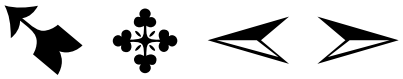
Erfahrungen mit der werkzeug-gestützten Änderung von COBOL-Systemen

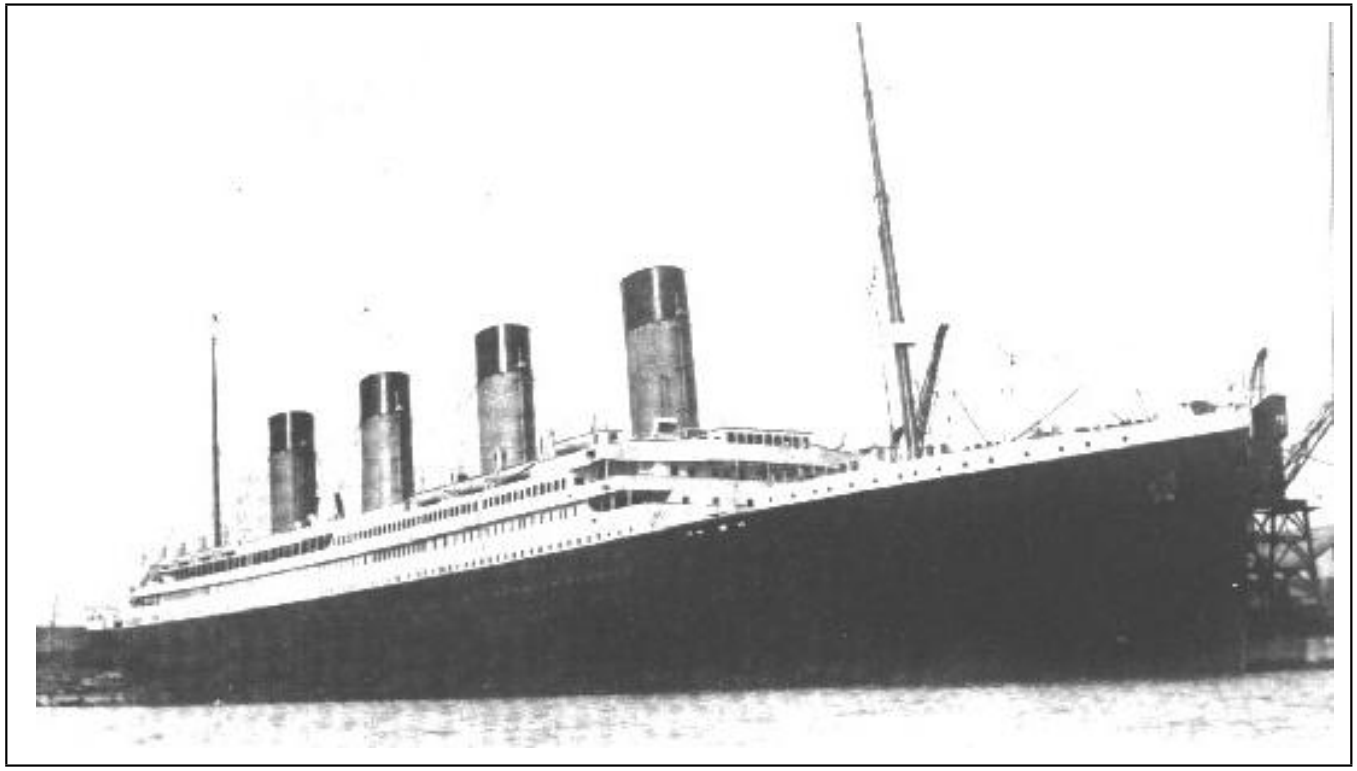
Dr. Jürgen Vollmer

– CoCoLab –

vollmer@cocolab.de • www.cocolab.de



Was ist das Problem?



CoCoLab = Compiler Compiler Laboratory

Gegründet: 1994

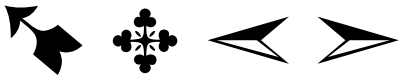
Mitarbeiter: Dr. Josef Grosch, Dr. Jürgen Vollmer

Ziele:
- Entwicklung & Vertrieb der **Cocktail** Toolbox

  Cocktail = Compiler Toolkit Karlsruhe
- Entwicklung von Compilern für Programmiersprachen
- Analyse von Programmsystemen, Reengineering, (Re-)Dokumentation, . . .

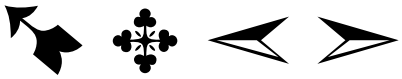
Basis:
- Cocktail Toolbox
- Parser z.B. für: COBOL, CICS, Clist, Rexx, JCL, DL/1, PL/1, Fortran, SQL (DB2, Oracle, Informix, . . .), C/C++/C#, Java, Natural, . . .
- Programm-Analysen für diese Sprachen

Kunden: Banken, Versicherungen, Telekommunikation, Softwarehäuser, . . .

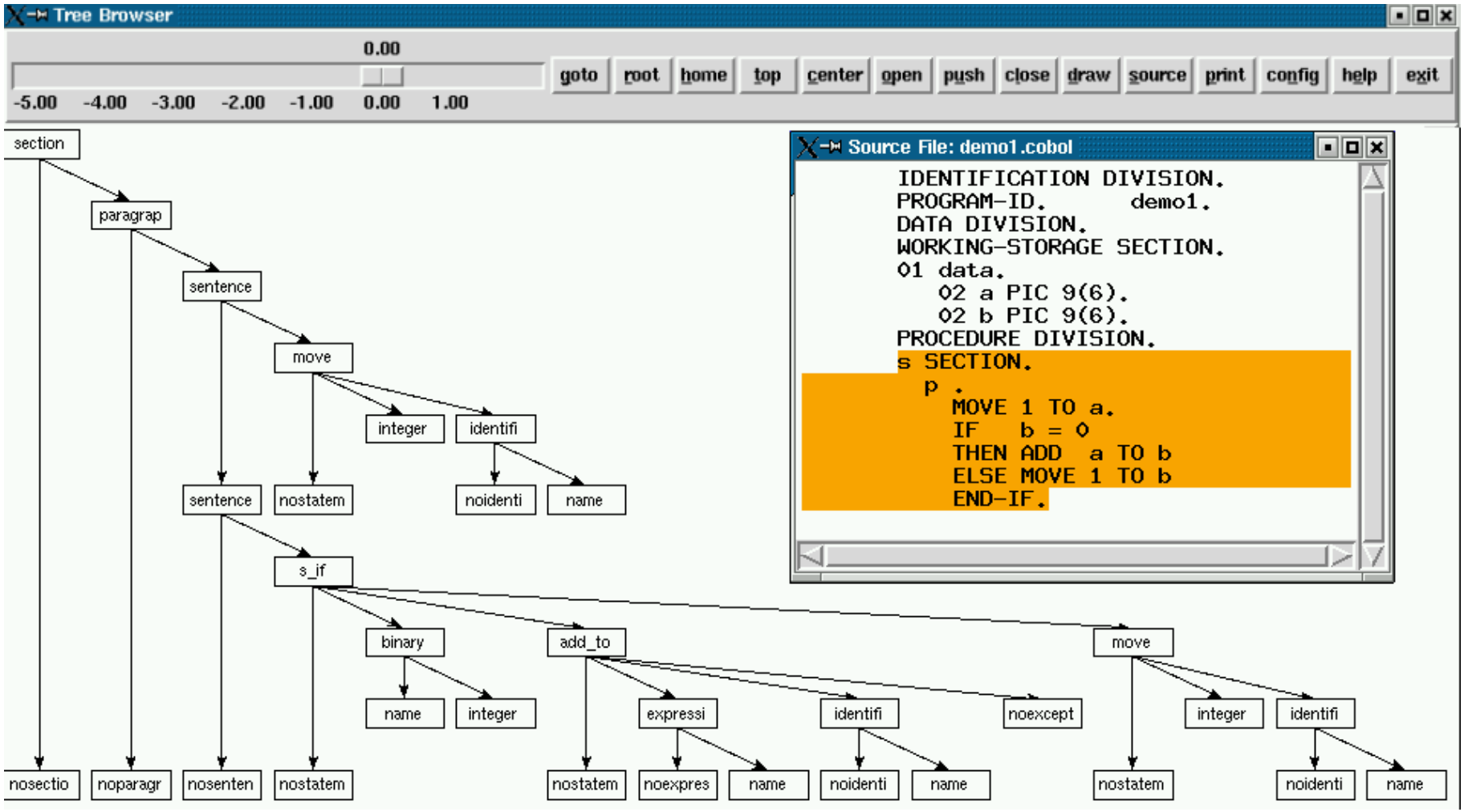



Der CoCoLab-Ansatz

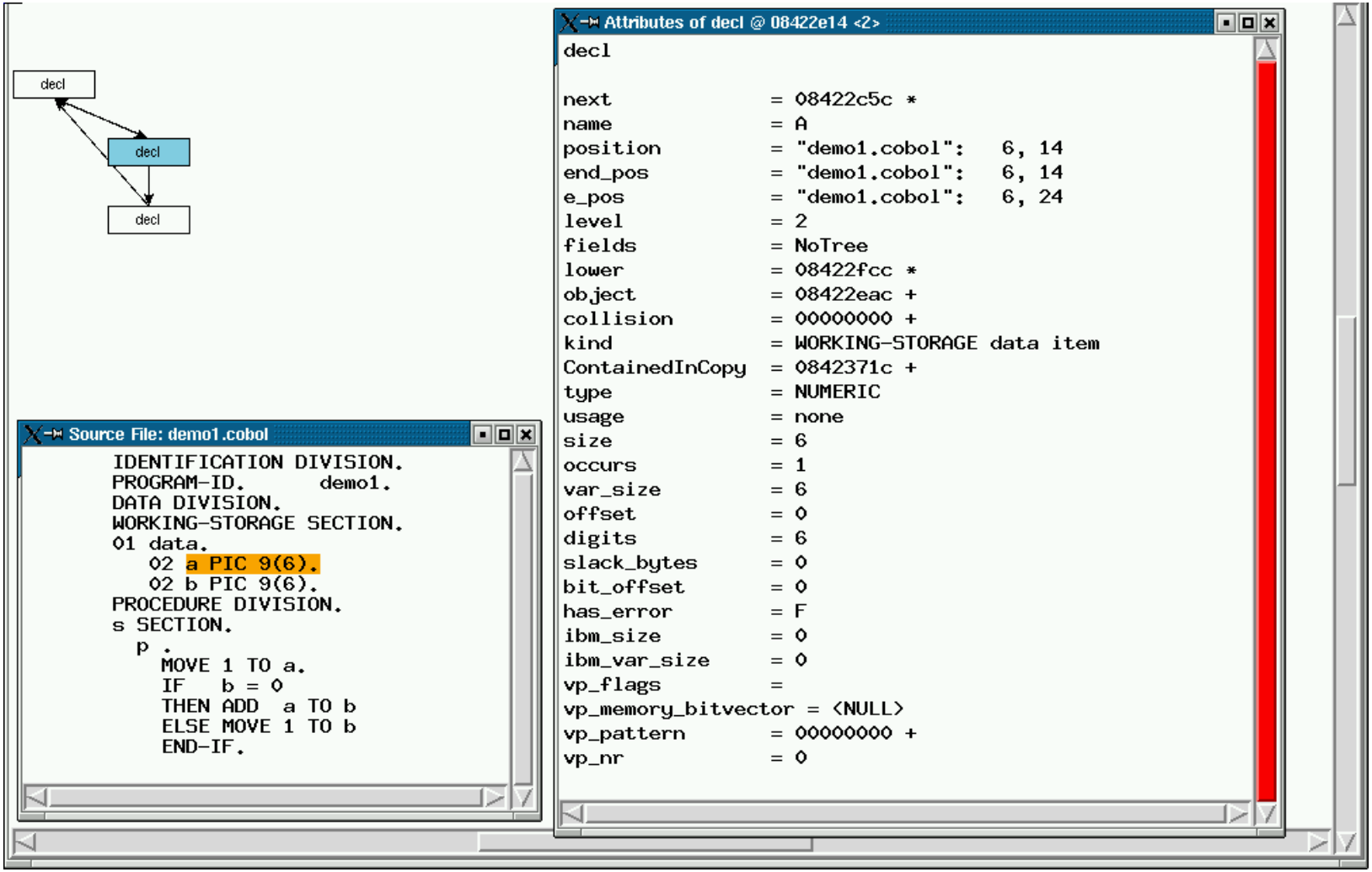
- Interne Darstellungsformen eines Programmes:
  1. Liste von Programmzeilen
     - $\Rightarrow$ einfache Textersetzung möglich
     - $\Rightarrow$ keine Information über die Struktur ds Programmes
  2. Übersetzerbau (*compiler construction*)
     - $\Rightarrow$ Strukturinformation
     - $\Rightarrow$ „Bedeutungs-Information“
     - $\Rightarrow$ mehr Aufwand
     - $\Rightarrow$ **CoCoLab & Cocktail**



Interne Programm Darstellung – Abstrakter Syntaxbaum

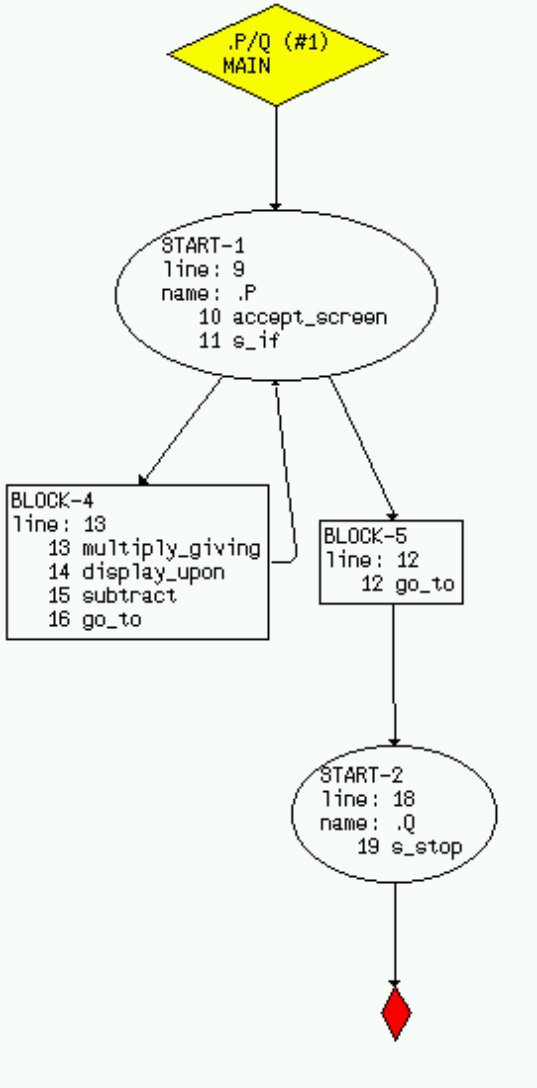


Interne Programm Darstellung – Berechnete Information



Interne Programm Darstellung – Kontrollflußgraph

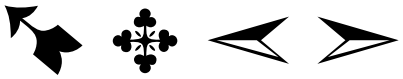
```
 1    IDENTIFICATION DIVISION.
 2    PROGRAM-ID.      demo2.
 3    DATA DIVISION.
 4    WORKING-STORAGE SECTION.
 5    01 data.
 6       02 a PIC 9(6).
 7       02 b PIC 9(6).
 8     PROCEDURE DIVISION.
 9     p.
10         ACCEPT a.
11         IF   a <= 0
12         THEN GO TO q.
13         MULTIPLY A BY A GIVING b
14         DISPLAY b
15         SUBTRACT 1 FROM a.
16         GO TO p.
17
18       q.
19         STOP RUN.
```



Interne Programm Darstellung – Datenflußanalyse

Lesen einer Variablen an einer bestimmten Stelle im Programm:

- Welche Zuweisungen erreichen diese Benutzung?
- Welche konstanten Werte hat eine gegebene Benutzung?



Anwendung: (Re)-Dokumentation

Aufgabenstellung:

Welche Bedeutung hat ein CALL?

```
CALL "handle-file" USING param, args.
```

Beispiel:

Öffnen/Lesen/Schreiben/Schliessen einer Datei.

Erwünschte Ausgabe:

CALL in Zeile 100 öffnet Datei abc zum Lesen im Mode xyz.

Aufgabe des Benutzers:

Spezifikation der Tatsache:

Feld von 1. USING Argument auf Offset 5 legt Aktion fest

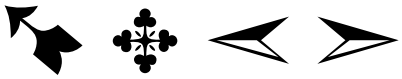
Feld von 1. USING Argument auf Offset 6 die Art der Datei

Feld von 1. USING Argument auf Offset 7-20 den Namen der Datei

Zulösende Probleme:

Finde alle Zuweisungen an relevante Parameter, bestimme deren Werte

(Datenflußanalyse + Konstantenpropagation)



Einfache Transformation

Beispiele:

- Einfügen/Ersetzen syntaktischer Klauseln, z.B. `END-IF`
- Systematisches Ersetzen von bestimmten Bezeichnern:

```
01 data1.
   02 part1.
      10 x PIC 9. unverändert
01 data2.
   02 x PIC 9.   ⇒   01 y PIC 9.
MOVE x OF data1 TO x OF data2. ⇒
MOVE x OF data1 TO y OF data2.
```

- Modifikation von Programmnamen von aufgerufenen Programmen:

```
01 prog X(10).
MOVE "A" to prog.  ⇒  MOVE "B" to prog.
CALL prog.
CALL "A"   ⇒  CALL "B"
```

Transformation: Wertpapierkennummern (WKN)

Problemstellung:

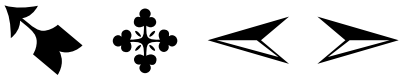
numerisch 6 stellen $\Rightarrow$ alphanumerisch 6 Stellen

Technische Aufgaben:

- Umstellung der Datenbanken
- Umstellung der Dateien
- Umstellung der COBOL-Programme
- Compile & Link & Test

Organisatorische Aufgaben:

- (Programm)-Clusterbildung
- Reihenfolge der Umstellung
- Verwaltung der Quellen
- Fachliche Änderungen während Umstellung
- Parallelbetrieb Alt/Neu




Transformation: Wertpapierkennummern / Programme

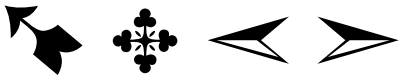
Umstellung der COBOL-Programme:

- Festlegung der Umstellungsregeln
  - Festlegung des Initialwertes (`SPACE` oder `ZERO`)
  - Umstellung der `PICTURE` Klausel
  - Umstellung der Anweisungen
- Welche Variablen enthalten WKN?
  - Finden der „initialen Treffer“

    ⇒ Suchliste der Art `*WKN*` oder `W6N21`
  - Propagation der Eigenschaft „enthält WKN“

    ⇒ `MOVE` *`variable`* `TO` *`WKN-variable`*

    ⇒ `MOVE` *`WKN-variable`* `TO` *`variable`*



WKN-Umstellungsregeln

level name PIC 9(*count*)
⇒
level name PIC X(*count*)

```
01 src  PIC 9(d)   ⇒ PIC X(d)
01 dst  PIC 9(s)   ⇒ PIC X(s)
MOVE src TO dst
⇒ s = d: keine Änderung
   s < d: MOVE ZERO TO dst
          MOVE src  TO dst(d-s+1:s)
   s > d: MOVE src(s-d+1:d) TO dst
```

---

level name PIC ±9(*count*)
⇒
level name
   level+1 *name*-SIGN PIC X(1) VALUE SPACE.
   level+1 *name*-VALUE PIC X(*count*).

```
01 src  PIC  9(d)  ⇒ ...
01 dst  PIC +9(s)  ⇒ ...
MOVE src TO dst
⇒ s = d: MOVE src   TO dst-VALUE
          MOVE SPACE TO dst-SIGN
   s < d: MOVE ZERO  TO dst-VALUE
          MOVE src   TO dst-VALUE(d-s+1:s)
          MOVE SPACE TO dst-SIGN
   s > d: MOVE src(s-d+1:d) TO dst-VALUE
          MOVE SPACE TO dst-SIGN
```

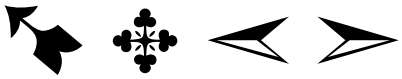
WKN-Umstellung – Technische Probleme

- Umgang mit REDEFINES (statische Aliase)

```
01 WKN PIC 9(6).
01 WKN-X    REDEFINES WKN X(6).
01 WKN-PART REDEFINES WKN.
   02 WKN-1-3 PIC 9(3).
   02 WKN-2-6 PIC 9(3).

01 memory PIC X(100).
01 DATA1 REDEFIENS memory.
   10 WKN PIC 9(6). ...
01 DATA2 REDEFIENS memory.
   10 XYZ PIC 9(6). ...
```

- Mehrdeutig benutzte Variablen
- Umgang mit falsch positiv erkannten WKN-Variablen
  - ⇒ Ausschlußliste der Art: `*-WPK-SA`
  - ⇒ IGNORE-ALIAS: `XYZ OF DATA2`
  - ⇒ Heuristiken (z.B. WKN hat zwischen 6 und 20 Bytes),
    aber: 175 verschiedene PICTURE Klauseln für WKN gefunden
- „Richtiges“ Einfügen der Anweisungen (terminierender Punkt).
- „Schönes“ Layout der erzeugten Texte.



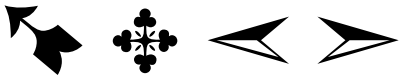

Aufwand: Wertpapierkennummern

Anzahl Programme:	15.000 (gesamt)	1.000 (WKN-relevant)
Anzahl COPY-Strecken:	30.000 (gesamt)	1.500 (WKN-relevant)
Anzahl Fundstellen:	800.000 (ohne Heuristiken)	300.000 (mit H.)
Lines Of Code:	2.300.000 (Programme), 300.000 (COPY's)	
Ergebnisse:	- ca. 4.050 Umstellungen	
	- keine Transformationsregel für ca. 5.900 Stellen	
	- ca. 24.700 Warnungen	
Geschätzer Aufwand:	vollständig manuelle Umstellung:	
	1,5 Tage / Programm	
	0,1 Tag / COPY-Strecke	
	= ca. 1.700 Tage	
Realer Aufwand:	halbautomatische Umstellung:	
	8 Personen, 5,5 Monate á 20 Tage	
	+ Werkzeug 200 Tage	
	= 1080 Tage	
Ersparnis:	$\Rightarrow$ca. 600 Tage, d.h. ca 30% schneller/billiger	

Lehren aus den bisherigen Projekten

- Nur ungenaue Spezifikation des Problems durch den Kunden.
- Spezifikationen ändern sich.
- Kunde: „Das Feature XYZ wird bei uns nicht benutzt.“

  Erfahrung & Analyse: doch!
- Kundenspezifische Anpassung der Tools ist enorm wichtig.
- Verständigungsproblem: Compilerbauer $\leftrightarrow$ COBOL-Programmierer



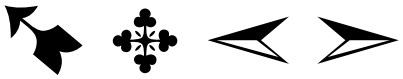

These: Vollautomatische Umstellung ist nicht möglich

Halbautomatische Umstellung bedeutet:

- Werkzeug liefert Analysedaten für manuelle Umstellung.
- Werkzeug transformiert in eindeutigen „Situationen“.
- Werkzeug markiert problematische Stellen zur manuellen Nachbearbeitung.

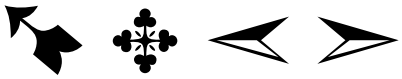
Korrolar:

Vollautomatische Umstellung ist für triviale Probleme möglich.




Zusammenfassung

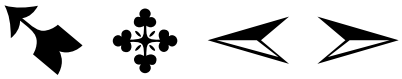
- Erfolg mit Techniken aus dem (optimierenden) Compilerbau.
- Halbautomatische Umstellung ist möglich.
- Vollautomatische Umstellung nur für triviale Probleme.
- Erstellung von Transformation-Werkzeug lohnt sich.



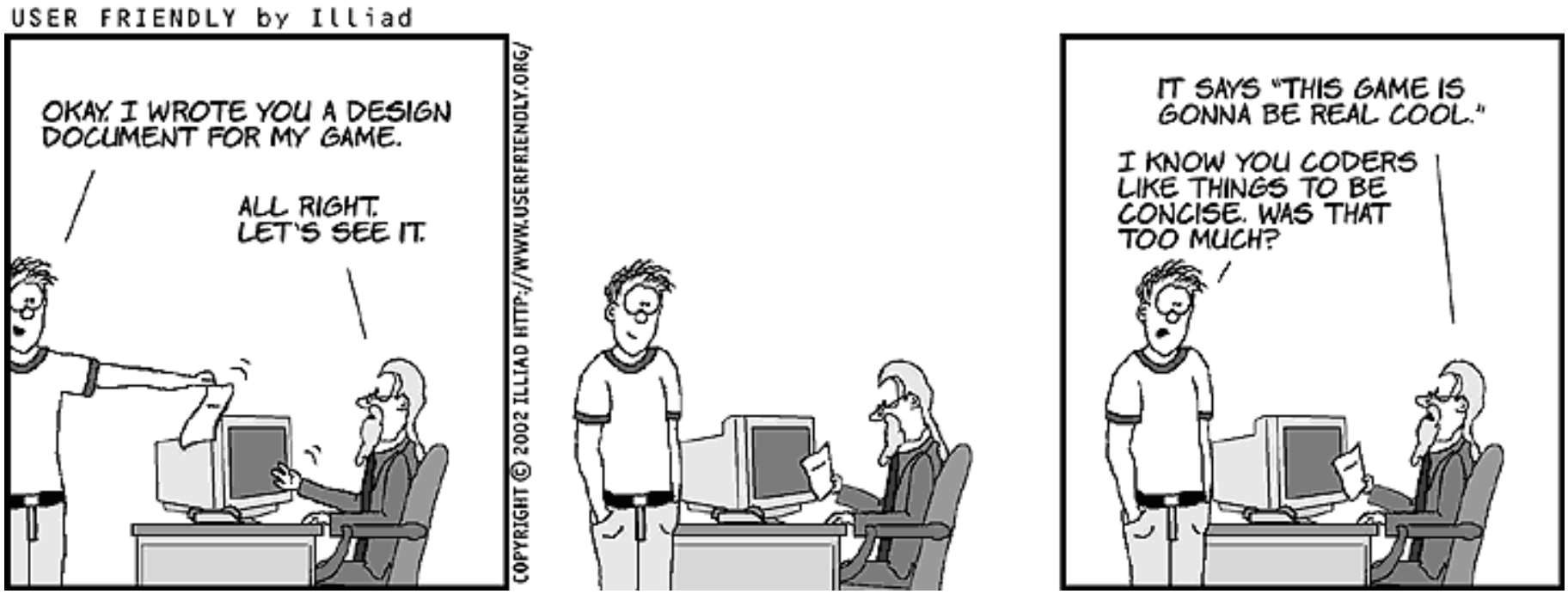

Ausblick

Aufgaben:

- (bessere) Erklärungskomponente für Analyseergebnisse
- (bessere) Visualisierung der Analyseergebnisse
- Programmübergreifende Analyse
- Entwicklung von Heuristiken, um komplexere Trafo’s zu erlauben




Zu guter Letzt: Die Rolle von Design Dokumenten ☺



Dank an alle beteiligten Mitarbeiter der Projektpartner!