Visualisierung feingranularer Abhängigkeiten

Jens Krinke

FernUniversität in Hagen
Fachbereich Elektrotechnik und Informationstechnik
Fach Softwaretechnik

6. Workshop Software-Reengineering, Bad Honnef, Mai 2004

Visualisierung

Graphische Visualisierung von Abhängigkeiten zwischen größeren Einheiten ist ein wichtiges (und erfolgreiches) Hilfsmittel.

- UML-Diagramme
- Abhängigkeiten zwischen Klassen, Objekten, Komponenten etc.
- Visualisierung von Eigenschaften (Metriken)

Feingranulare Abhängigkeiten

Feingranulare Abhängigkeiten sind Abhängigkeiten zwischen den Anweisungen und Ausdrücken eines Programms.

Datenabhängigkeiten existieren zwischen Anweisungen, die die gleiche Variable benutzen oder definieren,

Kontrollabhängigkeiten existieren zwischen Anweisungen, die über die Ausführung anderer Anweisungen entscheiden.

Ähnliche Definitionen gibt es für den interprozeduralen Fall.

Layout von Abhängigkeitsgraphen

Darstellung von Abhängigkeiten durch Abhängigkeitsgraphen mit Hilfe allgemeiner Graphlayouter, wie z.B.

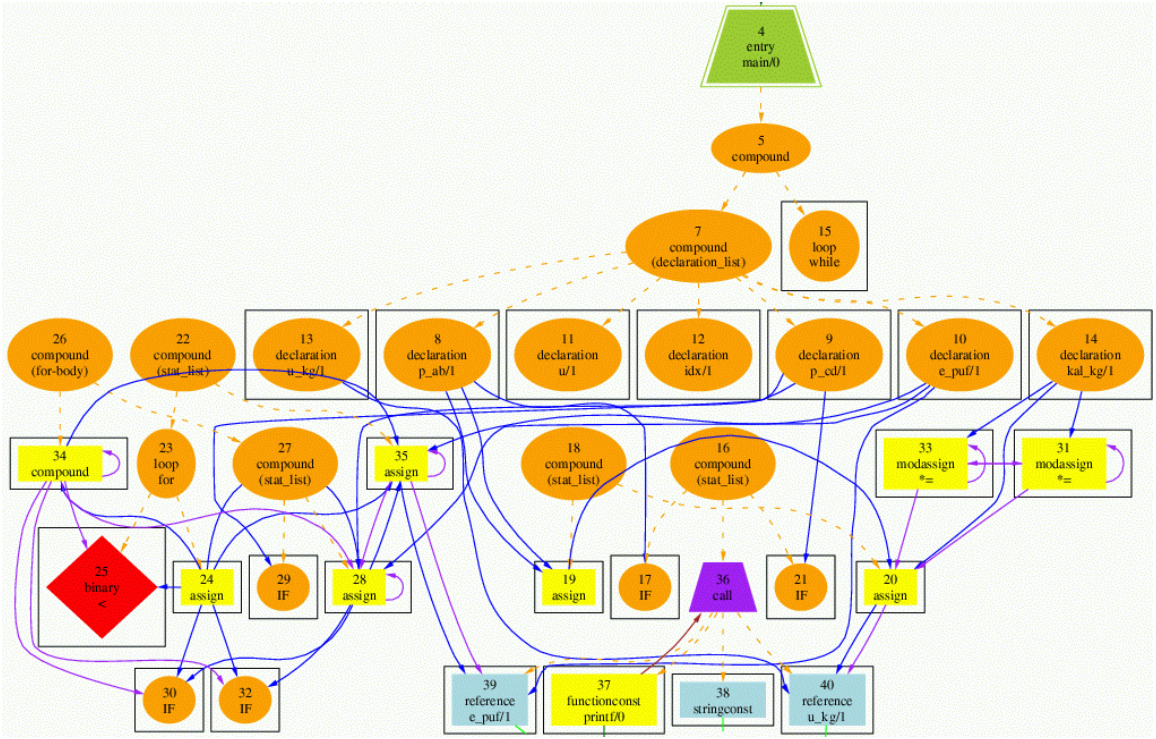
- daVinci
- VCG
- dot

Erfahrungen:

✘ Verwendung von Standard-Layoutern führt zu keinen verständlichen Graphen.

✘ Entwickler finden sich im Graph nicht zurecht.

Beispiel (dot)



Eigenschaften feingranularer Abhängigkeiten

Entwickler erwarten spezielle Eigenschaften:

- ✘ Entwickler brauchen die *syntaktische* Struktur um sich im Programm zurechtzufinden.
- ➜ Abstraktionen wie Clustering oder Falten können nicht mehr auf die Programmstruktur (Quelltext) abgebildet werden.

Ohne Bezug zum Quelltext ist ein Verständnis auf Anweisungs-Ebene nicht möglich.

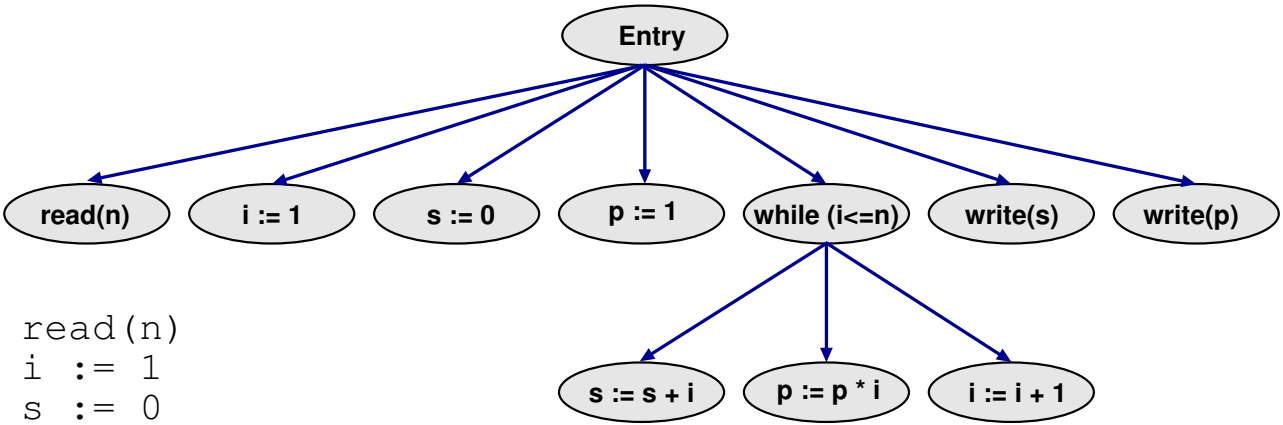
Deklaratives Layout

- Kontrollabhängigkeiten bilden das Gerüst ähnlich einem abstrakten Syntaxbaum.
- Die Reihenfolge der Anweisungen im Quelltext entspricht der horizontalen Reihenfolge der Knoten.
- Datenabhängigkeiten werden nachträglich geroutet und ändern das Layout nur wenig.

Umsetzung:

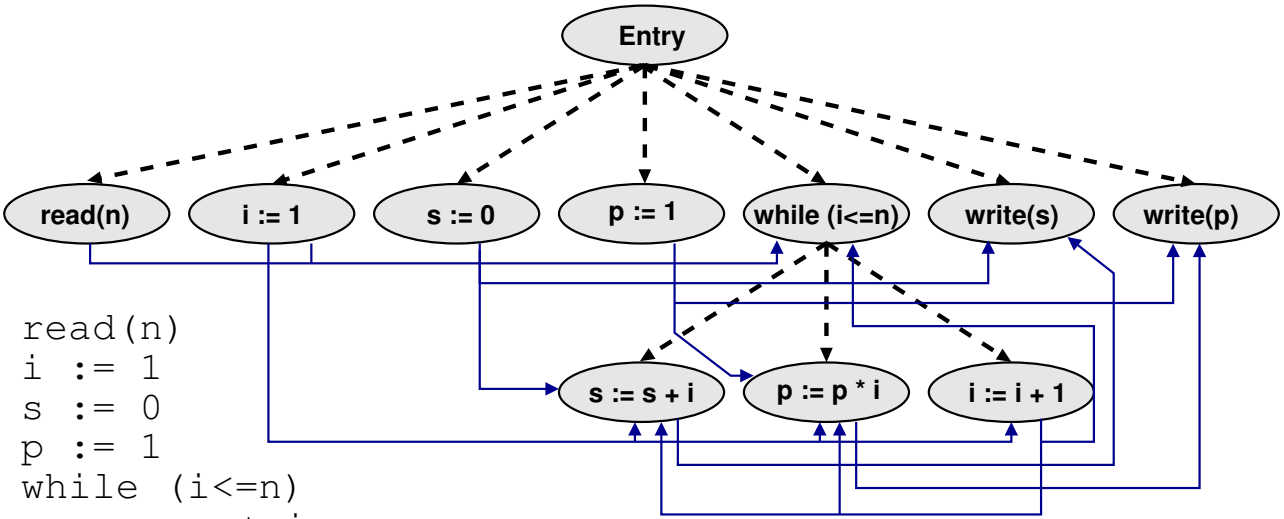
1. Layout der Kontrollabhängigkeiten durch Sugiyama
2. Layout der Datenabhängigkeiten als Manhattan-Layout

Beispiel-Layout



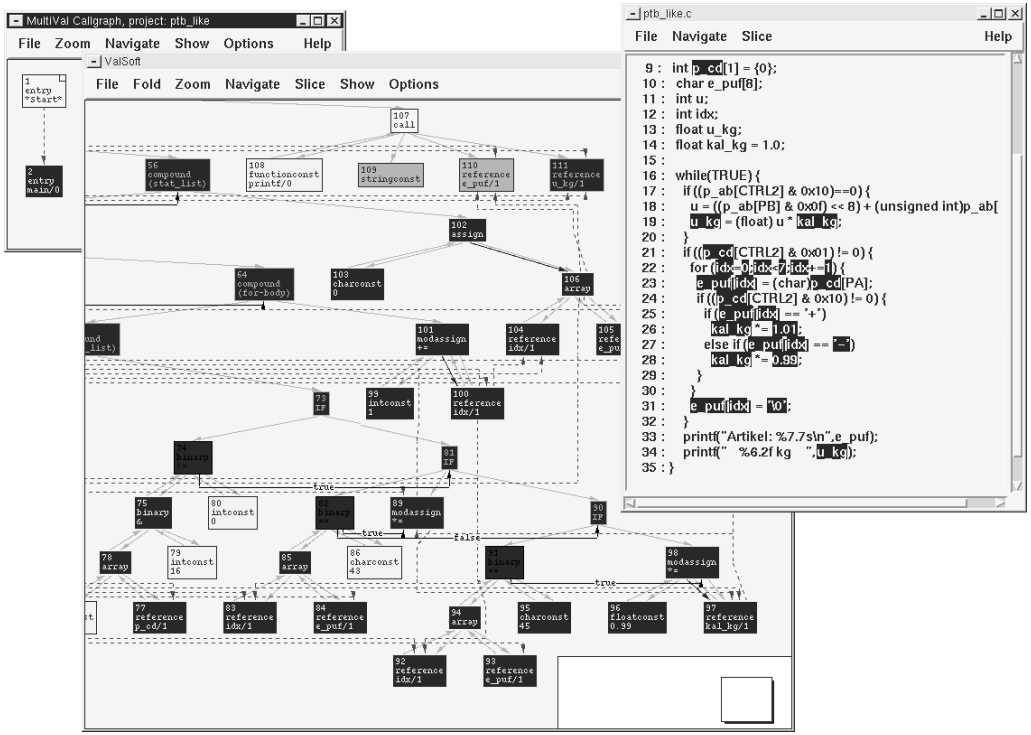

```
read(n)
i := 1
s := 0
p := 1
while (i<=n)
  s := s + i
  p := p * i
  i := i + 1
write(s)
write(p)
```

Beispiel-Layout




```
read(n)
i := 1
s := 0
p := 1
while (i<=n)
   s := s + i
   p := p * i
   i := i + 1
write(s)
write(p)
```

GUI


MultiVal Callgraph, project: ptb_like
File Zoom Navigate Show Options Help
ValSoft
File Fold Zoom Navigate Slice Show Options
ptb_like.c
File Navigate Slice Help
9 : int p_cd[1] = {0};
10 : char e_puf[8];
11 : int u;
12 : int idx;
13 : float u_kg;
14 : float kal_kg = 1.0;
15 :
16 : while(TRUE) {
17 : if ((p_ab[CTRL2] & 0x10)==0) {
18 : u = ((p_ab[PB] & 0x0f) << 8) + (unsigned int)p_ab[
19 : u_kg = (float) u * kal_kg;
20 : }
21 : if ((p_cd[CTRL2] & 0x01) != 0) {
22 : for (idx=0;idx<7;idx++) {
23 : e_puf[idx] = (char)p_cd[PA];
24 : if ((p_cd[CTRL2] & 0x10) != 0) {
25 : if (e_puf[idx] == '+')
26 : kal_kg *= 1.01;
27 : else if (e_puf[idx] == '-')
28 : kal_kg *= 0.99;
29 : }
30 : }
31 : e_puf[idx] = '\0';
32 : }
33 : printf("Artikel: %7.7s\n",e_puf);
34 : printf(" %6.2f kg ",u_kg);
35 : }

Slices

Slices enthalten alle Anweisungen eines Programms,
die eine bestimmte Anweisung beeinflussen können.

Berechnung durch Erreichbarkeit im Abhängigkeitsgraphen.

Probleme:

- ✘ Slices sind (immer noch) zu groß.
- ✘ Slices enthalten keine quantitativen Aussagen.
- ➜ Slicing wird nicht zum Programmverstehen benutzt.

Grund:

Slices fehlt es an *Lokalität* und *Abstraktion*.

Lokalität

Lokalität in Slices kann erreicht werden,
indem die *Distanz* der Abhängigkeiten zur aktuellen Anweisung gemessen und berücksichtigt wird.

Die Distanz eines Knoten zur aktuellen Anweisung
ist die Anzahl der Kanten auf dem kürzesten Pfad.

- Slices werden nur bis zu einer Distanz berechnet
- Slices werden mit der Distanz der Knoten visualisiert

➜ Beide Varianten können kombiniert werden.

➜ Beides kann sowohl im Graphen,
als auch im Quelltext visualisiert werden.

Textuelle Darstellung

```
01: const unsigned TRUE = 1;
02: const unsigned CTRL2 = 0;
03: const unsigned PB = 0;
04: const unsigned PA = 1;
05: void printf();
06: void main()
07: {
08:   int p_ab[2] = {0, 1};
09:   int p_cd[1] = {0};
10:   char e_puf[8];
11:   int u;
12:   int idx;
13:   float u_kg;
14:   float kal_kg = 1.0;
15:
16:   while(TRUE) {
17:     if ((p_ab[CTRL2] & 0x10)==0) {
18:       u = ((p_ab[PB] & 0x0f) << 8) + (unsigned int)p_ab[PA];
19:       u_kg = (float) u * kal_kg;
20:     }
21:     if ((p_cd[CTRL2] & 0x01) != 0) {
22:       for (idx=0;idx<7;idx++) {
23:         e_puf[idx] = (char)p_cd[PA];
24:         if ((p_cd[CTRL2] & 0x10) != 0) {
25:           if (e_puf[idx] == '+')
26:             kal_kg *= 1.01;
27:           else if (e_puf[idx] == '-')
28:             kal_kg *= 0.99;
29:         }
30:       }
31:       e_puf[idx] = '\0';
32:     }
33:     printf("Artikel: %7.7s\n   %6.2f kg    ",e_puf,u_kg);
34:   }
35: }
36:
```

Abstrakte Darstellung

Am Anfang des Programmverstehens steht nicht die Frage nach *gibt es eine Abhängigkeit zwischen X und Y?*, sondern eher

- *Was sind die wichtigen Variablen und Prozeduren?*
- *Welchen Einfluss hat eine Variable oder Prozedur?*
- *Wo sind starke Kopplungen zwischen Variablen bzw. Prozeduren?*
- *Was sind die ‘Hot Spots’?*

Chopping

Ein *Chop* enthält die Anweisungen eines Programms,
die an einer Beeinflussung einer Anweisung
durch eine andere Anweisung beteiligt sind.

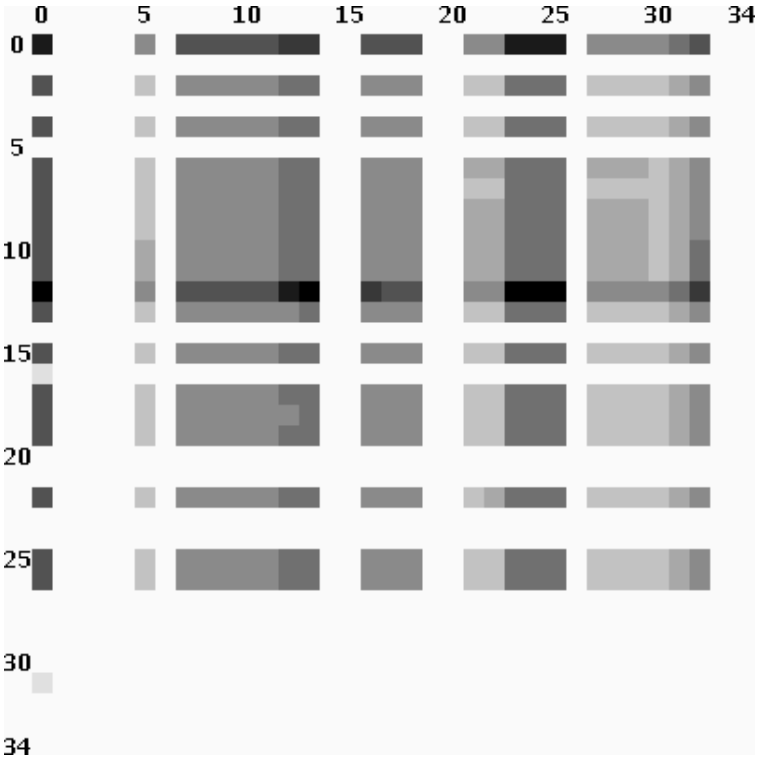
✔ Chops können wie Slices berechnet und benutzt werden.

✘ Chops haben die gleichen Probleme wie Slices.

Zur Abstraktion werden Chops zur Berechnung
einer Kopplungsmatrix herangezogen:

- Statt Anweisungen: Variablen oder Prozeduren als Kriterien
- Die Anzahl der beteiligten Anweisungen wird gemessen
- Die Anzahl wird als Maßzahl in einer Matrix visualisiert

Chop-Visualisierung



`ansitape`

Variablen:

2: `stdin`
3: `stdout`
4: `stderr`

12: `tcb`
(tape control block)

Zusammenfassung

- Visualisierung von feingranularen Abhängigkeiten
  mit Hilfe von Standard-Layoutern
  führt zu keinen verständlichen Graphen.
- Visualisierung mit speziellem Layout hilft deutlich weiter.
- Slices fehlt es an Lokalität und Abstraktion.
- Lokalität:
  messen, benutzen und visualisieren der Distanz.
- Abstraktion:
  messen und visualisieren des Einflussbereichs
  zwischen Variablen bzw. Prozeduren.