



Ein Werkzeug zur Überdeckungsmessung für kontrollflussbezogene Testverfahren

Hendrik Seffler

HU Berlin

Entwicklung eines Werkzeugs zur Überdeckungsmessung für kontrollflussbezogene Testverfahren

- ⑥ Analyse der Struktur von Softwaresystemen
- ⑥ nicht-funktionales Testen
- ⑥ Entwurf von Grund auf (Entwicklung durch alle Phasen der Softwareentwicklung)

Warum?

Test im Rahmen dieses Projektes bisher mit *ATOS*

- ⑥ funktionsorientiertes Testen
- ⑥ Testautomatisierung
- ⑥ keine Prüfung der Struktur

Warum?

Fragen, die *ATOS* nicht beantwortet (nicht beantworten kann)

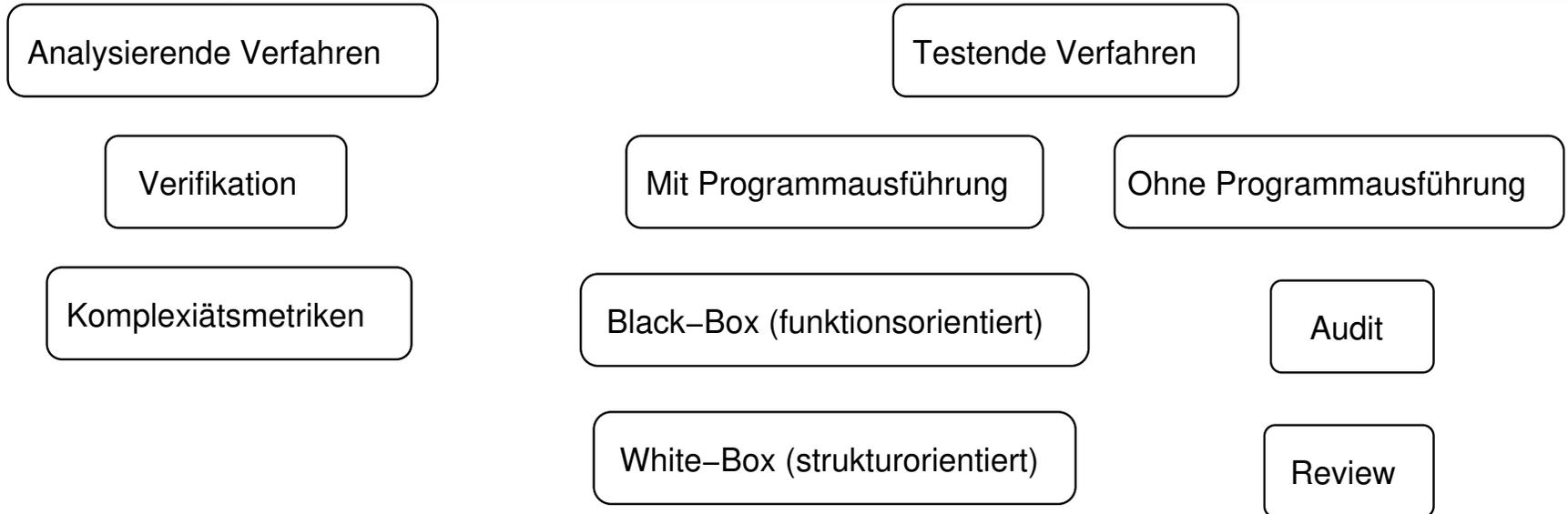
- ⑥ Alles, was Kenntnis des Quellcodes voraussetzt
- ⑥ Wurden alle Anweisungen ausgeführt?
- ⑥ Warum wurden welche Flüsse durch das Programm gewählt?
- ⑥ Welche Teile des Programms sind besonders komplex (fehleranfällig)?

Wie?

Kenntnis der Interna eines Programms notwendig

- ⑥ Parsen des Quellcodes
- ⑥ Darstellung von Kontrollflüssen

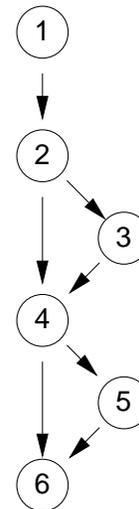
Qualitätssicherung



Was will man wissen?

Ermittlung von Kontrollflüssen

```
void example(boolean a, boolean b)
{
    if (a)
        do();
    if(b)
        doAlso();
}
```



Ziel: Interne Datenstruktur und Visualisierung

Was will man wissen?

Statische ...

- ⑥ zyklomatische Komplexität
- ⑥ essentielle Komplexität

... und dynamische Analyse

- ⑥ Anweisungs-/Zweigüberdeckung
- ⑥ Bedingungsüberdeckung
- ⑥ Pfadtests

Was ist gefordert?

- ⑥ Analyse von Komplexitätsmaßzahlen und Überdeckungsmaßen
- ⑥ Zielsprachen: Java & C++
- ⑥ Darstellung und Auswertung

Was ist notwendig?

- ⑥ Portabilität → Java
- ⑥ Analyse des Quellcodes durch einen Parser → *JavaCC*
- ⑥ *Instrumentierung* des geparsten Codes
- ⑥ Darstellung von Graphen → *JGraph*

Entwurf von mehreren unterschiedlichen Modulen

- ⑥ GUI - Visualisierung
- ⑥ Kontrollflussgraphen
- ⑥ Datenanalyse

Der grundsätzliche Aufbau der GUI basiert auf der Idee von *Sichten*

- ⑥ Quellcode, Kontrollfluss, Struktur, Testfälle, Überdeckung

Aus diesen Sichten werden jeweils komplexere Sichten zusammengesetzt

- ⑥ Komplexität (statische Analyse)
- ⑥ Überdeckung (dynamische Analyse)
- ⑥ Architektur (dynamische Analyse)

Neben der Auswertung existieren noch einige Hilfsfunktionen

- ⑥ Projekterstellung
- ⑥ Instrumentierung von Quellcode
- ⑥ Projektreport
- ⑥ Hilfe/Dokumentation

Die GUI greift auf einen gemeinsamen Datenbestand zu

- ⑥ Kontrollflussgraphen
- ⑥ Testfalldaten (Knotendurchläufe, Bedingungen, Pfade)

Die Aufgabe des Parsers ist es, Quellcode in einer der unterstützten Sprachen einzulesen und in eine sprachunabhängige Struktur abzubilden.

Zugriff über Operationen, die unabhängig von der GUI auf den Daten operieren

- ⑥ GraphMaker
- ⑥ ComplexityChecker
- ⑥ CoverageChecker

Trennung zwischen Darstellung und Auswertung

Design - GraphMaker

Die Klasse `GraphMaker` bereitet Daten für die interne Darstellung und die Darstellung am Bildschirm auf

- ⑥ Visualisierung
- ⑥ Analyse von Komplexität
- ⑥ Finden von Pfaden
- ⑥ Anweisungs- und Kantenüberdeckung

Design - ComplexityChecker

Mit `ComplexityChecker` wird Komplexität von Graphen überprüft

Bereitstellung einer einfachen Schnittstelle auf komplexe Algorithmen

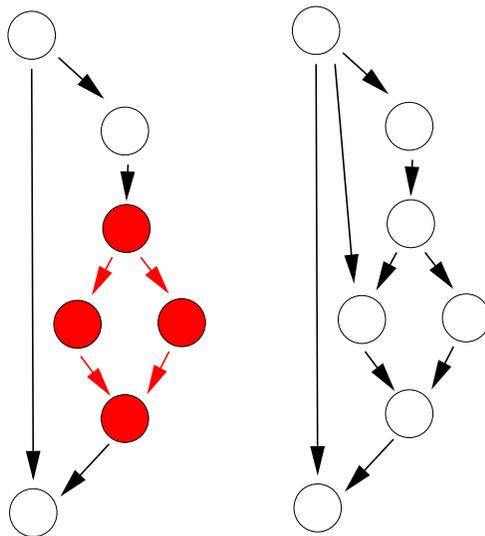
- ⑥ `getCyclomaticComplexity (Function f)`
- ⑥ `getEssentialComplexity (Function f)`

Design - ComplexityChecker

- ⑥ zyklomatische Komplexität

$$v(G) = |E| - |V| + 2$$

- ⑥ essentielle Komplexität
sukzessive Entfernung von *Primes*

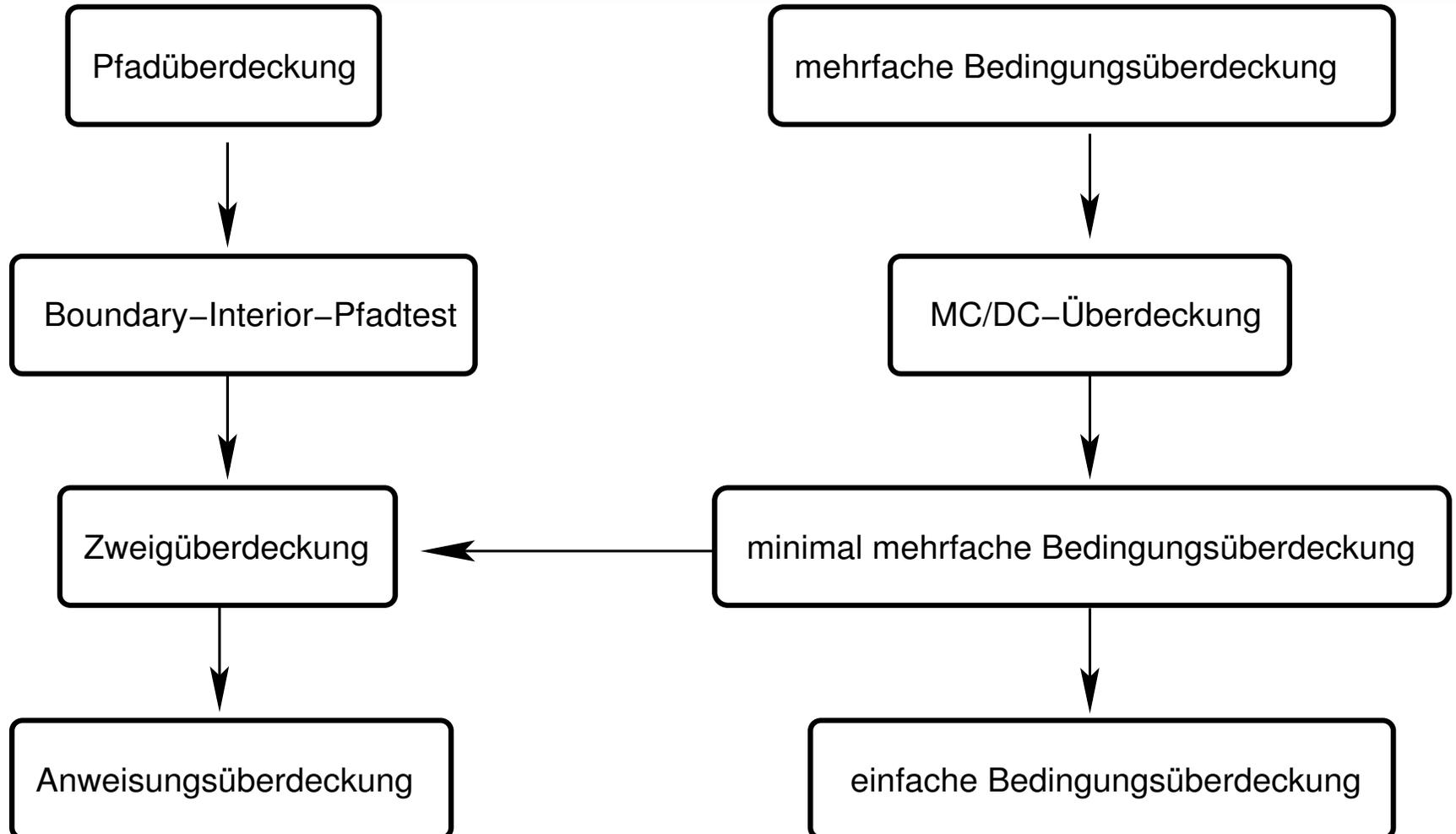


Design - CoverageChecker

Analyse von Überdeckung aus verschiedenen Quellen

- ⑥ Betrachtung von Knoten des Graphen
- ⑥ Analyse der einzelnen Bedingungen
- ⑥ Untersuchung von Pfaden

Design - CoverageChecker



Implementierung - Vorgehen

Iteratives Vorgehen, da Parser und Auswertung parallel entwickelt

- ⑥ Wahl einer Entwicklungsumgebung
- ⑥ Versionsverwaltung
- ⑥ Quellcode-Dokumentation

Implementierung - Größe

Umfang des entwickelten Systems nach Code-Zeilen

- ⑥ GUI: 6746 Zeilen
- ⑥ Kontrollflussgraph: 2100 Zeilen
- ⑥ Projekt-Verwaltung: 2037 Zeilen
- ⑥ Überprüfung: 1668 Zeilen

Sprachunabhängiges Framework zum Arbeiten auf Quellcode verschiedener Sprachen. Einheitliche Schnittstelle und Zugriff auf Programmstruktur über Kontrollflussgraphen und Bedingungen.

- ⑥ Überdeckungsmaße
- ⑥ Komplexität

Auf dieser Basis sind weitere Erweiterungen denkbar

- ⑥ weitere Überdeckungsmaße
- ⑥ Aufrufgraphen
- ⑥ Profiling

Demonstration

The screenshot displays the SBTWAN (Software-Based Test Workbench) interface for analyzing code coverage. The window title is "SBTWAN - /home/hendrik/test/dokuTest.spr".

Left Panel (Überdeckung): Shows a project tree with folders for "Foo" and "Test". The file "werteZiffernfolgeAus(6,1)" is selected.

Center Panel (Control Flow Graph): A graph showing the execution flow of the code. Nodes are represented by green and red rectangles, connected by arrows. Some paths are highlighted in red, indicating execution by a specific test case.

Right Panel (Code): Displays the source code for the method `werteZiffernfolgeAus`. The code is as follows:

```
public double werteZiffernfolgeAus (String inZiffernString){
    double wert = 0.0;
    double genauigkeit = 1.0;
    String woBinIch = "VorDemKomma";
    boolean fehlerfrei = true;
    int position =1;

    while (position <= inZiffernString.length() & fehlerfrei){
        String zchn = inZiffernString.substring(position-1, position);
        if (zchn.matches("\\d")){
            if (woBinIch.equals("NachDemKomma")){
                genauigkeit = genauigkeit / 10.0;
                wert = 10.0 * wert + Double.parseDouble(zchn);
            }
            else if ((boolean)zchn.equals("") & woBinIch.equals("VorDemKomma")){
                woBinIch = "NachDemKomma";
            }
            else{
                fehlerfrei=false;
            }
            position++;
        }
        double result;
        if (!fehlerfrei || inZiffernString.length()==0 ||
            ((woBinIch.equals("NachDemKomma") & inZiffernString.length()==1)){
            result= -1.0;
        }
        else{
            result = wert * genauigkeit;
        }
        return result;
    }
}
```

Far Right Panel (Testfälle): A list of test cases, including "TEST_9" and "TEST_92".

Bottom Panel: Shows coverage statistics for various test cases:

- CO : :Nein (86%)
- C1 : :Nein (84%)
- BI : :Nein (33%)
- C2 : :Nein (30%)
- C3 : :Nein (28%)
- MMDC : :Nein (68%)
- MCDC : :Nein (30%)

The status bar at the bottom indicates "Überdeckungsansicht geöffnet".

Fragen?

⑥ Fragen?