendung dieser Konzepte auf bestimmte Problemstellungen genauso einfach ist.

Unserem Projekt legten wir die Arbeit von de Barros und Harper [1] zugrunde, die zeigt, wie ein Relationenmodell in eine **Z**-spezifische Form gebracht werden kann. Die Übertragung erscheint zunächst als Methodenmonismus. Er erhält erst dann eine Berechtigung, wenn die Relationenschemata zu größeren Einheiten zusammengefaßt wer-

den, die die bereits beschriebenen Deklarationen und Prädikate erben. Für die Darstellung komplexer Zusammenhänge, wie z. B. die Beziehung zwischen mehrstufig über Assoziationen nur indirekt miteinander verbundene Relationen mußten hingegen eigene Darstellungsformen gefunden werden. Es fehlte nicht an formalen Grundkonzepten, wohl aber an genügend vielen darauf fußenden Anwendungsverfahren. Es stehen zu wenige auf den betreffenden Problembereich zugeschnittene Darstellungskonzepte zur Verfügung. Es überrascht daher nicht, wenn Bowen in seinen „zehn Geboten" zur Verwendung formaler Methoden fordert: *'Thou shalt have a formal methods guru on call'* [3, 59].

Bezüglich des Problembereichs *Betriebliche Informationssysteme* ergab sich noch ein weiteres Manko: Temporale Aspekte können nicht direkt ausgedrückt werden. Es kann also nicht explizit offengelegt werden, in welche Richtung sich eine Entität innerhalb einer Relation verändern kann, was z. B. bei *langen Transaktionen* notwendig ist. Man denke etwa an die möglichen Zustandsübergänge von einer Anmeldung bis zur abgelegten Prüfung, bei der nur ganz bestimmte Korrekturen möglich sein sollen.

7 Folgerungen

Die Veranstaltungsreihe hat gezeigt, *daß* und *wie* man die formale Spezifikation eines „betrieblichen" Informationssystems in einer Lehrveranstaltung vermitteln kann. Es ergeben sich viele Gemeinsamkeiten des Problembereiches mit Projekten aus der Praxis. Die in den Veranstaltungen mit der Sprache **Z** gesammelten Erfahrungen können daher auf praktische Anwendungen übertragen werden.

Die *Schwierigkeiten* bei der Anwendung von **Z** lassen sich ebenfalls verallgemeinern. Aufgrund der Universalität von **Z** benötigt der Anwender für konkrete Problemklassen Vorgaben und Muster, in welcher Form er sich am besten ausdrücken sollte. Die Frage, *wie* man spezifiziert, kann nicht auf ihn abgewälzt werden, da seine primäre Aufgabe darin besteht, einen Ausschnitt der Realität zu erfassen und nicht die Ausdrucksform selbst zu finden. Der Spezifizierer benötigt aber auch Vorgaben für den Aufbau des *gesamten* Spezifikationsdokumentes. Hinzu kommt, daß die Beschränkung auf Mengenlehre und Prädikatenlogik für unsere Problemklasse nicht ausreicht, da auch temporale und modale Aspekte spezifiziert werden müssen.

Insgesamt hat es sich gerade für die Zielgruppe *Studenten der Wirtschaftswissenschaften* als fruchtbar erwiesen, formale Spezifikationsmethoden zum Gegenstand von Lehrprojekten zu machen. Es wird hierdurch ein gewisser Immunisierungseffekt gegen den Eindruck erreicht, mit graphisch orientierten Methoden zur Geschäftsprozeß- und Datenmodellierung sei die Arbeit beim fachlichen Entwurf von Informationssystemen bereits geleistet. Vielmehr lernt der Teilnehmer auch, daß sich manche Sachverhalte besser formal als graphisch erfassen lassen.

Danksagung. Wir danken Frau Debora Weber-Wulff für wichtige Hinweise und die engagierte Betreuung der Endfassung als Gutachterin.

Literatur

[1] deBarros, R.S.M.; Harper, D.J.: A Method for the Specification of Relational Database Applications. In: [10], 261-285.

[2] Bjørner, D.; Hoare, C.A.R.; Langmaack (eds.): VDM '90, VDM and Z – Formal Methods in Software Development. Springer, Berlin – Heidelberg et al. 1990.

[3] Bowen, J.P.; Hinchey, M.G.: Ten Commandments of Formal Methods. In: Computer 28(1995) 4, 56-63.

[4] Denert, E.: Dokumentenorientierte Software-Entwicklung. In: Informatik Spektrum 16(1993) 3, 159-164.

[5] Diller, A.: Z - An Introduction to Formal Methods. John Wiley & Sons, Chichester et al. 1990.

[6] Finney, K.: Mathematical Notation in Formal Specification: Too Difficult for the Masses? In: IEEE Trans. on Software Engineering, 22(1996) 2, 158-159.

[7] Hall, A.: Seven Myths of Formal Methods. IEEE Software, 7(1990) Sept., 11-19.

[8] Heisel, M.; Weber-Wulff, D.: Korrekte Software: Nur eine Illusion? In: Informatik F&E 9(1994) 4, 192-200.

[9] Lewis, T.: Where Is Software Headed? In: Computer 28(1995) 8, 20-21.

[10] Nichols, J.E. (Hrsg.): Z User Workshop. York 1991, Springer, Berlin - Heidelberg et al. 1992.

[11] Ortner, E.: Unternehmensweite Datenmodellierung als Basis für integrierte Informationsverarbeitung in Wirtschaft und Verwaltung. In: Wirtschaftsinformatik 33(1991) 4, 269-280.

[12] Schäfer, J.P.; Grauer, M. (Hrsg.): Universitätsverwaltung und Wirtschaftsinformatik. Proceedings, Siegen Okt. 1995.

[13] Spitta, Th.: Software Engineering und Prototyping. Springer, Berlin - Heidelberg et al. 1989.

[14] Spitta, Th.; Mordau, J.: Entwicklung und Ergebnisse eines allgemeingültigen Fachkonzeptes für die Prüfungsverwaltung an Hochschulen. In: [12]

[15] Spivey, J.M.: Understanding Z – a specification language and its formal semantics. University Press, Cambridge 1989.

[16] Spivey, J.M.: The Z Notation – A Reference Manual. Prentice Hall, New York, London et al. 1992.

[17] Würges, H.: Some Remarks on the Use of Abstract Specifications for Operating Systems. In: ACM Software Engineering Notes 3(1978) 3, 8-12.

Projektergebnisse, verfügbar unter http://www.wiwi.uni-bielefeld.de/~spitta/Homepage.html

[I] Spitta, Th.; Mordau, J.: BI-Projekt 'SP': Fachkonzept. Universität Bielefeld, Fakultät für Wirtschaftswissenschaften, Arbeitspapier WS 1994/95.

[II] Mordau, J.; Spitta, Th.: BI-Projekt 'MS': Spezifikation einer Datenbankanwendung mit Implementierungsansatz. Universität Bielefeld, Fakultät für Wirtschaftswissenschaften, Arbeitspapier SS 1995.

[III] Mordau, J.; Spitta, Th.: BI-Projekt 'MS': Spezifikation einer standardisierten Datenbankanwendung. Universität Bielefeld, Fakultät für Wirtschaftswissenschaften, Arbeitspapier WS 1995/96.

Dipl.-Kfm. Jörn Mordau, Prof. Dr. Thorsten Spitta
Universität Bielefeld, Fak. für Wirtschaftswissenschaften,
Postfach 10 01 31,
D-33501 Bielefeld
E-mail: {JMordau|ThSpitta@wiwi.uni-bielefeld.de}