

Code-Coverage auf Embedded Systems

Daniel Fischer

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

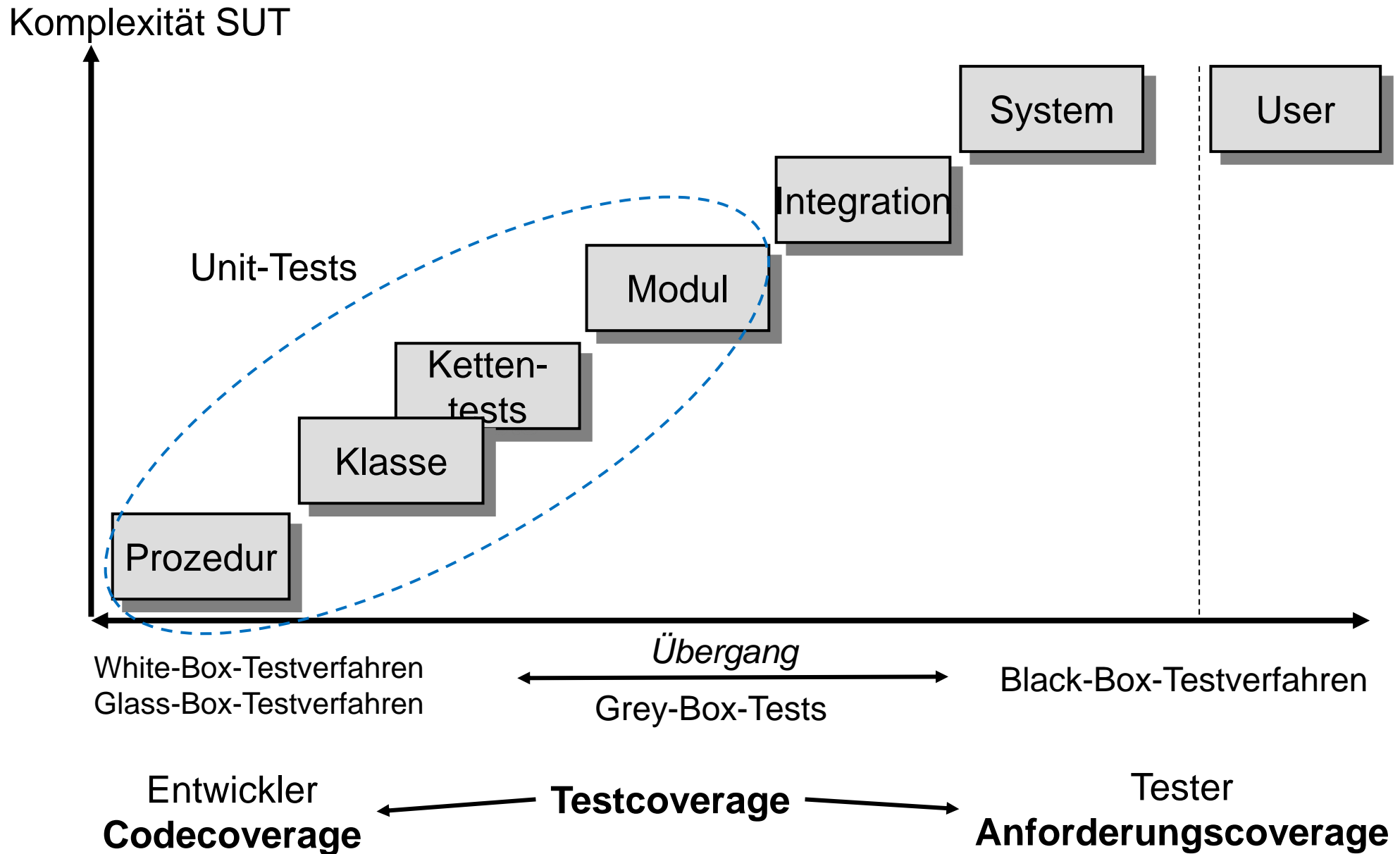
Andreas Behr

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

1. Grundlagen
2. Coveragestufen
3. Normen
4. Instrumentierung
5. Herausforderungen kleine Targets
6. Beispiel



Ursache-Wirkungsgraph

Classification Tree
Method (CTM)

Realtime
Testing

Rare Event Testing

Last Tests

Recovery Tests
Stress Tests

**Kontrollflussorientiertes
Testen**

Statistisches Testen

Äquivalenzklassen
Mehrdimensionale Äquivalenzklassen
Grenzwertanalyse
Test besonderer Werte
Informelle Tests
Smoke Tests
Basis

Advanced

Back-to-Back Testing

CRUD

Rare Event Testing

Mutation Testing

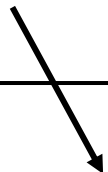
Zufallsgesteuerter
Test

Monkeytest

Fuzzing (Fuzz Testing)

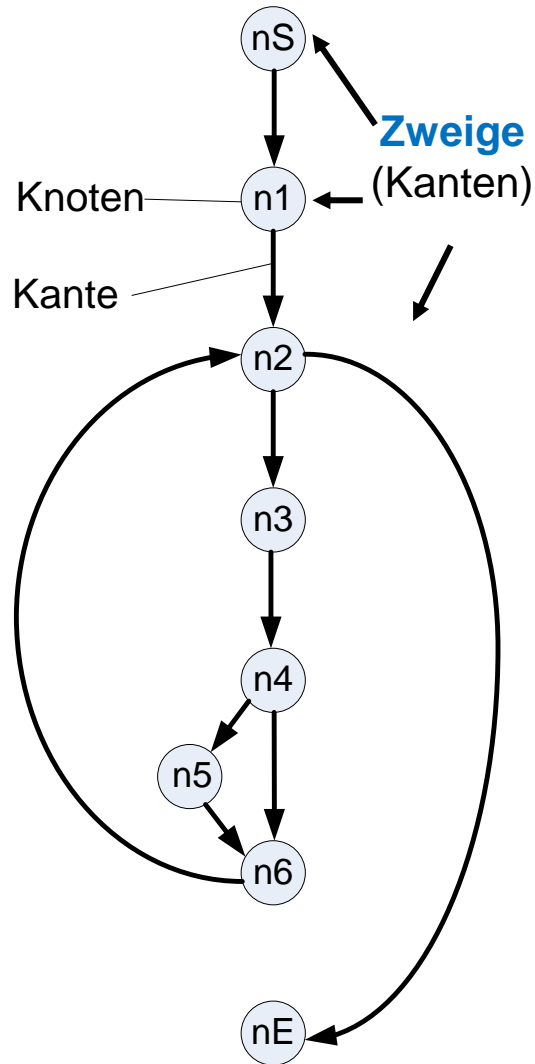
Evolutionäres Testen

Pairwise Testing

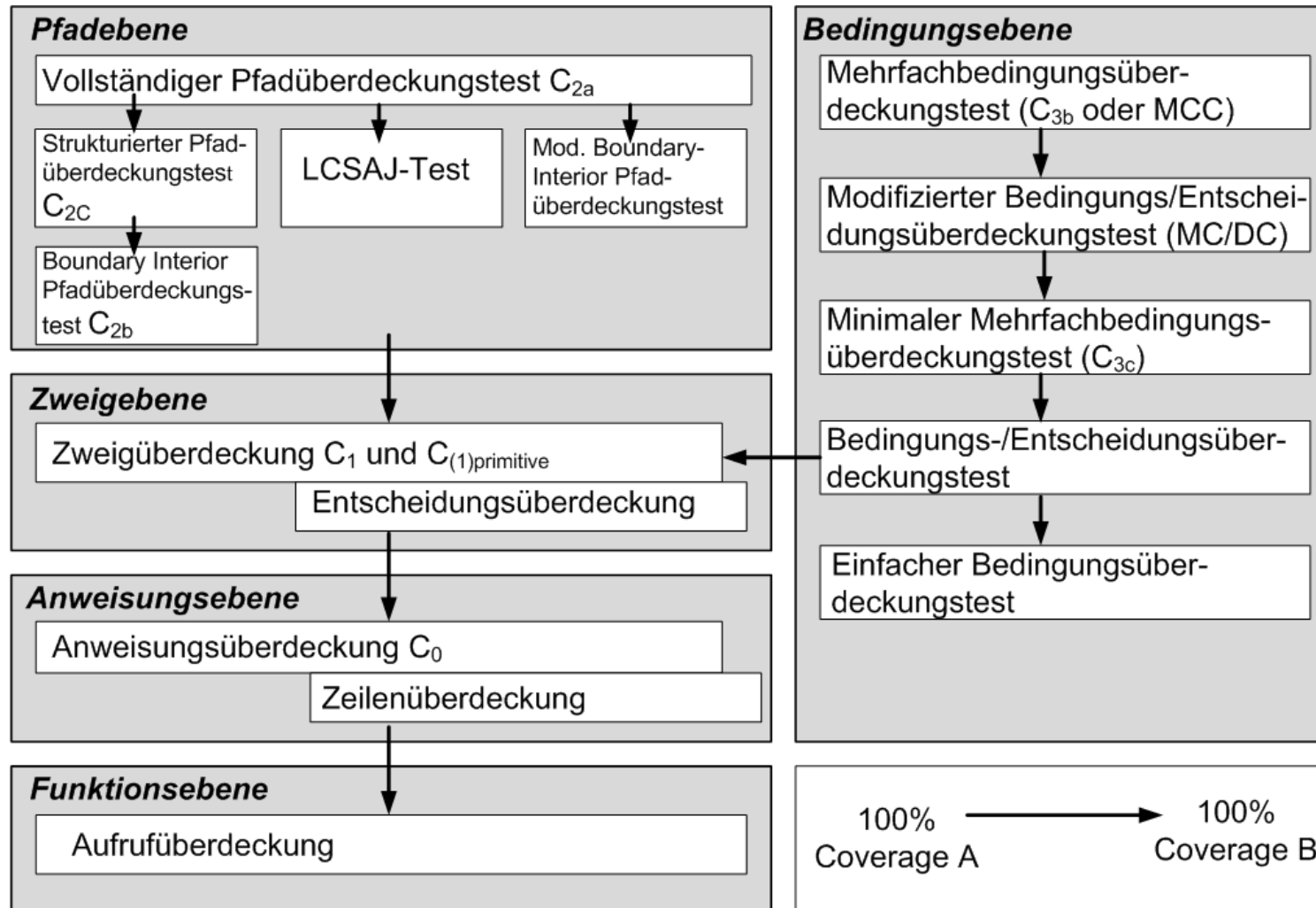


Etablierte Testtechnik für sicherheitskritische Embedded Systems
Testendekriterium (White-Box-Testverfahren)
Notwendig für die Erfüllung verschiedener Normen

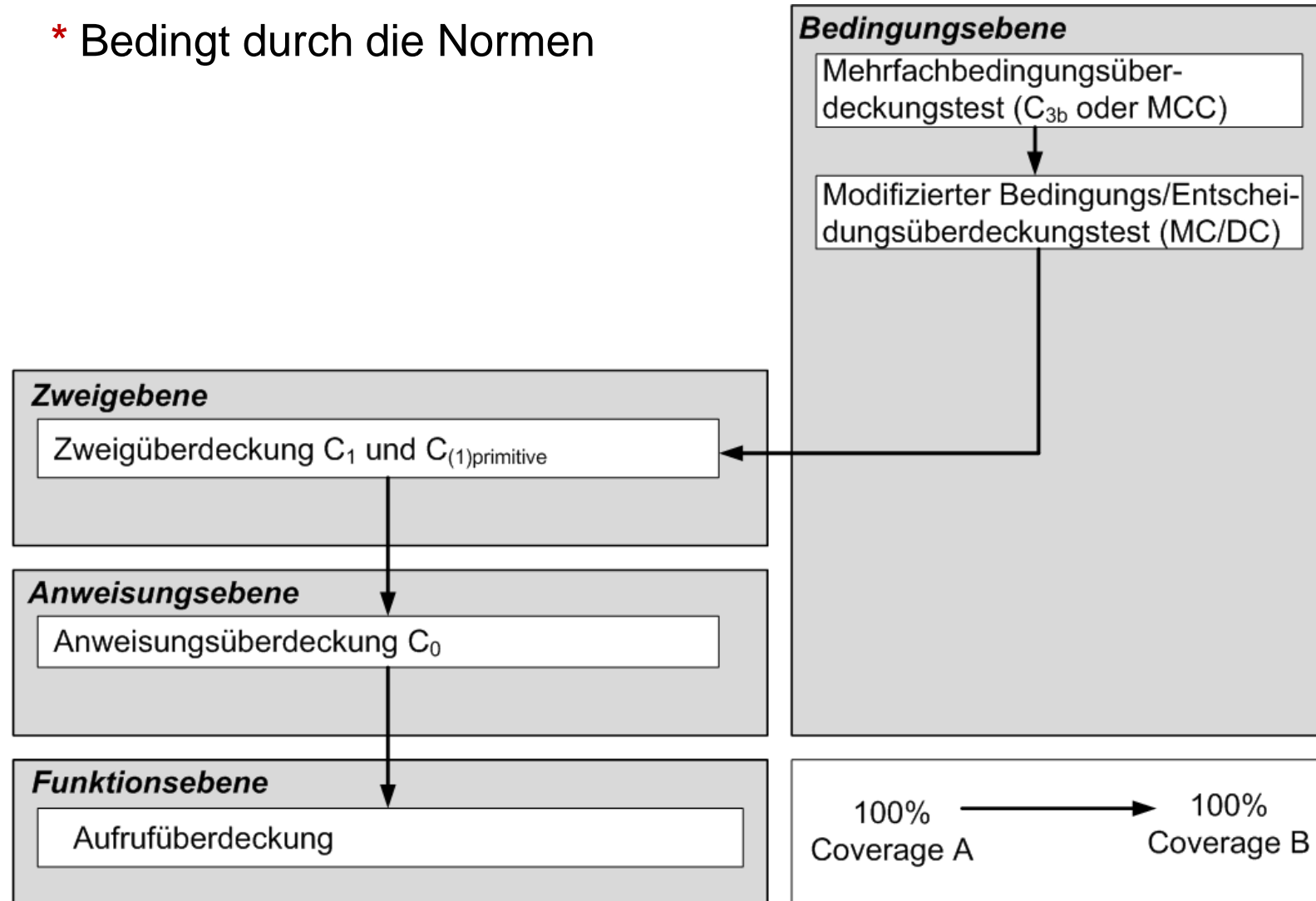
Pfade



```
void ZaehleZchn (int& VokalAnzahl, int& GesamtAnzahl) ← Funktionen  
{  
  unsigned char Zchn;  
  Zchn = getch(); ← Anweisungen (Knoten)  
  while ((Zchn>='A')&&(Zchn<='Z')&&(GesamtAnzahl<INT_MAX))  
  {  
    GesamtAnzahl = GesamtAnzahl + 1; ← Bedingungen  
    if ((Zchn=='A')||((Zchn=='E')||((Zchn=='I')||((Zchn=='O')||((Zchn=='U'))))  
    {  
      VokalAnzahl = VokalAnzahl + 1;  
    }  
    Zchn = getch();  
  }  
}
```



* Bedingt durch die Normen



```
int goo( int a, int b, int c)
{
    int x;

    if (((a>0) || (b>0)) && (c>0))
    {
        x = 1;
    }
    else
    {
        x = 0;
    }

    return x;
}
```

```
UCUNIT__TESTCASE_BEGIN("Function Coverage");
UCUNIT__CHECKLIST_BEGIN(UCUNIT__ACTION_WARNING);
UCUNIT__CHECK_IS_EQUAL(1, goo(1,0,1));
UCUNIT__CHECKLIST_END();
UCUNIT__TESTCASE_END("Function Coverage");
```

```
=====
TESTCASE:Function Coverage:BEGIN
-----
CHECK: Line 22:IsEqual<1, goo<1,0,1>>:PASSED
CHECK: Line 23:Checklist<>:PASSED
-----
TESTCASE:Function Coverage:PASSED
=====

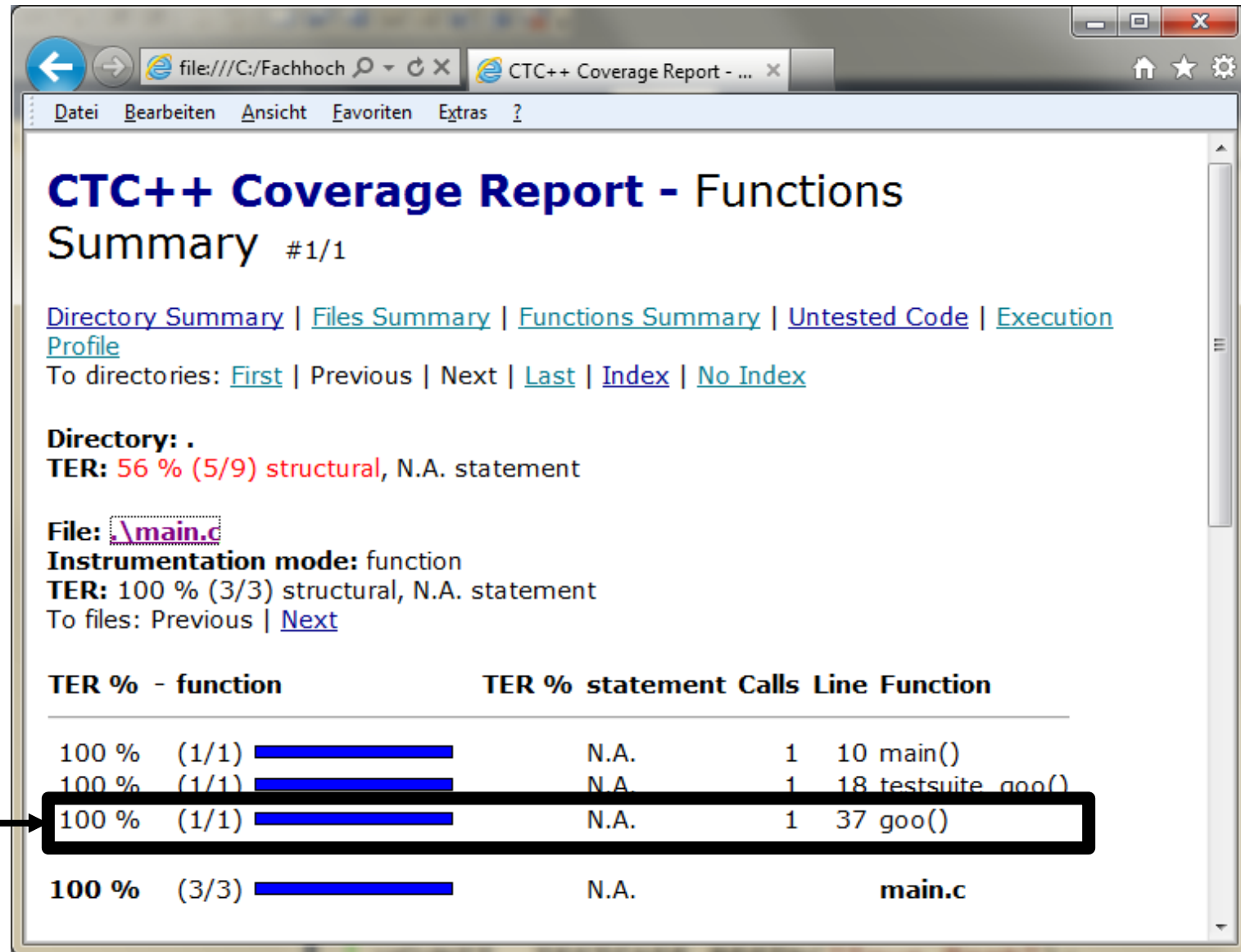
*****
Testcases: failed: 0
           passed: 1
Checks:    failed: 0
           passed: 1
*****
_
```


TER

Test
Effectiveness
Ratio

TER entspricht
der entsprechend
gewählten Coverage ...

Für die zu testende
Funktion goo ()
wurde eine
Aufrufüberdeckung
von 100% erreicht.



```
int goo( int a, int b, int c)
{
    int x;

    if (((a>0) || (b>0)) && (c>0))
    {
        x = 1;
    }
    else
    {
        x = 0;
    }

    return x;
}
```

```
UCUNIT__TESTCASE_BEGIN("Statement Coverage");
UCUNIT__CHECKLIST_BEGIN(UCUNIT__ACTION_WARNING);
UCUNIT__CHECK_IS_EQUAL(1, goo(1,0,1));
UCUNIT__CHECKLIST_END();
UCUNIT__TESTCASE_END("Statement Coverage");
```

```
=====
TESTCASE:Statement Coverage:BEGIN
-----
CHECK: Line 22:IsEqual<1, goo<1,0,1>>:PASSED
CHECK: Line 23:Checklist<>:PASSED
-----
TESTCASE:Statement Coverage:PASSED
=====

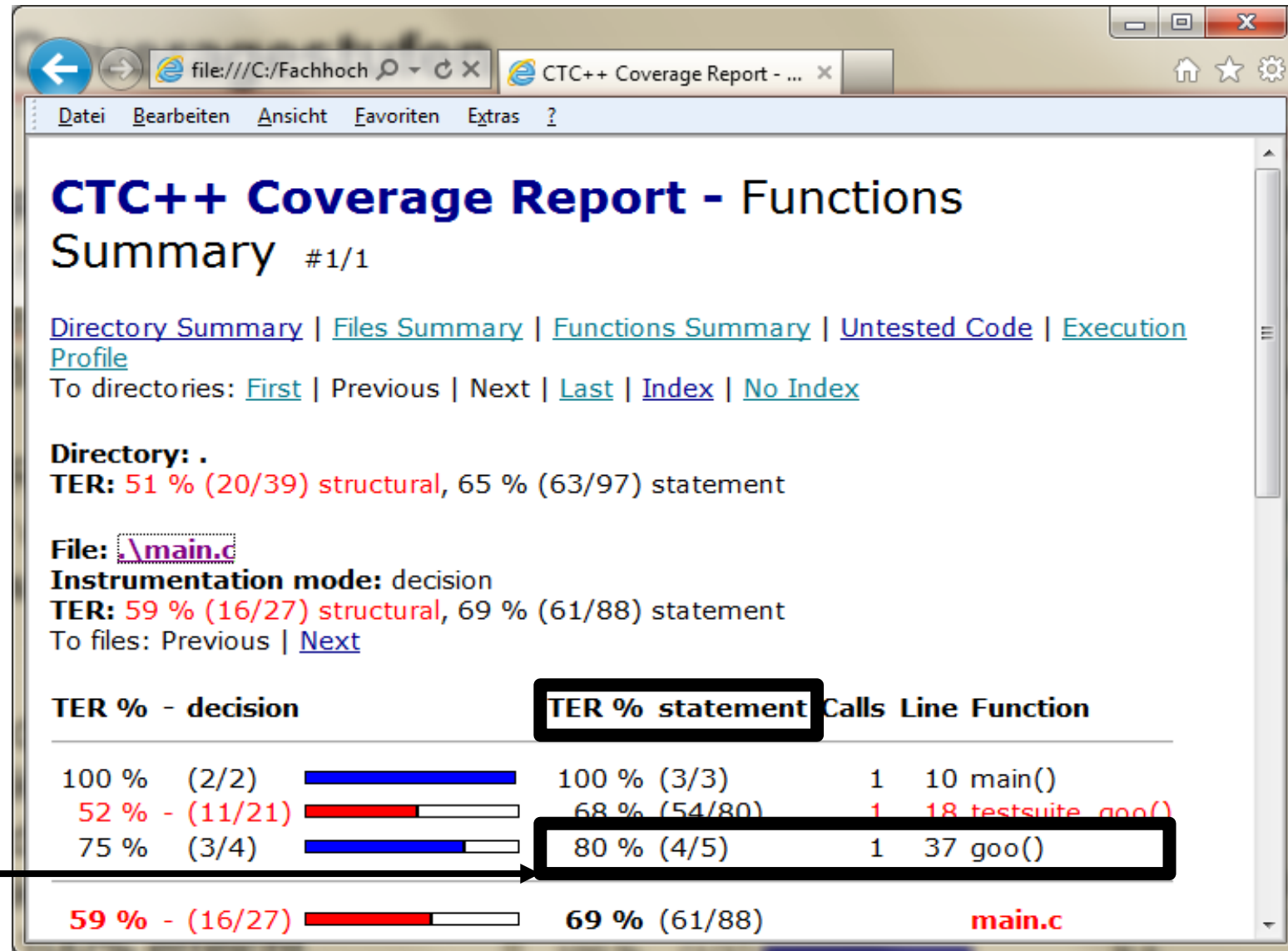
*****
Testcases: failed: 0
           passed: 1
Checks:    failed: 0
           passed: 1
*****
```

TER

Test
Effectiveness
Ratio

TER entspricht
der entsprechend
gewählten Coverage ...

Für die zu testende
Funktion goo ()
wurde nur eine
Anweisungs-
überdeckung
von 80% (4/5) erreicht.



```
int goo( int a, int b, int c)
{
    int x;

    if (((a>0) || (b>0)) && (c>0))
    {
        x = 1;
    }
    else
    {
        x = 0;
    }

    return x;
}
```

```
UCUNIT__TESTCASE_BEGIN("Branch Coverage");
UCUNIT__CHECKLIST_BEGIN(UCUNIT__ACTION_WARNING);
UCUNIT__CHECK_IS_EQUAL(1, goo(1,0,1));
UCUNIT__CHECK_IS_EQUAL(0, goo(1,0,0));
UCUNIT__CHECKLIST_END();
UCUNIT__TESTCASE_END("Branch Coverage");
```

```
=====
TESTCASE:Branch Coverage:BEGIN
-----
CHECK: Line 22:IsEqual<1, goo<1,0,1>>:PASSED
CHECK: Line 23:IsEqual<0, goo<1,0,0>>:PASSED
CHECK: Line 24:Checklist<>:PASSED
-----
TESTCASE:Branch Coverage:PASSED
=====

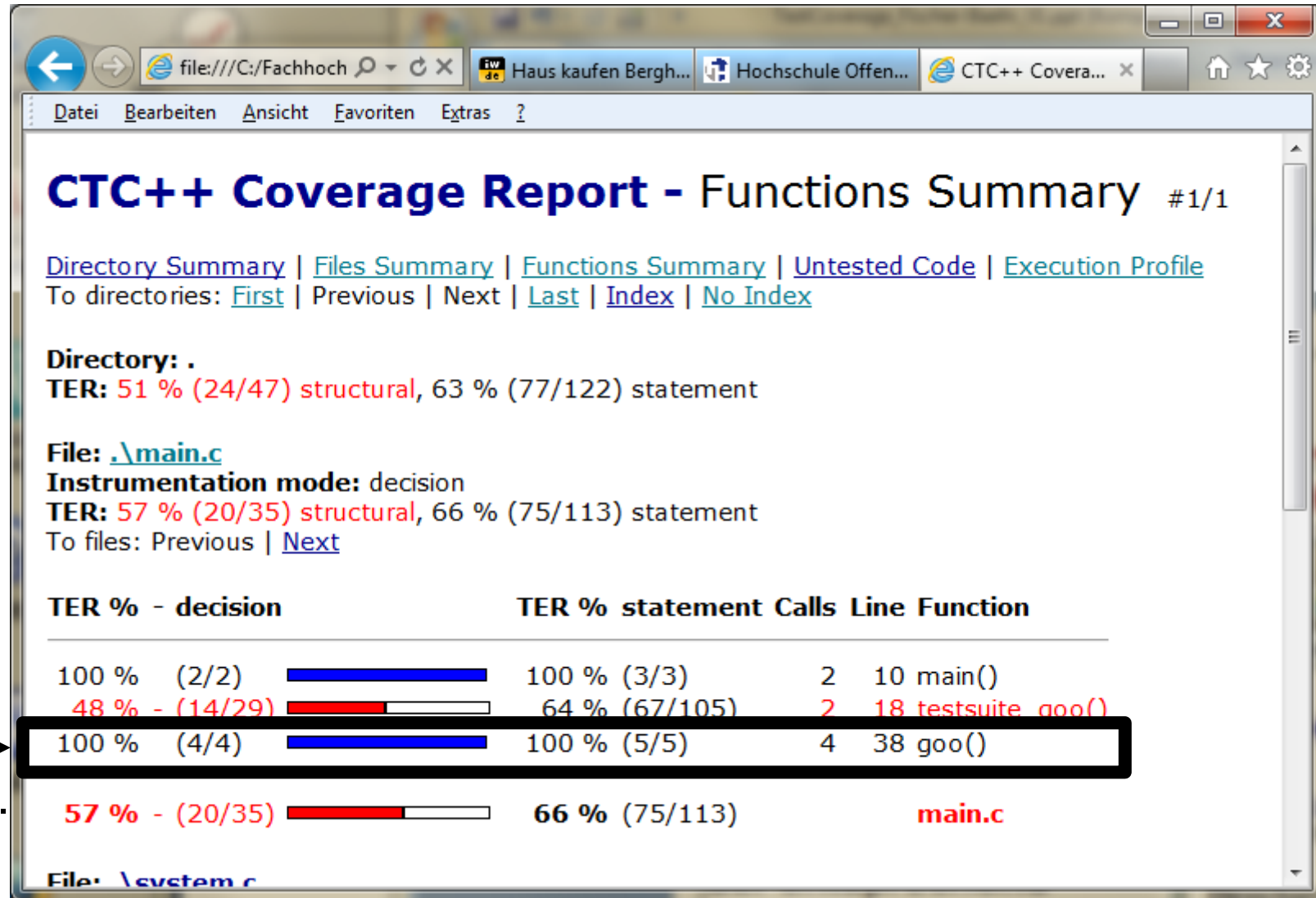
*****
Testcases: failed: 0
           passed: 1
Checks:    failed: 0
           passed: 2
*****
-
```

TER

Test
Effectiveness
Ratio

TER entspricht
der entsprechend
gewählten Coverage ...

Für die zu testende
Funktion goo ()
wurde eine
Zweigüberdeckung-
überdeckung →
von 100% (4/4) erreicht.



```
int goo( int a, int b, int c)
{
  int x;

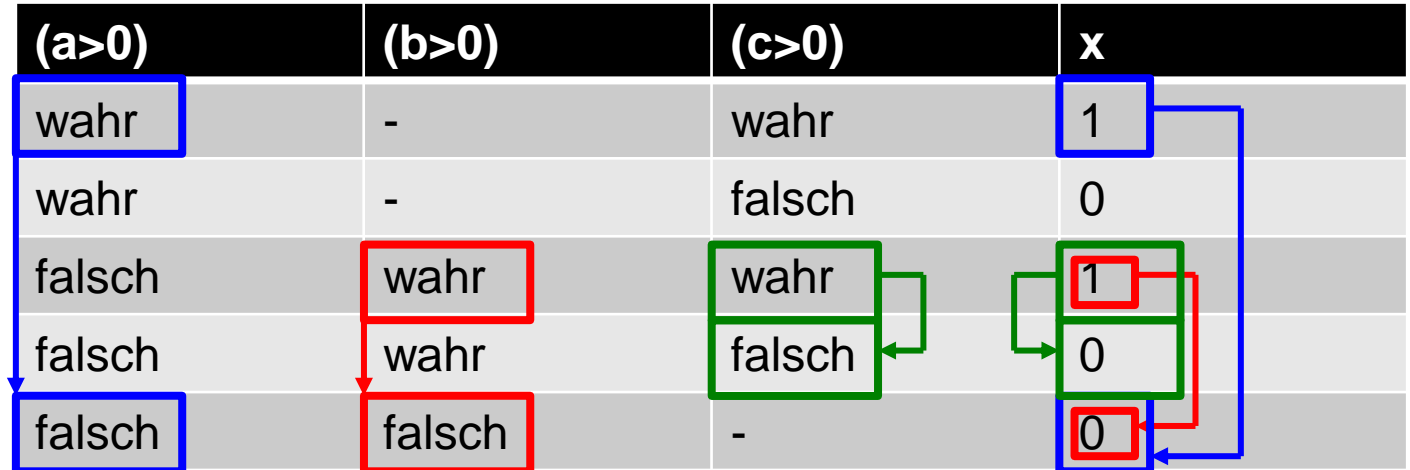
  if (((a>0) || (b>0)) && (c>0))
  {
    x = 1;
  }
  else
  {
    x = 0;
  }

  return x;
}
```

DO-178B Definition:

„Every decision has taken all possible outcomes at least once, and every condition in a decision is shown to independently affect that decision's outcome.“

(a>0)	(b>0)	(c>0)	x
wahr	-	wahr	1
wahr	-	falsch	0
falsch	wahr	wahr	1
falsch	wahr	falsch	0
falsch	falsch	-	0



Unvollständige Evaluation in C/C++

-wird nicht ausgewertet, kann wahr oder falsch sein

n+1 Tests sind notwendig, n := Anzahl atomarer Bedingungen


```
int goo( int a, int b, int c)
{
    int x;

    if (((a>0) || (b>0)) && (c>0))
    {
        x = 1;
    }
    else
    {
        x = 0;
    }

    return x;
}
```

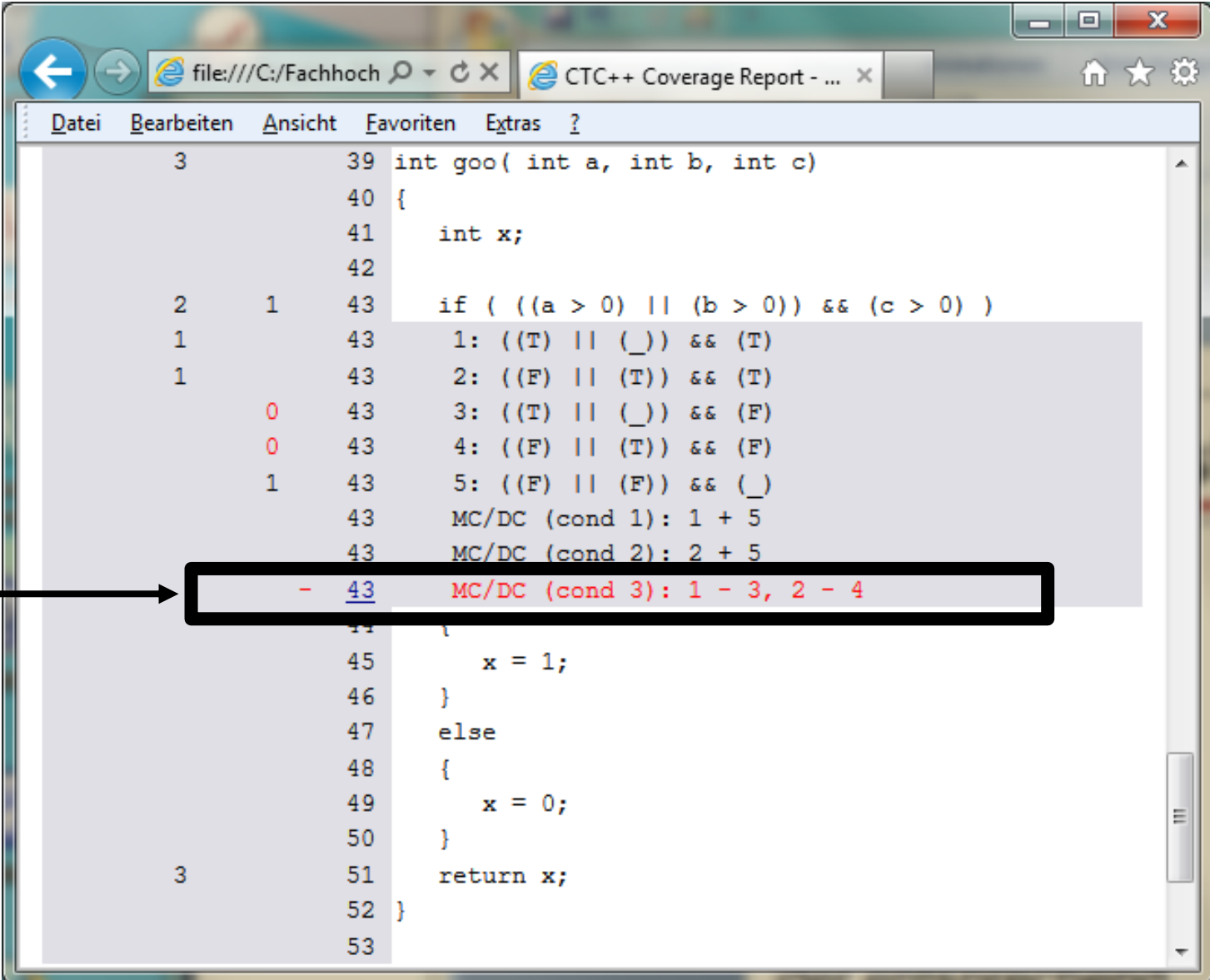
```
UCUNIT__TESTCASE_BEGIN("MC/DC Coverage");
UCUNIT__CHECKLIST_BEGIN(UCUNIT__ACTION_WARNING);
UCUNIT__CHECK_IS_EQUAL(1, goo(1,0,1));
UCUNIT__CHECK_IS_EQUAL(1, goo(0,1,1));
UCUNIT__CHECK_IS_EQUAL(0, goo(0,0,0));
UCUNIT__CHECKLIST_END();
UCUNIT__TESTCASE_END("MC/DC Coverage");
```

```
=====
TESTCASE:MC/DC Coverage:BEGIN
-----
CHECK: Line 22:IsEqual<1, goo<1,0,1>>:PASSED
CHECK: Line 23:IsEqual<1, goo<0,1,1>>:PASSED
CHECK: Line 24:IsEqual<0, goo<0,0,0>>:PASSED
CHECK: Line 25:Checklist<>:PASSED
-----
TESTCASE:MC/DC Coverage:PASSED
=====

*****
Testcases: failed: 0
           passed: 1
Checks:    failed: 0
           passed: 3
*****
```

Für die letzte atomare Bedingung (cond 3) wurde der Nachweis für MC/DC nicht erbracht. Es wird angezeigt, welche Testpaare noch notwendig sind.

3 atomare Bedingung
-> 4 Testfälle



```
file:///C:/Fachhoch CTC++ Coverage Report - ...
Datei Bearbeiten Ansicht Favoriten Extras ?
3 39 int goo( int a, int b, int c)
40 {
41     int x;
42
43     if ( ((a > 0) || (b > 0)) && (c > 0) )
1 43     1: ((T) || (_)) && (T)
1 43     2: ((F) || (T)) && (T)
0 43     3: ((T) || (_)) && (F)
0 43     4: ((F) || (T)) && (F)
1 43     5: ((F) || (F)) && (_)
43     MC/DC (cond 1): 1 + 5
43     MC/DC (cond 2): 2 + 5
- 43     MC/DC (cond 3): 1 - 3, 2 - 4
44     {
45         x = 1;
46     }
47     else
48     {
49         x = 0;
50     }
3 51     return x;
52 }
53
```



```
int goo( int a, int b, int c)
{
    int x;

    if (((a>0) || (b>0)) && (c>0))
    {
        x = 1;
    }
    else
    {
        x = 0;
    }

    return x;
}
```

(a>0)	(b>0)	(c>0)	x
wahr	-	wahr	1
wahr	-	falsch	0
falsch	wahr	wahr	1
falsch	wahr	falsch	0
falsch	falsch	-	0

Alle Testfälle (hier fünf) werden ausgeführt. Normen fordern Multicondition Coverage nicht ein.

Aber: Hat man 100% Multicondition Coverage, so ergibt sich auch 100% MC/DC (Subsumption).

Statt vier Testfälle (n+1) werden alle fünf Testfälle ausgeführt und man erhält automatisch eine 100%ige MC/DC Testüberdeckung.

SIL: Safety Integrity Level

Verfahren/Maßnahme		SIL 1	SIL 2	SIL 3	SIL 4
...
7a	Strukturabhängige Tests mit einer Testabdeckung (Eingangspunkte) 100% Aufrufüberdeckung	++	++	++	++
7b	Strukturabhängige Tests mit einer Testabdeckung (Anweisungen) 100% Anweisungsüberdeckung	+	++	++	++
7c	Strukturabhängige Tests mit einer Testabdeckung (Verzweigungen) 100% Zweigüberdeckung	+	+	++	++
7d	Strukturabhängige Tests mit einer Testabdeckung (Bedingungen) 100% MC/DC oder MCC?	+	+	+	++

Auszug Tabelle B.2 aus der DIN EN 61508-3

- ++ Besonders empfohlene Maßnahmen, bei Nichtverwendung ist dies zu begründen
- + Empfohlene Maßnahmen

ASIL: Automotive Safety Integrity Level

Methods		ASIL			
		A	B	C	D
1a	Statement coverage <i>Anweisungsüberdeckung</i>	++	++	+	+
1b	Branch coverage <i>Zweigüberdeckung</i>	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage) <i>MC/DC</i>	+	+	+	++

Tabelle 12 (Software Unit Level) aus der ISO 26262-6

Methods		ASIL			
		A	B	C	D
1a	Function coverage <i>Aufrufüberdeckung</i>	+	+	++	++
1b	Call coverage <i>Call Pair Coverage</i>	+	+	++	++

Tabelle 15 (Software Architectural Level) aus der ISO 26262-6

- ++ Besonders empfohlene Maßnahmen
- + Empfohlene Maßnahmen

DO-178B/C

Level	Auswirkungen	Coveragestufen	Anteile Systeme	Anteile Software
A	„Catastrophic“	MC/DC, C ₁ , C ₀	20-30%	40%
B	„Hazardous/Severe“	C ₁ , C ₀	20%	30%
C	„Major“	C ₀	25%	20%
D	„Minor“	-	20%	10%
E	„No Effect“	-	10%	5%

Anweisungsüberdeckung C₀, Zweigüberdeckung C₁, Modifizierter Bedingungs-/ Entscheidungsüberdeckungstest MC/DC

IEC 62304

„... it might be **desirable** to use white box methods to more efficiently accomplish certain tests, initiate stress conditions or faults, or increase code coverage of the qualification tests.“ (IEC 62304, Kapitel B.5.7 Software System testing)



- Einbau von globalen Zähler (Integer-Arrays) in Programmcode
- Speicherung der Information: Wo welcher Zähler instrumentiert ist!
- Inkrementierung der Zähler beim Durchlauf
- Speicherung der Zählerstände
- Spätere Auswertung der Zählerstände für den Report

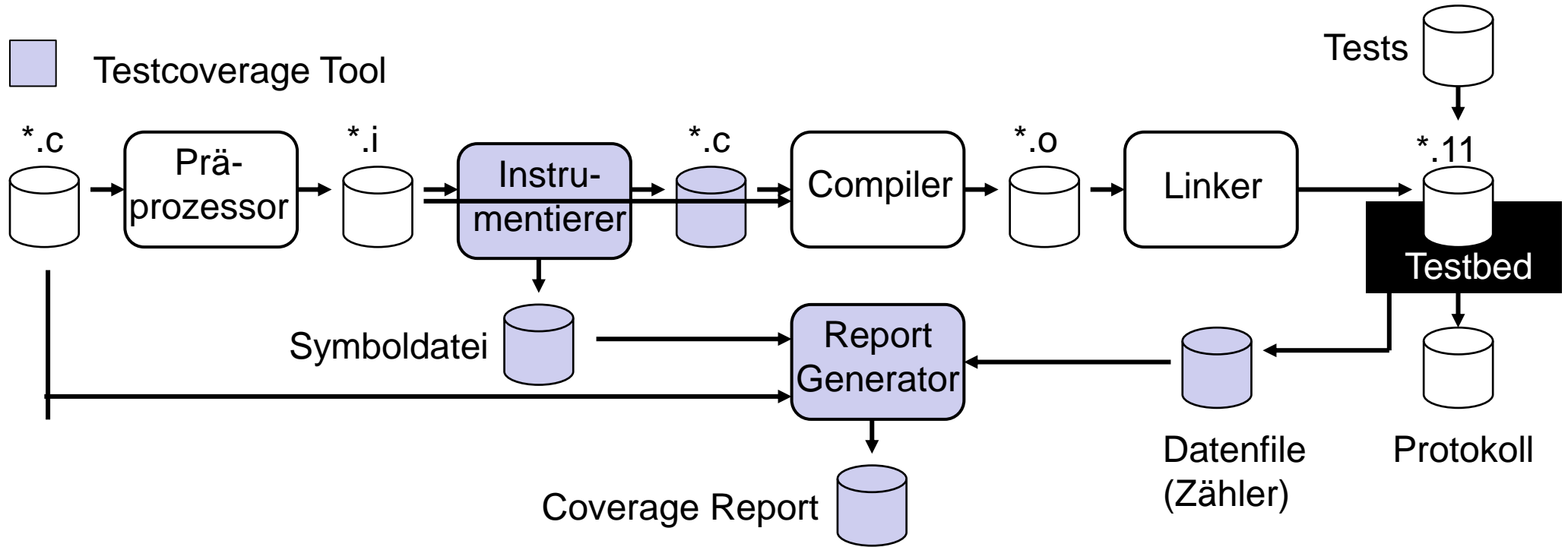
Ohne Komma-Operator

```
if ( a<0 )
{ /* ... */ }
else
{ /* ... */ }
```

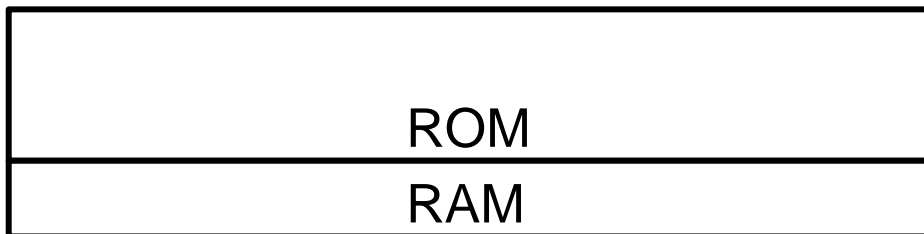
Mit Komma-Operator

```
if ( (a<0&&(zaehler1++||1)) || (zaehler2++&&0) )
{ /* ... */ }
else
{ /* ... */ }
```

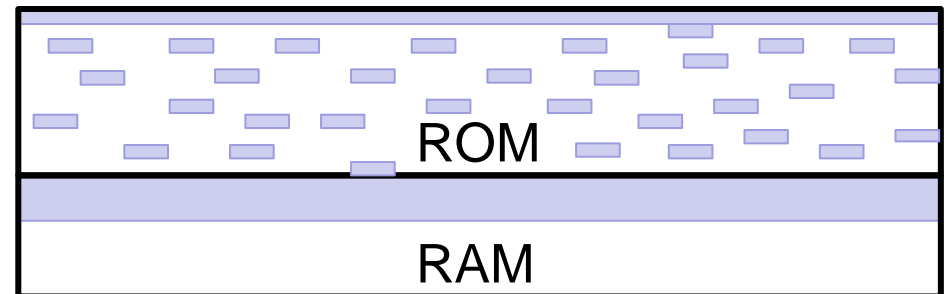
```
if ( (a<0) ? (zaehler1++,1) : (zaehler2++,0) )
{ /* ... */ }
else
{ /* ... */ }
```



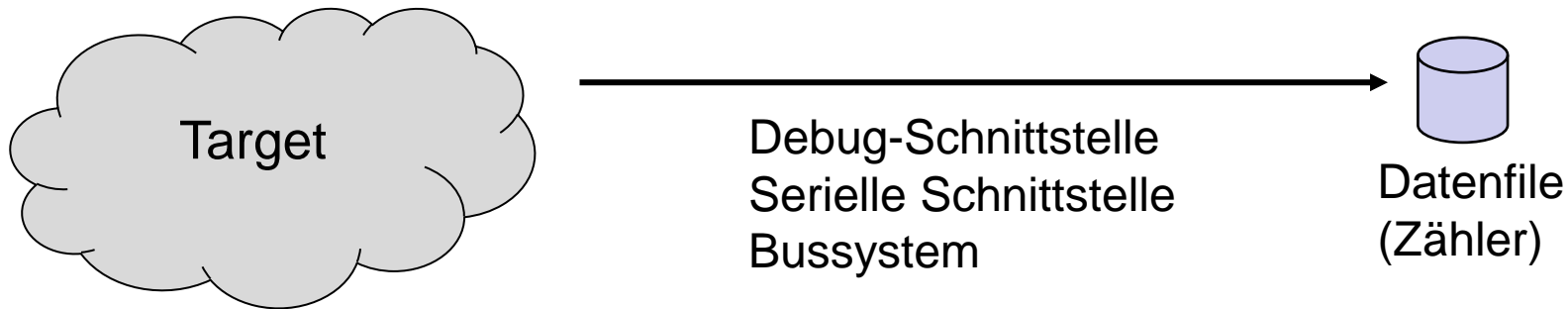
**Speicherbedarf Target
ohne Instrumentierung**



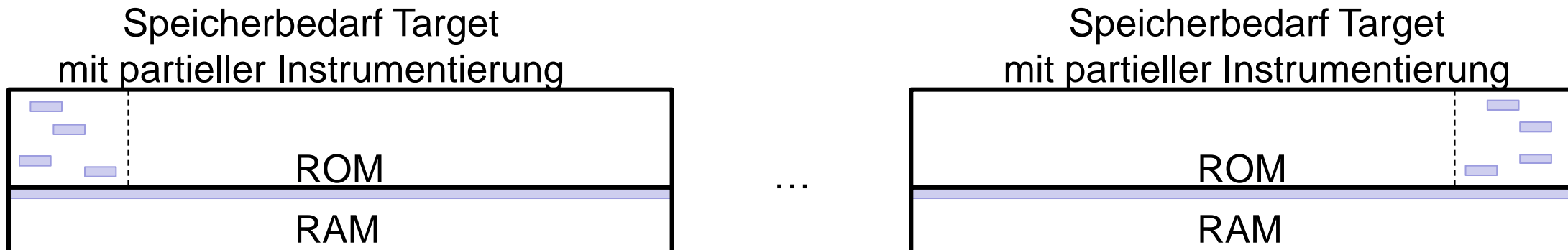
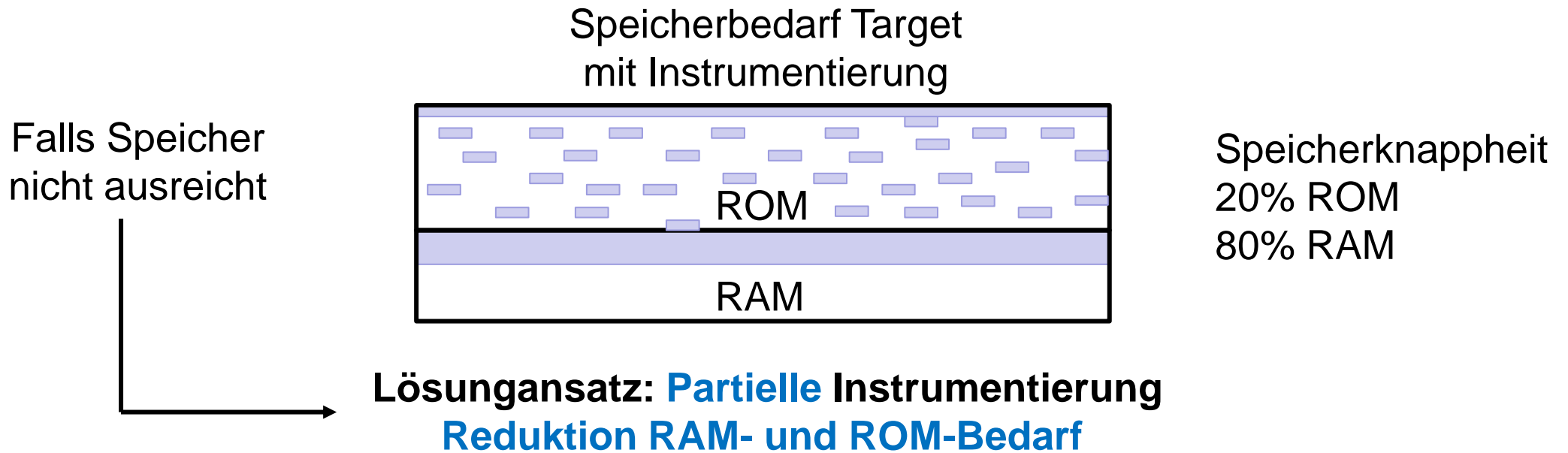
**Speicherbedarf Target
mit Instrumentierung**



- RAM
 - ROM
- } Ursache Speicherknappheit: 80 % RAM, 20 % ROM (prakt. Erfahrungen)
- **Meist kein Dateisystem** (Zähler müssen im RAM gehalten werden)



- Begrenzte **Anzahl an Schnittstellen** auf dem Target (Auslesen der Zähler)
Zusätzliche Schnittstelle für Tests im HW-Design berücksichtigen (Design for Test)



Testautomatisierung der Black- und White-Box-Tests sinnvoll/notwendig

Reduktion RAM-Bedarf

- **32-, 16- oder 8-Bit Zähler?**

Einsparung RAM ... aber höhere Wahrscheinlichkeit von Überläufen

- **Ein einzelnes Bit als Flag (Bit-Coverage):**

Wird immer nur gesetzt, Einsparung von RAM, Information wie oft Codestelle durchlaufen wurde geht verloren (Information meist nicht notwendig)

Reduktion ROM-Bedarf

- Geringste geforderte Instrumentierung (Funktions-, Zweig- und Bedingungebene) wählen

- Nutzung Unterstützung der Hardware bei Bit-Coverage:

-Keine Unterstützung-

```
MOV 0x200, %reg1
OR 2, %reg1
MOV %reg1, 0x200
```

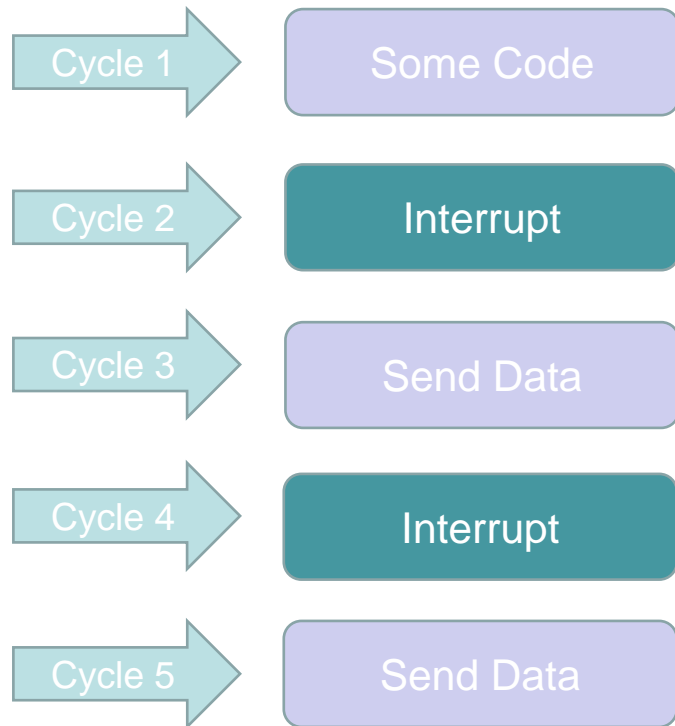
-Unterstützung durch HW-

```
ORL 0x200, 2
```

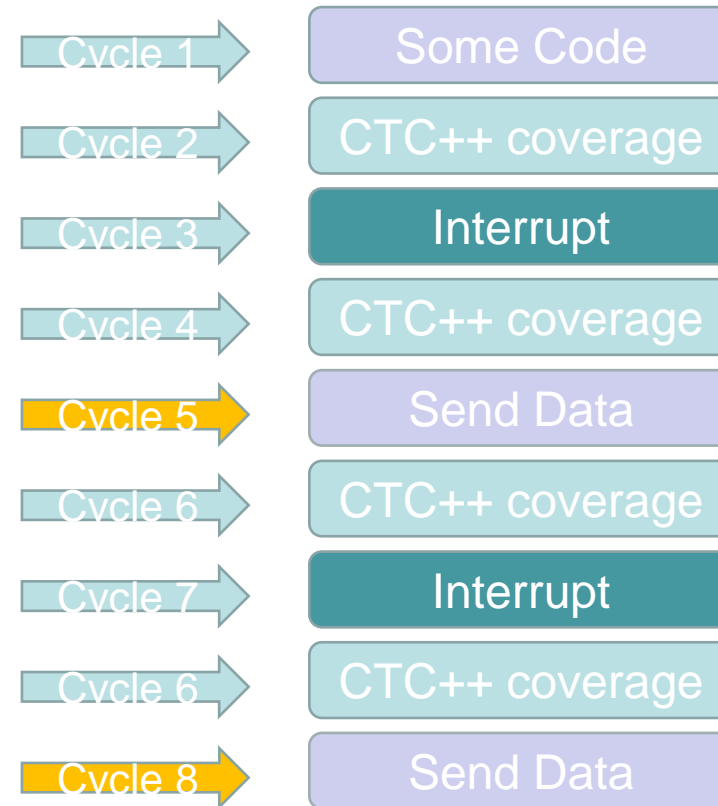
C51

```
SETB 0x1602
```

Melexis



Uninstrumented Execution Profile



Instrumented Execution Profile

```
int goo( int a, int b, int c)
{
  int x;

  if (((a>0) || (b>0)) && (c>0))
  {
    x = 1;
  }
  else
  {
    x = 0;
  }

  return x;
}
```

ROM-Bedarf

Ohne Instrumentierung:	60 Byte
Funktionsüberdeckung:	67 Byte
Zweigüberdeckung:	118 Byte
Bedingungsüberdeckung:	285 Byte

Einfaches Beispiel mit geringem Code und großen Instrumentierungs-overhead (im Mittel 30% Overhead).

Zusätzlicher RAM-Bedarf ohne Bit-Coverage

Funktionsüberdeckung:	1 Integer	Integer:
Zweigüberdeckung:	4 Integer	32 Bit (unsigned long)
Bedingungsüberdeckung:	7 Integer	als default

Zusätzlicher RAM-Bedarf mit Bit-Coverage

Funktionsüberdeckung:	1 Bit
Zweigüberdeckung:	4 Bit
Bedingungsüberdeckung:	7 Bit

Codecoverage wird zukünftig immer wichtiger (Normen und Testendekriterium)

Es gibt verschiedene Coverage Stufen (unterschiedlicher Testaufwand pro Coverage Stufe)

Ansätze wurden gezeigt, um die Herausforderungen (Speicherknappheit, kein Filesystem und begrenzte Anzahl von Schnittstellen) auf kleinen Targets zu meistern. Hierin unterscheiden sich die verschiedenen Coverage-Tools.

Empfehlung: Führen Sie im Vorfeld eine Evaluierung von Coverage-Tools auf ihrem Target durch.

Roland Bär
baer@verifysoft.com

Andreas Behr
behr@verifysoft.com

Daniel Fischer
daniel.fischer@hs-offenburg.de


TECHNOLOGY


Hochschule Offenburg
University of Applied Sciences

Oder besuchen Sie uns am
Messestand!