Interaktive graphische Fehlerbehebung mit UML

Vortrag von Jan Sebastian Siwy am 28. Juni 2004
im Rahmen des Seminars “Visualisierung verteilter Systeme”

Überblick

Fehlerbehebung sehr komplex

- Überblick
- Details

unzureichende Unterstützung durch gängige Werkzeuge

Ziel: dynamische Übertragung des Programmzustands auf UML-Diagramme

Überblick

Grundlage

- Timothy Jacobs und Benjamin Musial
  - Air Force Institute of Technology
  - Veröffentlichung 2003

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

1. Einleitung

Ziele

- Überblick
  - Entwurf im Gedächtnis
  - Ursache ≠ Symptom
- Überwachung der Anwendung

1. Einleitung

Grafische Darstellung

- Informations- und Kontrollflüsse
  - Zusammenhang zwischen Prozessen und Daten
- UML-Objektdiagramme
  - allgemein bekannt

1. Einleitung

Grafische Darstellung

- verschiedene Abstraktionsstufen
- Animationen und Farben
  - Nachvollziehbarkeit

1. Einleitung

Verteilte Systeme

- noch höhere Komplexität

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

2. Hintergrund

Strukturierung / Visualisierung

- Flussdiagramme (?)
- Klassendiagramme
- Objektdiagramme
- Formatierung von Quellcode

2. Hintergrund

Kriterien für grafische Darstellung

- Umfang *(scope)*
- Abstraktion *(abstraction)*
- Spezifikationsmethoden *(spec. method)*
- Schnittstelle *(interface)*
- Darstellung *(presentation)*

2. Hintergrund

Einstufung gängiger Werkzeuge

- Visual C++ (1998)
- ProDev Workshop (2002)

Einstufung von Forschungsprojekte

- meist nur mit Kern “Präsentation”

JAN und DDD (später)

2. Hintergrund

Einstufung von UML

- umfangreich
  - verschiedene Diagrammtypen
- abstrakt und ausdrucksstark

2. Hintergrund

Verbreitung von UML:

- CASE-Werkzeuge
- hohe Verbreitung vom Entwurf
- Aber: kaum bei Analyse und Fehlerbehebung

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

3. Anwendung

Entwickelte Software basiert auf:

- ArgoUML
- Java Platform Debugger Architecture (JPDA)
- eigene Entwicklungen

3. Anwendungen

Arbeitsweise

- Informationen über Exemplare über JPDA extrahieren
- Objektdiagramm modifizieren
  - Layout anpassen
  - Methodenaufrufe sichtbar machen

3. Anwendungen

Fischaugen-Effekt

- unterschiedliche Detailstufen
- Fokus auf ein Objekt
- je weiter weg, desto kleiner

3. Anwendungen

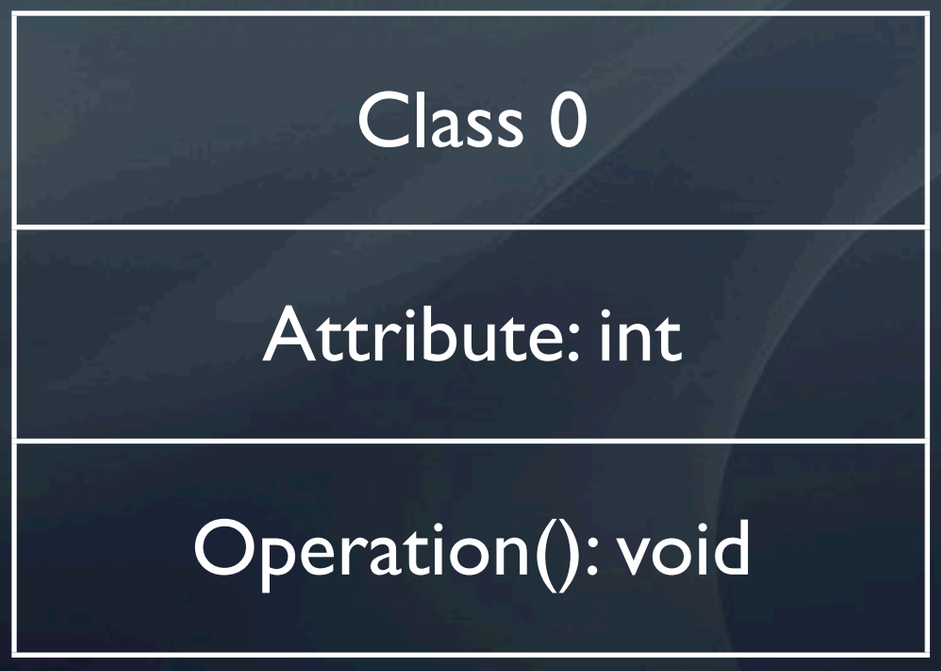
Fischaugen-Effekt

- LOD *(level of details)*
  - Größe und Art der Informationen
- DOI *(degree of interest)*
  - Häufigkeit der Benutzung
  - Entfernung vom ausgewählten Exemplar

3. Anwendungen

Fischaugen-Effekt

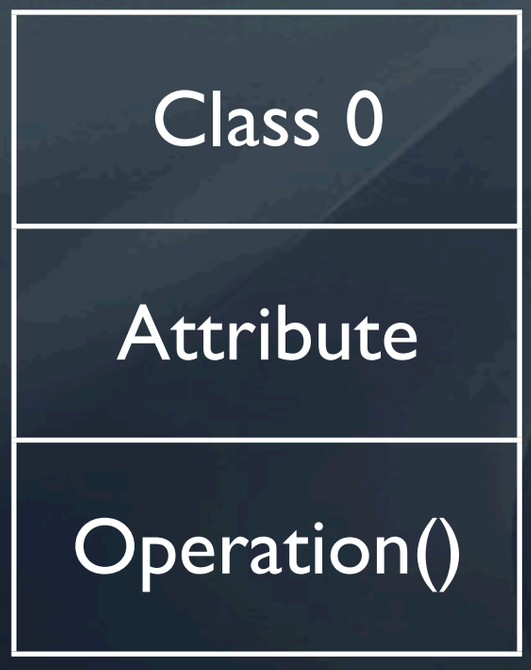
- LOD 0



3. Anwendungen

Fischaugen-Effekt

- LOD 1



3. Anwendungen

Fischaugen-Effekt

- LOD 2

Class 0

3. Anwendungen

Fischaugen-Effekt

- LOD 3

3. Anwendungen

Fischaugen-Effekt

- LOD 4

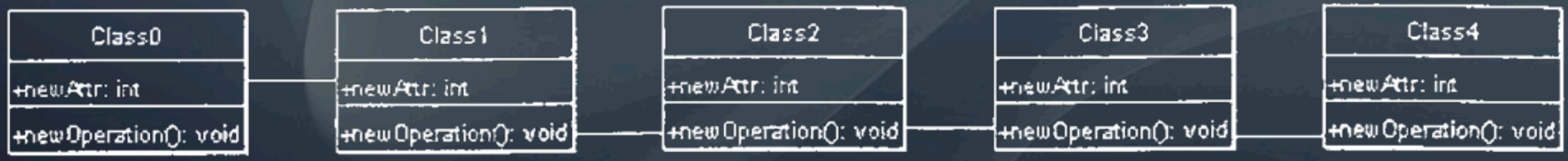
3. Anwendungen

Fischaugen-Effekt

- LOD 5

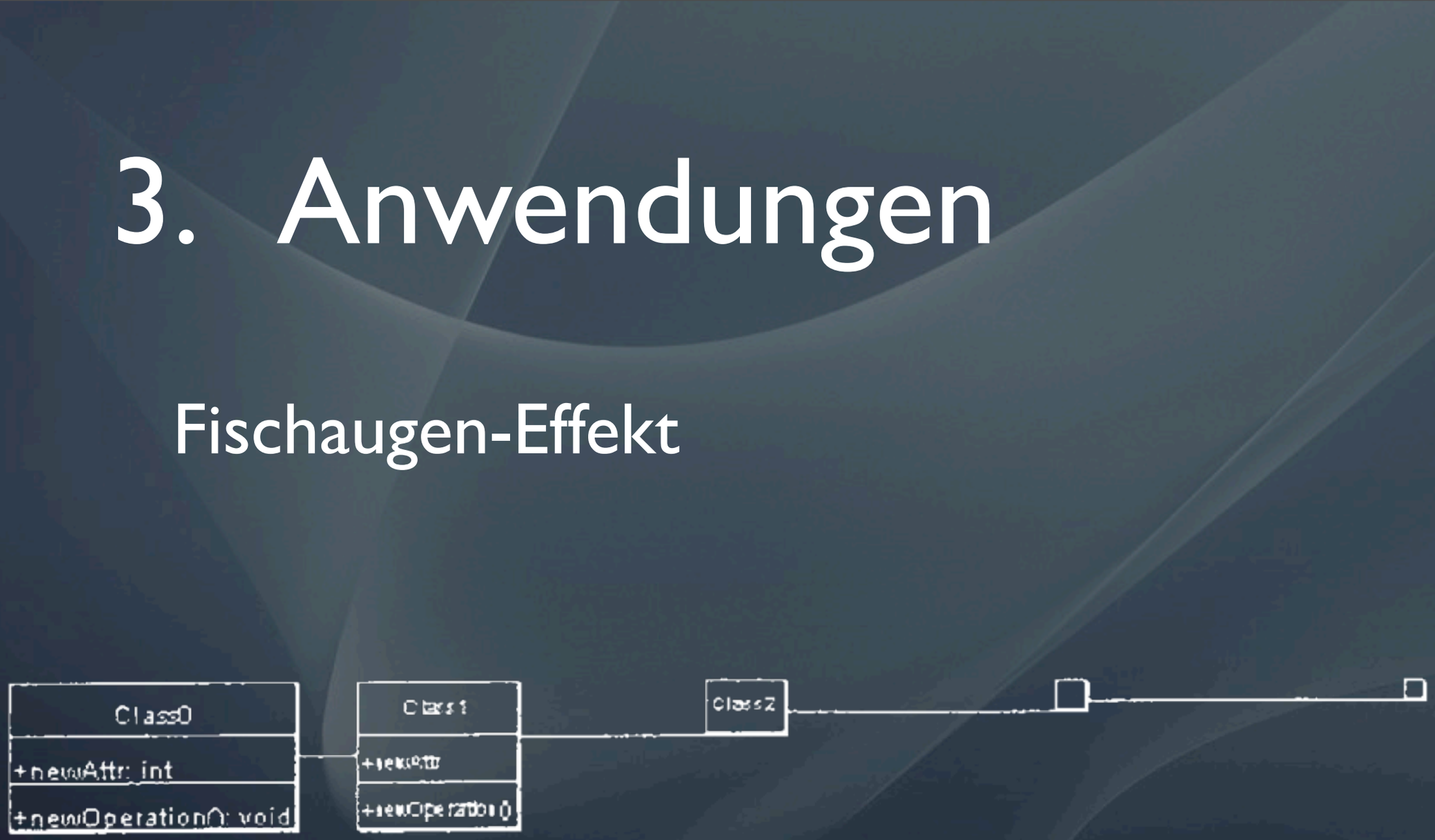
3. Anwendungen

Fischaugen-Effekt



3. Anwendungen

Fischaugen-Effekt



3. Anwendungen

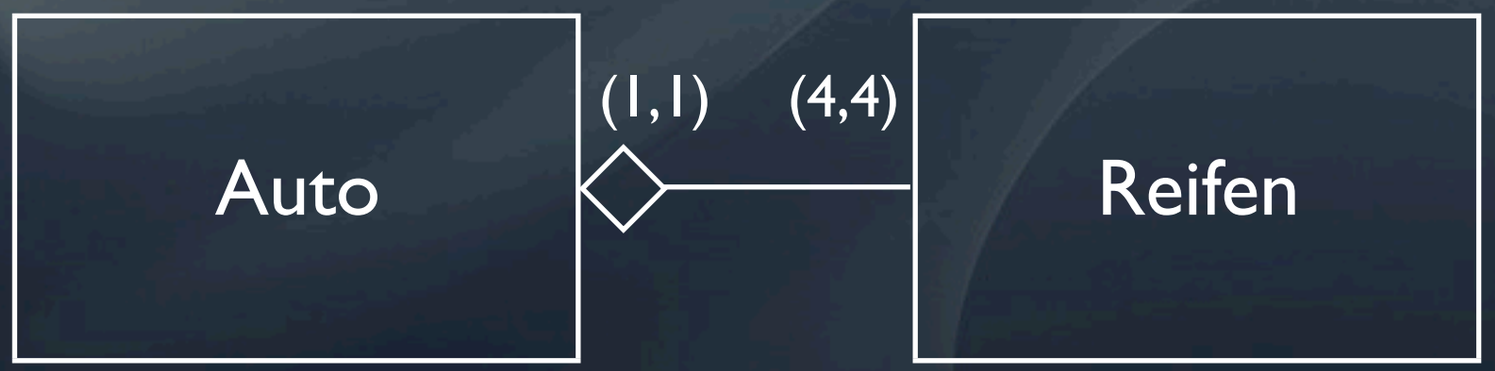
Aussparung

- Aggregation

Exkurs

Aggregation

- Beispiel: Auto mit 4 Reifen



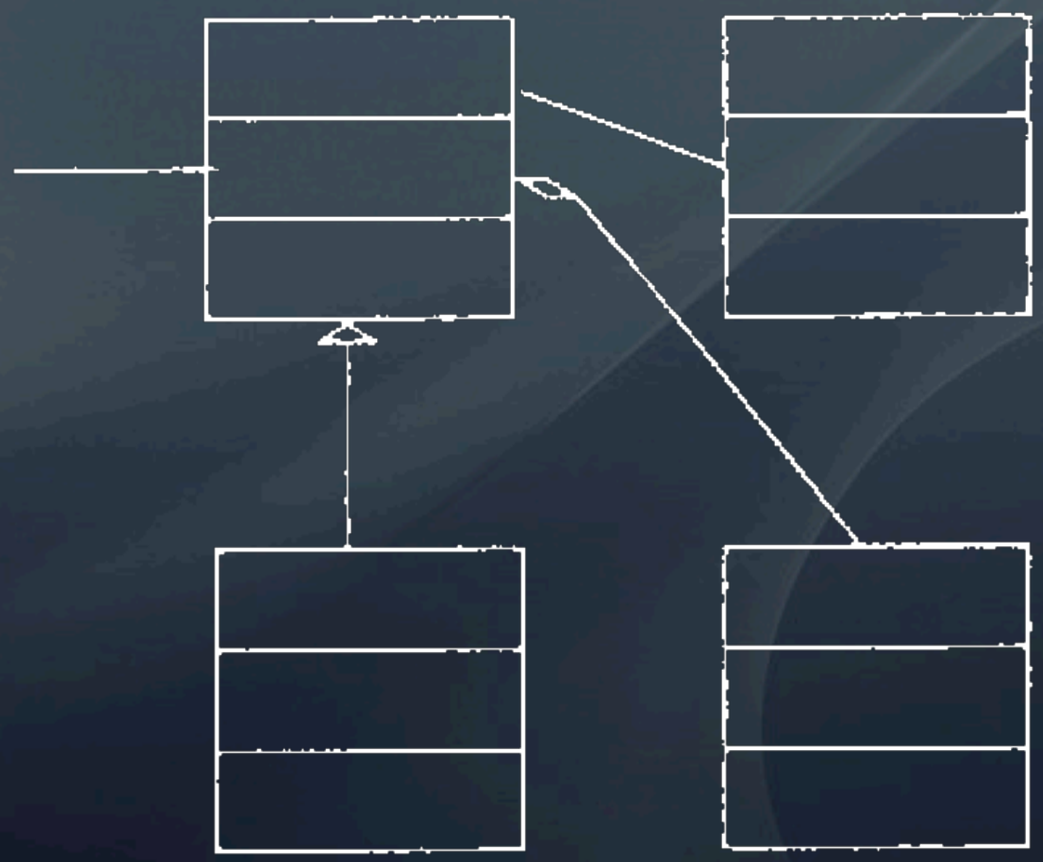
3. Anwendungen

Aussparung

- Aggregation
  - Darstellung der Hauptkomponente
  - Bestandteile nur andeuten

3. Anwendungen

Aussparung



3. Anwendungen

Aussparung

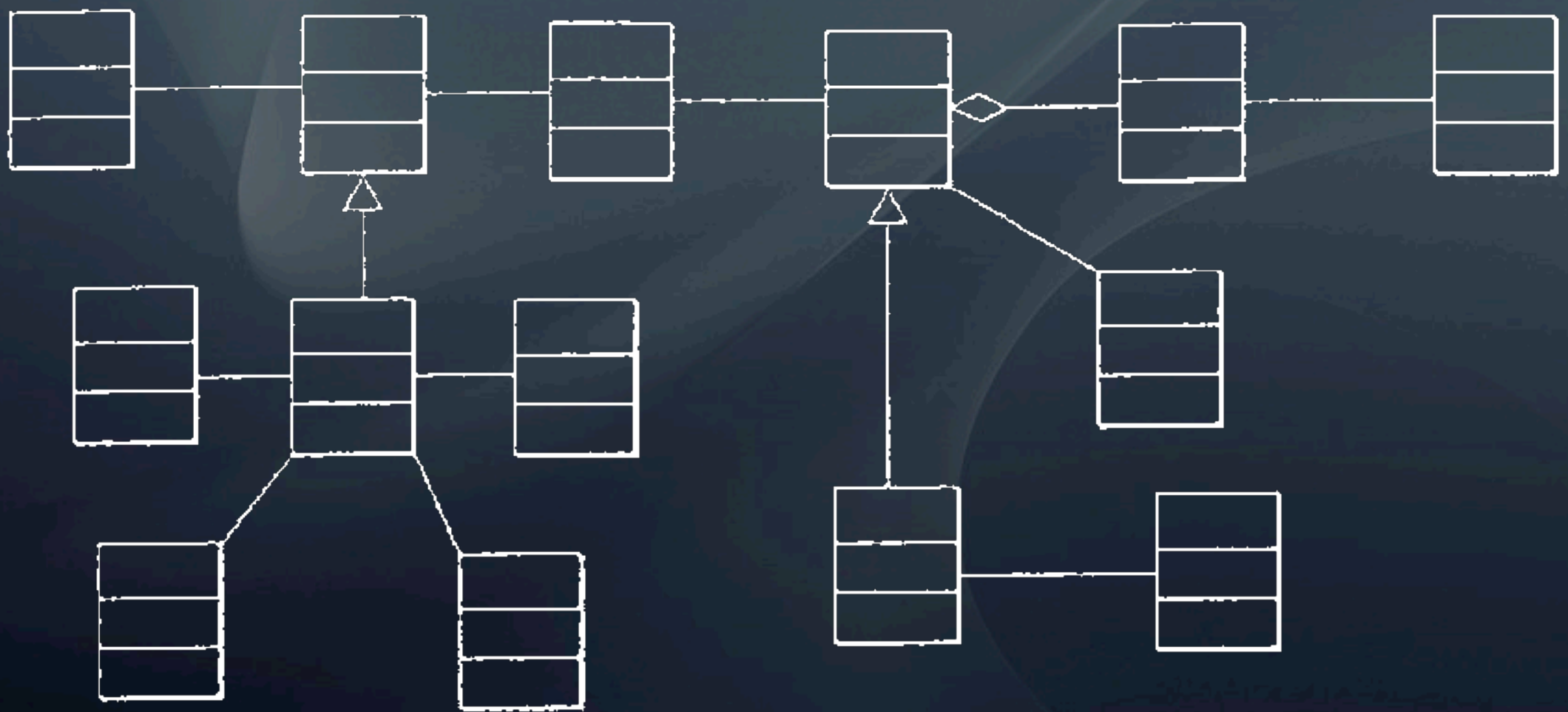
3. Anwendungen

Layout

- automatische Anordnung
- hierarchische Darstellung

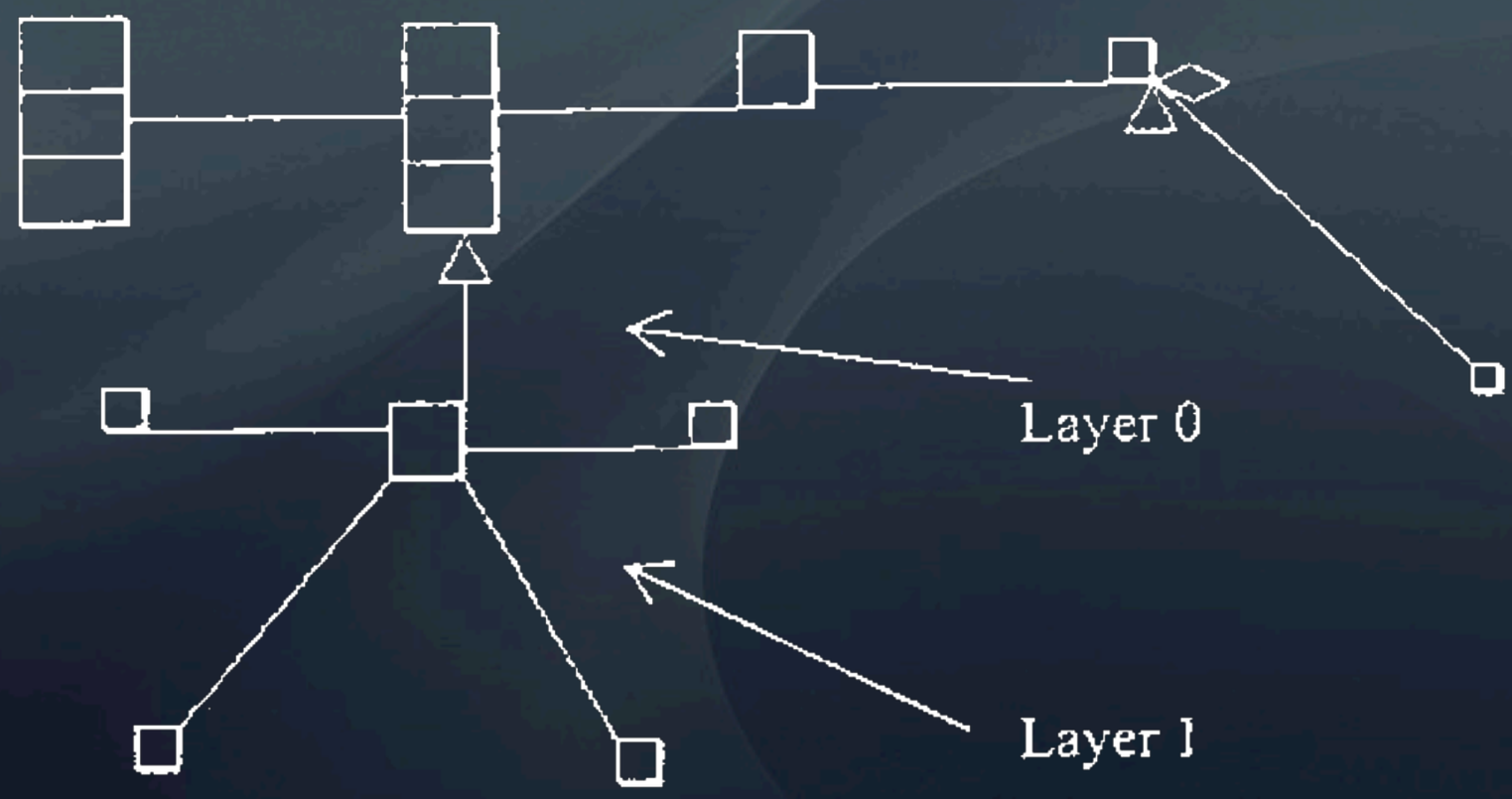
3. Anwendungen

Layout



3. Anwendungen

Layout



3. Anwendungen

Layout

3. Anwendungen

Layout

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

4. Graph. Fehlerbehebung

"focus + context"

- mehr Exemplare gleichzeitig sichtbar
- bessere Übersicht
- entfernte Exemplare mit weniger Details

— aber Fokus veränderbar

4. Graph. Fehlerbehebung

Arbeitsweise

- Herausfinden, wo etwas passiert
- Fokus ändern
- Kontrollfluss beobachten
- Variablen auslesen
- ggf. Quellcode einer Klasse einblenden

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

5. Diskussion

"focus + context"

- Beibehaltung der UML-Semantik
- Verbesserung des Platzbedarfs
  - einfach Exemplare: 24 vs. 75 pro "Bildschirm"

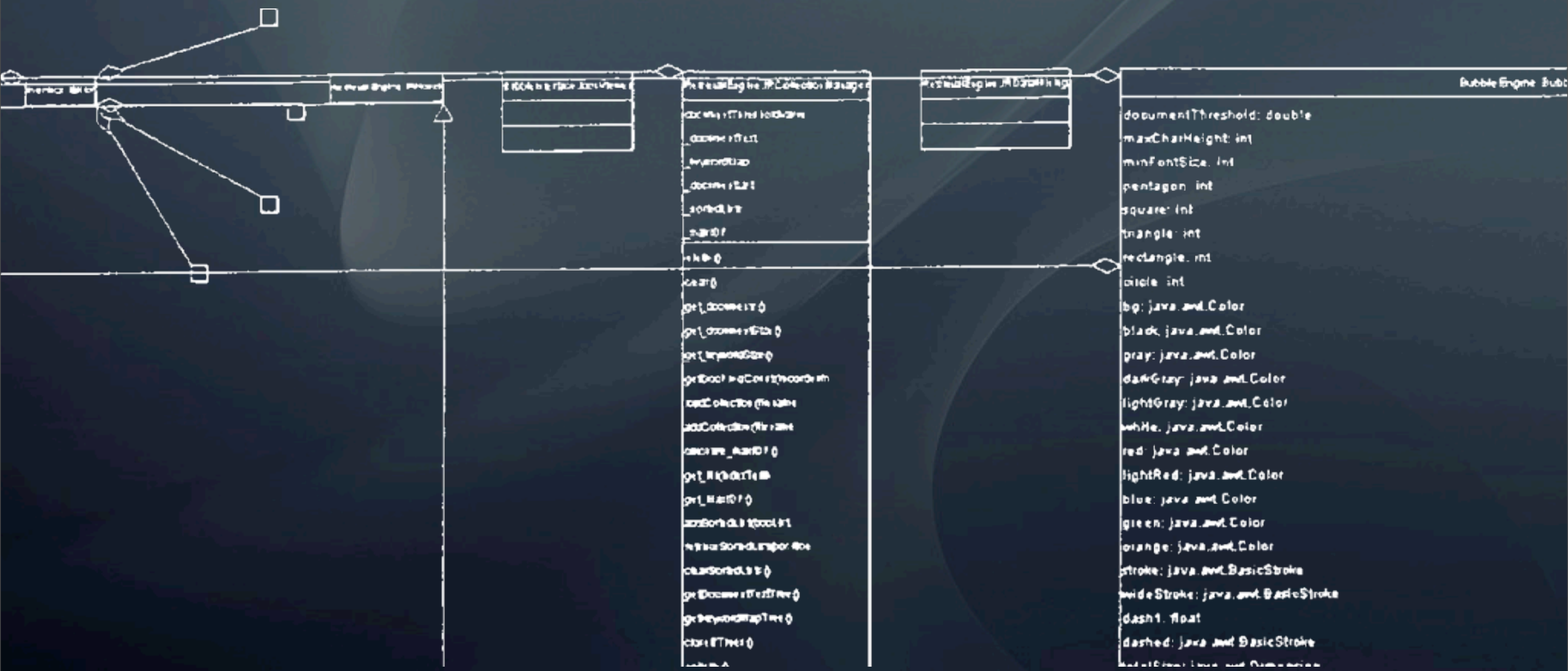
5. Diskussion

"focus + context"

- großen Exemplare im Vordegrund problematisch

5. Diskussion

“focus + context”



5. Diskussion

Einordnung gemäß genanter Kriterien

- Umfang: Struktur zur Laufzeit
- Abstraktion: unterschiedliche Detailgrade
- Spezifikationsmethode: unbekannt
- Schnittstelle: Fokus ändern
- Darstellung: UML und Quellcode

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

6. Verteilte Systeme

6. Verteilte Systeme

Was hat dies mit verteilten Systemen zu tun?

6. Verteilte Systeme

Was hat dies mit verteilten Systemen zu tun?

Nichts!

6. Verteilte Systeme

Was hat dies mit verteilten Systemen zu tun?

Nichts!

Wird in späteren Versionen realisiert

Exkurs

JAN (Java Animation)

Exkurs

JAN

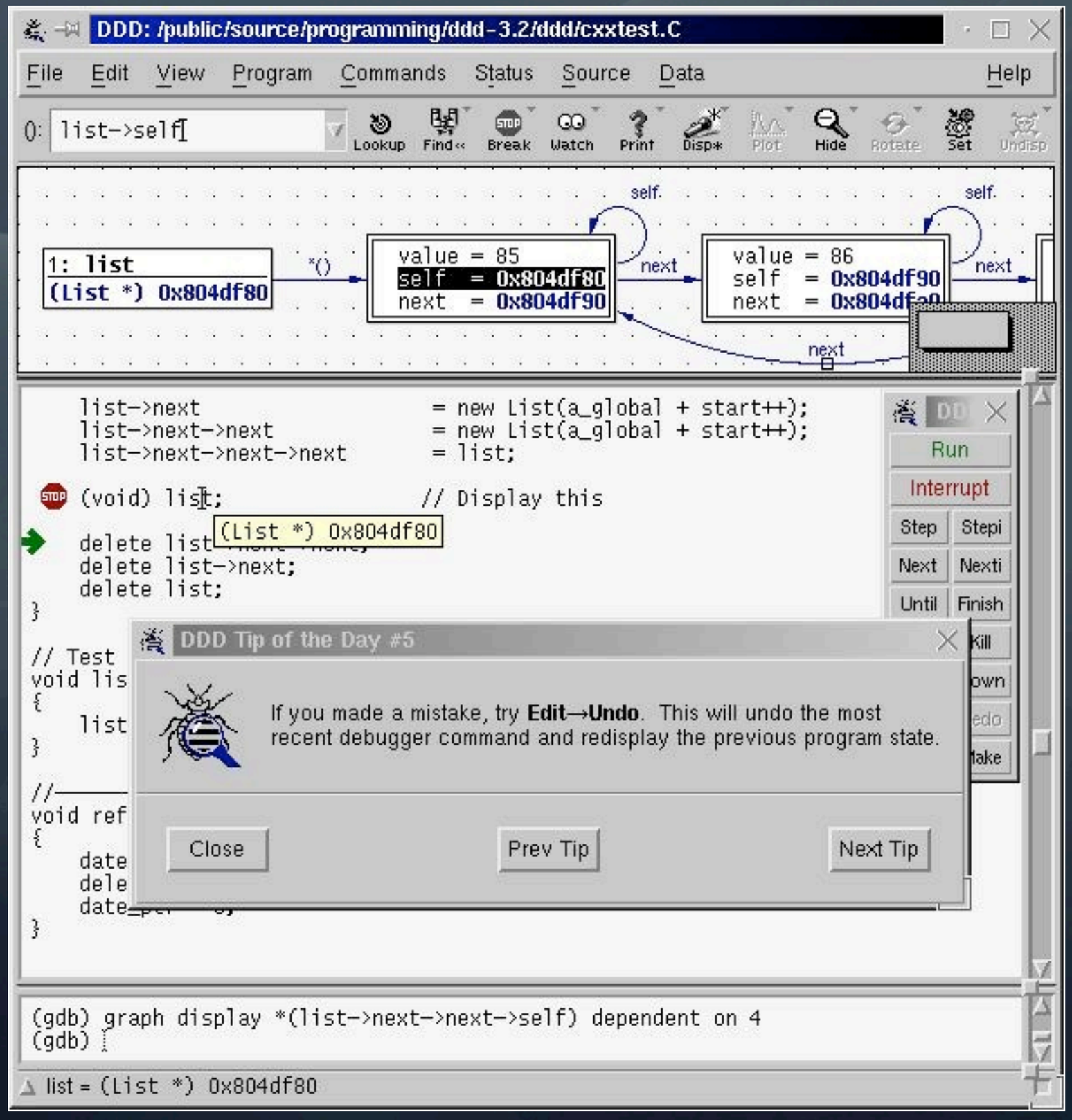
- Vorteile
  - Sequenzdiagramm
    Komposition gut erkennbar
- Nachteil
  - Modifikation am Quellcode notwendig
    Layout

Exkurs

DDD (Data Display Debugger)

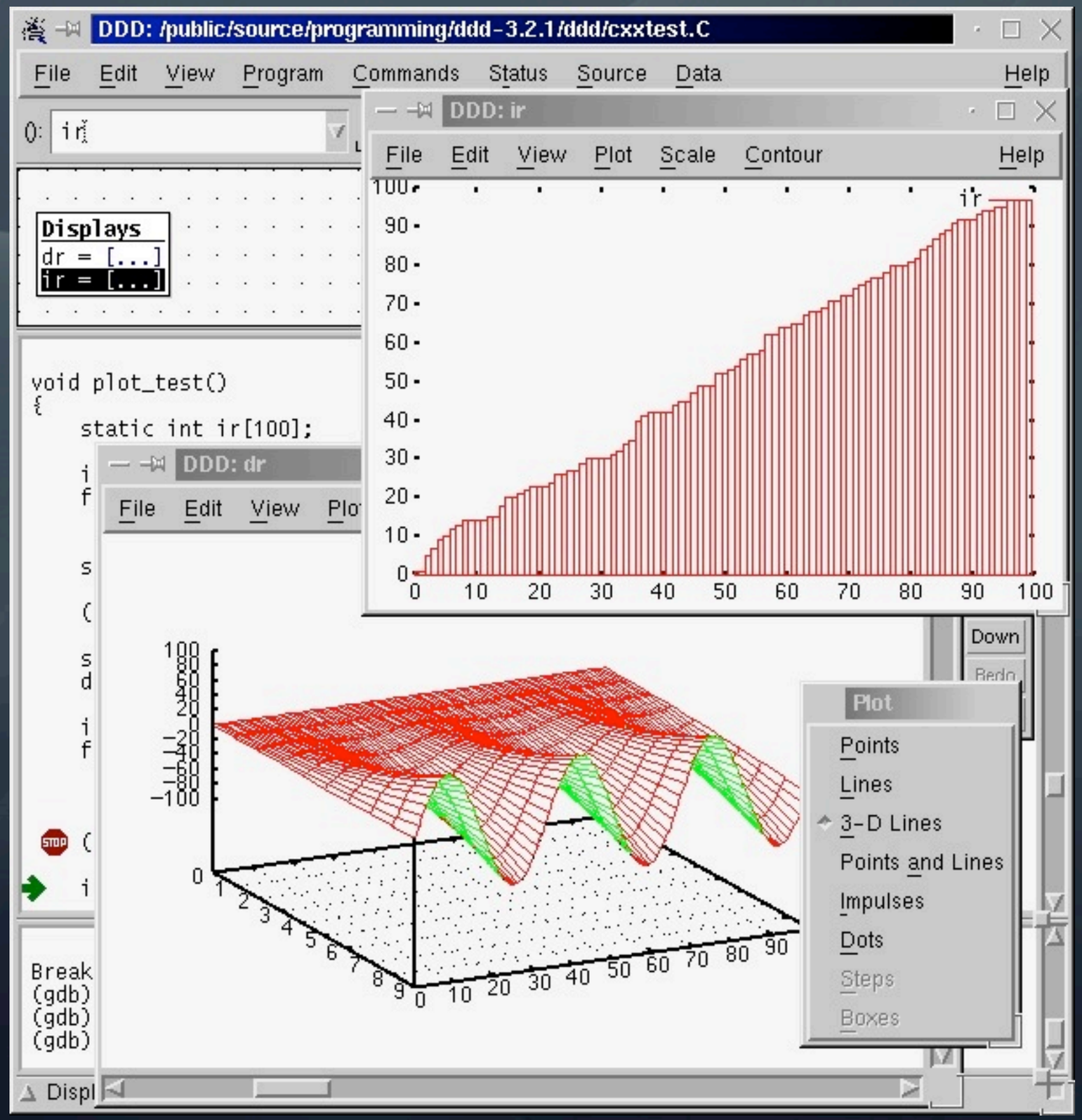
Exkurs

DDD


DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
(): list->self
Lookup Find« Break Watch Print Disp* Plot Hide Rotate Set Undisp
self.
self.
1: list
(List *) 0x804df80
*()
value = 85
self = 0x804df80
next = 0x804df90
next
value = 86
self = 0x804df90
next
next
list->next = new List(a_global + start++);
list->next->next = new List(a_global + start++);
list->next->next->next = list;
(void) list; // Display this
(List *) 0x804df80
delete list;
delete list->next;
delete list;
}
// Test
void lis
{
list
}
//
void ref
{
date
dele
date
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Kill
own
edo
lake
DDD Tip of the Day #5
If you made a mistake, try Edit→Undo. This will undo the most recent debugger command and redisplay the previous program state.
Close
Prev Tip
Next Tip
(gdb) graph display *(list->next->next->self) dependent on 4
(gdb)
list = (List *) 0x804df80

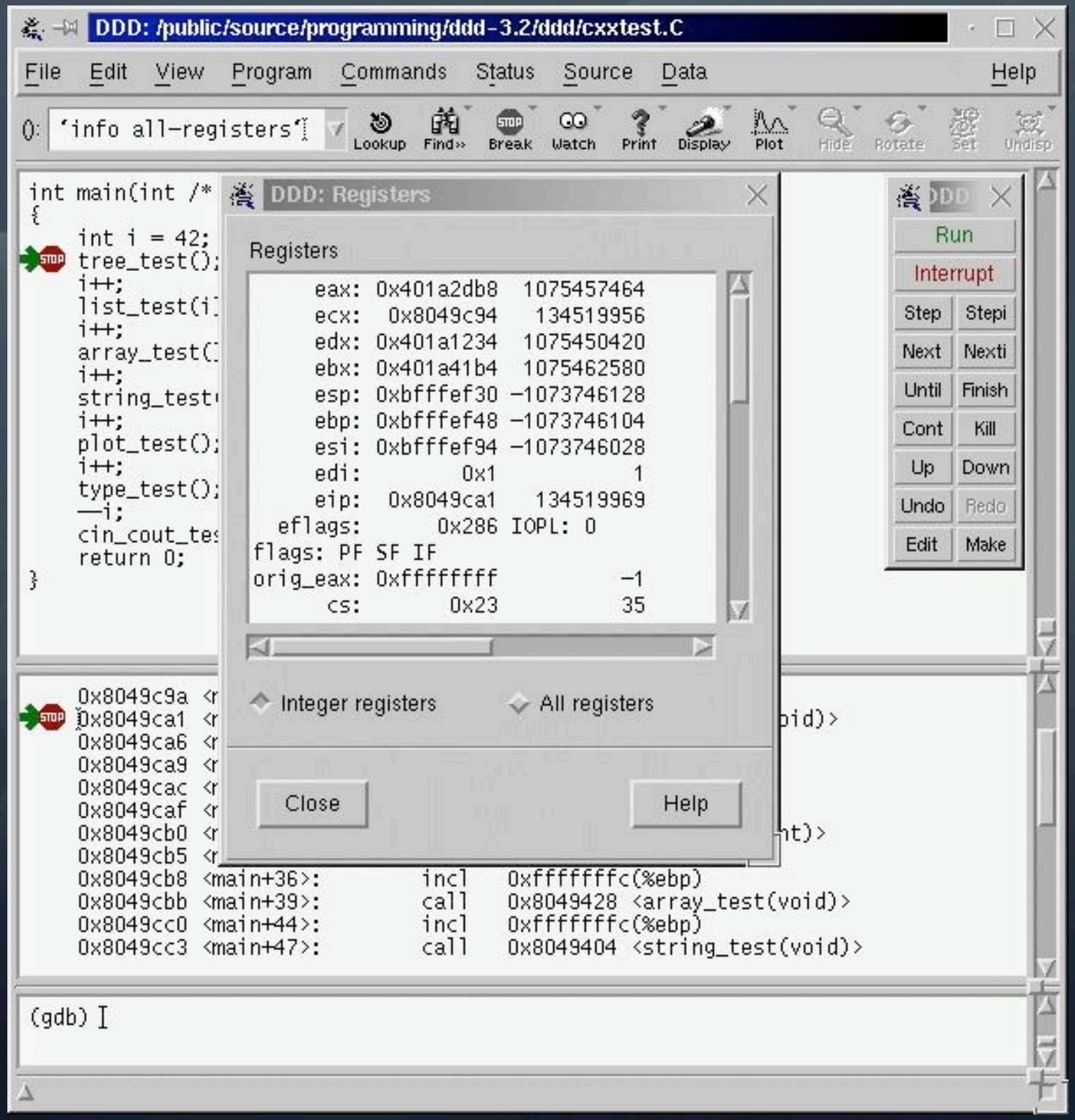
Exkurs

DDD


DDD: /public/source/programming/ddd-3.2.1/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
0: ir
Displays
dr = [...]
ir = [...]
void plot_test()
{
static int ir[100];
DDD: dr
File Edit View Plo
DDD: ir
File Edit View Plot Scale Contour Help
ir
Down
Redo
Plot
Points
Lines
3-D Lines
Points and Lines
Impulses
Dots
Steps
Boxes
Break
(gdb)
(gdb)
(gdb)
Displ

Exkurs

DDD


DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
0: 'info all-registers'
Lookup Find» Break Watch Print Display Plot Hide Rotate Set Undisp
int main(int /*
{
int i = 42;
tree_test();
i++;
list_test(i
i++;
array_test(
i++;
string_test
i++;
plot_test();
i++;
type_test();
--i;
cin_cout_te
return 0;
}
DDD: Registers
Registers
eax: 0x401a2db8 1075457464
ecx: 0x8049c94 134519956
edx: 0x401a1234 1075450420
ebx: 0x401a41b4 1075462580
esp: 0xbfffef30 -1073746128
ebp: 0xbfffef48 -1073746104
esi: 0xbfffef94 -1073746028
edi: 0x1 1
eip: 0x8049ca1 134519969
eflags: 0x286 IOPL: 0
flags: PF SF IF
orig_eax: 0xffffffff -1
cs: 0x23 35
Integer registers
All registers
Close
Help
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
0x8049c9a <r
0x8049ca1 <r
0x8049ca6 <r
0x8049ca9 <r
0x8049cac <r
0x8049caf <r
0x8049cb0 <r
0x8049cb5 <r
0x8049cb8 <main+36>: incl 0xfffffffc(%ebp)
0x8049cbb <main+39>: call 0x8049428 <array_test(void)>
0x8049cc0 <main+44>: incl 0xfffffffc(%ebp)
0x8049cc3 <main+47>: call 0x8049404 <string_test(void)>
(gdb)

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

7. Zusammenfassung

Abstraktionsebenen

- Zusammenhänge (grafisch) und Details (Quellcode)

UML

- nicht nur Entwurf, sondern auch Analyse und Fehlerbehebung
- Fischaugen-Effekt

Inhalt

1. Einleitung
2. Hintergrund
3. Anwendung
4. Grafische Fehlerbehebung
5. Diskussion
6. Verteilte Systeme
7. Zusammenfassung
8. Kommentar

8. Kommentar

Irreführung

- Hat (bisher) nichts mit verteilten Systemen zu tun

Inkonsequent

- Klassen vs. Objekte

8. Kommentar

Programm nicht verfügbar

- Bewertung kaum möglich
- Nicht überprüfbar

Danke!

Diskussion