

# Vorlesung Software-Reengineering

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik  
Fachbereich Mathematik und Informatik  
Universität Bremen

Wintersemester 2005/06

## Überblick I

① Dynamische Analyse

- ① Dynamische Analyse
  - Dynamische Analyse
  - Information durch dynamische Analyse
  - Testfälle
  - Instrumentierung
  - Probleme der Instrumentierung
  - Weiterverarbeitung dynamischer Information
  - Anwendungsbeispiele
  - Vergleich mit statischer Analyse
  - Wiederholungsfragen

## Dynamische Analyse

Bisher: statische Analyse

- Aliasing
  - Polymorphismus
    - dynamisches Binden
    - Überladen
  - Reflection
  - verteilte Systeme
- viele Entscheidungen erst zur Laufzeit
- statische Analyse problematisch

statisch = ruhend, unbeweglich – bezieht sich hier auf den Quellcode. Dieser ändert sich nicht während der Analyse.  
dynamisch = ändert sich – hier ändert sich der Zustand des Programms (d.h. des Arbeitsspeichers etc.) mit jeder ausgeführten Anweisung.

Aliasing: mehrere Zeiger auf das gleiche Objekt. Problem: Erst zur Laufzeit ist wirklich bekannt, auf welches Objekt der Zeiger konkret zeigt.

Polymorphismus: auch hier entscheidet sich erst zur Laufzeit durch den Typ des Objekts, welche Methode konkret aufgerufen wird.

Reflection is the ability of a running program to examine itself and its software environment, and to change what it does depending on what it finds.

Verteiltes System: Menge interagierender Prozesse ohne gemeinsamen Speicher, d.h. mit Kommunikation untereinander.

## Dynamische Analyse

### Definition

**Dynamische Analyse:** Analyse der Eigenschaften eines laufenden Programms.

### Ablauf:

- Ausführen des Programms
- Beobachtung der Ausführung
- Analyse der Ergebnisse

### Notwendig:

- Wahl der Testfälle
- Instrumentierung
- Analysemethoden/-werkzeuge

## Definition

**Dynamische Analyse:** Analyse der Eigenschaften eines laufenden Programms.

## Ablauf:

- Ausführen des Programms
- Beobachtung der Ausführung
- Analyse der Ergebnisse

## Notwendig:

- Wahl der Testfälle
- Instrumentierung
- Analysemethoden/-werkzeuge

Eine dynamische Analyse hat jeder schon verwendet: Testen. Auch dabei wird das Programm ausgeführt, das Verhalten des Programms während der Ausführung beobachtet und aus den Beobachtungen Rückschlüsse gezogen (ist die Spezifikation erfüllt oder nicht?).

Auch beim Testen ist die Wahl der Testfälle entscheidend: Möglichst alle Teile des Programms sollen in allen Varianten mindestens einmal ausgeführt worden sein. Eine hohe Testabdeckung ist also wünschenswert. Genauso ist es bei den meisten dynamischen Analysen.

Für weitergehende Analysen ist in der Regel ein Blick in den inneren Zustand des Programms notwendig. Daher müssen Vorkehrungen getroffen werden, um diesen Zustand zugreifbar zu machen. Zum Beispiel müssen evtl. Variablenwerte ausgegeben oder das Passieren bestimmter Punkte erfasst werden. Dazu ist eine Instrumentierung notwendig.

## Was messen?

### Beobachtung der Ausführung durch Messen

- Codeabdeckung oder Häufigkeit
  - Anweisungen, Zweige, Pfade, Aufrufe, ...
- berechnete Werte
- Laufzeit, Speicherverbrauch
- Reihenfolge von Operationen
- ...

Problem: Messung ändert Verhalten

Testsuite bestimmt

- Kosten (Zeit und Raum)
- Genauigkeit (was wird nie ausgeführt)

Vollständige Testabdeckung?

Vollständige Testabdeckung ist im allgemeinen nicht erreichbar, sobald ein Programm Schleifen enthält, die beliebig oft durchlaufen werden können. Dann müsste es unendlich viele Testfälle geben.



Problem häufig: zu viele Daten!

Lösungen:

- selektive Instrumentierung
- sofortige Verarbeitung (online)
- Umcodierung
  - verlustfreie Kompression
  - Filterung
  - Vorverarbeitung (Trace Summarization)

## Auswertung

- Aggregation
  - Statistiken
- Inferenz: Ableitung neuen Wissens
  - maschinelles Lernen
- Visualisierung
- online/offline

# Beispiel: Profiling

- Ziel: Performanceengpässe finden
- Messen der Zeiten
  - Zeit vor/nach Ausführung
  - Mitzählen, Summieren
- PC sampling

% cumulative	self	self	self	total		
time	seconds	seconds	calls	ms/call	ms/call	name
33.34	0.02	0.02	7208	0.00	0.00	open
16.67	0.03	0.01	244	0.04	0.12	offtime
16.67	0.04	0.01	8	1.25	1.25	memccpy
16.67	0.05	0.01	7	1.43	1.43	write
...						

2006-01-11

- Vorlesung Software-Reengineering
  - Dynamische Analyse
    - Anwendungsbeispiele
      - Beispiel: Profiling

## Beispiel: Profiling

- Ziel: Performanceengpässe finden
- Messen der Zeiten
  - Zeit vor/nach Ausführung
  - Mitzählen, Summieren
- PC sampling

% cumulative	self	self	self	total		
time	seconds	seconds	calls	ms/call	ms/call	name
33.34	0.02	0.02	7208	0.00	0.00	open
16.67	0.03	0.01	244	0.04	0.12	offtime
16.67	0.04	0.01	8	1.25	1.25	memccpy
16.67	0.05	0.01	7	1.43	1.43	write
...						

PC sampling: in festen Zeitintervallen (z.B. alle 10ms) wird ermittelt, worauf gerade der Programmzähler (PC) zeigt. Der Zähler für die entsprechende Anweisung (oder die Funktion, zu der die Anweisung gehört) wird um eins erhöht. Dies wird über den gesamten Programmlauf wiederholt. Funktionen, in denen sehr viel Zeit verbraucht wurde, haben damit am Ende einen hohen Zählerstand und können somit als Performance-kritisch identifiziert werden.

Vorteil: wesentlich geringerer Overhead als beim Merken der Zeiten bei jedem Funktionseintritt/-austritt.

Statisches Slicing: Erreichbarkeit im System-Dependency-Graph

Dynamisches Slicing: Verfolgen von Werten zur Laufzeit

Jeder Wert wird annotiert:

- wo wurde er berechnet?
- aus welchen Werten?

Dann: Rückwärtsverfolgung der Verweise

- wo wurde er berechnet?
- aus welchen Werten?

Beispiel:

```
a = 5;
if (x) {
  a++;
}
```

```
print(a);
```

Zur Laufzeit ist immer bekannt, ob x true oder false war, d.h. ob bei print a=5 oder a=6 ist.

# Beispiel: Aspect Mining (Breu 2004)

- Untersuchung von Aufrufrelationen:
    - Reihenfolge
    - Schachtelung
  - mehrfaches Auftreten
  - verschiedene Aufrufkontexte (opt.)
- Aspekt-Kandidaten

```

B() {
    C() {
        G() {}
        H() {}
    }
}
A() {}
B() {
    C() {}
}
B() {
    C() {}
    G() {}
    H() {}
}
A() {}
B() {

```

2006-01-11

- Vorlesung Software-Reengineering
  - └─ Dynamische Analyse
    - └─ Anwendungsbeispiele
      - └─ Beispiel: Aspect Mining (Breu 2004)

Beispiel: Aspect Mining (Breu 2004)

- Untersuchung von Aufrufrelationen:
  - Reihenfolge
  - Schachtelung
- mehrfaches Auftreten
- verschiedene Aufrufkontexte (opt.)
- Aspekt-Kandidaten

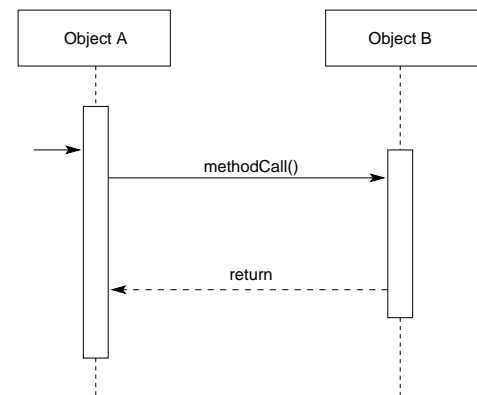
```

BO {
    CO {
        GO {}
        HO {}
    }
}
AO {}
BO {
    CO {
    }
}
AO {
    BO {
        CO {}
    }
}

```

An aspect is a modular unit that crosscuts the structure of other modular units.  
 Examples: Logging, Synchronisation, Exception handling, Communication

- Aufzeichnen von Methodenaufrufen
  - Aufrufer, Aufgerufener (Klasse, Instanz), Methode
- Pattern matching
  - Suchen ähnlicher Muster
  - Klassen, Instanzen, Methoden, Struktur
- Auswahl über Klassen/Methoden
  - Collaboration Browser
- Visualisierung als Sequenzdiagramm



## Beispiel: Invariantenerkennung (Ernst 2000)

```
public class StackAr {
    Object[] theArray;
    int topOfStack;
    ...
}
```

Invarianten:

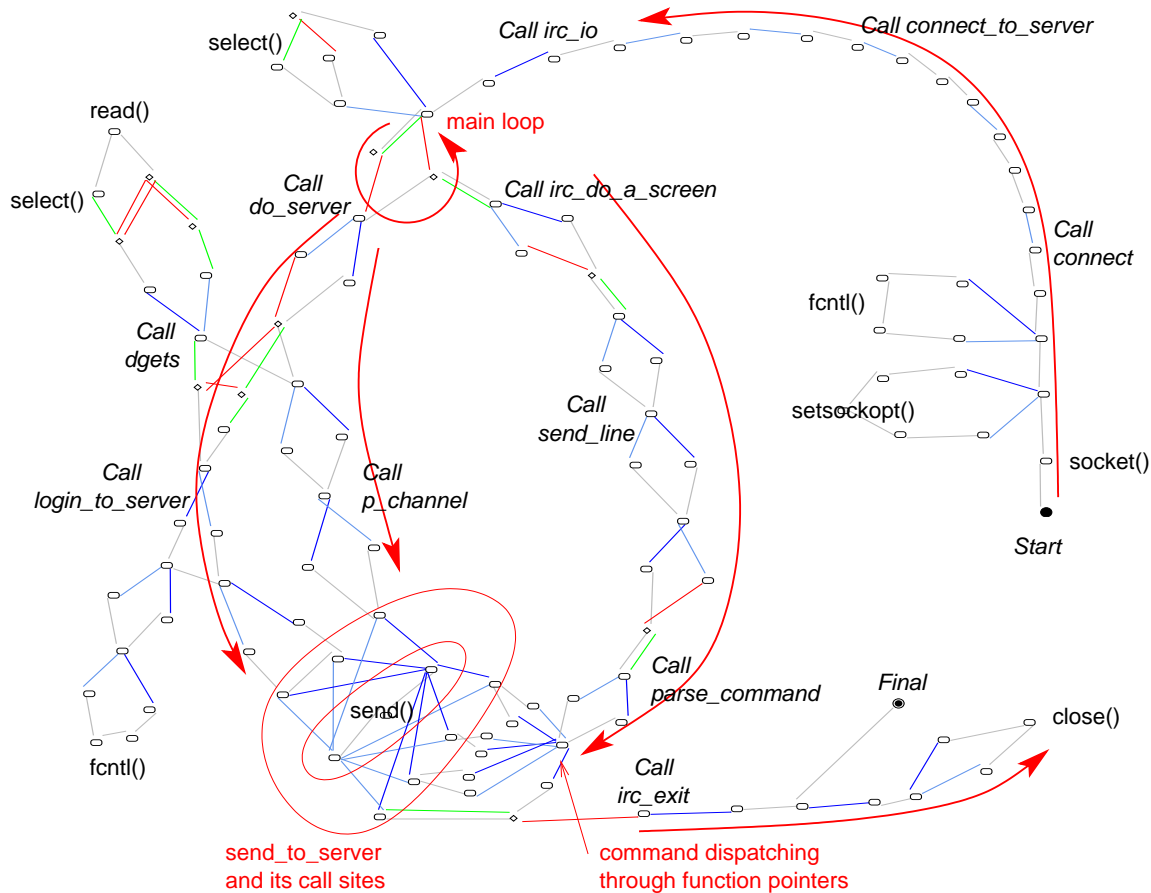
- `theArray != null`
- `theArray.getClass() == java.lang.Object[].class`
- `topOfStack >= -1`
- `topOfStack <= theArray.length - 1`
- `theArray[0..topOfStack] elements != null`
- `theArray[topOfStack+1..] elements == null`

- Betrachtung aller Variablen
- auch abgeleitete Variablen
  - sum, min, max
  - first, second, last
  - $s[i]$ ,  $s[i-1]$ , Teilsequenzen  $s[0..i]$ ,  $s[0..i-1]$
  - Anzahl Aufrufe einer Funktion
- Instanziierung und Testen aller potentiellen Invarianten
- Filterung der übrigen Invarianten
  - hinreichend signifikant?

## Beispiel: Dynamic Object Process Graph (Quante und Koschke 2006)

- Rekonstruktion des Kontrollflussgraphen
- Projektion auf ein Objekt
- Instrumentierung von
  - Verzweigungen
  - Routinenaufrufen
  - Operationen auf Objekt
- Verwandt zu Slicing

# Beispiel: Dynamic Object Process Graph



## Statische Analyse von Programmverhalten

- Analyse des Codes
- Modell des Programmzustands erstellen
- Mögliches Verhalten folgern
  - abstrakte Ausführung

### Abstrakte Ausführung:

- meist Datenflussanalyse
- Transferfunktion für jede Anweisung
  - Änderung des abstrakten Zustands?
- Beispiel:  $y = x++;$

## Wahl der abstrakten Domäne

- Domäne: {even, odd, either}  
 $\langle x \text{ odd}, y \text{ either} \rangle \quad y = x++;$      $\langle x \text{ even}, y \text{ odd} \rangle$   
 $\langle x \text{ even}, y \text{ either} \rangle \quad y = x++;$      $\langle x \text{ odd}, y \text{ even} \rangle$
- Domäne: {prime, nonprime, anything}  
 $\langle x \text{ prime}, y \text{ anything} \rangle \quad y = x++;$      $\langle x \text{ anything}, y \text{ prime} \rangle$
- Domäne: Menge von möglichen nat. Zahlen  
 $\langle x \in \{3,5,7\}, y \in \mathbb{N} \rangle \quad y = x++;$      $\langle x \in \{4,6,8\}, y \in \{3,5,7\} \rangle$
- Domäne: symbolische Auswertung  
 $\langle x_n, y_n \rangle \quad y = x++;$      $\langle x_{n+1} = x_n + 1, y_{n+1} = x_n \rangle$

## Wahl der abstrakten Domäne

### Abstraktion bestimmt

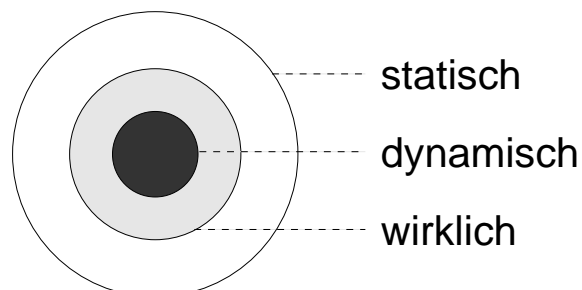
- Kosten (Zeit und Raum)
- Genauigkeit (Informationsverlust)

# Statisch vs. Dynamisch

Statisch	Dynamisch
abstrakte Domäne aufwändig wenn genau	konkrete Ausführung kann aufwändig sein
konservativ wegen Abstraktion	präzise keine Abstraktion
vollständig (plus mehr) da konservativ	unvollständig nicht verallgemeinerbar
begrenzte Sichtweite eher lokal	kann entfernte Beziehungen aufdecken
Abstraktion bestimmt Kosten, Genauigkeit	Testsuite bestimmt Kosten, Genauigkeit

# Statisch vs. Dynamisch

- dynamisch: zu konkret, nicht verallgemeinerbar  
→ Teilmenge
- statisch: zu abstrakt, daher ungenau  
→ Obermenge
- Wahrheit liegt (meist) dazwischen



Beispiel:

```
y = f(x);  
if (y < 0)  
    y = -y;
```

Dynamisch:  $y \in \{0, 2, 7, 43, 79\}$

Statisch:  $0 \leq y \leq \text{MAX\_INT}$

Wirklich:  $0 \leq y \leq 100$

## Kombination

- Aggregation
  - dynamisch  $\rightarrow$  statisch: Übergeneralisierung erkennen
  - statisch  $\rightarrow$  dynamisch: gezielte Instrumentierung
- Hybride Analysen
- Analoge Analysen
  - gleiches Problem, andere Domäne

Anwendung	dynamisch	statisch
Typprüfung	×	×
Speicherzugriffsprüfung	×	×
Slicing	×	×
Merkmalsuche	×	×
Protokollerkennung	×	×
Protokollverifikation	×	×
Metriken	Verhalten	Struktur
- Profiling	×	—
- Testabdeckung	×	—
Finden toten Codes	—	×
Klonerkennung	—	×
Zeigeranalyse	leicht	schwierig

## Wiederholungs- und Vertiefungsfragen I

- Nennen Sie Vor- und Nachteile dynamischer Analyse gegenüber statischer Analyse
- Was kann durch dynamische Analyse alles erfasst werden?
- Wie kann dynamische Analyse durchgeführt werden?
- Welche Probleme hat die Instrumentierung?
- Nennen Sie Beispiele für Anwendungen dynamischer Analyse.

- 1 **Agrawal und Horgan 1990** AGRAWAL, Hiralal ; HORGAN, Joseph R.: Dynamic Program Slicing. In: Proceedings of the Conference on Programming Language Design and Implementation. White Plains, New York, USA, Juni 1990
- 2 **Breu 2004** BREU, Silvia: Aspect Mining Using Event Traces, Universität Passau, Diplomarbeit, März 2004
- 3 **Ernst 2000** ERNST, Michael: Dynamically Discovering Likely Program Invariants. Seattle, Washington, USA, University of Washington, Department of Computer Science and Engineering, Dissertation, August 2000
- 4 **Quante und Koschke 2006** QUANTE, Jochen ; KOSCHKE, Rainer: Dynamic Object Process Graphs. In: Proceedings of the European Conference on Software Maintenance and Reengineering. Bari, Italy, März 2006

- 5 **Richner und Ducasse 2002** RICHNER, Tamar ; DUCASSE, Stéphane: Using Dynamic Information for the Iterative Recovery of Collaborations and Roles. In: Proceedings of the International Conference on Software Maintenance. Montreal, Canada : IEEE Computer Society Press, Oktober 2002, S. 34–43