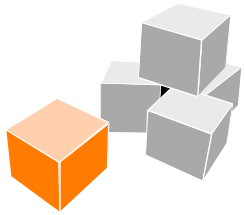
TECHNISCHE
UNIVERSITÄT
DARMSTADT



Software
Technology
Group
TU Darmstadt | FB Informatik


Dr.-Ing. Michael Haupt

in Zusammenarbeit mit
Dipl.-Inform. Christoph Bockisch
Stephan Ritzkowski
Prof. Dr.-Ing. Mira Mezini


# Omniscient Debugging und Slicing für Java

# Überblick

- das Fachgebiet Softwaretechnik

- Probleme beim Debugging
- "Debugging rückwärts" mit Omniscient Debugging

- statisches und dynamisches Slicing
- Fehlersuche mit Omniscient Slicing

- Zusammenfassung

# Das Fachgebiet Softwaretechnik

***"teile und herrsche"* als Konstruktionsprinzip**

modulare und entkoppelte Softwarearchitekturen
aspektorientierte und komponentenorientierte Softwareentwicklung
service-orientierte Softwarearchitekturen und Produktlinien
dynamisch anpassbare Software

**zwei komplementäre Wege zur Beherrschung der Softwarekomplexität**

**intelligente Konstruktionswerkzeuge**

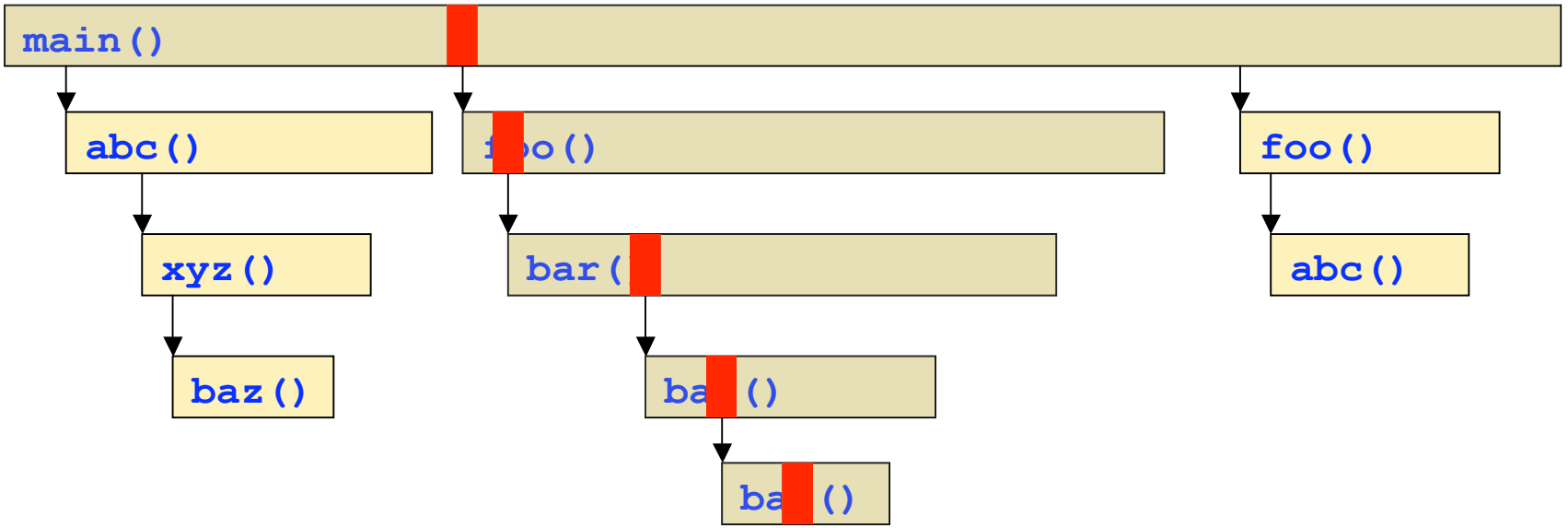
intelligentes Wissensmanagement und -Retrieval für Entwicklungsumgebungen
integrierte erweiterbare statische Analyseplattformen
Softwareexploration und -visualisierung für leichtere Wartung und Re-Engineering

# Aktuelle Aktivitäten

- neue Programmiersprachen
  - inklusive geeigneter Laufzeit- und Entwicklungsumgebungen
  - Schwerpunkt: aspektorientierte Programmierung
  - CaesarJ, Steamloom
- Anwendung der neuen Programmiersprachen
  - Software-Produktlinien
  - modulare Komponenten-Middleware
  - aspektorientierte Web-Service-Komposition
  - modellbasierte Entwicklung mobiler Anwendungen
- Entwicklungsumgebungen
  - als Plattformen für Software Knowledge/Information Engineering
  - Exploration, Visualisierung, Analyse

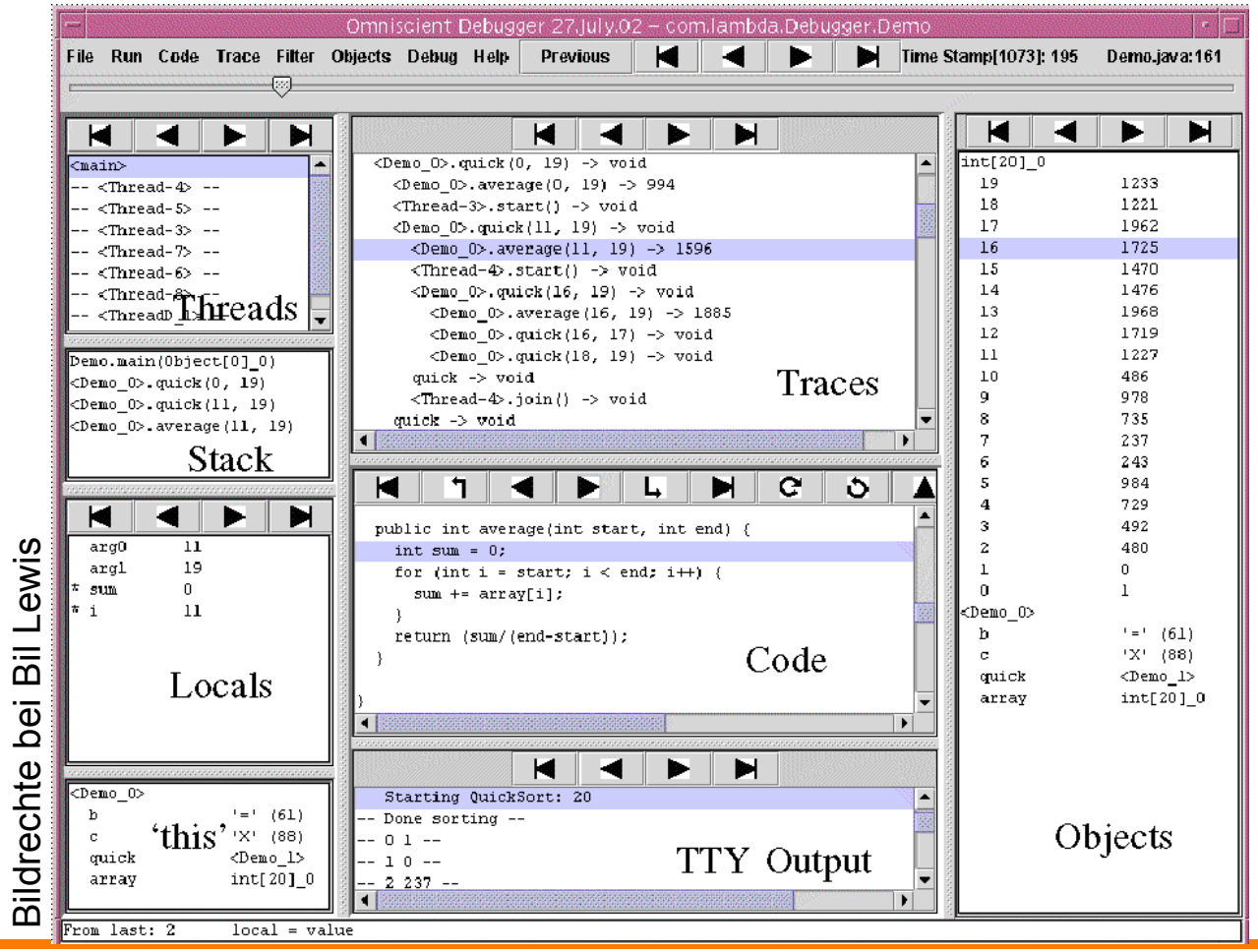
# "Traditionelles" Debugging

- Breakpoints: Markierung potenziell interessanter Stellen
  - Ausführung der Anwendung bis zum Breakpoint
  - Untersuchung des Zustands, (ggf. schrittweise) Weiterausführung
- nicht (oder nur schwer) beantwortbare Fragen
  - "Woher kommt diese `null`-Referenz?"
  - "Welche Methode wurde vor dieser aufgerufen?"
  - "Wie wirkt sich diese Zuweisung im weiteren Verlauf aus?"



# Omniscient Debugging

- "debugging backwards in time"
  - vorgestellt von Bil Lewis, 2002
  - erlaubt Navigation durch komplette Ausführungsvergangenheit
  - Punkte in der Ausführung (samt Zuständen) werden aufgezeichnet




Bildrechte bei Bil Lewis

# Omniscient Debugging

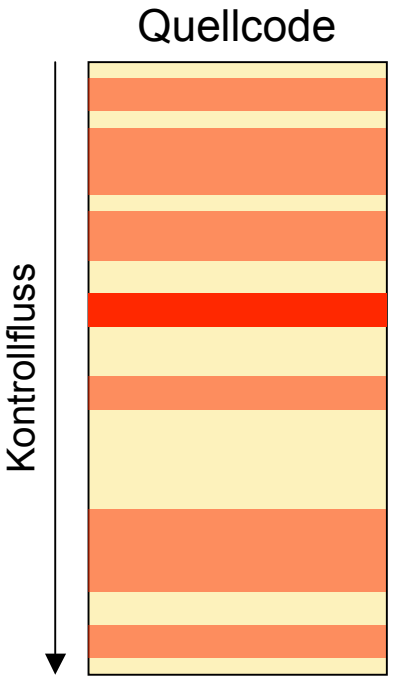
- "debugging backwards in time"
  - vorgestellt von Bil Lewis, 2002
  - erlaubt Navigation durch komplette Ausführungsvergangenheit
  - Punkte in der Ausführung (samt Zuständen) werden aufgezeichnet

- Navigation
  - ODB erlaubt schrittweise Suche nach Fehlerquellen
  - erwünscht: unmittelbare Übersicht aller möglichen Fehlerquellen

- Zukunft der Ausführung
  - vorausschauendes Debugging ist im ODB nicht möglich
  - erwünscht: Übersicht möglicher Einflüsse bestimmter Zustände

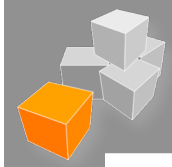
# Slicing

- Slice: die Menge der Anweisungen,
  - die den Zustand an dieser Anweisung beeinflussen (backward slice)
  - deren Zustand von dieser Anweisung beeinflusst wird (forward slice)




- statisches Slicing
  - basiert auf statischer Programmanalyse
  - Slices enthalten potenziell beeinflussende / beeinflusste Anweisungen

    → Slices sind größer als nötig
- dynamisches Slicing
  - basiert auf einem konkreten (terminierten) Programmlauf
  - Slices enthalten tatsächlich beeinflussende / beeinflusste Anweisungen



# Slicing

```
void foo(int x) {
    if(x < 10)
        System.out.println(„kleiner als 10“);
    else
        System.out.println(„größer als 10“);
}

void bar() {
    int k = (int) (Math.random() * 100.0);
    foo(k);
}
```

statischer Forward Slice

```
void foo(int x) {
    if(x < 10)
        System.out.println(„kleiner als 10“);
    else
        System.out.println(„größer als 10“);
}

void bar() {
    int k = (int) (Math.random() * 100.0);
    foo(k);
}
```

dynamischer Forward Slice (k=5)

```
void foo(int x) {
    if(x < 10)
        System.out.println(„kleiner als 10“);
    else
        System.out.println(„größer als 10“);
}

void bar() {
    int k = (int) (Math.random() * 100.0);
    foo(k);
}
```

# Omniscient Slicing – Backtrac

- Kombination von Omniscient Debugging mit Slicing
  - Anwendungslauf generiert vollständigen Trace
  - erreicht durch Instrumentation zur Ladezeit (class loader/agent)
  - keine Interaktion zur Laufzeit: „post mortem“-Debugging

- Backtrac
  - realisiert als Eclipse-Plugin
  - OD-ähnliche Navigation über die Ausführungszeit
  - zusätzliche Möglichkeit, dynamische Slices zu berechnen

  → **DEMO**

# Hintergründe zu Backtrac

- Projekt Steamloom am FG Softwaretechnik
  - JVM mit Unterstützung für dynamische AOP
  - (De)Installation von Aspekten zur Laufzeit
  - warum? → Sprachmechanismen gehören in die VM!
  - minimale Aspekt-Infrastruktur, hohe Geschwindigkeit
- dynamische Aspekte
  - Verhalten der Anwendung ändert sich mit Aspektzuständen
  - dynamisches Slicing bietet gute Analysemöglichkeiten
- Unterstützung in Backtrac
  - Debugging für dynamische Aspektorientierung (→ Steamloom)
  - Backtrac bietet auch Aspekt-Slicing

# Eigenschaften von Backtrac

- Backtrac ist ein Prototyp
  - entstanden im Rahmen einer Diplomarbeit
  - einige OD- und Slicing-Features (noch) nicht implementiert
- Kritikpunkte
  - Größe von Traces (derzeit: reines Textformat)
  - interner Speicherbedarf für Slicing-Graphen
- zukünftige Arbeiten
  - Erweiterung der OD-Fähigkeiten
  - Verbesserung der Slicing-Unterstützung
  - Anpassung für weitere AOP-Ansätze
  - Integration mit Lewis‘ Omniscient Debugger

# Vielen Dank für Ihre Aufmerksamkeit!

http://www.st.informatik.tu-darmstadt.de/