Konstruktive Methoden der Softwareentwicklung

Friedbert Jochum

Interner Forschungsbericht SG 2/2001 der Fachgruppe Systemgestaltung
Fachbereich Informatik, Fachhochschule Köln
Februar 2001

Kurzfassung

Im vorliegenden Beitrag wird ein Lehrkonzept vorgestellt, bei dem zielgerichtete, begründete Handlungsweisen (konstruktive Methoden) für die Entwicklung von Software im Vordergrund stehen. Neben einer hohen *Produktqualität* wird bei solchen Methoden auch eine hohe *Prozessqualität* angestrebt. Dies macht Softwareentwicklung nachvollziehbar und damit auch besser lehr- und lernbar. Der gewählte methodische Ansatz ist im Kern universell insofern, dass sich die Grundkategorien weder an einem speziellen Anwendungsgebiet noch an einem Programmierparadigma orientieren.

Viele Lehrbücher der Softwaretechnik folgen einer *Abbildtheorie*, wonach es gilt, bei der Softwareentwicklung die Anwendungswelt durch ein (formales) Modell und später durch die Software selbst möglichst naturgetreu nachzubilden. Dies zeigt sich z.B. auch in dem oft vertretenen Anspruch der objektorientierten Programmierung, Anwendungsobjekte durch Softwareklassen zu repräsentieren. Dabei ist im konkreten Fall meist völlig unklar, wie die relevanten Anwendungsobjekte und deren Eigenschaften gefunden werden können.[1] Was fehlt, sind explizite Verfahrensregeln, nach denen Softwareentwickler systematisch *vorgehen* und bei Schwierigkeiten schrittweise bis zu den Fehlern *zurückgehen* könnten. Das „Original“ wird unkritisch (d.h. „so wie man glaubt, wie es ist“) durch Abstraktion in ein Modell überführt.

Die Abstraktion, als mental/psychologischer Prozeß verstanden, ist nicht weiter nachvollziehbar. Die Adäquatheit des Modells läßt sich daher nicht durch das Herstellungsverfahren garantieren sondern kann erst im Nachhinein während der Anwendung von Prototypen oder des „fertigen“ Systems empirisch überprüft werden. Da Benutzer und Entwickler in der Regel verschiedene Personen mit höchst unterschiedlichen individuellen Interessen, Perspektiven und Kompetenzen sind, besteht die Gefahr, daß nicht die „Anwendungswelt“ nachgebildet wird, sondern das, was die Entwickler dafür halten. Dies gilt in abgeschwächter Weise auch für Ansätze, die zwar die Notwendigkeit einer begrifflichen Gliederung des Anwendungsgebiets unter Einbeziehung der zukünftigen Benutzer betonen, selbst aber den *Prozess* der Begriffsbildung nicht explizit machen (vgl. z.B. /MAOD 97/).

Der *konstruktive Ansatz* versucht diese Schwierigkeiten zu umgehen. Er steht sowohl in der Zielsetzung als auch in der Vorgehensweise in unmittelbarem Gegensatz zur Abbildtheorie.

[1] In Meyer /MEYE 90/ wird das Auffindung relevanter Objekte sogar als akademische Diskussion abgetan, mit der Entwerfer normalerweise ihre Zeit nicht verbringen sollten (S. 55).

Software dient hier nicht der möglichst originalgetreuen Abbildung von Weltausschnitten, sondern der Bewältigung (fach-)praktischer Probleme. Ob Tabellenkalkulation, Computer Aided Design oder Electronic Business[2]: immer geht es elementar darum, *menschliches Handeln* computertechnisch zu unterstützen. Generelles *Konstruktionsziel* ist also eine *computergestützte Praxis*, in der Benutzer den Computer als Arbeitsmittel *zweckgerichtet* für übergeordnete Aufgaben einsetzen und automatisierbare Teilaufgaben softwaretechnisch auf dem Computer operationalisiert sind. Die Software beschreibt somit operationalisierte (Teil-) Handlungen (Operationen), die zur Ausführungszeit interaktiv in menschliche (Teil-) Handlungen eingebettet sind, wobei die Interaktionen immer symbolvermittelt erfolgen. Um ein möglichst effektives und effizientes Arbeiten der Benutzer zu ermöglichen (*Benutzungsrationalität*), muss die Software nicht nur funktional und prozessual sondern auch sprachlich/begrifflich optimal in den fachlichen Kontext eingebunden sein. Methodischer Unterbau der konstruktiven Softwareentwicklung ist folglich das fachliche *Handeln und Reden der Benutzer*. Darüber hinaus sollte die Software aufgrund ihrer Architektur leicht änderbar, erweiterbar und wiederverwendbar (*Wartungsrationalität*) und ggf. leicht portierbar (*Übertragungsrationalität*) sein. Hierbei ist das fachliche *Handeln und Reden der Entwickler* der methodische Unterbau. Es werden also zwei Konstruktionsziele angestrebt: (a) die Unterstützung der Benutzertätigkeit, und (b) die Unterstützung der Entwicklertätigkeit. Dem wird im Projektmodell durch die beiden aufeinander aufbauenden Teilaktivitäten *Aufgabenrekonstruktion* und *Systemkonstruktion* Rechnung getragen.

Der *Konstruktionsprozess* insgesamt wird als *sprachlogische Handlung* verstanden und nicht als mental/psychologischer Vorgang. Es wird kein „Original“ als objektiv gegeben postuliert. Die durch die Software zu unterstützenden Aufgaben werden vielmehr auf der Grundlage der Fachsprache der Benutzer von allen Beteiligten gemeinsam (im Idealfall von den Auftraggebern, den zukünftigen Benutzern und den Entwicklern) *sprachkritisch* rekonstruiert, d.h. normsprachlich neu beschrieben. Durch diese schrittweise sprachlogische *Herstellung* und Beschreibung durch geklärte Fachtermini wird die Aufgabenstellung besser verstanden als bei einer unkritischen Repräsentation durch ein formales Modell. Jeder einzelne Konstruktionsschritt wird in einem Dialog argumentativ begründet, wobei nur solche sprachlichen Konstrukte verwendet werden dürfen, die vorher *explizit*, *eindeutig*, *zirkelfrei* und *einvernehmlich* eingeführt wurden.

Prädikation, Abstraktion und Komposition sowie deren Umkehrungen sind die elementaren Sprachhandlungen, die beim Konstruktionsprozess durchgehend zur Anwendung kommen. Bei der *Prädikation*, die den methodischen Anfang der Konstruktion bildet, werden (außersprachliche) Gegenstände (Dinge, Geschehnisse) und Wörter (Prädikatoren) durch Zeigehandlung oder Demonstration bei gleichzeitiger Benennung miteinander verknüpft. Dies ist nur dann erforderlich, wenn eine Verständigung mit sprachlichen Mitteln nicht gelingt. Bei der *Abstraktion* (Umkehrung *Konkretion*) werden zum Beispiel bzgl. bestimmter inhaltlicher Aspekte (z.B. Synonymie) mehrere Prädikatoren unter einen Begriff subsumiert oder aus mehreren Artbegriffen auf einer höheren Sprachebene ein Gattungsbegriff gebildet (Ober/Unterbegriffs-Beziehung) oder aus spezialisierten Klassen eine Oberklasse (Vererbungsbeziehung). Bei der Algorithmisierung wird von den Inhalten ganz abstrahiert. Begriffe werden nur noch nach strukturellen Aspekten zusammengefasst: Dingbegriffe zu einer Datenstruktur, Geschehnisbegriffe zu einer Operation. Bei der *Komposition* (Umkehrung *Partition*) werden mehrere (konkrete oder abstrakte) Gegenstände zu einem neuen Gegenstand auf der gleichen Sprachebene zusammengefügt (z.B. Klassen aus

[2] Eine Liste, die sich beliebig mit alten oder neuen Schlagwörtern verlängern ließe: DTP, CIM, CAD, CAE, CASE, CSCW, EC, EB, CBT, Computertomographie, Information Retrieval, Workflow Management, Online Banking etc.

Attributen und Methoden, Aggregationen oder Assoziationen aus mehreren Klassen). Während die Abstraktionshandlung auf einer *Gleichheitsrelation* basiert, liegt der Kompositionshandlung eine *Abhängigkeitsrelation* zugrunde.

Neben der Prädikation sind zur praktischen (außersprachlichen) Verständnissicherung und Validierung die beiden Teilaktivitäten Aufgabenrekonstruktion und Systemkonstruktion über *evolutionäre Prototypen*[3] miteinander verkoppelt. Wir sprechen hierbei von einem *dialogisch/praktischen Begründungsverfahren*, bei dem Reden und Handeln miteinander verknüpft sind, und bei dem insbesondere jeder einzelne Konstruktionsschritt explizit dokumentiert ist.

Aufgrund des konstruktiven Aufbaus und der Verwendung normierter sprachlicher Mittel sind die Resultate *transsubjektiv* nachvollziehbar. Auftretende Verständnisschwierigkeiten (etwa durch mehrdeutige, vage, inkonsistente, unvollständige oder fehlende Begriffe) können frühzeitig als Konstruktionsfehler erkannt, bis zu den Ursprüngen zurückverfolgt und durch erneute Rekonstruktionszyklen behoben werden. Dies ist für die praktische Anwendung von großem Vorteil, da sich oft erst während der Rekonstruktion zurückliegende Missverständnisse aufgrund eines unterschiedlichen Vorverständnisses herausstellen, so dass die methodische Lücke nachträglich geschlossen werden muss.

Grundlage des konstruktiven Ansatzes ist eine „handlungstheoretisch erweiterte Fassung“ der *konstruktiven Wissenschaftstheorie*. Diese liefert einen allgemeinen Leitfaden für den intersubjektiv nachvollziehbaren und gültigen Aufbau von Theorien[4]. Sie basiert im Wesentlichen auf der Abstraktionstheorie von Frege und wurde insbesondere von Lorenzen in der heutigen Form vorgelegt (vgl. /KALO 96/ und /LORE 87/).

Die Forderung nach einer Anwendung der konstruktiven Wissenschaftstheorie in der Informatik ist nicht neu (vgl. z.B. /LUFT 81/, /LUKÖ 94/). In die gleiche Richtung zielt auch die von Schefe /SCHE 99/ erhobene Forderung, in der Softwaretechnik das Paradigma der abbildenden Modellierung durch ein Paradigma der normierenden Beschreibung abzulösen. Ch. Floyd /FLOY 92/ plädiert für eine Softwaretechnik auf der Grundlage des Radikalen Konstruktivismus[5]. Die konstruktive Wisenschaftstheorie von Lorenzen wurde in der Informatik bisher in den Bereichen Datenbanksysteme (vgl. /WEDE 91/ und /GRAB 99/), objektorientierte Schemaentwicklung (vgl. /WEDE 92/), objektorientierter Fachentwurf (vgl. /ORTN 93/ und /SCHI 97/), Information Engineering (vgl. /CUWY 88/), Expertensystem-Entwicklung (vgl. /GUNI 94/) und Workflow-Management-Systeme (vgl. /WEDE 96/) eingeführt. Ein Überblick über Ansätze in der Softwaretechnik vor dem Hintergrund des Radikalen Konstruktivismus findet sich in /FLOY 93/.

Die im vorliegenden Beitrag bevorzugte *handlungstheoretische Grundlegung* der konstruktiven Wissenschaftstheorie von Lorenzen folgt einem Vorschlag von Janich (vgl.

---

[3] Die frühzeitige Erstellung von Prototypen kann zwar auch ohne konstruktive Methode Verständnisdefekte aufdecken, macht sie aber nicht sprachlich explizit, da sie im Code verborgen sind. Prototyping gewinnt daher erst durch die methodische Einbettung in den sprachkritischen Konstruktionprozess an Aussagekraft.

[4] Nach Naur /NAUR 85/ kann auch die Programmierung als Theoreienbildung angesehen werden, wobei der Entstehungsprozeß selbst einen wichtigen Anteil am Verständnis des Resultats hat.

[5] Die grundlegenden Unterschiede zwischen der konstruktiven Wissenschaftstheorie und dem aus der theoretischen Biologie stammenden Radikalen Konstruktivismus werden ausführlich in /JAN 96/ diskutiert. Nachteilig für unsere Belange ist, dass der Radikale Konstruktivismus Konstruktionen als mentale Prozesse beschreibt, die von außen nicht zugänglich bzw. nachvollziehbar sind.

/JANI 97).[6] Sie ermöglicht eine konsequentere methodische Verankerung der Softwareentwicklung in der Praxis der Benutzer. Der von Jacobson /JAKO 92/ eingeführte „use case-getriebene“ Ansatz, der auch in UML als Use-case-Diagramm seinen Niederschlag gefunden hat, ist ebenfalls als handlungsorientiert einzustufen. Was jedoch fehlt, ist eine konstruktive, sprachkritische Vorgehensweise. Ebenso wie bei den aufgaben- bzw. tätigkeitsorientierten Ansätzen aus arbeitswissenschaftlicher Sicht von Paech /PAE 00/ bzw. von Dahme und Raeithel /DARA 97/.

Der hier grob skizzierte konstruktive Ansatz wurde sowohl in der akademischen Lehre als auch in Projekten[7] und bei Mitarbeiterschulungen in der Softwarebranche praktisch erprobt. Die konsequente Anwendung in Projekten liefert eine zweckrationale, für alle Beteiligten transparente Systemlösung. Es hat sich jedoch gezeigt, dass ein sprachkritisches Vorgehen nicht in allen Fällen empfehlenswert oder möglich ist. So z.B. bei der Entwicklung von Software für Kinder oder für Menschen mit sprachlicher oder geistiger Behinderung. Hier kann ein handlungtheoretisch geleitetes evolutionäres Prototyping von größerem Nutzen sein (vgl. /MÜLL 01/).

In der Langfassung dieses Berichts, die z.Z. in Bearbeitung ist, wird in Kapitel 2 das allgemeine Konstruktionsziel *zweckrationales menschliches Handeln* und die daraus abgeleiteten speziellen Software-Qualitätskriterien *Benutzungsrationalität, Wartungsrationalität* und *Übertragungsrationalität* herausgearbeitet. Kapitel 3 führt die grundlegenden Konstruktionshandlungen *Prädikation, Abstraktion* und *Komposition* sowie deren Umkehrungen *Konkretion* und *Partition* ein. In Kapitel 4 wird das *dialogisch/praktische Begründungsverfahren* erläutert. Kapitel 5 umreißt anhand eines *kleinen Projektmodells* die Vorgehensweise der konstruktiven Softwareentwicklung im Gesamtzusammenhang. In Kapitel 6 werden *praktische Beispiele und Erfahrungen* diskutiert. Eine Zusammenfassung der wichtigsten Aspekte erfolgt in Kapitel 7.

Literatur

/CUWY 88/ Curth, M.; Wyss, H.: Information Engineering – Konzeption und praktische Anwendung, Müchen 1988.

/DARA 97/ Dahme, Ch.; Raeithel, A.: Ein tätigkeitstheoretischer Ansatz zur Entwicklung von brauchbarer Software, in: Informatik-Spektrum 20: 5-12 (1997).

/FLOY 92/ Floyd, Ch.: Software Development as Reality Construction, in : Floyd, Ch.; Züllighofen, H.; Budde, R.; Keil-Slavik, R. (Eds.): Software Development and Reality Construction, Berlin, Heidelberg u.a. 1992, pp. 86 – 100.

/FLOY 93/ Floyd, Ch.: Prototyping, Erkenntnis, Realitätskonstruktion, in: Züllighoven, H.; Altmann, W.; Doberkat E. (Hrsg.): Requirements Engineering ´93: Prototyping, Stuttgart 1993.

/FRWI 95/ Franke, M.; Wilms, H.: Die sprachkritische Rekonstruktion einer Produktentwicklung in der Textilindustrie für die Erstellung eines objektorientierten Softwaresystems, Diplomarbeit am Fachbereich Informatik der FH Köln, April 1995.

/GRAB 99/ Grabenbauer, G.: Konstruktive Datenmodellierung – Die Konstruktion von Datenbankstrukturen aus dv-technischer und fachspezifischer Sicht, Aachen 1999.

/GUNI 94/ Gunia, H.: Sprachkritische Entwicklung von Expertensystemen, Dissertation, Uni Erlangen-Nürnberg, 1994.

[6] Die gegenüber der konstruktiven Wissenschaftstheorie weiterführenden Positionen werden ausführlich in /HAJA 96/ diskutiert.

[7] Stellvertretend für viele sei hier ein mit der Fa. Falke durchgeführtes Projekt genannt (vgl. /FRWI 95/).

/HAJA 96/ Hartmann, D.; Janich, P. : Methodischer Kulturalismus, in: Hartmann, D.; Janich, P. (Hrsg.): Methodischer Kulturalismus – Zwischen Naturalismus und Postmoderne, S. 9 – 69, Frankfurt/M, 1996.
/JAKO 92/ Jacobson, I.: Object-Oriented Software Engineering – A Use Case Driven Approach, Addison-Wesley, 1992.
/JANI 96/ Janich, P.: Konstruktivismus und Naturerkenntnis, Frankfurt/M, 1996.
/JANI 97/ Janich, P.: Kleine Philosophie der Naturwissenschaften, München, 1997.
/KALO 96/ Kamlah, W; Lorenzen, P.: Logische Propädeutik – Vorschule des vernünftigen Redens, Verlag J.B. Metzler, Stuttgart, 3. Auflage, 1996 (1. Auflage 1967).
/LORE 87/ Lorenzen, P.: Lehrbuch der konstruktiven Wissenschaftstheorie, Mannheim, Wien, Zürich, 1987.
/LUFT 81/ Luft, A.L: Software-Engineering und konstruktive Wissenschaftstheorie – Ein Beitrag zur Methodologie des Software Engineering, in: Angewandte Informatik 23 (1981), S. 93 – 99.
/LUKÖ 94/ Luft, A.L.; Kötter, R.: Informatik – Eine moderne Wissenstechnik, Mannheim, Leipzig u.a., 1994.
/MAOD 97/ Martin, J.; Odell, J.: Object-Oriented Methods: A Foundation: Uml-Edition, Englewood Cliffs, 1997.
/MEYE 90/ Meyer, B.: Objektorientierte Softwareentwicklung, München, Wien, 1990.
/MÜLL 01/ Müller, K.: Prototypische Entwicklung einer individuellen computerbasierten Lernhilfe für einen geistigbehinderten Jungen, Diplomarbeit am Fachbereich Informatik der FH Köln, Januar 2001.
/NAUR 85/ Naur, P.: Programming as Theory Building, Microprocessing and Microprogramming 15 (1985), 253-261.
/ORTN 93/ Ortner, E.: KASPER – Konstanzer Sprachkritikprogramm für das Software-Engineering, Fachbericht 36-93 der Universität Konstanz, Fachbereich Informationswissenschaft, Konstanz, 1993.
/PAE 00/ Paech, B.: Aufgabenorientierte Softwareentwicklung – Integrierte Gestaltung von Unternehmen, Arbeit und Software, Berlin, Heidelberg, 2000.
/SCHE 99/ Schefe, P: Softwaretechnik und Erkenntnistheorie, in: Informatik Spektrum 22: 122-135 (1999).
/SCHI 97/ Schienmann , B.: Objektorientierter Fachentwurf – Ein terminologieorientierter Ansatz für die Konstruktion von Anwendungssystemen, Stuttgart, Leipzig, 1997.
/WEDE 91/ Wedekind, H.: Datenbanksysteme I – eine konstruktive Einführung in die Datenverarbeitung in Wirtschaft und Verwaltung, Mannheim, Zürich, Berlin, 1991 (3., durchgesehene Auflage).
/WEDE 92/ Wedekind, H.: Objektorientierte Schemaentwicklung, Mannheim, Leipzig u.a., 1992.
/WEDE 96/ Wedekind, H.: Die Erweiterung eines Workflow-Management-Systems um eine Dialogkomponente, in: Ortner, E.; Schienmann, B.; Thoma, H. (Hrsg.): Natürlichsprachlicher Entwurf von Informationssystemen, GI-Workshop, Tutzing, 28.-30. Mai 1996.

Anschrift der Autors:

Prof. Dr. Friedbert Jochum
Fachhochschule Köln - University of Applied Sciences Cologne -
Fachbereich Informatik
Am Sandberg 1
D-51643 Gummersbach

E-mail: jochum@gm.fh-koeln.de
Homepage: http://www.gm.fh-koeln.de/~jochum