

..... the current state of the art of computer programming reflects inadequacies in our stock of paradigms, and in the way our programming languages support, or fail to support, the paradigms of their user communities.
(Robert W. Floyd, 1978)

Auf dem Weg zu idealen Programmierwerkzeugen

aus Erfahrung in Programmierung und SW-Projekten

C. Crasemann, J. Brauer und H. Krasemann, 2005-2009

Programming languages are the fundamental technology of programming. To make fundamental improvements in programming you need a fundamentally better programming language.
(J. Edwards, MIT, 2004)

Auf dem Weg zu idealen Programmierwerkzeugen

Inhalt


- über mich, H. Krasemann
- was ist denn ideal? meine Maßstäbe
- Defizite von Programmiersprachen
- falsche (Hierarchie von) Abstraktionen
- viele Aspekte sind (noch) nicht orthogonal
- Fortschritt in SW durch späte Bindung
- die Modellierungslücke
- DSLs: radikal genug?
- Literatur

Auf dem Weg zu idealen Programmierwerkzeugen über mich

- Physiker, nicht Informatiker
 - FORTRAN, Reduce, Schoonship, 68k Assembler
- Praxis seit 1981 (SCS – T-Systems, selbstständig)
- Schnellkurs Informatik mit TAOCP (Knuth)
- frühe Smalltalk-Projekte (und KI-Sprachen)
 - Smalltalk/V, KEE, ART, Interlisp D, ...
- Bau eines kommerziellen Grafik-FW und Tools in Smalltalk
 - ObjectDrawing
- Konzepte für ein Design-(Re)engineering-Werkzeug in Smalltalk
 - ADvance, ein Add-On für VisualWorks
- extensive Metaprogrammierung in Smalltalk für Werkzeuge
- Projekte von der Versicherung bis zum Verkehr
- 2000-3 und 2006-9: Steuerung von Containerterminals in Hamburg

Auf dem Weg zu idealen Programmierwerkzeugen was ist denn ideal? meine Maßstäbe

- Erfahrungen von Informatikern
 - sind sehr divergent (Arbeitsgebiete, Programmiersprachen, ..)
 - aber auch verschieden relevant
- Ideale sind
 - nicht allgemeingültig, sondern subjektiv
 - begründet in der eigenen (industriellen) Erfahrung
- orientiert an
 - Kosten
 - der Programmierung und der Tests (Fehlermengen etc.)
 - der Wartung und der Dokumentation
 - Adäquadheit, Mächtigkeit, Flexibilität, ...
- geben Anlass zu fruchtbarer (kontroverser) Diskussion



Unterstützung
der Spezifikation

Paradigmen der Programmiersprachen bestimmen die Mächtigkeit (Floyd 78)

<p>1978</p> <ul style="list-style-type: none">■ Strukturierte Programmierung■ Abstrakte Datentypen■ Rekursion und Coroutinen■ Hierarchie von Sprachen (Reduce/Lisp, Cobol/Assembler)	<ul style="list-style-type: none">■ Lisp-Makros
<p>Meta</p>	
<p>1986</p> <ul style="list-style-type: none">■ logische Inferenz (Prolog)■ Objekte (neue eigene Operatoren)■ Polymorphie (late binding)■ Vererbung (Abstraktionsdimension)	

Defizite der Programmiersprachen

Feedback fehlt

■ Die Programmstruktur ist versteckt

Sotograph

- Visualisierung der Konstruktion fehlt („*Ansicht einer SW*“)
 - Kopplung, Modularisierung, Verteilung
- Automatische Qualitäts-Metriken fehlen („*Statik einer SW*“)
 - Größe (LOC), Zahl der Teile (Module, Klassen, etc.)
 - Kopplung, statische und dynamische Abhängigkeiten

■ Die Laufzeitumgebung verheimlicht alles („*Performanz einer SW*“)

Smalltalk 80
DrScheme

- Zeitverhalten: Reaktionszeiten könnten automatisch gemessen werden
- Profiling: welcher Code wird wie oft ausgeführt
- Fehleranfälligkeit: Fehlerspeicher - wann - wo - wieviele
- Durchsatz und Ressourcenverbrauch (CPU, Speicher, Netzwerk-Bandbreite)

Defizite der Programmiersprachen

falsche Hierarchie und fehlende Konzepte

- falsche Konzepte
 - Klassen
 - statische Typen
- fehlende Abstraktionen
 - Design Patterns beweisen fehlende Konzepte
- rekursive Kapselung
- Constraints
- umfassende Schnittstellen-Kontrakte
- Zustandsmaschinen
 - Nebeneffekte sind die größte Fehlerquelle

Metaprogr. ist

- schwach (Java) oder
- gar nicht möglich

A paradigm at an even higher level of abstraction than the structured programming paradigm is the construction of a hierarchy of languages, where programs in the highest level language operate on the most abstract objects, and are translated into programs on the next lower level language.
(Robert W. Floyd, 1978)

Defizite der Programmiersprachen fehlende Orthogonalität (und Optionalität)

- Konfiguration
- Schnittstellen (Kontrakte, Typen)
- Dokumentation
- Persistenz
- Verteilung
- Synchronisation
- Optimierung

Metaprogr. ist

- schwach (Java) oder
- gar nicht möglich

Die optionalen Annotationen erlauben uns, zwischen dem „Konstruktionskontext“ und dem „Betriebskontext“ zu unterscheiden. Im Konstruktionskontext ist zugunsten von Flexibilität, schneller Rückmeldung und Wiederverwendung sehr viel erlaubt (es gibt viele erlaubte Programme), im Betriebskontext dagegen viel weniger (es gibt weniger korrekte Programme).

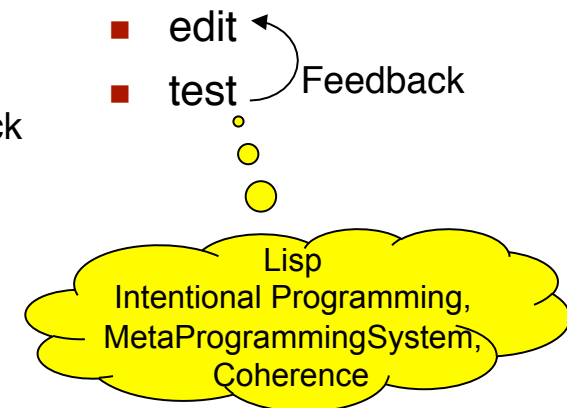
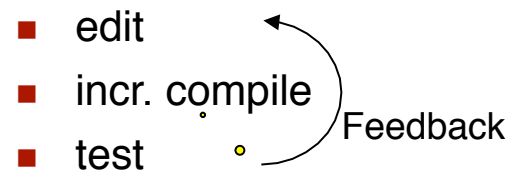
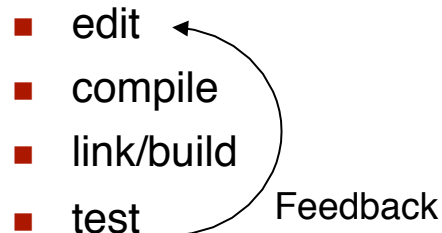
Insbesondere können wir während der Konstruktion problemlos mit undefiniertheiten aus Sicht des Betriebskontextes umgehen. Der Übergang vom Konstruktionskontext zum Betriebskontext ist fließend, realisiert durch viele kleine Schritte des Zuschaltens von Annotationen. (CBK)

Defizite der Programmiersprachen oder: zu frühe Bindung

One way to think about progress in software is that a lot of it has been about finding ways to *late-bind*.
(Alan Kay, HOPL)

Fortschritt der Informatik

- HW: RAM, Index Register, MMU, Time Sharing, ...
- Variablen abstrahieren vom Speicher
- Rekursion bindet Variablen spät an Prozeduren
- Message Dispatch zur Laufzeit (Polymorphie, ink. Kompilieren)
- Prolog Inferenz findet Lösungen und bindet sie dann!
- Der Erstellungsprozess wird schlank durch späte Bindung!



Auf dem Weg zu idealen Programmierwerkzeugen Die Modellierungslücke überwinden

Anwendungssprachen
Domain Specific Languages (DSL)

Konzepte der Anwendung

Modellierungslücke

OO-Sprachen
3 GL-Sprachen
Assembler
Maschinencode

Standard-Algorithmen
Abstrakte Datentypen
Basis-Algorithmen
Basis-Datentypen
Symbolische Adressen
Register

Die Modellierungslücke überwinden

Zu Fuß

Instead of simply writing your application in the base language, you build on top of the base language a language for writing programs like yours, then write your program in it. (P. Graham, Hackers & Painters)

Anwendungssprachen
Domain Specific Languages (DSL)

Problemspezifische
Klassen und Methoden
(Operatoren)

OO-Sprachen

3 GL-Sprachen

Assembler

Maschinencode

Konzepte der Anwendung

eigene Erfahrung mit Smalltalk
Berichte von Lisp-Programmierern

Standard-Algorithmen

Abstrakte Datentypen

Basis-Algorithmen

Basis-Datentypen

Symbolische Adressen

Register

Die Modellierungslücke überwinden

Generieren

Anwendungssprachen
Domain Specific Languages (DSL)

Generieren

OO-Sprachen

3 GL-Sprachen

Assembler

Maschinencode

Konzepte der Anwendung

UML

MDA (OMG)

OpenArchitecture (M. Völter)

Standard-Algorithmen

Abstrakte Datentypen

Basis-Algorithmen

Basis-Datentypen

Symbolische Adressen

Register

Die Modellierungslücke überwinden Metaprogrammieren

Anwendungssprachen
Domain Specific Languages (DSL)

Metaprogrammieren

OO-Sprachen
3 GL-Sprachen
Assembler
Maschinencode

Konzepte der Anwendung

LISP Makros, CLOS
Rebol-Dialecting
Ruby, Python, Grails, ...
In Squeak: Prolog/V, Lisp, Scheme, Forth

Standard-Algorithmen
Abstrakte Datentypen
Basis-Algorithmen
Basis-Datentypen
Symbolische Adressen
Register

Die Modellierungslücke überwinden Metaprogrammieren: Beispiel Prolog/V

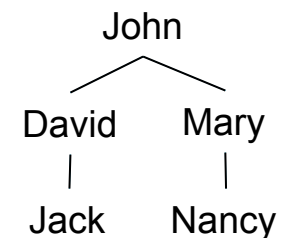
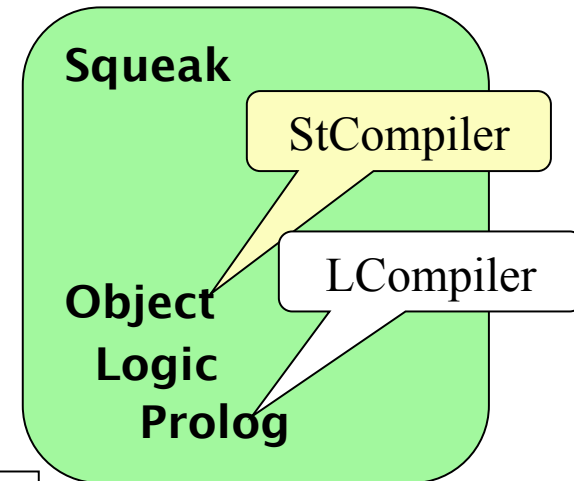
Der Prolog Prompt ist eine Smalltalk Methode

```
Family new :? grandPa('John', x)
(('Nancy') ('Jack'))
```

Family

```
=====rules
grandPa(x, y) :- father(x, z), OR(father(z, y), mother(z, y)).
```

```
=====facts
father('John', 'Mary').
father('John', 'David').
father('David', 'Jack').
mother('Mary', 'Nancy').
```



Die Modellierungslücke überwinden Domain Specific Languages (DSLs)

Anwendungssprachen
Domain Specific Languages (DSL)

Konzepte der Anwendung

MetaEdit

MPS Meta Programming System

IPS Intentional Programming System

OO-Sprachen

3 GL-Sprachen

Assembler

Maschinencode

Standard-Algorithmen

Abstrakte Datentypen

Basis-Algorithmen

Basis-Datentypen

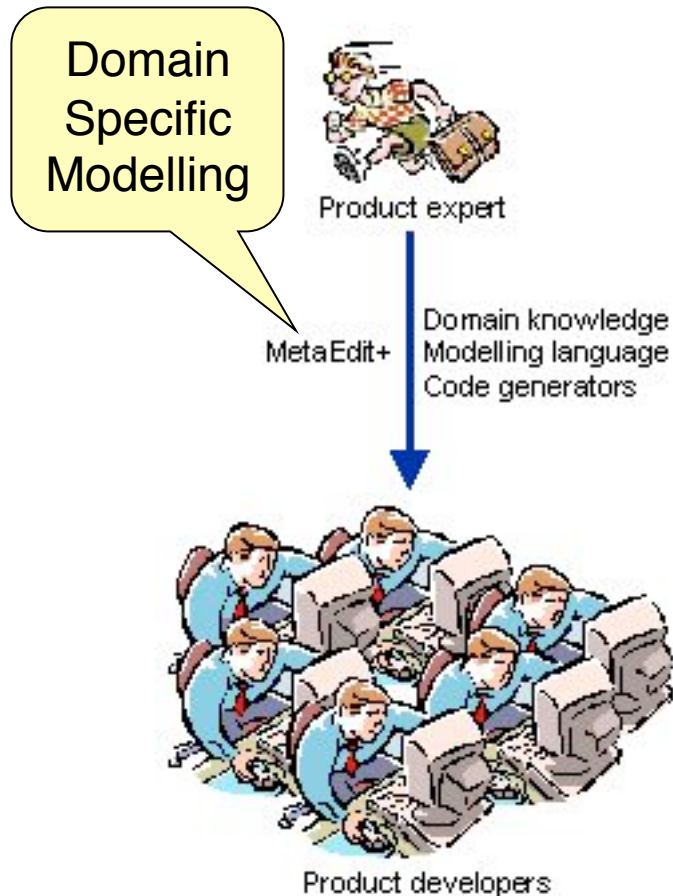
Symbolische Adressen

Register

Die Modellierungslücke überwinden

DSM: 100% generieren im Spezialfall

built with
Smalltalk!



- Method Workbench für Architekten/ Designer
 - Konzepte definieren
 - Regeln festlegen
 - Symbole entwerfen (die Notation)
 - Generatoren bauen
- MetaEdit+ für Entwickler
 - entwerfen wie mit UML, OMT, Fusion, ...
 - vielseitige Sichten
 - 100% generieren: C++, St, Java, Delphi, SQL
 - Integrieren mit Soap / Webservice / .net
 - Dokumente automatisch in Word, HTML
 - Pflege im grafischen Modell !

Die Modellierungslücke überwinden

LOP: Language Oriented Programming

- LOP Language Oriented Programming
 - MPS Meta Programming System 1.0 (**Java**)
 - IPS Intentional Programming System 1.0 (**.net**)
- Programm = Syntaxbaum (Parsetree)
- neue Strukturen erscheinen direkt
- Editor und Debugger sind zwingend
- Benutzer-Notationen
 - mathematische Formeln,
 - Notenschrift,
 - chemische Strukturformeln,
 - Schaltungssymbole,
 - UML-Symbole
 - Zustandsdiagramme
 - ...

Sergey Dmitriev,
JetBrains,
PlugIn für IntelliJ Idea

Charles Simonyi,
Intentional Software

Aber: Das MPS Tutorial beschreibt ein triviales Beispiel mit 90 Bildschirmseiten .
Der Aufwand für die Erstellung eines solch trivialen Beispiels ist ernüchternd.
(B. Humm)

Ausblick: eine Vision die umfassende Language Workbench

Vertikale DSLs, produktspezifisch, z.B.		Produktmaschine	Transport-Editor	Gleisbild-Editor	Spreadsheet	...
Metaprogrammierung						
GP und Workflow-DSL - Regeln			ZM-DSL - Zustandsmaschinen			
orthogonal	Struktur-DSLs - Komponenten, Konfiguration					
	Schnittstellen-DSL - Version, Art (Queries, Kdo.s, Notif.), Inhalte (Ont. Syntax, Zeit)					
	Constraint-DSL - für Konsistenz von Domäne und User Interface					
	Test-DSL - Umweltsimulation, Trigger (Ereignisse), Daten					
	Verteilungs-DSL – was läuft auf welchem Knoten					
	Persistenz-DSL – was ist wo persistent					
Ansicht, Statik, Performanz-Sichtbarkeit						

Ausblick: Muss es radikaler sein?

Zustand, späte Bindung, Codegröße

- Sprache und EU / LU : Coherence / Subtext . . .
MIT: Jonathan Edwards
 - echtes Instant Feedback (wie bei IPS, MPS)
 - programmieren auf dem Syntaxbaum (kein Compiler)
 - edit = compile = link = run
 - attackiert eines der großen Programmierprobleme: Zustände
 - Inferenz über die Reihenfolge der Nebeneffekte
 - Programming by example zur Laufzeit

- STEPS towards the Reinvention of Programming
Viewpoints Research Institute, Alan Kay
 - Experimente in Metaprogrammierung
 - Ziele
 - die richtigen Abstraktionen finden
 - mit viel weniger Code



Ausblick auf die idealen Programmierwerkzeuge

Zusammenfassung der Themen

- Die Defizite sind klar:
 - Feedback / Sichtbarkeit von SW (Statik, Performanz, ...)
 - Falsche Hierarchie und fehlende Konzepte
 - Orthogonalität und späte Bindung
 - Expressivität von SW (Codegröße, DSLs, Meta, ...)
- Die Lösungen sind noch nicht klar:
 - Revival der Lisp/Smalltalk-Techniken ohne Lisp oder Smalltalk
 - Ruby, Python, Rebol, ...
 - als Alternative zum Generieren UML → Java
- Es gibt etwas Neues: Language Workbenches
 - MetaEdit+, MPS, IPS
- Reicht das aus oder brauchen wir mehr?
 - subtext
 - Steps

Auf dem Weg zu idealen Programmierwerkzeugen

Literatur

Crasemann, Brauer und Krasemann

Auf dem Weg zu idealen Programmierwerkzeugen, Informatik Spektrum 31 (2008) S. 580
und Referenzen dort

Floyd The Paradigms of Programming. Comm. ACM 22(8): 455-460 (1979)

Alan Kay The Early History of Smalltalk, in HOPL-II, ACM-Press, 1996

Paul Graham

Hackers & Painters. Essays on the Art of Programming, O'Reilly 2004

Czarnecki & Eisenecker

Generative Programming, Addison Wesley, 2000, Kap. 11,

Fowler www.martinfowler.com/articles/languageWorkbench.html

Intentional Software

www.intentsoft.com

JetBrains www.jetbrains.net, dort MPS klicken

www.onboard.jetbrains.com/is1/articles/04/10/lop/

DSLs (*Literaturliste*) homepages.cwi.nl/~arie/papers/dslbib/

(*in Lisp*) lisp.dyndns.org/news?ID=NEWS-2005-07-08-1

Squeak www.squeak.org ; (*Prolog*) wiki.squeak.org/squeak/1000

Subtext www.subtextual.org/ { [] | subtext2.html | demo1.html }