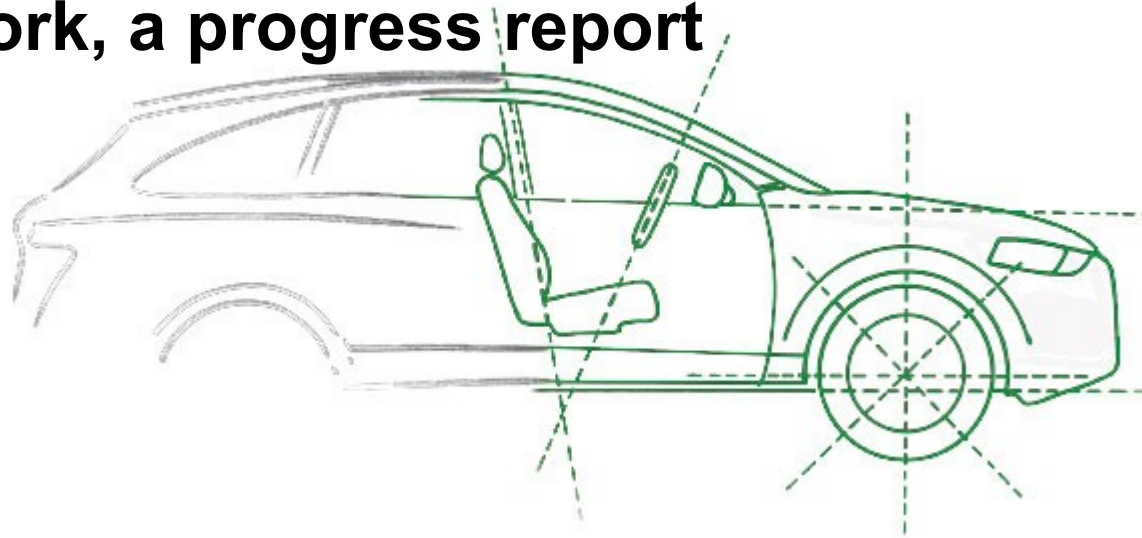




Evolutionary white-box software test with the EvoTest Framework, a progress report



Peter M. Kruse
April 1st, 2009



Table of Contents

- Motivation
- White Box Testing / Structural Testing
- Evolutionary Structural Testing
- EvoTest
- Case Studies
- Application
- Results
- Way forward
- Conclusion



Motivation

- Subsystems of today's luxury cars have up to 15 million lines of code
- Test automation needed
- Can Evolutionary Testing be the solution?
 - Some prototypes exists
 - Still mainly research-based
- Why?



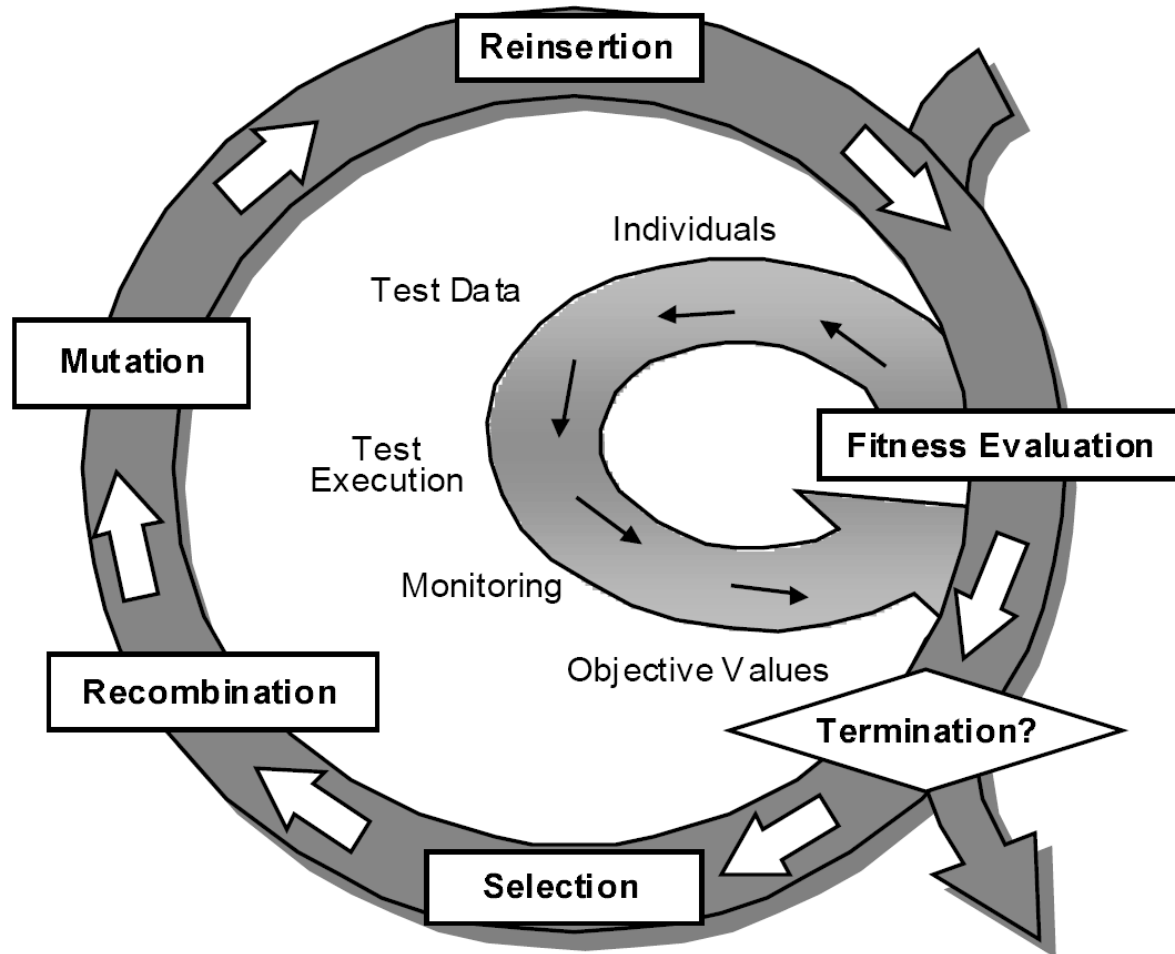
White Box Testing / Structural Testing

- Aim:
 - Maximal code coverage ¹
 - Little effort as possible
 - Efficient selection of test cases
- Usage:
 - During unit-testing phase of a software project
- Problem:
 - Finding test cases which exercise all branches is a complex task

¹⁾ As required by ISO 26262, IEC 61508, DO-178B



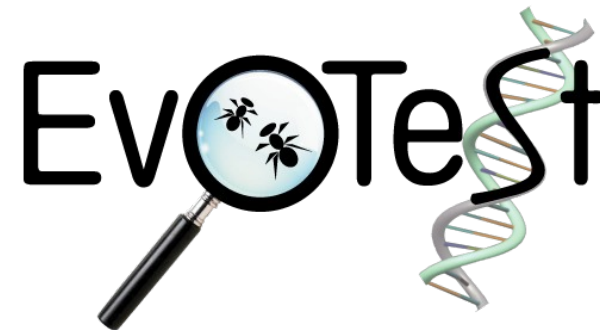
Evolutionary Structural Testing





EvoTest

- EU funded research project (IST-33472)
- Aim:
 - Find solutions for software testing
 - Using evolutionary adaptive techniques
- EvoTest Framework:
 - Automated structural testing (White Box)
 - Functional testing (Black Box)
 - Signal Generator Component
 - ...





Case Studies

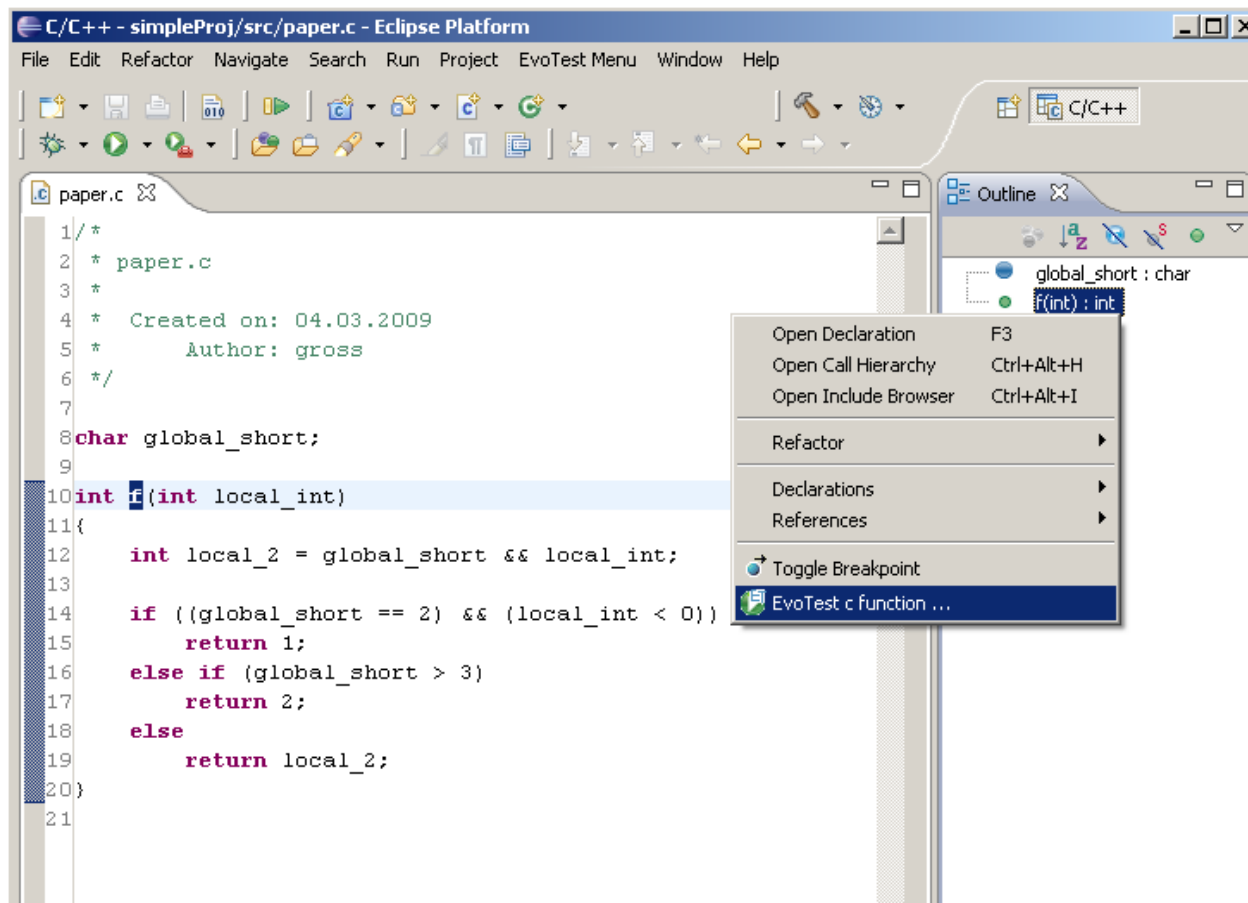
- 4 case studies
- Embedded Software Module
- Automotive Industry
- Implemented in C Language

Case Study	Description
A	Active Brake Assist software module (<i>dSpace TargetLink</i>)
B	Adaptive Headlight control software module (<i>dSpace TargetLink</i>)
C	Door-Lock control software module (<i>ETAS ASCET</i>)
D	Electric window control software module (<i>ETAS ASCET</i>)



Application

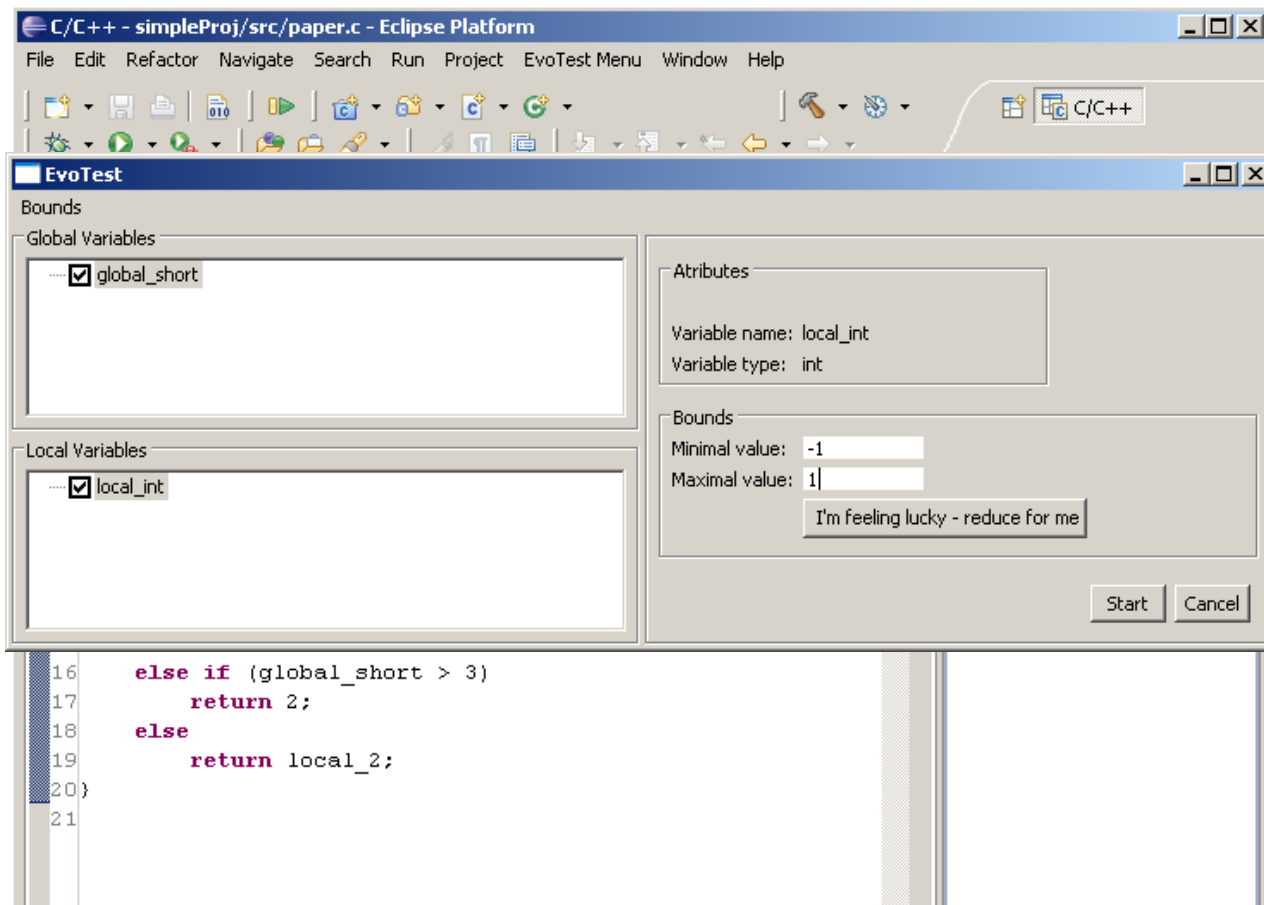
- Integration with Eclipse CDT





Application

- Integration with Eclipse CDT



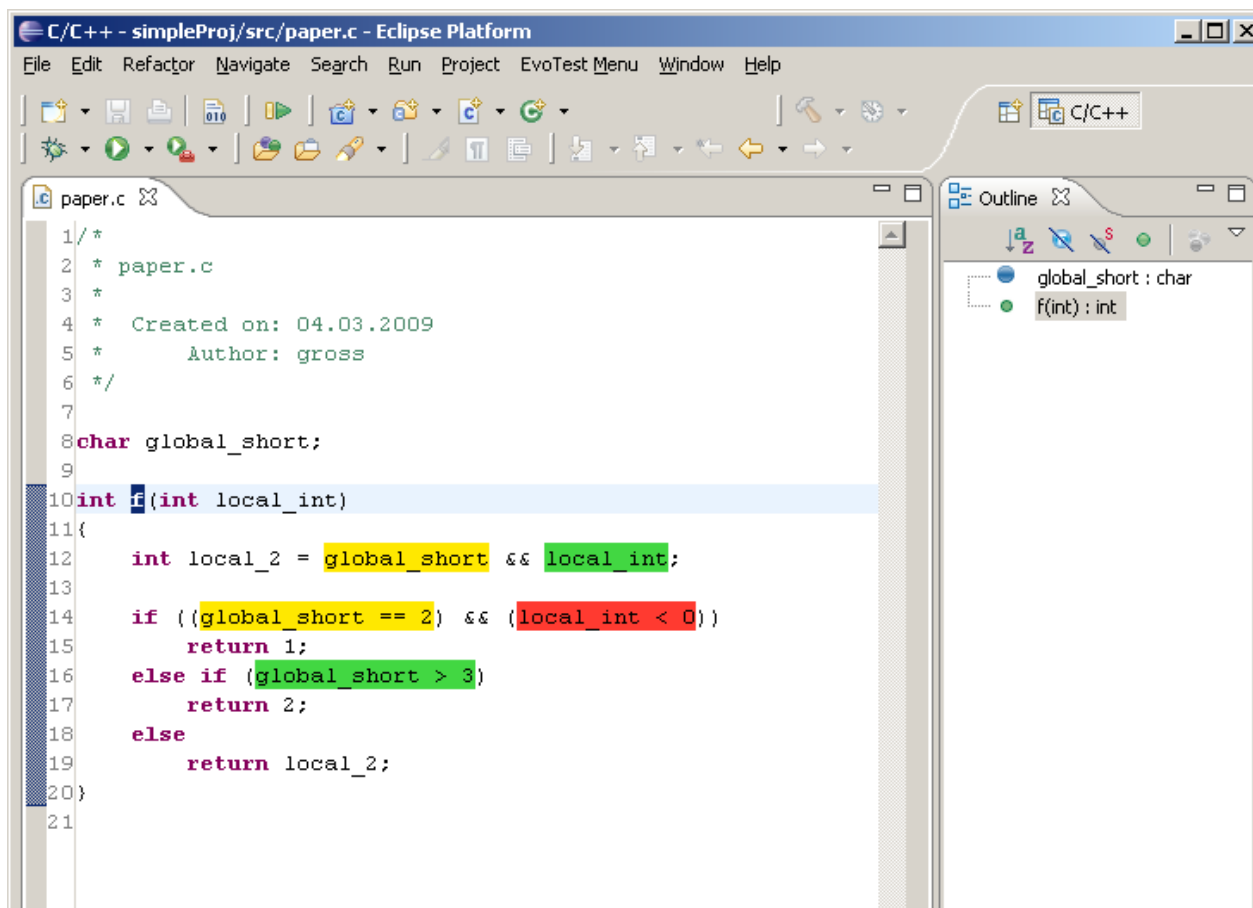
The screenshot shows the Eclipse IDE interface with the EvoTest dialog box open. The dialog box is titled "EvoTest" and has a "Bounds" section. It is divided into "Global Variables" and "Local Variables" sections. In the "Global Variables" section, the variable "global_short" is listed with a checked checkbox. In the "Local Variables" section, the variable "local_int" is listed with a checked checkbox. To the right of these sections, there are fields for "Attributes", "Variable name: local_int", and "Variable type: int". Below these, there are "Bounds" fields for "Minimal value: -1" and "Maximal value: 1". A button labeled "I'm feeling lucky - reduce for me" is located below the bounds fields. At the bottom right of the dialog box, there are "Start" and "Cancel" buttons. The background shows a code editor with the following code:

```
16     else if (global_short > 3)
17         return 2;
18     else
19         return local_2;
20 }
21
```



Application

- Result view



```
C/C++ - simpleProj/src/paper.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project EvoTest Menu Window Help
C/C++
paper.c
1/*
2 * paper.c
3 *
4 * Created on: 04.03.2009
5 * Author: gross
6 */
7
8char global_short;
9
10int f(int local_int)
11{
12    int local_2 = global_short && local_int;
13
14    if ((global_short == 2) && (local_int < 0))
15        return 1;
16    else if (global_short > 3)
17        return 2;
18    else
19        return local_2;
20}
21
```

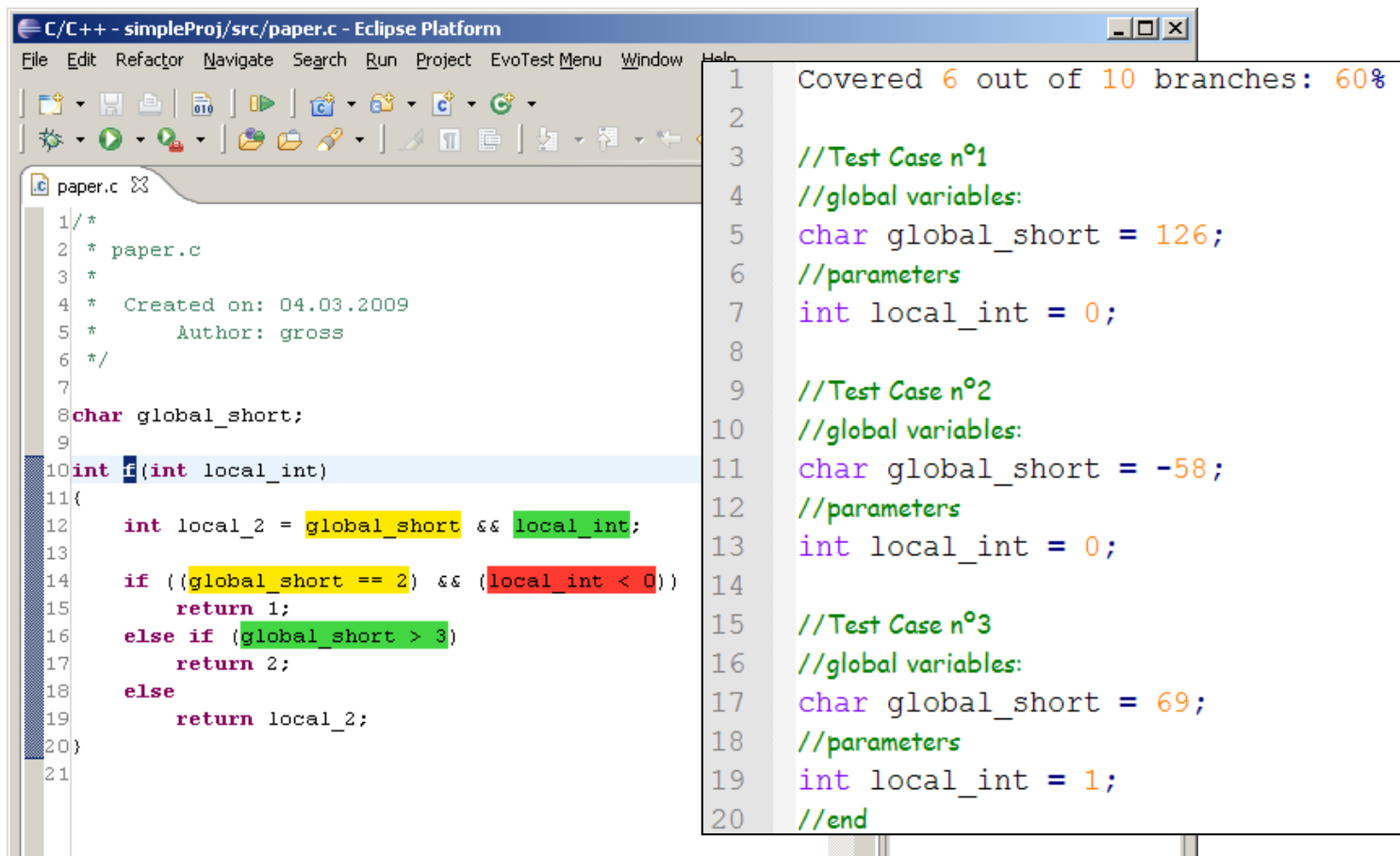
Outline

- global_short : char
- f(int) : int



Application

- Result view



The screenshot shows the Eclipse IDE interface. The main editor displays the source code for `paper.c`. The code includes a function `f` that takes an integer parameter `local_int` and returns an integer. The function uses a global variable `global_short` and a local variable `local_2`. The code is annotated with yellow and green highlights, indicating code coverage. The right-hand side of the IDE shows a coverage report for the function `f`, indicating that 6 out of 10 branches were covered, resulting in a 60% coverage rate. The report lists three test cases, each with its own set of global and local variables.

```
C/C++ - simpleProj/src/paper.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project EvoTest Menu Window Help
1 Covered 6 out of 10 branches: 60%
2
3 //Test Case n°1
4 //global variables:
5 char global_short = 126;
6 //parameters
7 int local_int = 0;
8
9 //Test Case n°2
10 //global variables:
11 char global_short = -58;
12 //parameters
13 int local_int = 0;
14
15 //Test Case n°3
16 //global variables:
17 char global_short = 69;
18 //parameters
19 int local_int = 1;
20 //end

paper.c
1 /*
2  * paper.c
3  *
4  * Created on: 04.03.2009
5  * Author: gross
6  */
7
8 char global_short;
9
10 int f(int local_int)
11 {
12     int local_2 = global_short && local_int;
13
14     if ((global_short == 2) && (local_int < 0))
15         return 1;
16     else if (global_short > 3)
17         return 2;
18     else
19         return local_2;
20 }
21
```



Results

Case Study	Total Functions	Functions containing <code>if</code> or <code>while</code> statements	Test case generation successful
A	2	2	1 (1)
B	77	44	34 (48)
C	486	70	30 (50)
D	197	67	3 (27)
All	762	183	68 (126)

- Coverage
 - 37 % with early version
 - 69 % with later version



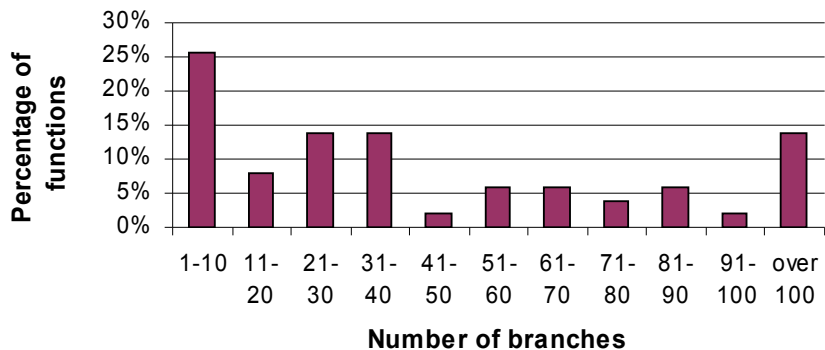
Results

Reason for failure to generate test cases	Functions	
Pointer to simple type	62	0
Array operations / Pointer to arrays	43	43
Pointer to void	12	12

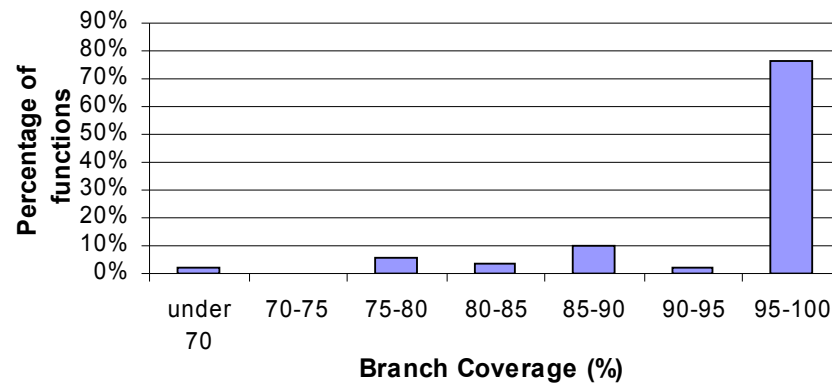


Coverage Rates

Distribution of branches in functions from Case Studies A & B



Distribution of branch coverage achieved by ETF Structural Test tool for Case Studies A and B





Way forward

- **Parameter Reduction**

- Large search space increases duration of test case generation
- Unnecessary parameters might be of unsupported type

➤ Using data-flow analysis, increased coverage of software modules could be achieved

As described by M. Harmen et al. “The Impact of input domain reduction on search-based data generation”, 2007

- **Support for pointers**

- Major reason for failure of test-case generation
- Pointers are too common to not support them

➤ Has been established in later version of tool

Approach described by M. Prutkina & A. Windisch “Evolutionary structural testing of software with pointers”, 2008



Way forward

- **Volatile variables**
 - Variables can be memory-mapped to I/O registers
 - Variable values can change asynchronously to program flow
- Since the function under test is executed in isolation, changes to volatile variables need to be simulated
- **Multi-Function Instrumentation**
 - Test case generation only for single isolated functions
 - Called functions are not yet instrumented
- Optimizing the order of evaluation of single functions can lead to a reduction in generated test data for the module as a whole
- **GUI Improvements**



Conclusion

- The tool requires the source code under test to be preprocessed
- Despite the tool still being a prototype, function coverage of 69% was possible
- Significant work is still required before evolutionary structural testing is ready for industrial application
- Nevertheless this promising approach deserves further research



- Thank you

Peter M. Kruse
Berner & Mattner Systemtechnik
peter.kruse@berner-mattner.com