# C++ OpenCppCoverage

Detlef Wilkening

10.11.2016

- **Warum überhaupt dieser Vortrag?**
  - Wir waren mal wieder in einer Gruppe unterwegs zu einem C++ Treffen
  - Und ich erzählte von einem meiner vielen Probleme:
    - Kennt jemand ein Coverage Tool?
      - Für Windows
      - Arbeitet mit Microsoft Visual Studio zusammen
      - Ist umsonst
  - Damals kannte ich kein solches Coverage Tool

- **Warum überhaupt dieser Vortrag?**
  - Wir waren mal wieder in einer Gruppe unterwegs zu einem C++ Treffen
  - Und ich erzählte von einem meiner vielen Probleme:
    - Kennt jemand ein Coverage Tool ?
      - Für Windows
      - Arbeitet mit Microsoft Visual Studio zusammen
      - Ist umsonst
  - Damals kannte ich kein solches Coverage Tool

  - Doch dann – danke an alle C++ User-Treffen – bekam ich einen Hinweis:
    - OpenCppCoverage
      - Für das Microsoft Visual Studio
      - Für Windows
      - Open-Source und frei
      - https://opencppcoverage.codeplex.com/
      - Aktuelle Version: 0.9.5.3

  - Möglicherweise geht es anderen ähnlich
  - Und Sie suchen auch ein solches Tool
  - Darum dieser Vortrag…

**Aber erstmal für alle die, die sich nicht so auskennen:**

- **Was ist Coverage überhaupt?**
- **Was ist ein Coverage-Tool?**

- **Man hat Tests**
  - Im Idealfall automatische Tests
    - Z.B. Unit-Tests
    - Können aber auch z.B. automatisierte GUI-Tests sein
  - Können aber auch manuelle Tests sein
- **Die Tests sind zu 100% erfolgreich**

**Aber:**

- **Wie gut sind die Tests?**
- **Was sagen die 100% Erfolg aus?**
- **Hat man einfach nicht alles getestet?**
- **Oder nur die einfachen Dinge?**

**Hier hilft ein Coverage-Tool**

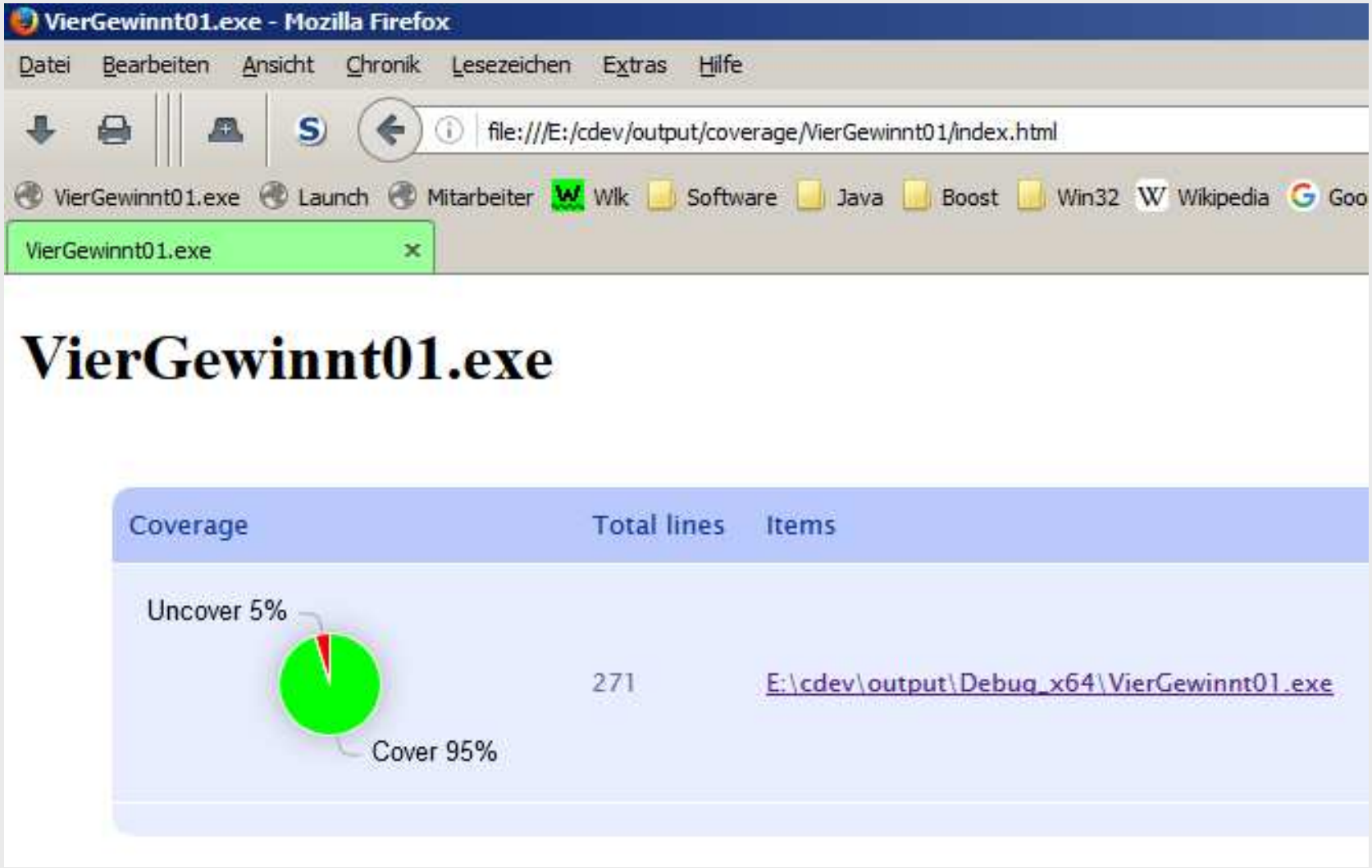
## Coverage-Tool

- **Zeigt, welcher Code bei einem Programmlauf abgedeckt wurde**
  - Der Programmablauf ist meist ein automatischer Test – muss aber nicht
- **Im Idealfall ist der gesamte Code abgedeckt**
  - Aber Achtung
    - Nur weil 100% Code-Abdeckung erreicht ist
    - Muss der Code nicht fehlerfrei sein
  - Dabei gibt es noch mehr zu beachten
  - Aber dazu später mehr

## Kleines Beispiel

- **Vier-Gewinnt Programm**
  - Mit automatischen Unit-Tests
- **HTML Outputs mit OpenCppCoverage**
  - Dateien-Seite (siehe Screens 2/3) ist nicht vollständig
  - Beispiel besteht aus mehr Dateien

- **Beispiel Screens 1 / 3**



- **Beispiel Screens 2 / 3**

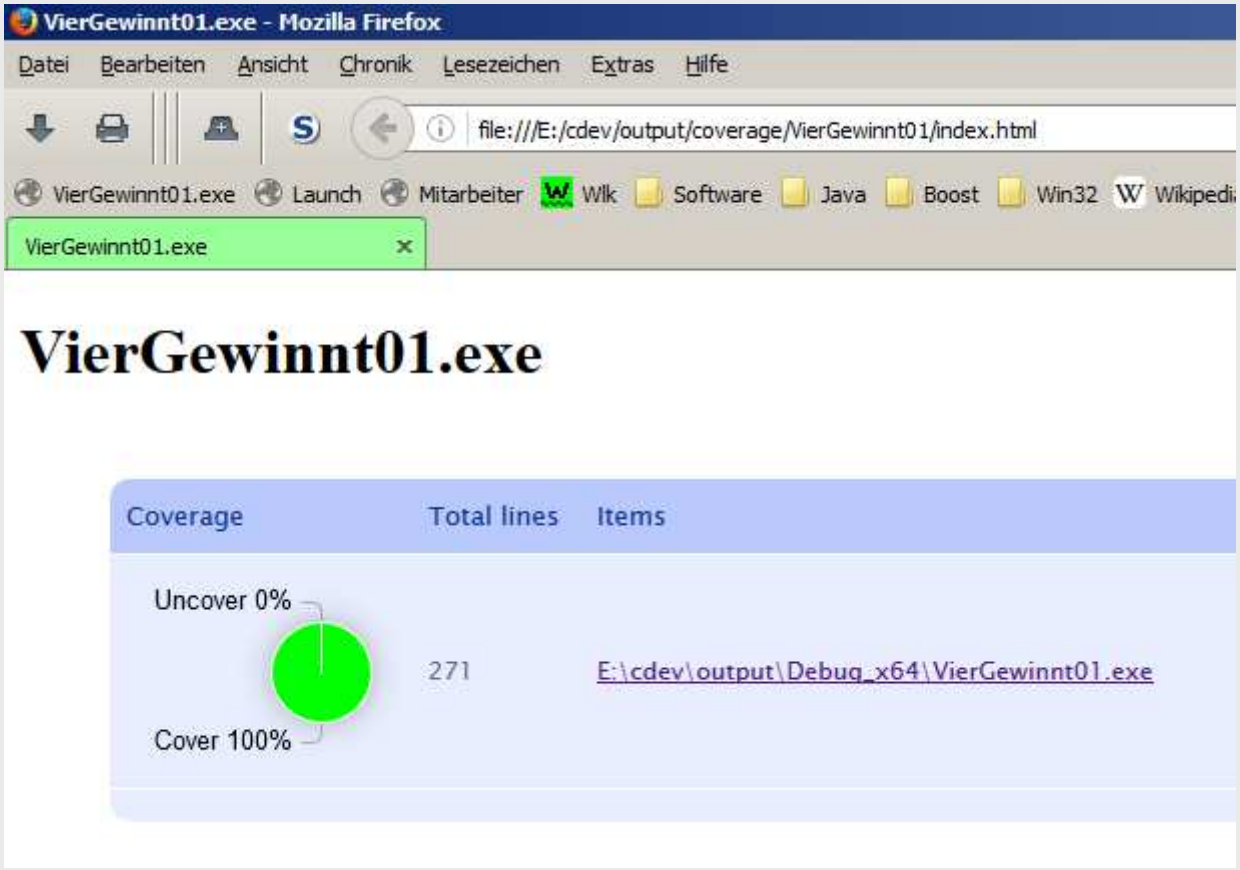
| Coverage | Total lines | Items |
|---|---|---|
|   | 75 | e:\cdev\cprojects\themen\viergewinnt\viergewinnt01\compi_analyse.cpp |
|   | 12 | e:\cdev\cprojects\themen\viergewinnt\viergewinnt01\compi_rnd.cpp |
|   | 28 | e:\cdev\cprojects\themen\viergewinnt\viergewinnt01\game.cpp |
|   | 88 | e:\cdev\cprojects\themen\viergewinnt\viergewinnt01\board.cpp |
|   | 18 | e:\cdev\cprojects\themen\viergewinnt\viergewinnt01\board.h |

- **Beispiel Screens 3 / 3**

compi_analyse.cpp

```
 99.    assert(!moves.empty());
100.    uniform_int_distribution<vector<Move>::size_type> dist(0, moves.size()-1);
101.    return moves[dist(engine)];
102. }
103.
104. //-----------------------------------------------------------------------------------------------
105.
106. int CompiAnalyse::simulateEnemyMove(const Board& b, int halflevel) const
107. {
108.    const Color color = !getColor();
109.
110.    int badestMoveResult = Max;
111.    for (int col = 0; col < Cols; ++col)
112.    {
113.       if (b.isNotColEmpty(col))
114.       {
115.          continue;
116.       }
117.
118.       Board bt(b);
119.       Move m(col, color);
120.       auto res = bt.set(m);
121.       switch (res)
122.       {
123.       case Board::Result::Win:
124.          return -Max+halflevel;
125.       case Board::Result::Remis:
126.          assert(bt.isFull());
127.          return 0;
128.       }
129.       assert(res == Board::Result::Open);
130.
```

- **Das ist natürlich ein gefakter Testlauf und Ergebnis**
- **Selbstverständlich hat der Test in Wirklichkeit 100% Coverage**
- **Ich habe für das Beispiel einige Tests disabled ;-)**



- **Welche Coverage-Arten gibt es?**
  - Function coverage
    - Wurde jede Funktion benutzt?
  - Statement coverage
    - Wurde jede Anweisung ausgeführt?
  - Line coverage
    - Wurde jede ausführbare Programmzeile ausgeführt
  - Branch coverage
    - Wurden alle Zweige bei Verzweigungen (If, Switch,…) ausgeführt?
  - Condition coverage
    - Wurden alle Boolschen-Ausdrücke zu false bzw. true ausgewertet?
      - Siehe Beispiel gleich
  - Path coverage
    - Wurde jeder mögliche Pfad durch eine Funktion/Modul/… ausgeführt?
      - Siehe Beispiel gleich
  - Loop coverage
    - Wurde jede Schleife 0-mal, 1-mal und mehr als 1-mal ausgeführt?
  - Und weitere
    - Parameter value coverage, LCSAJ coverage, State coverage, …

- **In der Praxis wohl am wichtigsten**
  - Statement coverage
    - Enthält damit z.B. „Function coverage“ und „Branch coverage“
  - Condition coverage & Path coverage
    - Wurden alle Boolschen-Ausdrücke zu false bzw. true ausgewertet?
    - Siehe Beispiel unten
      - Es gibt 4 Möglichkeiten für die Condition coverage
      - Aber es gibt nur 3 mögliche Pfade durch die Funktion
        - » True und False fügen keinen weiteren Pfad hinzu
  - Alle anderen sind entweder zu einfach oder schnell sehr aufwändig

```
int fct(int x, int y)
{
   if ((x>0) && (y>0))
   {
       return x;
   }
   return y;
}
```

# OpenCppCoverage

- **Coverage-Tool speziell für die Microsoft Visual-Studio Compiler**
  - Ab Visual Studio 2008 (inkl.)
    - Sollte eigentlich auch mit älteren Versionen funktionieren
  - Basiert intern auf den Microsoft Programm-Datenbank-Dateien (*.pdb)
- **Features**
  - Frei, Open-Source
  - Not intrusive
    - Programm muss nicht instrumentiert oder neu compiliert werden
  - Line Coverage
  - Binär, HTML & Cobertura Reports
- **Features – von mir bislang nicht genutzt oder verifiziert**
  - Geringer Performance-Overhead
  - Visual Studio Plugin existiert
  - Coverage Zusammenfassung
  - Child-Prozess Coverage
  - Jenkins Support
    - Durch Cobertura Ausgabe-Format

- **Aufruf**
  - Kommandozeile: „ OpenCppCoverage“
  - Optionen, u.a.:
    - Executable
    - Pfade zu den Sourcen
    - Reg-Expr für Sourcen, die nicht „gecoveragt“ werden sollen
    - Module, die nicht „gecoveragt“ werden sollen
    - Ausgabe-Typ
    - Ausgabe-Verzeichnis
    - Ausgabe-Verzeichnis mit Nutzung von Datum & Uhrzeit

- **Beispiel:**
  - OpenCppCoverage
  - --sources E:\cdev\cprojects\Themen\Coverage\Coverage01
  - --modules e:\cdev
  - --excluded_sources .test.cpp
  - --export_type=html:e:\cdev\output\coverage\Coverage01
  - -- e:\cdev\output\Debug_x64\Coverage01.exe

- **Beispiel „Coverage01“**
  - Einfaches Beispiel
  - Ausführlich
    - Wenn auch langweilig
  - Output HTML
- **Folien**
  - 1. Code (diese hier)
  - 2. Einstiegs-Seite & Detail-Seite
  - 3. Code & Coverage
- **Hinweis**
  - Bei späteren Beispielen zeige ich nur was wichtig ist
  - Hier mal alles

Coverage01.cpp

Coverage01.cpp E:\cdev\cprojects\Themen\Coverage\Coverage01

```
#include <iostream>
using namespace std;

void f1()
{
   cout << "- f1" << endl;
}

void f2()
{
   cout << "- f2" << endl;
}

int main()
{
   cout << "Coverage 01" << endl;
   f1();
}
```

# Coverage01.exe

| Coverage | Total lines | Items |
|---|---|---|
|   | 10 | E:\cdev\output\Debug_x64\Coverage01.exe |

# Coverage01.exe

| Coverage | Total lines | Items |
|---|---|---|
|   | 10 | e:\cdev\cprojects\themen\coverage\coverage01\coverage01.cpp |

Coverage01.cpp

Coverage01.cpp E:\cdev\cprojects\Themen\Coverage\Coverage01

```
#include <iostream>
using namespace std;

void f1()
{
   cout << "- f1" << endl;
}

void f2()
{
   cout << "- f2" << endl;
}

int main()
{
   cout << "Coverage 01" << endl;
   f1();
}
```

coverage01.cpp

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  void f1()
5.  {
6.     cout << "- f1" << endl;
7.  }
8.
9.  void f2()
10. {
11.    cout << "- f2" << endl;
12. }
13.
14. int main()
15. {
16.    cout << "Coverage 01" << endl;
17.    f1();
18. }
```

- **Aber ich hatte hier auch einige Probleme**
  - Relative Pfade haben bei mir zum Teil nicht funktioniert
    - Vielleicht war ich auch nur zu dumm, sie korrekt anzugeben
    - Habe aber einiges probiert
    - Darum im Aufruf-Beispiel weiter vorne alle Pfade absolut
  - Ausgabe-Verzeichnis-Name mit Nutzung eines Zeitstempels (Datum & Uhrzeit)
    - Hat bei mir nur funktioniert, wenn kein spezielles Verzeichnis
      - Ohne Angabe eines Ausgabe-Verzeichnis hat es bei mir problemlos funktioniert
      - Mit nicht

- **Und ansonsten:**
  - Ich finde den HTML Output nicht optimal
    - Lässt sich selbst mit den „TableTools2“ nicht optimal filtern und sortieren
    - Habe aber die anderen Formate (gerade Cobertura) oder das Visual-Studio Plugin auch noch nicht ausprobiert…

- **Ich weiß: OpenCppCoverage ist OpenSource – man könnte selber…**
  - Ich habe sogar mal in die Sourcen reingeschaut, ganz ganz kurz nur
  - Aber hatte bislang einfach keine Lust mich dort zu vertiefen

- **Es gab noch mehr Probleme und Unschönheiten**
  - Z.B. was das Line-Coverage anging…
  - Und bei Exceptions…
  - Und bei der Integration von Dateien…

- **Beispiele dafür auf den nächsten Folien**

- **Beispiel „Coverage02“**
  - Thema „Abdeckung boolscher-Ausdrücke“
  - Wird von OpenCppCoverage nicht unterstützt
- **Test**
  - 4 gleiche Funktionen (f1, f2, f3, f4)
  - Jeweils ein if mit 2 boolschen Ausdrücken
  - Werden jeweils so aufgerufen, dass alle Bool-Variationen vorkommen

Coverage02.cpp

main | int main()

```
#include <iostream>
using namespace std;

bool f1(int x, int y)
{
    if (x > 0 && y > 0)
    {
        return true;
    }
    return false;
}
```

```
int main()
{
    cout << "Coverage 02
    f1(0, 0);
    f2(0, 1);
    f3(1, 0);
    f4(1, 1);
}
```

```
 4. bool f1(int x, int y)
 5. {
 6.     if (x > 0 && y > 0)
 7.     {
 8.         return true;
 9.     }
10.     return false;
11. }
12.
13. bool f2(int x, int y)
14. {
15.     if (x > 0 && y > 0)
16.     {
17.         return true;
18.     }
19.     return false;
20. }
21.
```

```
22. bool f3(int x, int y)
23. {
24.     if (x > 0 && y > 0)
25.     {
26.         return true;
27.     }
28.     return false;
29. }
30.
31. bool f4(int x, int y)
32. {
33.     if (x > 0 && y > 0)
34.     {
35.         return true;
36.     }
37.     return false;
38. }
39.
```

```
40. int main()
41. {
42.     cout << "Cover
43.     f1(0, 0);
44.     f2(0, 1);
45.     f3(1, 0);
46.     f4(1, 1);
47. }
```

- **Beispiel „Coverage03“**
  - Weiterhin Thema „Abdeckung boolscher-Ausdrücke“
  - Auch der Zeilen-Umbruch-Trick bringt hier nix
  - OpenCppCoverage hat **keine** Condition coverage

```
bool f1(int x, int y)
{
    if (x > 0
        && y > 0)
    {
        return true;
    }
    return false;
}
```

```
 4. bool f1(int x, int y)
 5. {
 6.     if (x > 0
 7.         && y > 0)
 8.     {
 9.         return true;
10.     }
11.     return false;
12. }
13.
14. bool f2(int x, int y)
15. {
16.     if (x > 0
17.         && y > 0)
18.     {
19.         return true;
20.     }
21.     return false;
22. }
```

```
24. bool f3(int x, int y)
25. {
26.     if (x > 0
27.         && y > 0)
28.     {
29.         return true;
30.     }
31.     return false;
32. }
33.
34. bool f4(int x, int y)
35. {
36.     if (x > 0
37.         && y > 0)
38.     {
39.         return true;
40.     }
41.     return false;
42. }
```

- **Beispiel „Coverage04“**
  - Zeilen-Coverage ist ein echtes Problem bei z.B. Lambda-Ausdrücken

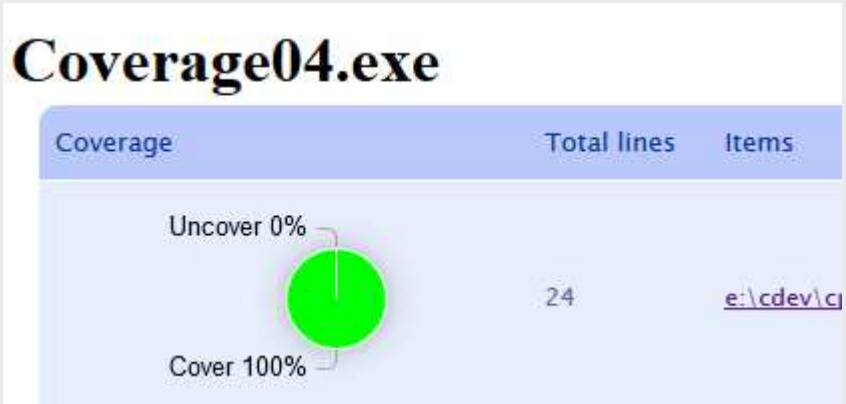
```
void f1()
{
    auto x = [](){ cout << "f1 Lambda" << endl; };
}

void f2()
{
    auto x = [] () { cout << "f2 Lambda" << endl;
    x();
}

void f3()
{
    auto x = []()
    {
        cout << "f3 Lambda" << endl;
    };
}
```

```
void f4()
{
    auto x = []()
    {
        cout << "f4 Lambda" << endl;
    };
    x();
}

int main()
{
    cout << "Coverage 04, Lambda" <<
    f1();
    f2();
    f3();
    f4();
}
```

- **Lambdas gelten als abgedeckt**
  - Wenn sie in einer Zeile stehen, wo Code ausgeführt wird
- **Auch wenn der Lambda-Ausdruck selber nie ausgeführt wurde**

```
4.  void f1()
5.  {
6.     auto x = [](){ cout << "f1 Lambda" << endl; };
7.  }
8.
9.  void f2()
10. {
11.    auto x = [] () { cout << "f2 Lambda" << endl; };
12.    x();
13. }
```

- **Hier hilft der Zeilen-Umbruch-Trick etwas**
- **Aber Achtung**
  - Nicht ausgeführte Lambdas fallen dem User vielleicht auf
    - Nicht grün markiert, sondern unmarkiert
  - Nicht dem Tool
    - Sie sind nicht rot und werden nicht als nicht-abgedeckt gemeldet!
    - Warum sind sie nicht rot markiert?
  - Die Coverage ist im Beispiel fehlerhaft 100%




```
15. void f3()
16. {
17.     auto x = []()
18.     {
19.         cout << "f3 Lambda" << endl;
20.     };
21. }
22.
23. void f4()
24. {
25.     auto x = []()
26.     {
27.         cout << "f4 Lambda" << endl;
28.     };
29.     x();
30. }
31.
```

- **Beispiel „Coverage05“**
  - Auch wenn mehrere Anweisungen „normal“ in einer Zeile stehen
    - Ist die Coverage-Anzeige manchmal fehlerhaft
    - Manchmal funktioniert sie auch korrekt

```
37.    for (int i = 0; i < 4; ++i)
38.    {
39.       cout << i;
40.       break; ++i;
41.    }
42.    cout << endl;
```

```
44.    for (int i = 0; i < 4; ++i)
45.    {
46.       cout << i;
47.       if (i == 0) break; ++i;
48.    }
49.    cout << endl;
```

```
51.    for (int i = 0; i < 4; ++i)
52.    {
53.       cout << i;
54.       if (i == -1) ++i;
55.    }
56.    cout << endl;
```

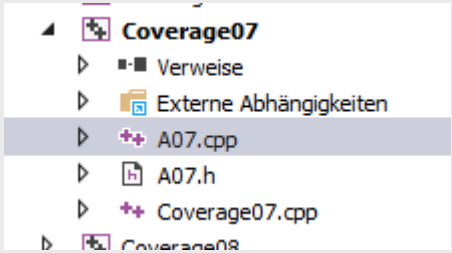
- **Beispiel „Coverage06“**
  - Ein anderes Problem sind das Werfen von Exceptions
  - Wenn in dem Scope ein Objekt mit Destruktor existiert
  - Der „scheinbar“ fehlende Destruktor-Aufruf wird angemeckert
    - Ist ein bekannter „Bug“, den OpenCppCoverage nicht fixen kann
    - Da der Code im PDB nun mal drin ist…

```
 1. #include <exception>
 2. #include <iostream>
 3. using namespace std;
 4.
 5. int main()
 6. {
 7.    cout << "Coverage 06 - Objekt mit Destruktor, Exception" << endl;
 8.
 9.    try
10.    {
11.       string s("Exception");
12.       throw exception(s.c_str());
13.    }
14.    catch (const exception& e)
15.    {
16.       cout << e.what() << endl;
17.    }
18. }
```

- **Beispiele „Coverage07“ … „Coverage15“**
  - Problematisch ist auch, dass OpenCppCoverage manchmal Dateien ignoriert
    - Daher nicht anzeigt, dass sie gar nicht benutzt wurden
    - Daher die Coverage hier 0 % ist
  - Dafür gibt es jetzt einige Beispiele…
    - Aber Achtung – manche Beispiele funktionieren auch, aber manche auch nicht

- **Hinweis**
  - Mir ist technisch klar,
    warum OpenCppCoverage die fehlende Coverage nicht melden kann
    - Dazu später mehr
  - Aber das ist einfach nicht, was ich von einem Coverage-Tool erwarte

- **Beispiel „Coverage07“**




A07.h

```
#ifndef A07_H
#define A07_H

class A07
{
public:
   void fct();
};

#endif
```

A07.cpp

```
#include "A07.h"
#include <iostream>
using namespace std;

void A07::fct()
{
   cout << "A07::fct" << endl;
}
```

Coverage07.cpp

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Coverage 07 - Klasse A07, implementiert im Source, ungenutzt" <
}
```

- **Beispiel „Coverage07“**
  - Das klappt

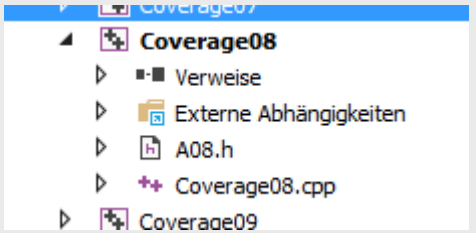
## Coverage07.exe

| Coverage | Total lines | Items |
|---|---|---|
|   | 6 | E:\cdev\output\Debug_x64\Coverage07.exe |

## Coverage07.exe

| Coverage | Total lines | Items |
|---|---|---|
|   | 3 | e:\cdev\cprojects\themen\coverage\coverage07\a07.cpp |
|   | 3 | e:\cdev\cprojects\themen\coverage\coverage07\coverage07.cpp |

- **Beispiel „Coverage08“**
  - Quasi identisch, nur inline im Header implementiert



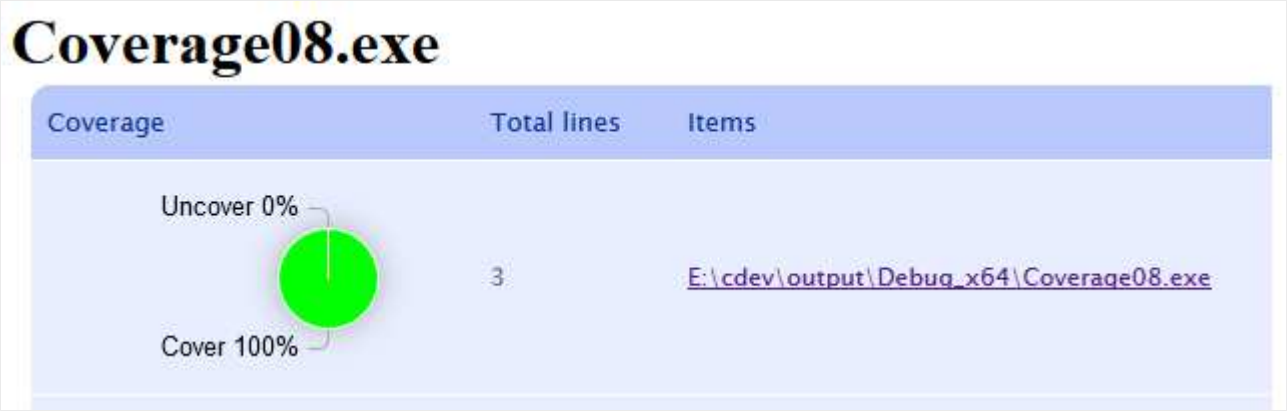

A08.h

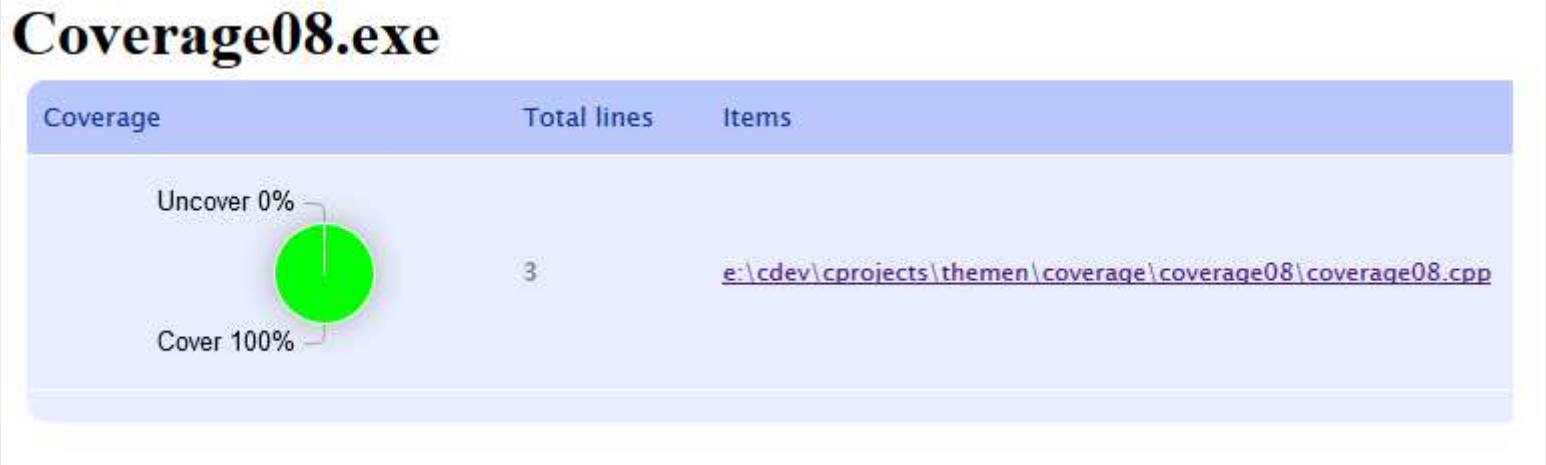
```
#ifndef A08_H
#define A08_H

#include <iostream>

class A08
{
public:
   void fct()
   {
      std::cout << "A08::fct" << std::endl;
   }
};

#endif
```

Coverage08.cpp

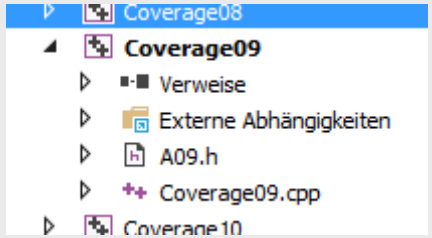
```
#include <iostream>
using namespace std;

int main()
{
   cout << "Coverage 08 -
}
```

- **Beispiel „Coverage08“**
  - Wird nicht angezeigt?



- **Beispiel „Coverage09“**
  - Wird die Funktion genutzt…




A09.h | Coverage09.cpp | Coverage08.cpp | A08.h

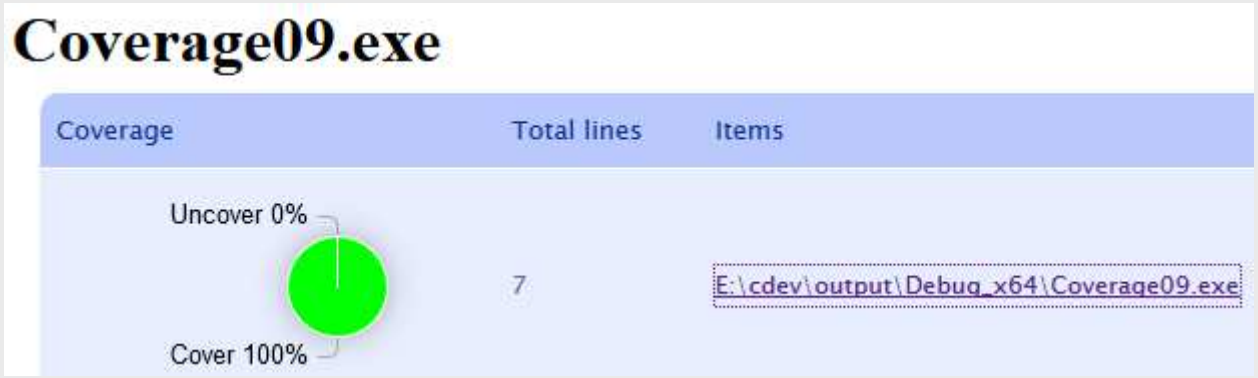
A09.h — E:\cdev\cprojects\Themen\Coverage\Coverage09\A09.h

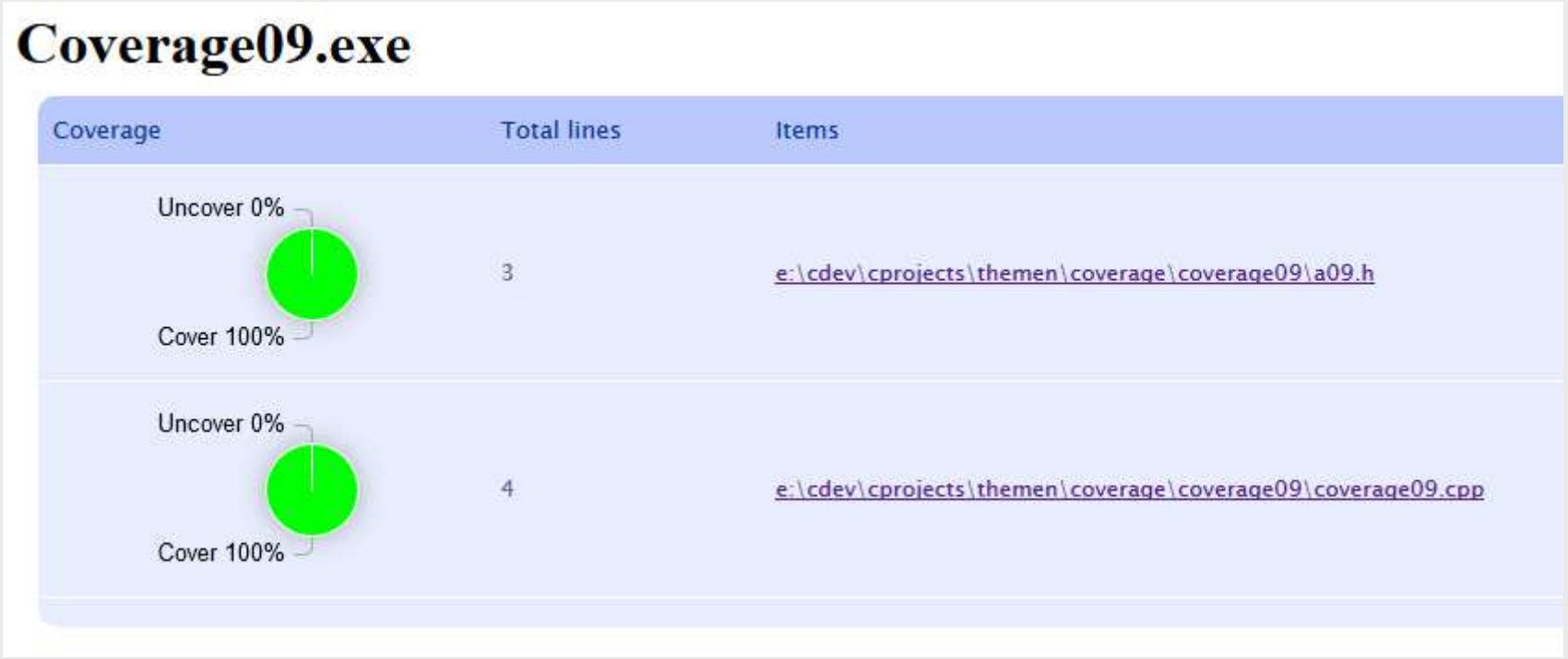
```
#ifndef A09_H
#define A09_H

#include <iostream>

class A09
{
public:
   void fct()
   {
      std::cout << "A09::fct" << std::endl;
   }
};

#endif
```

Coverage09.cpp — E:\cdev\cprojects\Ther

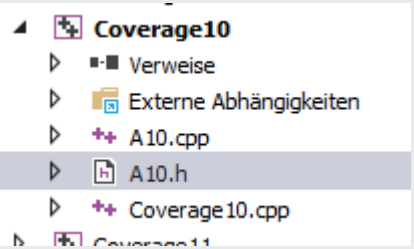
```
#include "A09.h"
#include <iostream>
using namespace std;

int main()
{
   cout << "Coverage 09
   A09 a;
   a.fct();
}
```

- **Beispiel „Coverage09“**
  - Alles okay



- **Beispiel „Coverage10“**
  - Und wie ist das, wenn man nur den Source „nutzt“?




Coverage10.cpp A10.cpp A10.h

A10.h E:\cdev\cprojects\Themen\Coverage

```
#ifndef A10_H
#define A10_H

#include <iostream>

class A10
{
public:
   void fSource();

   void fHeader()
   {
      std::cout << "A10::fHeade
   }
};

#endif
```

Coverage10.cpp A10.cpp A10.h

A10.cpp E:\cdev\cprojects\Themen\Cover

```
#include "A10.h"
#include <iostream>

void A10::fSource()
{
   std::cout << "A10::fSource
}
```

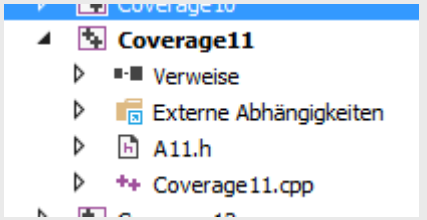
```
int main()
{
   cout << "Coverage 10 - Klasse A10
   A10 a;
   a.fSource();
}
```

- **Beispiel „Coverage10“**
  - Er sieht den fehlenden Teil im Header nicht

## Coverage10.exe

| Coverage | Total lines | Items |
|---|---|---|
|   | 3 | e:\cdev\cprojects\themen\coverage\coverage10\a10.cpp |
|   | 4 | e:\cdev\cprojects\themen\coverage\coverage10\coverage10.cpp |

- **Beispiel „Coverage11“**
  - Und bei Templates
  - Immerhin ist dort die Implementierung immer im Header?




Coverage11.cpp | A11.h

A11.h | E:\cdev\cprojects\Themen\Coverage\Coverage11\A11.h

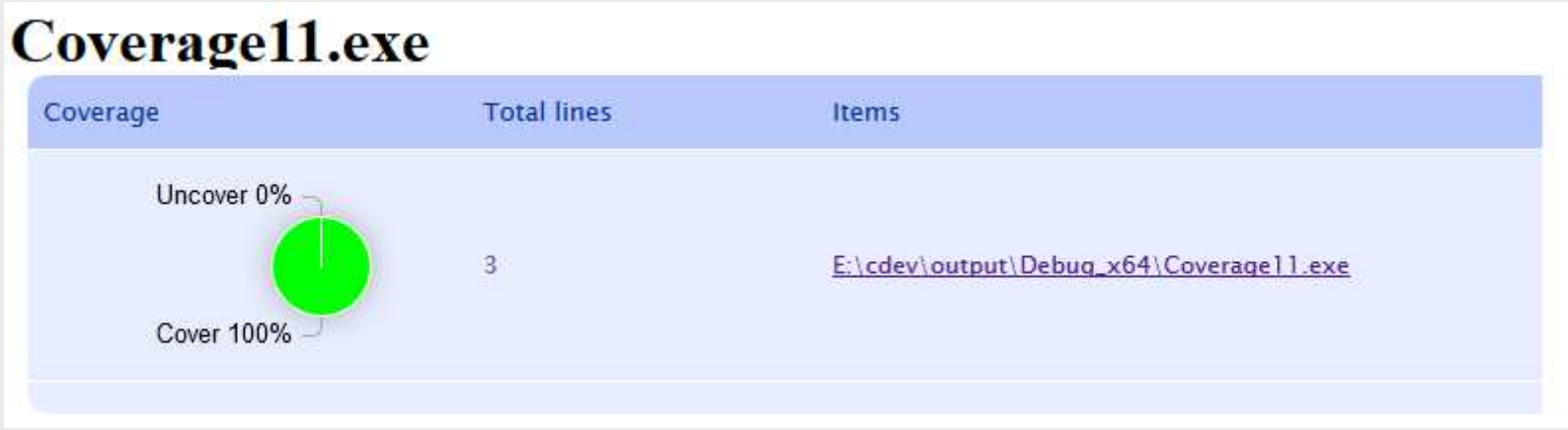
```
#ifndef A11_H
#define A11_H

#include <iostream>

template<class T> class A11
{
public:
   void fct()
   {
      std::cout << "A11::fct" << std::endl;
   }
};

#endif
```

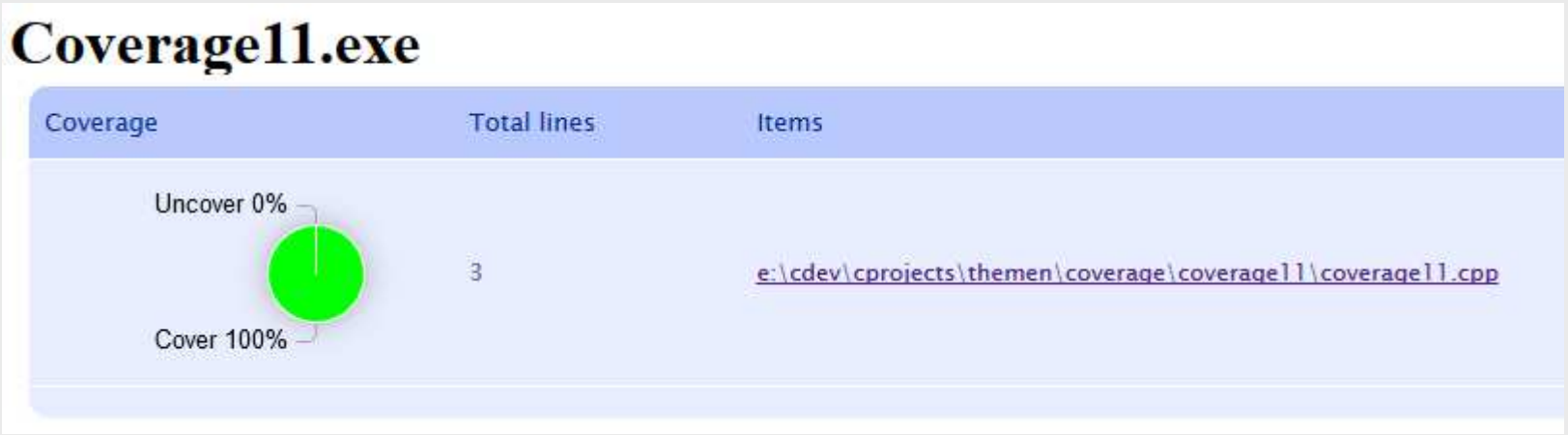
Coverage11.cpp | A11.h

Coverage11.cpp | E:\cdev\cprojects\Ther

```
#include "A11.h"
#include <iostream>
using namespace std;

int main()
{
   cout << "Coverage 11
}
```

- **Beispiel „Coverage11“**
  - Ganz böse – auch nicht ☹



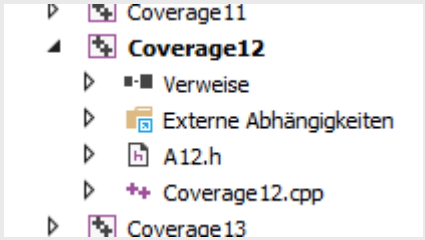



- **Beispiel „Coverage12“**
  - Und wenn ich ein Objekt erzeuge, nur die Funktion nicht aufrufe…

Coverage12.cpp | A12.h

A12.h | E:\cdev\cprojects\Themen\Coverag

```
#ifndef A12_H
#define A12_H

#include <iostream>

template<class T> class A12
{
public:
    void fct()
    {
        std::cout << "A12::fct"
    }
};

#endif
```




```
int main()
{
    cout << "Coverage 12 - Templa

    A12<int> a;
}
```

- **Beispiel „Coverage12“**
  - Kein Einfluß – immer noch fehlerhaft

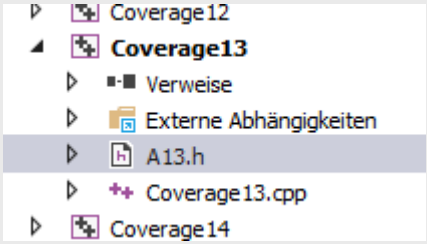
**Coverage12.exe**

| Coverage | Total lines | Items |
|---|---|---|
| Uncover 0% / Cover 100% | 3 | E:\cdev\output\Debug_x64\Coverage12.exe |

**Coverage12.exe**

| Coverage | Total lines | Items |
|---|---|---|
| Uncover 0% / Cover 100% | 3 | e:\cdev\cprojects\themen\coverage\coverage12\coverage12.cpp |

- **Beispiel „Coverage13“**
  - Ändert denn ein „=default“ Default-Konstruktor was?



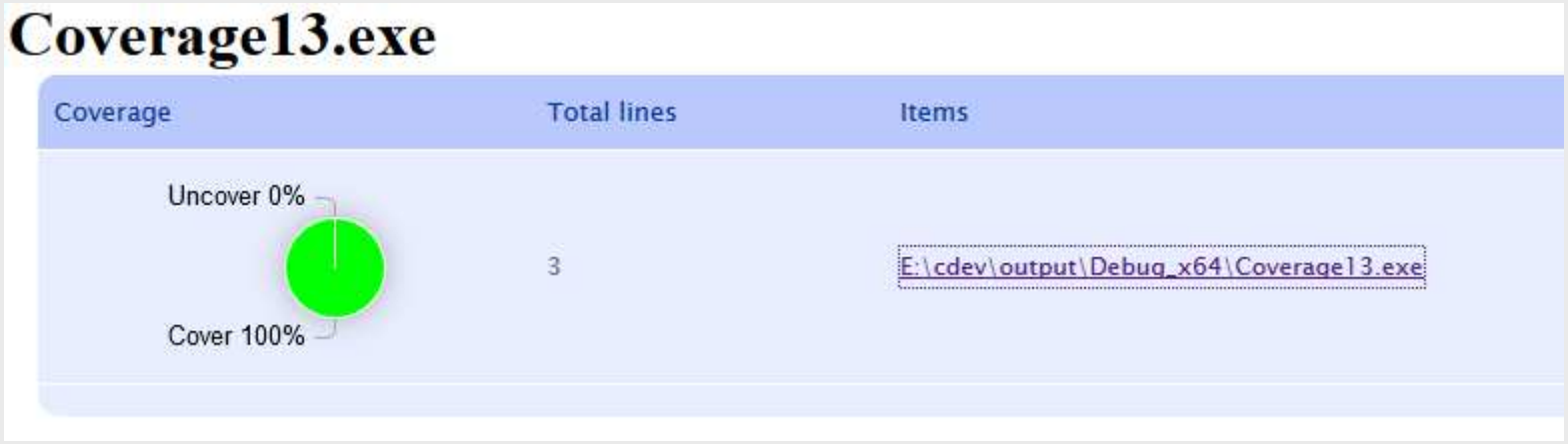

Coverage13.cpp | A13.h

A13.h | E:\cdev\cprojects\Themen\Coverage\Cov

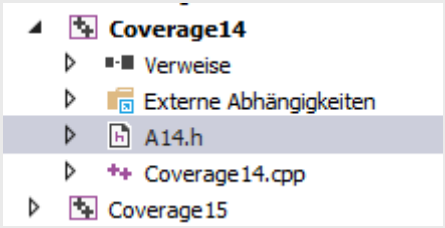
```
#ifndef A13_H
#define A13_H

#include <iostream>

template<class T> class A13
{
public:
   A13() = default;

   void fct()
   {
      std::cout << "A13::fct" <<
   }
};

#endif
```

```
int main()
{
   cout << "Coverage 13 - Temp

   A13<int> a;
}
```

- **Beispiel „Coverage13“**
  - Immer noch nichts – böse, böse ☹




- **Beispiel „Coverage14“**
  - Und wenn ich die Funktion nutze?




Coverage14.cpp | A14.h

A14.h — E:\cdev\cprojects\Themen\Coverage\Coverag

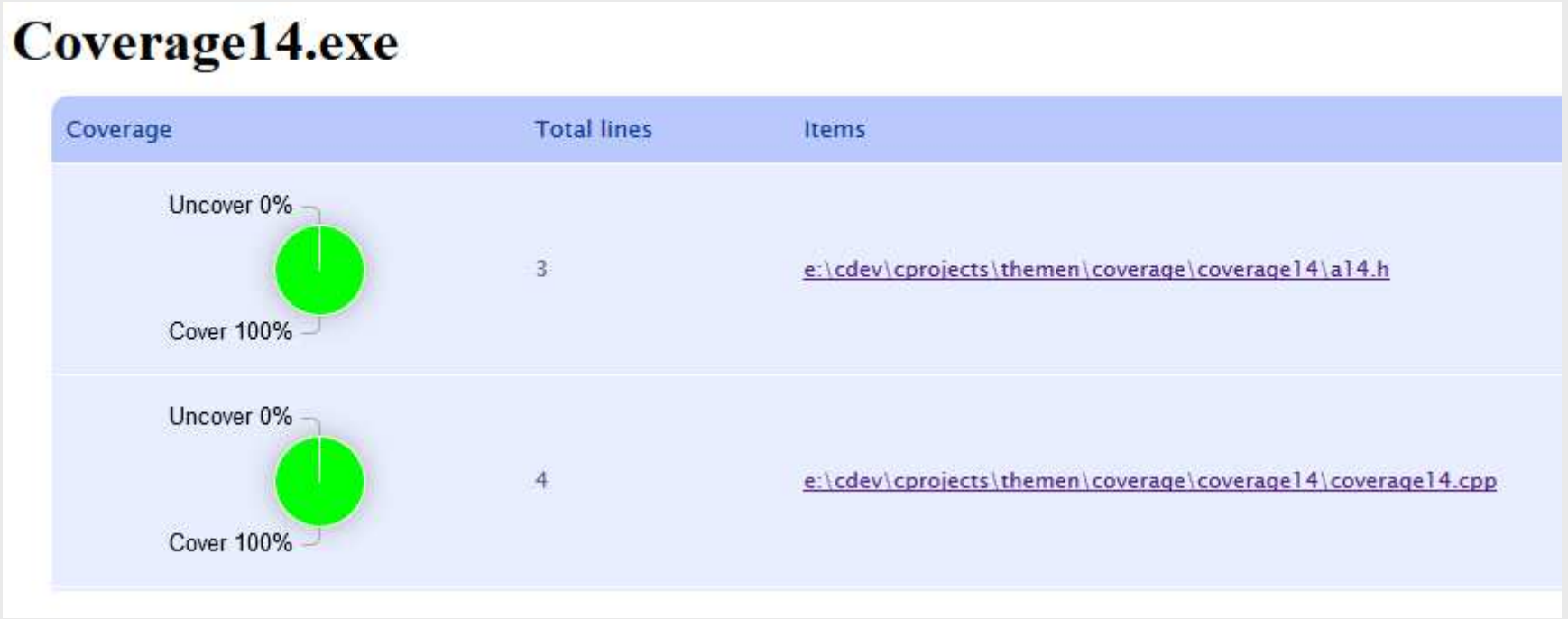
```
#ifndef A14_H
#define A14_H

#include <iostream>

template<class T> class A14
{
public:
    A14() = default;

    void fct()
    {
        std::cout << "A14::fct" << s
    }
};

#endif
```

Coverage14.cpp | A14.h

Coverage14.cpp — E:\cdev\cprojects\Themen\C

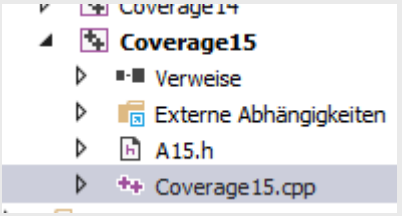
```
#include "A14.h"
#include <iostream>
using namespace std;

int main()
{
    cout << "Coverage 14 - T

    A14<int> a;
    a.fct();
}
```

- **Beispiel „Coverage14“**
  - Dann klappt es ☺



- **Beispiel „Coverage15“**
  - Und bei mehreren Funktionen, die nicht alle genutzt werden?

```
#ifndef A15_H
#define A15_H

#include <iostream>

template<class T> class A15
{
public:
   A15() {}

   void f()
   {
      std::cout << "A15::f" << st
   }

   void g()
   {
      std::cout << "A15::g" << st
   }
};

#endif
```




Coverage15.cpp | A15.h

Coverage15.cpp — E:\cdev\cprojects\Themen\Cov

```
#include "A15.h"
#include <iostream>
using namespace std;

int main()
{
   cout << "Coverage 15 - Te

   A15<int> a;
   a.f();
}
```

- **Beispiel „Coverage15“**
  - Auch wieder ganz böse – er merkt‘s nicht ☹

Coverage15.exe

| Coverage | Total lines | Items |
|---|---|---|
| Uncover 0% / Cover 100% | 9 | E:\cdev\output\Debug_x64\Coverage15.exe |

Coverage15.exe

| Coverage | Total lines | Items |
|---|---|---|
| Uncover 0% / Cover 100% | 4 | e:\cdev\cprojects\themen\coverage\coverage15\a15.h |
| Uncover 0% / Cover 100% | 5 | e:\cdev\cprojects\themen\coverage\coverage15\coverage15.cpp |

- **Beispiel „Coverage15“**
  - Wieder nur unmarkiert, nicht rot markiert
    - Hier Funktion „g()“

```
 1. #ifndef A15_H
 2. #define A15_H
 3.
 4. #include <iostream>
 5.
 6. template<class T> class A15
 7. {
 8. public:
 9.    A15() {}
10.
11.    void f()
12.    {
13.       std::cout << "A15::f" << std::endl;
14.    }
15.
16.    void g()
17.    {
18.       std::cout << "A15::g" << std::endl;
19.    }
20. };
21.
22. #endif
```

- **Technisch ist mir das schon klar**
  - Der Compiler optimiert nicht genutzte Inline-Funktionen weg
  - Der Compiler instanziiert nicht genutzte Template-Funktionen nicht
    - Das darf er auch gar nicht
    - Sonst wäre er nicht standard-konform, und auch nicht nutzbar
  - Und damit ist dieser Code nicht Teil des PDB Files vom Visual-Studio
    - Und ich denke, das trifft auf nicht genutzte Lambdas auch zu
      - Siehe Beispiel Coverage04 und Coverage05

- **Aus Sicht der Coverage ist dies aber nicht erwartungs-konform**
- **Und stellt das Tool natürlich in Frage!?**

# Fazit „OpenCppCoverage“ 1/2

- **Es ist ein Coverage-Tool für das Visual-Studio**
  - Umsonst und offen
- **Es funktioniert prinzipiell einfach und akzeptabel**
  - Meine ersten Versuche klappten auf Anhieb
  - Meine Batch-Dateien sind kurz und waren schnell geschrieben
    - Vielleicht sollte ich mal das Visual-Studio Plugin ausprobieren…
  - Läuft seitdem – mit den Schwächen (s.u.) – problemlos
- **OpenCppCoverage hat aber auch einige Schwächen**
  - Line Coverage ist nicht mein Traum
    - Lambdas, Ifs, …
    - Aufpassen, dass wirklich alles „gecoveragt“ ist
  - Exceptions mit Destruktor im Scope
    - Ich bekomme manche Tests nicht auf 100%
  - Dateien/Funktionen werden nicht integriert
    - User muss aufpassen, dass wirklich alles „gecoveragt“ ist
  - Relative Pfade funktionieren nicht oder nur eingeschränkt
  - Zeitstempel im Ausgabe-Namen bei expliziter Angabe funktioniert nicht

# Fazit „OpenCppCoverage“ 2/2

- **Ich hätte gerne ein besseres Tool**
  - Kennt jemand eine Alternative?
  - Aber umsonst
- **Kann aber mit den Schwächen leben**
  - Passe halt "ein bisschen" auf
- **Unterm Strich benutze ich es zuhause ständig**
- **Und habe dadurch schon einiges gelernt und gefunden**
  - Stellen, die nicht getestet wurden
  - Stellen, die gar nicht erreicht werden konnten
    - Und die ich dann entfernt habe
    - Natürlich mit einer entsprechenden Absicherung
  - Ablaufpfade, die mit gar nicht klar waren
    - Da Code – typischerweise Guards – nicht durchlaufen wurde
- **Ich möchte nicht mehr darauf verzichten**
  - Aber würde mich natürlich über eine bessere Alternative freuen

## Links

- https://opencppcoverage.codeplex.com/
- https://de.wikipedia.org/wiki/Testabdeckung
- https://en.wikipedia.org/wiki/Code_coverage

**Kennt eigentlich jemand einen guten Profiler für Windows / Visual-Studio?**

**Natürlich umsonst und Open-Source!**

**Hat schon mal jemand den im Visual-Studio integrierten Profiler genutzt und kann darüber berichten?**