# Formale Beschreibungstechniken und praktische Softwaretechnik – eine unglückliche Verbindung ?

## Sieben Thesen

Heinrich Hussmann
Technische Universität Dresden
Fakultät Informatik
Email hussmann@inf.tu-dresden.de

### 1. Einleitung

Praktiker der Softwaretechnik haben meist eine klare Einstellung zu formalen Beschreibungstechniken: Sie empfinden solche Techniken als unverständlich und nutzlos. Auch unter Wissenschaftlern der Softwaretechnik verbreitet sich mehr und mehr die Ansicht, formale Beschreibungstechniken seien Musterbeispiele für die Verschwendung von Forschungsressourcen.

Wie kommen solche harten Urteile zustande? Wie hat das Gebiet der formalen Beschreibungstechniken zu solch einer Einschätzung beigetragen? Und vor allem: Ist es möglich und sinnvoll, durch gezielte Arbeit an formalen Beschreibungstechniken eine bessere Einbettung in die praktische Softwaretechnik zu erreichen?

In diesem Vortrag werden einige Thesen aufgestellt, die eine Annäherung an eine Beantwortung dieser (leider bereits häufig erfolglos diskutierten) Fragestellung ermöglichen. Die Betrachtung erfolgt von einem relativ selten anzutreffenden fachlichen Hintergrund aus, nämlich jeweils mehrjähriger Erfahrung sowohl in formalen Beschreibungstechniken als auch in praktischen Software-Projekten sowie in der akademischen Softwaretechnik.

Die im folgenden Text gemachten Aussagen sind in keiner Weise als Kritik an einzelnen Forschungsergebnissen und –projekten zu verstehen. Es wird versucht, pauschale Trendaussagen zu machen, weshalb naturgemäß eine differenzierte Behandlung einzelner Ansätze nicht möglich ist.

### 2. Der Krieg der Welten

Über lange Zeit blieb das Arbeitsgebiet der formalen Beschreibungstechniken nach außen hin (also z.B. zur Softwaretechnik hin) stark abgeschottet. Bis heute zeigt etwa die Mehrzahl der Lehrbücher zu Abstrakten Datentypen das Erscheinungsbild eines mathematischen Lehrtextes, trotz der eindeutig softwaretechnischen Zielsetzung. Dadurch wurde der Eindruck erweckt, von diesen Hilfsmitteln könne man nur nach ausgiebiger Beschäftigung mit den entsprechenden mathematischen Grundlagen profitieren.

Formale Ansätze zur Softwareentwicklung verstanden sich zu Beginn als radikale Alternative zur „konventionellen Softwareentwicklung“ [D76, BW82]. Es wurde die große (und nach wie vor faszinierende) Vision entwickelt, man könne Programmentwicklung systematisch als einen

Vorgang innerhalb eines mathematisch-logischen Kalküls begreifen, der von den (in logische Formeln transformierten) Anforderungen ausgehend zu einem lauffähigen und effizienten Programm führte.

Dieser radikale Grundansatz führte zu verschiedenen grundlegenden Problemen:

1. Es verfestigte sich die Idee, formale Beschreibungstechniken seien besonders geeignet für die frühen Phasen der Softwareentwicklung, also Anforderungsermittlung, Anforderungsspezifikation und Produktdefinition. Es ist allerdings intuitiv leicht verständlich, daß die frühen Phasen der Softwareentwicklung einen besonders *niedrigen* Grad der Formalität aufweisen, und daß der Grad der Formalität erst im Laufe der Entwicklung allmählich ansteigt. Weiter unten wird erläutert, daß es wesentliche Anwendungsgebiete formaler Beschreibungstechniken in relativ späten Phasen der Entwicklung gibt. Dennoch werden bis heute in den meisten Lehrbüchern für formale Methoden die frühen Phasen betont. (Ausnahmen, wie die Literatur zur Methode B, bestätigen hier die Regel.)

   **These 1:** *Formale Beschreibungstechniken sind wenig geeignet für frühe Phasen der Softwareentwicklung.*

2. Es entstand eine Abkopplung von der Entwicklung in der praktischen Softwaretechnik. Beschreibungstechniken wie sie z.B. in der Strukturierten Analyse (SA) und später in der Objektorientierten Modellierung (OOA/OOD und UML) verwendet wurden, stießen auf größte Skepsis in der „Gemeinde“ der formalen Methoden. Die Tatsache, daß mit diesen Ansätzen zum Teil *aufgrund ihrer niedrigen Formalität* die Entwicklung großer Softwaresysteme übersichtlicher und zielgenauer gestaltet werden konnte, wurde schlicht nicht wahrgenommen. Es entstand eine Konkurrenzsituation, wo eine gegenseitige Ergänzung sinnvoll gewesen wäre.

   **These 2:** *Die Anwendbarkeit formaler Beschreibungstechniken wird dadurch beschränkt, daß diese Techniken keine natürliche Erweiterung von semi-formalen Techniken darstellen.*

3. Es entstand eine dramatische interne Zersplitterung der formalen Beschreibungstechniken. Die Generalisierung verschiedener (für verschiedene Teilaspekte geeigneter) Ansätze zu allgemeingültigen Entwicklungsparadigmen führte zu inkompatiblen Grundkonzepten. Noch heute gibt es selbst für elementare Techniken der Softwarespezifikation eine Vielzahl nur für den Experten vergleichbarer Sprachen und Werkzeuge. Dies ist zum Teil auf die Finanzierung aus öffentlichen Forschungsgeldern zurückzuführen, die viele Spezifikationssprachen und –systeme jeweils nur für eine begrenzte Lebensdauer unterstützten. Dadurch ging (und geht) wertvolle Technologie in der Werkzeugunterstützung verloren, die mit einer nicht weiter geförderten Teilentwicklung untergeht.

   **These 3:** *Die Lehrbarkeit und Anwendbarkeit formaler Beschreibungstechniken leidet unter einem Mangel an anerkannten Standards und Austauschformaten.*

4. Die Entwicklungsgeschwindigkeit formaler Beschreibungstechniken wurde und wird nach wie vor bestimmt von den klassischen Arbeitszyklen wissenschaftlicher Arbeit. Eine wesentliche Innovation ist hier in einem Zeitrahmen von ca. 3-5 Jahren (d.h. der Dauer einer Promotion)

nach Definition der grundlegenden Idee zu erwarten. Die praktische Softwaretechnik ist dagegen eng gekoppelt mit der rasanten Entwicklung des gesamten Softwaremarktes. Dadurch werden neue Technologien oft innerhalb eines Zeitrahmens von 6 bis 9 Monaten nach der Ideenfindung zu einem wesentlichen Faktor (Beispiel: das XML-basierte Austauschformat XMI für Spezifikationen in der Modellierungssprache UML). Diese Kurzatmigkeit bringt natürlich eine Vielzahl inhärenter Gefahren mit sich, aber dennoch können formale Beschreibungstechniken einem solchen Entwicklungsdruck kaum standhalten und laufen damit Gefahr, bald als „völlig veraltet“ zu gelten. Ein typisches Beispiel ist hier der derzeitige Stillstand bei objektorientierten Erweiterungen von formalen Spezifikationssprachen: U.a. Object-Z, Z++, ZEST sind konkurrierende Erweiterungen von Z, die kaum mehr gepflegt und weiterentwickelt werden.

**These 4:** *Für die Weiterentwicklung von Spezifikationssprachen ist der Ansatz wissenschaftlicher Forschungsarbeit nur teilweise geeignet. Kommerzielle Nutzbarkeit ist der Schlüssel zu rascher Weiterentwicklung.*

## 3. Trend zur Formalisierung in der Softwaretechnik

Es existiert eine gewisse Gegenbewegung zu der oben erläuterten Entwicklung. Bei den Beschreibungstechniken, die in der praktisch orientierten Softwaretechnik verwendet werden, ist zu beobachten, daß im Laufe der historischen Entwicklung der Grad der Formalität in den Beschreibungstechniken *ansteigt*. Formalität wird hier verstanden als die Eigenschaft, daß rein syntaktische (formale) Eigenschaften des beschreibenden Dokuments einen wesentlichen Beitrag zur Semantik liefern.

Beispiele für diesen Trend sind:

- Die Technik der Strukturierten Analyse (ca. 1970-1985), die eine höhere Formalität gegenüber natürlichsprachigem Text erlaubt [MP84].
- Die Technik der objektorientierten Modellierung (ca. 1985-1995), die einen deutlich höheren Formalitätsgrad erlaubt als z.B. die Modellierung in der Strukturierten Analyse [R+91].
- Die Definition präziser statisch-semantischer Bedingungen für die moderne Spezifikationsnotation UML (1997) [RJB99] im Gegensatz zu unscharfen Festlegungen in vorgehenden Notationen [OMG98].
- Die Integration einer formalen Beschreibungstechnik, nämlich der Object Constraint Language OCL, in den aktuellen OMG-Standard der UML (1998) [WK99].

Die Erfahrung aus der Projektpraxis zeigt, daß spezielle Modellierungssprachen erst dann wesentlichen Nutzen bringen, wenn eine umfangreiche Unterstützung durch Analyse- und Transformationswerkzeuge besteht. Die notwendige Unterstützung wird mit derzeitigen CASE-Werkzeugen nur in Ansätzen erreicht, was zum Teil auf fehlende Präzision der Beschreibungstechniken zurückzuführen ist.

**These 5**: *Formalität von Beschreibungstechniken ist von grundsätzlichem Vorteil für die praktische Anwendbarkeit, wenn sie leistungsfähige Werkzeuge ermöglicht.*

Diese These mag als ein Widerspruch zu obiger These 1 erscheinen, die Formalität als ungeeignet für die frühen Entwicklungsphasen darstellte. Dieser Widerspruch wird in der praktischen Softwaretechnik durch einen *fließenden Übergang* zwischen relativ informellen und formalen Darstellungen aufgelöst. Als Beispiel kann hier die Darstellungsform von Klassendiagrammen, z.B. in UML, dienen. Klassendiagramme können z.B. verwendet werden, um grobe Skizzen eines Problembereichs und seiner Begriffe anzugeben, um eine grobe Übersicht über einen Softwareentwurf zu geben, aber auch um Software so weit im Detail zu spezifizieren, daß Teile des Programmcodes generiert werden können. Die Eigenschaft des fließenden Übergangs macht es auch möglich, schrittweise formalere Beschreibungstechniken in bestehende Entwicklungsprozesse und Dokumentationsstandards zu integrieren.

Da die klassischen formalen Beschreibungstechniken keine fließenden Übergänge zu weniger formalen Darstellungsformen unterstützten (von wenigen Forschungsansätzen abgesehen), entstanden. sozusagen als Ersatz, formale Erweiterungen semi-formaler Beschreibungstechniken. Einzelne formale Beschreibungstechniken wurden in die Standard-Modellierungssprache UML aufgenommen (z.B. Statecharts, Petri-Netze). Besonders auffällig ist die ebenfalls standardisierte Sprache OCL, die explizit die Ziele formaler Spezifikation erfüllen will, aber keinerlei Rückgriff auf die Tradition formaler Beschreibungstechniken nimmt. Damit entsteht eine ernstzunehmende Konkurrenz für klassische formale Beschreibungstechniken.

**These 6:** *Der Bedarf an Formalität in Beschreibungstechniken wird zunehmend durch Sprachen und Werkzeuge gedeckt, die ausschließlich im Bereich der praktisch orientierten Softwaretechnik verwurzelt sind.*

## 4. Formale Beschreibungstechniken in der Softwaretechnik

Über die oben angesprochene generelle Frage der Modellierung von Systemen hinaus existiert eine Vielzahl von Fragestellungen der Softwaretechnik, in denen der Einsatz formaler Hilfsmittel vielversprechend erscheint. Diese Anwendungsbereiche sind oft dadurch gekennzeichnet, daß Werkzeuge benötigt werden, die in der Lage sind, große Mengen formaler Beschreibungen von Software in geeigneter Weise zu analysieren und weiterzuverarbeiten.

Folgende Anwendungsgebiete (meist aus relativ späten Phasen im Entwicklungsprozeß) erscheinen in dieser Perspektive aussichtsreich:

- **Testunterstützung:** Das Vorhandensein einer formalen Spezifikation macht die Automatisierung von Testaktivitäten möglich. Hier ist ein hoher Grad der Automatisierung, d.h. ein hoher Produktivitätsgewinn, denkbar.
- **Reverse Engineering:** Die Rekonstruktion von Entwurfsmodellen und verschiedenen abstrakten Sichten aus einem gegebenen Programm ist eine Tätigkeit, die von einem rein formalen Ausgangspunkt (nämlich einem vorgegebenen Programmsystem) ausgeht. Formale Methoden erweisen sich hier als ausgesprochen leistungsfähig, siehe z.B. [DKV99].
- **Architekturen und Entwurfskritik:** Es ist anerkannt, daß Software bestimmte Strukturprinzipien befolgen sollte (z.B. niedrige Kopplung, hohe Kohäsion), um hohe Zuverlässigkeit und gute Wartbarkeit zu erreichen. Jedoch fehlt es an präzisen, für konkrete

Programme und Entwürfe überprüfbaren Kriterien für solche Prinzipien. (Beispiel: Eine formale algebraische Spezifikation einer objektorientierten Klassenspezifikation kann Verletzungen des Geheimnisprinzips aufdecken.)

- **Komponentenspezifikation:** Je mehr Software aus vorgegebenen (meist nur im Maschinencode verfügbaren) Komponenten zusammengesetzt wird, und je mehr Haftungsansprüche in der Softwareentwicklung eine Rolle spielen, desto wichtiger wird eine präzise („gerichtsfeste“) Beschreibung der Funktionalität von Softwarekomponenten.
- **Theoretischer Hintergrund semiformaler Beschreibungstechniken:** Es ist hilfreich, vor allem für Ausbildungszwecke, die Zusammenhänge zwischen verschiedenen semi-formalen Beschreibungstechniken präzise beschreiben zu können. Hierfür bieten sich einige formale Beschreibungstechniken an. Wichtig ist hier allerdings vor allem das Gebiet der Kombination von Beschreibungstechniken.
- **Ausbildung:** Formale Beschreibungstechniken können ein „Idealbild“ der Spezifikation von Modul- und Klassenschnittstellen liefern, mit dessen Hilfe sich Fertigkeiten erläutern und trainieren lassen, die im praktischen Entwurf von Systemen auch mit semi-formalen Hilfsmitteln hilfreich sind. Wer formale Spezifikation erlernt, gewinnt eine neue Sicht auch auf die Tätigkeit der informellen Spezifikation.

**These 7:** *In relativ späten Phasen der Softwareentwicklung entsteht ein zunehmender Bedarf an Werkzeugen, die mit formalen Softwarebeschreibungen operieren.*

Bewußt nicht aufgeführt wurden klassische Anwendungsgebiete für formale Beschreibungs- und Entwicklungstechniken wie Systeme mit höchsten Zuverlässigkeitsanforderungen. Für solche „Nischenmärkte“ ist die Einsetzbarkeit und Notwendigkeit formaler Beschreibungstechniken weitgehend unumstritten. Ethische Argumente erlauben alleine aufgrund dieses Anwendungsbereichs eine Rechtfertigung für Entwicklung und Einsatz formaler Methoden. Die obigen Ansätze versuchen jedoch, formale Beschreibungstechniken in einem weiteren Anwendungsfeld zu plazieren.

Aus der obigen Aufzählung wurde deutlich, daß bei vielen Aufgaben der Bedarf für Formalität in der Systembeschreibung in Zukunft steigen wird. Es ist eine Herausforderung an das Gebiet der formalen Beschreibungstechniken, einen wesentlichen Beitrag zu diesen Problemlösungen zu liefern.

## 5. Konsequenzen

In den oben aufgeführten sieben Thesen wurde deutlich, daß die Forschung in formalen Beschreibungstechniken aus historischen Gründen weniger Einfluß auf die praktische Softwaretechnik hat als theoretisch denkbar und wünschenswert wäre. Dieser Einfluß kann nach Ansicht des Autors in Zukunft durchaus stärker wirksam werden. Dazu müssen im Gebiet der formalen Beschreibungstechniken folgende Schwerpunkte gesetzt werden:

- Es ist möglichst weitgehende Kompatibilität mit semi-formalen und informellen Beschreibungstechniken anzustreben. Hierfür ist ein günstiger Entwicklungsstand erreicht, da anerkannte Standards in der Softwaretechnik entstanden sind (insbesondere UML).
- Es sind Lehrmaterialien zu entwickeln, die formale Beschreibungstechniken als natürliche Ergänzung und Erweiterung gängiger praxisrelevanter Entwicklungsmethoden darstellen.
- Formale Beschreibungstechniken müssen durch leistungsfähige Analyse- und Transformationswerkzeuge unterstützt werden, um ihre Stärken zur Geltung zu bringen.
- Bei Werkzeugen ist ein hoher Grad der Automatisierung anzustreben, um den Nutzerkreis zu vergrößern.
- Als Anwendungsgebiet für formale Beschreibungstechniken und die zugehörigen Werkzeuge sind spezielle Themen aus späten Entwicklungsphasen anzustreben.

In nicht allzu ferner Zukunft werden formal ausgerichtete Beschreibungstechniken und zugehörige Werkzeuge als selbstverständlicher Bestandteil der Softwaretechnik angesehen werden. Derzeit werden die Weichen gestellt, ob das traditionelle wissenschaftliche Arbeitsgebiet der formalen Beschreibungstechniken auf diese Entwicklung Einfluß nimmt.

## Literatur


[D76] E.W. Dijkstra, A Discipline of Programming, Prentice-Hall 1976.

[DKV99] A. van Deursen, P. Klint, C. Verhoef, Research Issues in the Renovation of Legacy Systems. In: P. Finance (ed.) FASE‘99 Proceedings, Lecture Notes in Computer Science Vol. 1577, Springer 1999, pp. 1-21.

[BW82] F.L. Bauer, H. Wössner, Algorithmic Language and Program Development, Springer 1982.

[MP84] S. McMenamin, J. Palmer, Essential Systems Analysis, Yourdon Press 1984.

[OMG98] UML Semantics, version 1.1, http://www.omg.org/cgi-bin/doc?ad/97-08-04

[R+91] J. Rumbaugh et al., Object-oriented Modeling and Design, Prentice-Hall 1991.

[RJB99] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley 1999.

[WK99] J. Warmer, A. Kleppe, The Object Constraint Language: Precise Modeling with UML. Addison-Wesley 1999.