Software-Reengineering

Prof. Dr. Rainer Koschke

Montag, 29.01.2007

```
1  // original: http://www.d.umn.edu/~gshute/C/examples/quicksort.C
2  #include <stdio.h>
3
4  int A[] = { 99, 43, 22, 17, 57, 32, 43, 19, 26, 48, 87, 12, 75,
5  const int numEntries = sizeof(A)/sizeof(int);
6
7  void lqksort(int ilo, int ihi) {
8      int pivot;           // pivot value for partitioning array
9      int ulo, uhi;        // indices at ends of unpartitioned regi
10     int ieq;             // least index of array entry with value
11     int tempEntry;       // temporary entry used for swapping
12
13     if (ilo >= ihi) {
14         return;
15     }
16     // Select a pivot value.
17     pivot = A[(ilo + ihi)/2];
18     // Initialize ends of unpartitioned region and least index o
19     // with value equal to pivot.
20     ieq = ulo = ilo;
21     uhi = ihi;
```

```
22      // While the unpartitioned region is not empty, try to reduc
23      while (ulo <= uhi) {
24          if (A[uhi] > pivot) {
25              // Here, we can reduce the size of the unpartitioned
26              // try again.
27              uhi--;
28          } else {
29              // Here, A[uhi] <= pivot, so swap entries at indices
30              // uhi.
31              tempEntry = A[ulo];
32              A[ulo] = A[uhi];
33              A[uhi] = tempEntry;
34              // After the swap, A[ulo] <= pivot.
35              if (A[ulo] < pivot) {
36                  // Swap entries at indices ieq and ulo.
37                  tempEntry = A[ieq];
38                  A[ieq] = A[ulo];
39                  A[ulo] = tempEntry;
40                  // After the swap, A[ieq] < pivot, so we need to
41                  // ieq.
42                  ieq++;
```

```
43                      // We also need to change ulo, but we also need
44                      // that when A[ulo] = pivot, so we do it after
45                      // statement.
46                  }
47                  // Once again, we can reduce the size of the unpart
48                  // region and try again.
49                  ulo++;
50              }
51          }
52          // Now, all entries from index ilo to ieq - 1 are less than
53          // and all entries from index uhi to ihi + 1 are greater tha
54          // pivot.  So we have two regions of the array that can be
55          // recursively to put all of the entries in order.
56          lqksort(ilo, ieq - 1);
57          lqksort(uhi + 1, ihi);
58      }
59
60      void qksort(void) {
61          lqksort(0, numEntries - 1);
62      }
```

```
1   void
2   bubbleSort (int list[], int len)
3   {
4     int sorted = FALSE;
5     while (!sorted)
6       {
7         int j;
8         sorted = TRUE;
9         for (j = 0; j < len - 1; j++)
10          {
11            if (list[j] > list[j + 1])
12              {
13                int t;
14                sorted = FALSE;
15                t = list[j];
16                list[j] = list[j + 1];
17                list[j + 1] = t;
18              }
19          }
20      }
21  }
```

LOC

lqksort qksort lqksort + qksort bubbleSort

LOC

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21

LOC

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21
Non-blank LOC	30	3	33	21

LOC

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21
Non-blank LOC	30	3	33	21
CLOC	25	0	25	0

LOC

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21
Non-blank LOC	30	3	33	21
CLOC	25	0	25	0
ELOC	19	1	20	9

Halstead

Länge	$N = N_1 + N_2$
Vokabular	$\mu = \mu_1 + \mu_2$
Volumen	$V = N \cdot \log_2 \mu$
Program Level	$L_{est} = (2/\mu_1) \cdot (\mu_2/N_2)$
Programmieraufwand	$E_{est} = V/L_{est}$

mit μ_1, μ_2 = Anzahl unterschiedlicher Operatoren, Operanden
N_1, N_2 = Gesamtzahl verwendeter Operatoren, Operanden

```
24
25 void qksort(void) {
26     lqksort(0, numEntries - 1);
27 }
```

$N = 5 + 11 = 16$

$M = 5 + 8 = 13$

$V = 16 \cdot \mathrm{ld}\ 13$

Halstead für qksort

Operanden qksort		
1	lqksort	1
2	0	1
3	numEntries	1
4	1	1
	Summe	4

Anzahl Operatoren = Anzahl Tokens - Anzahl Operanden = 16 - 4 = 12

$$N = 12 + 4 = 16$$

$$\mu = 8 + 4 = 12$$

$$V = 16 \cdot log_2 12 \approx 57$$

```
1  void lqksort(int ilo, int ihi) {
2      int pivot; int ulo, uhi; int ieq; int tempEntry;
3      if (ilo >= ihi) return;
4      pivot = A[(ilo + ihi)/2];
5      ieq = ulo = ilo; uhi = ihi;
6      while (ulo <= uhi) {
7          if (A[uhi] > pivot) {uhi--;}
8          else {
9              tempEntry = A[ulo];
10             A[ulo] = A[uhi];
11             A[uhi] = tempEntry;
12             if (A[ulo] < pivot) {
13                 tempEntry = A[ieq];
14                 A[ieq] = A[ulo];
15                 A[ulo] = tempEntry;
16                 ieq++;
17             }
18             ulo++;
19         }
20     }
21     lqksort(ilo, ieq - 1);
22     lqksort(uhi + 1, ihi);
23 }
```

$N =$

$M = 35$

Halstead für `lqksort`

$$\mu = 32 / 35$$

$$N = 163$$

$$V = 163 \cdot log_2 32 = 163 \cdot 5 \approx 815$$

```
1   void
2   bubbleSort (int list[], int len)
3   {
4     int sorted = FALSE;
5     while (!sorted)
6       {
7         int j;
8         sorted = TRUE;
9         for (j = 0; j < len - 1; j++)
10          {
11            if (list[j] > list[j + 1])
12              {
13                int t;
14                sorted = FALSE;
15                t = list[j];
16                list[j] = list[j + 1];
17                list[j + 1] = t;
18              }
19          }
20      }
21  }
```

Halstead für BubbleSort

$$\mu = 26$$

$$N = 100$$

$$V = 100 \cdot log_2 26 \approx 470$$

Metrikenübersicht

$V = N \cdot \mathrm{ld}\, \mu$

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21
Non-blank LOC	30	3	33	21
CLOC	25	0	25	0
ELOC	19	1	20	9
Halstead	815	≈ 57	$\neq 815 + 57$	≈ 470

Metrikenübersicht

	lqksort	qksort	lqksort + qksort	bubbleSort
LOC (wc)	52	3	55	21
Non-blank LOC	30	3	33	21
CLOC	25	0	25	0
ELOC	19	1	20	9
Halstead	815	≈ 57	$\neq 815 + 57$	≈ 470
McCabe	5	1	6	4

Empirische Studien

Maintainability Index (Coleman/Oman, 1994):

$$MI_1 = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot V(g') - 16.2 \cdot \ln(LOC)$$
$$MI_2 = MI_1 + 50 \cdot \sin\sqrt{2.46 \cdot perCM}$$

- V = average Halstead Volume per module
- $V(g')$ = average extended cyclomatic complexity per module
- LOC = average LOC per module
- $perCM$ = average percent of lines of comment per module
- MI_2 nur bei sinnvoller Kommentierung
- $MI < 65 \Rightarrow$ schlechte / $MI \geq 85 \Rightarrow$ gute Wartbarkeit

Empirische Studien

Maintainability Index (Coleman/Oman, 1994):

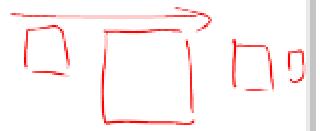
$$MI_1 = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot V(g') - 16.2 \cdot \ln(LOC)$$
$$MI_2 = MI_1 + 50 \cdot \sin\sqrt{2.46 \cdot perCM}$$

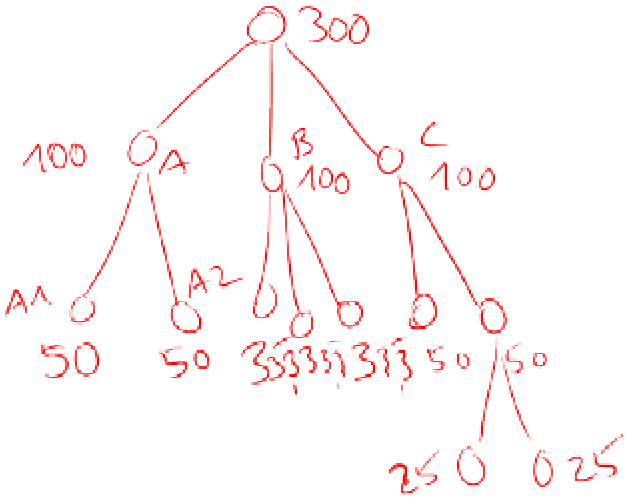
- V = average Halstead Volume per module
- $V(g')$ = average extended cyclomatic complexity per module
- LOC = average LOC per module
- $perCM$ = average percent of lines of comment per module
- MI_2 nur bei sinnvoller Kommentierung
- $MI < 65 \Rightarrow$ schlechte / $MI \geq 85 \Rightarrow$ gute Wartbarkeit

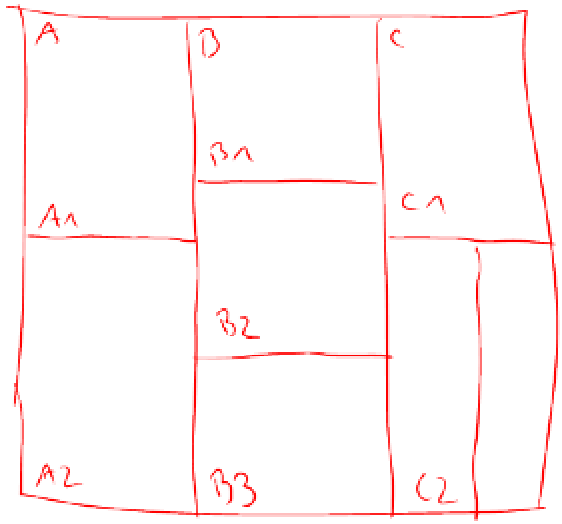
$$MI_1(lqksort) = 171 - 5.2 \cdot \ln(815) - 0.23 \cdot 5 - 16.2 \cdot \ln(30)$$
$$MI_2(lqksort) = MI_1(lqksort) + 50 \cdot \sin\sqrt{2.46 \cdot 25/50} \approx 81$$
$$MI_1(bubbleSort) = 171 - 5.2 \cdot \ln(470) - 0.23 \cdot 4 - 16.2 \cdot \ln(21)$$
$$MI_2(bubbleSort) = MI_1(bubbleSort) + 50 \cdot \sin\sqrt{2.46 \cdot 0/21} \approx 89$$

Aufgabe 3.5 *Software-Visualisierung*

Entwerfen Sie eine geeignete Software-Visualisierung für die Klonerkennung. Die Visualisierung soll helfen, einen schnellen Überblick über Source-Code-Elemente mit einem großen Anteil von Klonen zu gewinnen.
Die Software-Visualisierung soll es außerdem ermöglichen, ein Verständnis der Fortpflanzung der Klone zu erhalten (sowohl zeitlich – d.h. von einer Version zur anderen – als auch räumlich – d.h. von einer Quellcodeposition zur anderen unter besonderer Berücksichtigung von Duplizierung innerhalb von Dateien und über Dateien hinweg).
Beachten Sie auch die Unterscheidung der Klontypen (Typ 1, 2 und 3).



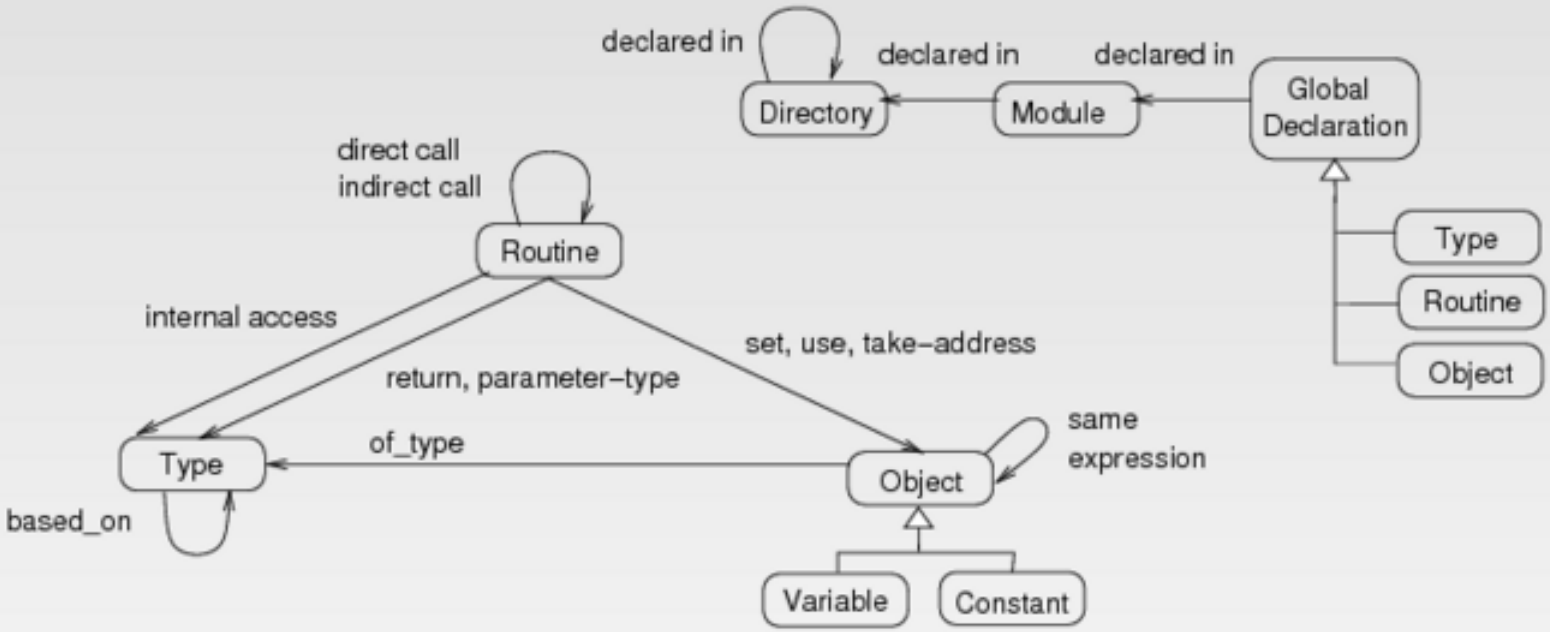




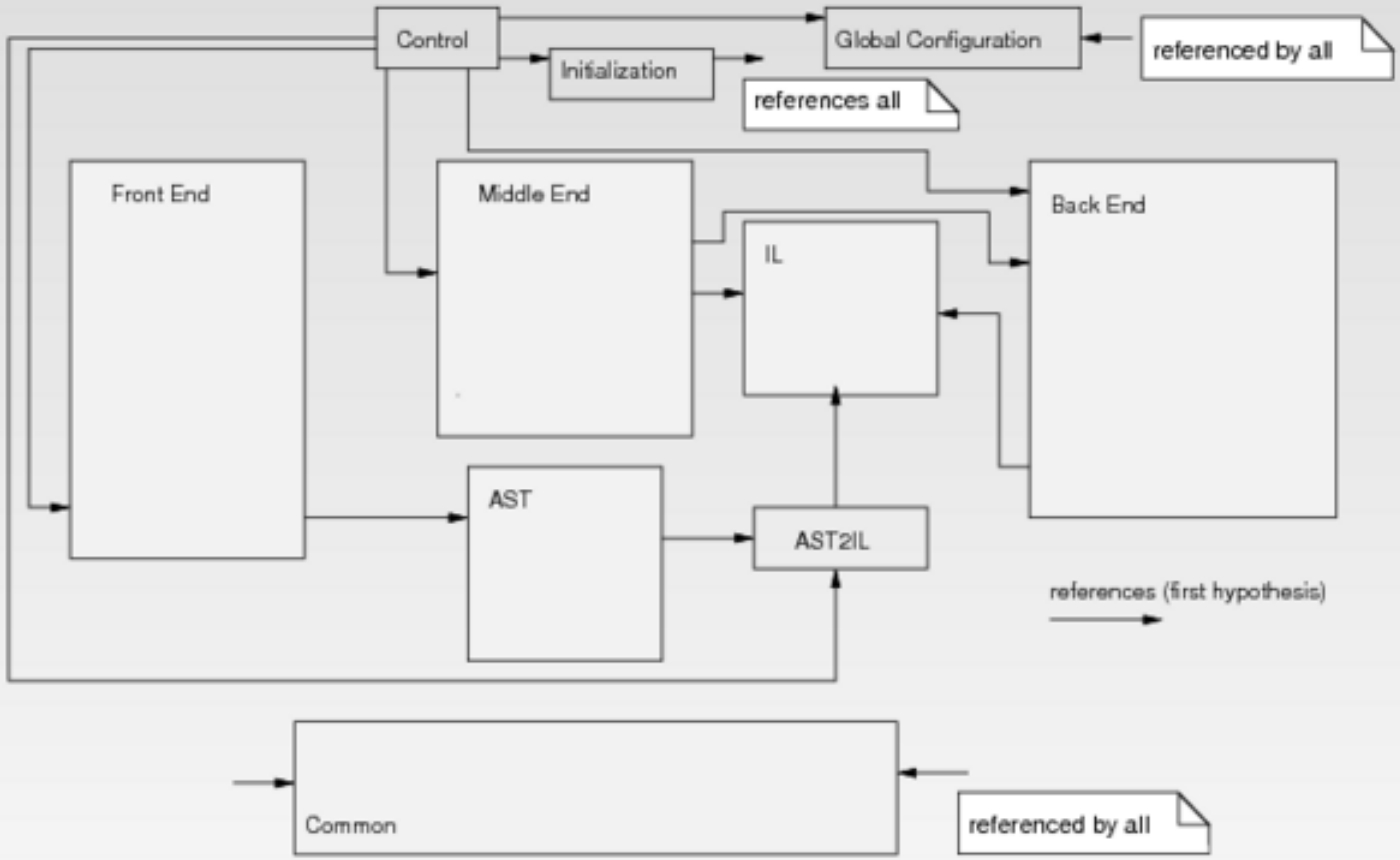
Fallstudie für C-Compiler (Koschke und Simon 2003)

System	KLOC	C-Units	Multi-Plattform	Aufwand
sdcc	100	49	ja	6 h
cc1	500	156	ja	8+ h

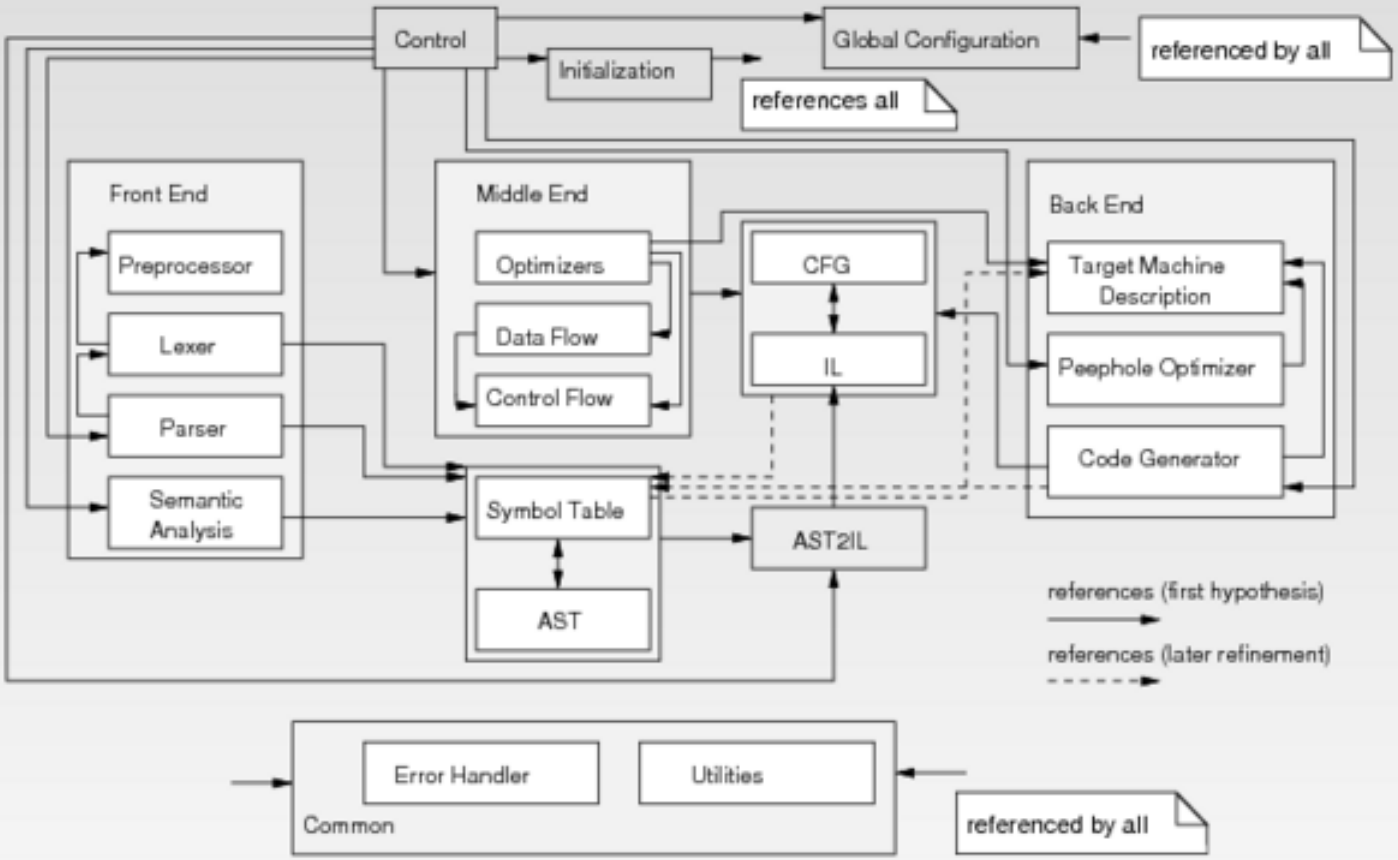
Bauhaus-Fakten-Extraktion



Compiler-Architektur (grobe Modulsicht)



Compiler-Architektur (detaillierte Modulsicht)



Resultate für *sdcc*

- Abbildung von Dateien relativ klar
- viele Divergenzen in der ersten Iteration
- Verfeinerung (5 zusätzliche Iterationen):
  - 45 globale Deklarationen nicht auf die konzeptionelle Komponente abgebildet, auf die Datei abgebildet wurde, die sie deklariert
  - die meisten davon auf *Global Declarations*
  - einige übersehene Abhängigkeiten in der hypothetischen Sicht

Resultate für *sdcc*

Architekturverletzungen:

- Symboltabelle referenziert Parser
  - Blocknummer von Deklarationen und Zeilennummer
- Back-End referenziert Parser
  - globale Variablen, die Kellergröße für Aktivierungsblöcke verwalten

Resultate für *sdcc*

Architekturmuster:

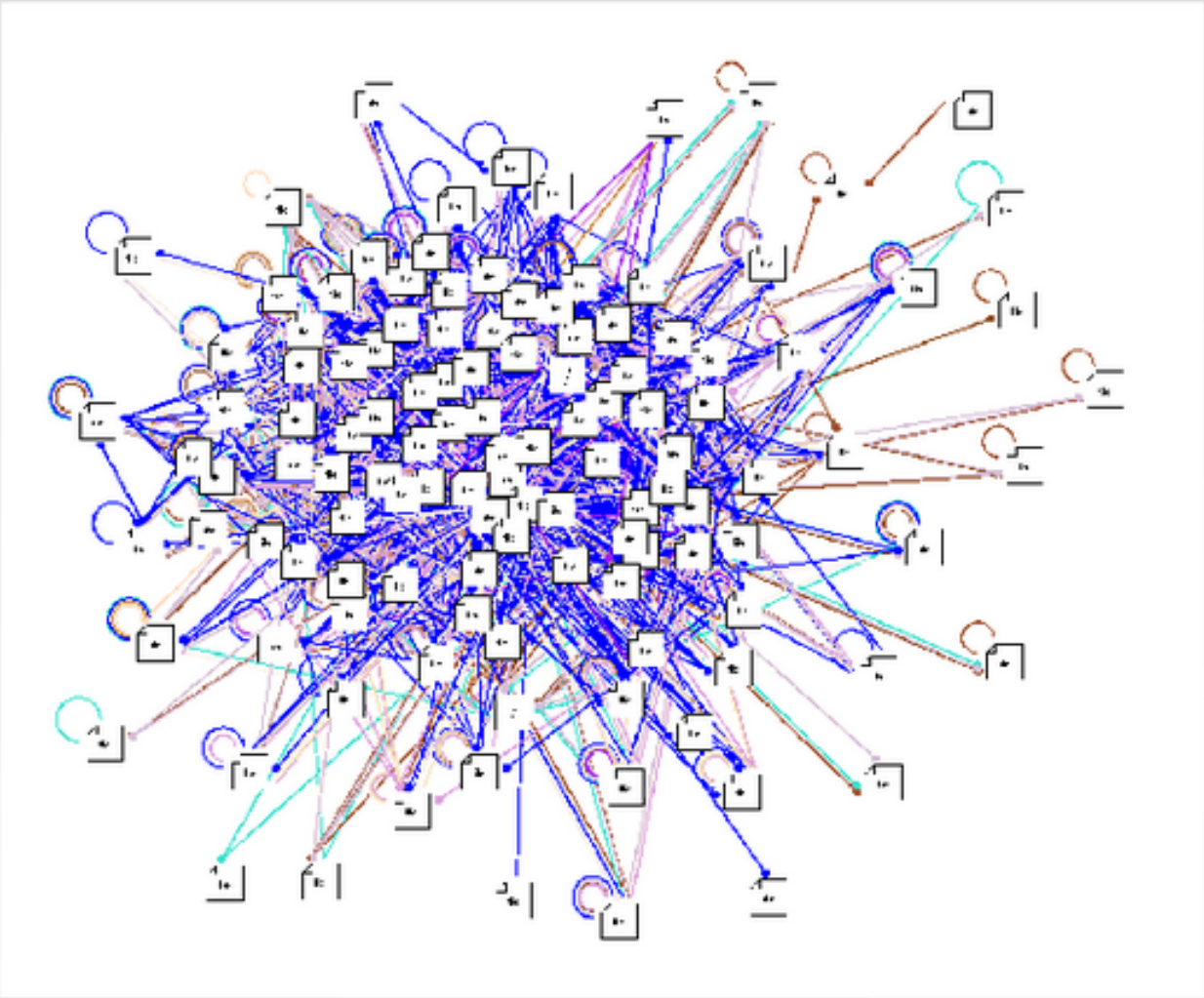
- Optimierung referenziert Back-End
  - um plattformabhängige Parameter zu erhalten
  - mittels Funktionszeigern

  → "anonyme" Abhängigkeit

Resultate für *cc1*

- nur Units abgebildet (keine Verfeinerung für globale Deklarationen)
- Front-End ist gut strukturiert and lose gekoppelt an Middle-/Back-End
- Back-End ist einfach identifizierbar
- Middle-End ist ein "big ball of mud"

Modulabhängigkeiten von *cc1*



Resultate für *cc1*

Architekturverletzungen:

- Middle-End referenziert Preprozessor
  - benutzt Hash-Tabelle des Preprozessors
  - (es gibt mindestens drei verschiedene Hash-Tabellen in *cc1*)
- viele, viele Divergenzen...

Gesammelte Erfahrungen

- Extraktion:
  - Funktionszeigeranalyse sehr wichtig
- Verfeinerungen der Modelle:
  - alle Modelle (hypothetische und konkrete Sicht, Abbildung)
- Einfluss der Strukturiertheit des analysierten Systems:
  - Kohäsion der Units erleichtert Abbildung enorm

Zusammenfassung

- hierarchisches hypothetisches Modell ist notwendig für große Systeme
- hypothetisches Modell und Abbildung müssen manuell erstellt werden
- Voraussetzungen:
  - hypothesische Sicht erfordert Wissen über Anwendungsbereich und potentielle Architektur
- Weitere notwendige Erweiterungen:
  - Abbildung von Relationen
  - Kontroll- und Datenflussabhängigkeiten statt nur Referenzen