Bug Hunting mit statischer Codeanalyse

Daniel Fischer
Hochschule Offenburg
Badstraße 24
77652 Offenburg
daniel.fischer@hs-offenburg.de
Tel.: 0781-205-148

Andreas Behr
Verifysoft Technology GmbH
In der Spöck 10-12
77656 Offenburg
behr@verifysoft.com
Tel.:0781-127-81189

Roland Bär
Verifysoft Technology GmbH
In der Spöck 10-12
77656 Offenburg
baer@verifysoft.com
Tel.:0781-127-81189

Hochschule Offenburg
University of Applied Sciences

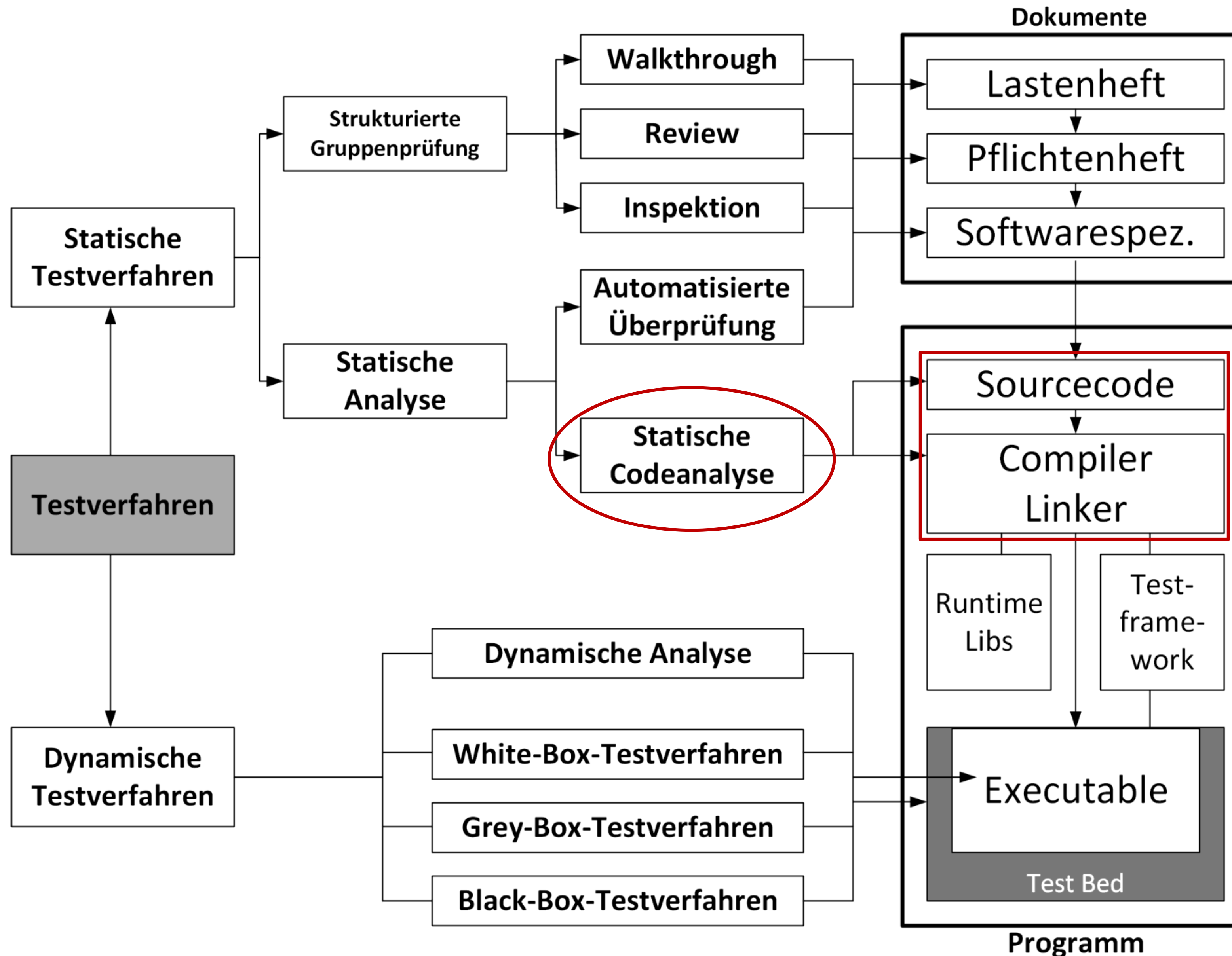
Verifysoft
TECHNOLOGY

Inhalt

1. Grundlagen
2. Statische Codeanalyse
   - Intraprozedural
   - Interprozedural
3. Nutzen der statischen Analyse
4. Exemplarische Beispiele
5. Zusammenfassung

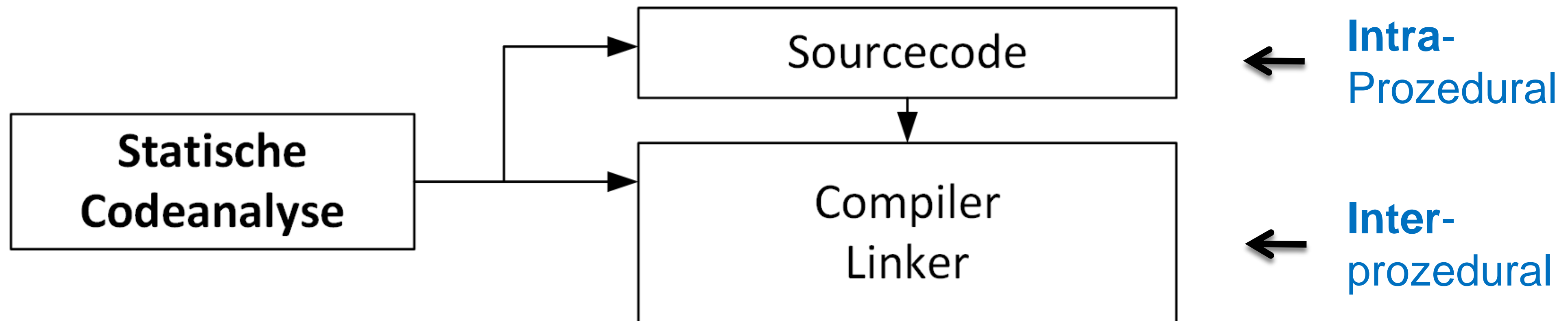
Grundlagen - Testverfahren

Embedded Software Engineering Kongress



Grundlagen – Statische Codeanalyse

Embedded Software Engineering Kongress



- Automatischer Review des Source-Codes
- Compiler/Linker wird benötigt für interprozedurale Analyse
- Programm muss jedoch nicht ausgeführt werden

Statische Codeanalyse

Embedded Software Engineering Kongress

Varianten der statischen Codeanalyse

- Konformität gegenüber Coding Guidelines
- Rules Checker (MISRA, CERT)
- Kontroll- und Datenflussanalyse
- Analyse und Aufdeckung von Concurrency Fehlern (Race Conditions,…)
- Analyse zur WCET Bestimmung (Worst Case Execution Time)
- Komplexitätsmetriken (Lines of Code, Zyklomatische Komplexität, …)
- Architekturanalyse (u.a. Nachweis der Softwareerosion)

Statische Codeanalyse

Varianten der statischen Codeanalyse

Intraprozedurale Analyse

- Lokaler Blick auf Funktion
- Findet einfache Fehler

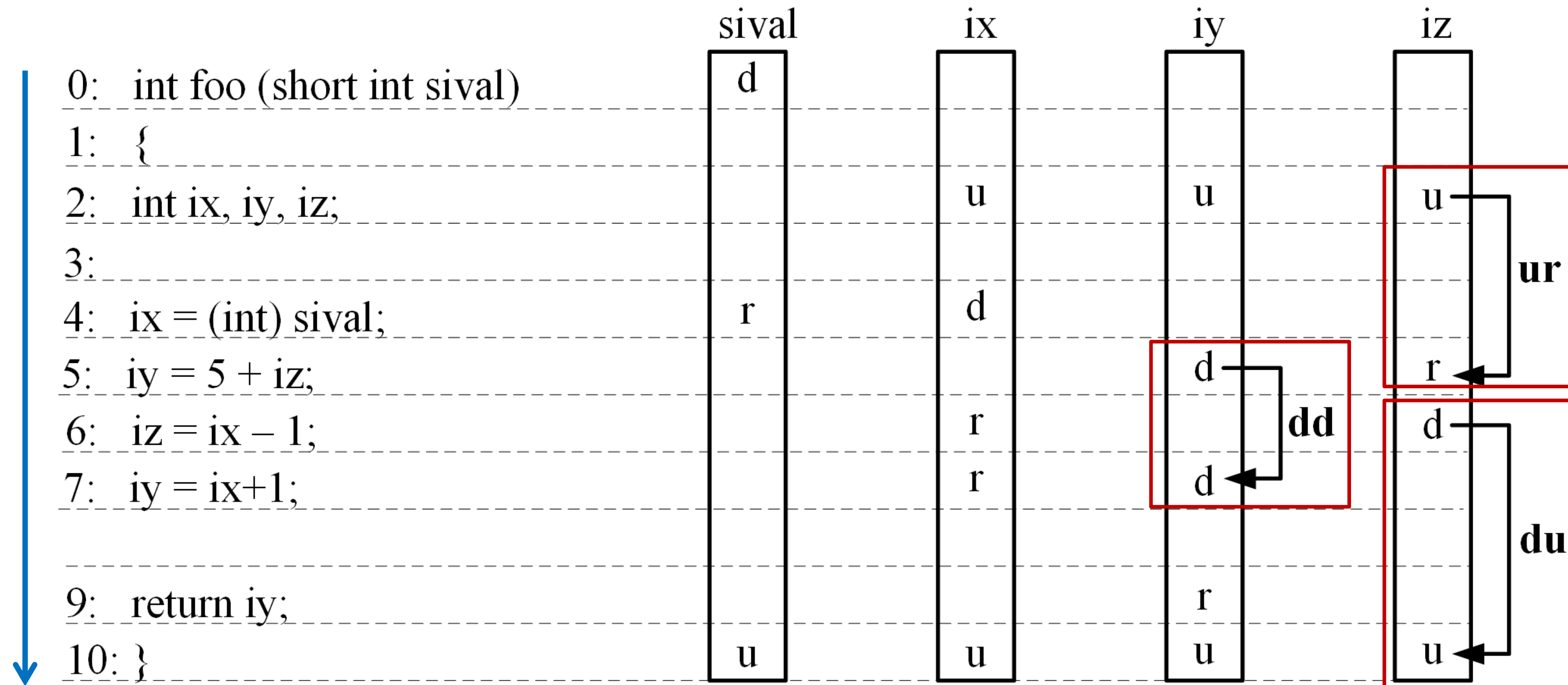
Interprozedurale Analyse

- Abstraktes Modell des Programms
- Vollständige Kontroll- und Datenflussanalyse modulübergreifend
- Findet nicht offensichtliche Fehler mit komplexem Zusammenhang

Datenflussanalyse – Anomalien im Datenfluss

Jede lokale und globale Variable wird spezifiziert

- u undefiniert
- d definiert
- r referenced



Zustände der Variablen über Zeit

Statische Codeanalyse - Datenflussanalyse

Embedded Software Engineering Kongress

Kombination von Daten- **und** Kontrollflussanalyse notwendig

```
void goo (int i)
{
    int *p;
    if (i==3)
        p = (int*) malloc(10*sizeof(int));
    aSubroutine(p);
    if (i==10)
        free(p);
}
```

- Kontrollflussanalyse notwendig
- Fehler in großen Funktionen leicht zu übersehen
- Interprozedurale Analyse um Fehler in Subroutinen zu finden (aSubroutine)
- Call by reference Problematik

Nutzen Statischer Codeanalyse

Grundlegender Vorteil statischer Analyse

- Keine Testfälle und Prozeduren zu implementieren
- Geringer Installationsaufwand
- Tools funktionieren meist „out of the box“

→Brauchbare Ergebnisse ohne Aufwand

→Ergebnisse variieren stark abhängig von eingesetzter Testsoftware (Qualitativ)

Nutzen Statischer Codeanalyse

Studie basierend auf Entwicklung von vier Betriebssystemen zeigt:

- 14.8 - 24.4% aller Postrelease Bugfixes fügen einen neuen Fehler ein
- 39% der Concurrency Bugfixes sind inkorrekt
- Bugfixes werden häufig von Entwicklern gemacht, die den Code nicht kennen. Deren Bugfixes fügen häufig neue Bugs ein

In der Wartungsphase finden meist keine umfassenden Nachtests statt
→ Vermeintlicher Bugfix fügt neue Fehler ein!

[YZPS11] How do Fixes Become Bugs?, Proceedings, 19th ACM SIGSOFT symposium, Hungary, 2011

Beispiele – Buffer Overrun

Standardsituationen unter der Lupe mit

Cppcheck
- Open Source

PC-lint
- Kommerziell
- MISRA

CodeSonar
- Kommerziell
- Vollständige Interproz. Kontroll-/Datenflussanalyse

Augenmerk bei der Auswahl des Tools
- Compiler / Generelle Kompatibilität
- Projektgröße
- „Innere Werte“ – Suchtiefe, …
- Anspruch
- Budget

Grober Richtwert:
„You get what you pay for“

Beispiele – Buffer Overrun

Embedded Software Engineering Kongress

Standard Situation Buffer Overrun 1

```
char buffer[10];

char ch = buffer[10];
```

Ergebnis

Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Buffer Overrun

Embedded Software Engineering Kongress

Standard Situation Buffer Overrun 2

```
char buffer[10];

for (int i = 0; i <= 10; i++)
  buffer[i] = ´b´;
```

<u>Ergebnis</u>

Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Buffer Overrun

Embedded Software Engineering Kongress

Standard Situation Buffer Overrun 3

```
char buffer[10];
char* pc;
int j = 0;

pc = buffer;

for (int i = 0; i <= 10; i++)
    *pc++ = ´c´;

j = 0;
for (int i = 0; i < 6; i++)
    buffer[j++] = ´d´;
```

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Buffer Overrun

Embedded Software Engineering Kongress

Standard Situation Buffer Overrun 4

```
char buffer[10];
char* pc;
int j = 0;
pc = buffer;

j = 0;
for (int i = 0; i < 6; i++)
    buffer[j++] = ´d´;

for (int i = 0; i < 6; i++)
    buffer[j++] = ´e´;
```

Ergebnis	
Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Not Initialized

Embedded Software Engineering Kongress

Standard Situation: Not Initialized 1

```
int notinit (int k)
{
   char* pc1;
   int iret;

   *pc1 = ´a´;
}
```

Ergebnis

Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Not Initialized

Embedded Software Engineering Kongress

Standard Situation: Not Initialized 2

```
int notinit (int k)
{
    char* pc1;
    int iret;

    if ( k > 42 )
        iret = 1;

    return iret;
}
```

<u>Ergebnis</u>

Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Dead Code

Standard Situation: Dead Code

```
void dead_code ( void )
{
   if ( x )
   {
      if ( y > 10 && !x )
      {
         x++;        /*dead code*/
         return;
      }
   }
}
```

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Memory Leak

Embedded Software Engineering Kongress

Standard Situation: Memory Leak

```
int simple_leak( void )
{
   char* p = (char* malloc(12));
   if ( !p )
      return 0;

   if ( !some_function() )
   {
      free(p);
      return -1;
   }

   if ( !some_other_function() )
      return -2; /*MEM Leak*/

   free(p);
   return 1;
}
```

Ergebnis	
Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Use After Free

Standard Situation: Use After Free 1

```
void simple_use_after_free( void )
{
    char* pc1;

    pc1 = (char*) malloc(10*sizeof(char));

    if ( pc1 )
    {
        pc1[0] = ´a´;
        free(pc1);
        pc1[0] = ´b´;
    }
}
```

Ergebnis

Cppcheck	gefunden
PC-lint	gefunden
CodeSonar	gefunden

Beispiele – Use After Free

Standard Situation: Use After Free 2

```
void simple_use_after_free( void )
{
    char* pc2;
    char* pc3;

    pc2 = (char*) malloc(10*sizeof(char));

    if ( pc2 )
    {
        pc3 = pc2;
        free(pc2);
        pc3[0] = ´b´;
    }
}
```

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Zusammenfassung

Gesamtübersicht der Ergebnisse

Exemplarischer Fehler	Cppcheck	PC-lint	CodeSonar
Buffer Overrun 1	ja	ja	ja
Buffer Overrun 2	ja	ja	ja
Buffer Overrun 3	nein	nein	ja
Buffer Overrun 4	nein	nein	ja
Not Initialized 1	ja	ja	ja
Not Initialized 2	ja	ja	ja
Dead Code	nein	nein	ja
Memory Leak	ja	ja	ja
Use After Free 1	ja	ja	ja
Use After Free 2	nein	nein	ja

Beispiele – Interprozedural

Embedded Software Engineering Kongress

Standard Situation: Double Free

```
void test_driver( void )
{
   int* pi1 = (int*) malloc(..);

   if(pi1)
      test_buffer_overrun(pi1);

   if(pi1)
      free(pi1);
}
```

Modul1.c

```
void test_buffer_overrun( int* p)
{
   if( p )
      free(p);
}
```

Modul2.c

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Interprozedural

Standard Situation: Buffer Overrun (static)

```
void test_driver( void )
{
    int* pi1 = (int*) malloc(..);
    int test[4];

    if(pi1)
        test_buffer_overrun(pi1, test);

    if(pi1)
        free(p1);
}
```

Modul1.c

```
void test_buffer_overrun( int* p, int
test[])
{
    test[4] = 3;
    if( p )
        free(p);
}
```

Modul2.c

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Interprozedural

Embedded Software Engineering Kongress

Standard Situation: Buffer Overrun (dynamic)

```
void test_driver( void )
{
   int* pi1 = (int*) malloc(..);
   int test[4];

   if(pi1)
      test_buffer_overrun(pi1, test);

   if(pi1)
      free(pi1);
}
```

Modul1.c

```
void test_buffer_overrun( int* p, int
test[])
{
   int* pdummy = p + 1;
   *pdummy = 42;
}
```

Modul2.c

<u>Ergebnis</u>

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Interprozedural

Standard Situation: Null Pointer Dereference

```
void test_driver( void )
{
   int* pi1 = NULL;
   test_deref(pi1);
}
```

Modul1.c

```
void test_deref( int* p )
{
   *p = 55;
}
```

Modul2.c

<u>Ergebnis</u>

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Interprozedural

Embedded Software Engineering Kongress

Standard Situation: Memory Leak

```
void test_driver( void )
{
    int* pi1 = NULL;
    test_free(pi1, 20);
}
```

Modul1.c

```
void test_free( int* p, int x )
{
    if( p && x < 10)
        free(p);
}
```

Modul2.c

Ergebnis

Cppcheck	nicht gefunden
PC-lint	nicht gefunden
CodeSonar	gefunden

Beispiele – Zusammenfassung

Abschliessende Gesamtübersicht

Exemplarischer Fehler	Cppcheck	PC-lint	CodeSonar
Double Free	nein	nein	ja
Buffer Overrun (static)	nein	nein	ja
Buffer Overrun (dynamic)	nein	nein	ja
Nullpointer Dereference	nein	nein	ja
Memory Leak	nein	nein	ja

→Deutliche Unterschiede beim Auffinden von Fehlern, die über Modulgrenzen hinweg auftreten.

→ Fehler in komplexem Zusammenspiel mehrerer Module sind zeitaufwändig und kostenintensiv.

Zusammenfassung

Embedded Software Engineering Kongress

- Statische Codeanalyse als effizientes Tool zur aktiven Fehlerbehebung

- Unterschiedliche Tools mit unterschiedlicher Performanz

  Interprozedural

  Intraprozedural

- Semantische Fehler können nicht entdeckt werden

  → Das Programm läuft fehlerfrei vs. Das Programm tut was es soll

Empfehlung

Evaluieren Sie unterschiedliche statische Codeanalyse Tools

Achten Sie auf die Inneren Werte: Suchtiefe, Intra/Interprozedural, …

Fragen

Embedded Software Engineering Kongress

Daniel Fischer
daniel.fischer@hs-offenburg.de

Roland Bär
baer@verifysoft.com

Andreas Behr
behr@verifysoft.com

Oder besuchen Sie uns am
Messestand!