SEMINAL Workshop, 9.-10. Oct. 2002

Connection between Complexity Measures and Evolutionary (Structural) Testability

Joachim Wegener, Harmen Sthamer, Kerstin Buhr
DaimlerChrysler AG, Research and Technology
Joachim.Wegener@DaimlerChrysler.com

- Introduction
- Experiments
  - Overview
  - Standard Complexity Measures
  - Relationship to Evolutionary Testability
- Conclusion

Definitions and Motivation

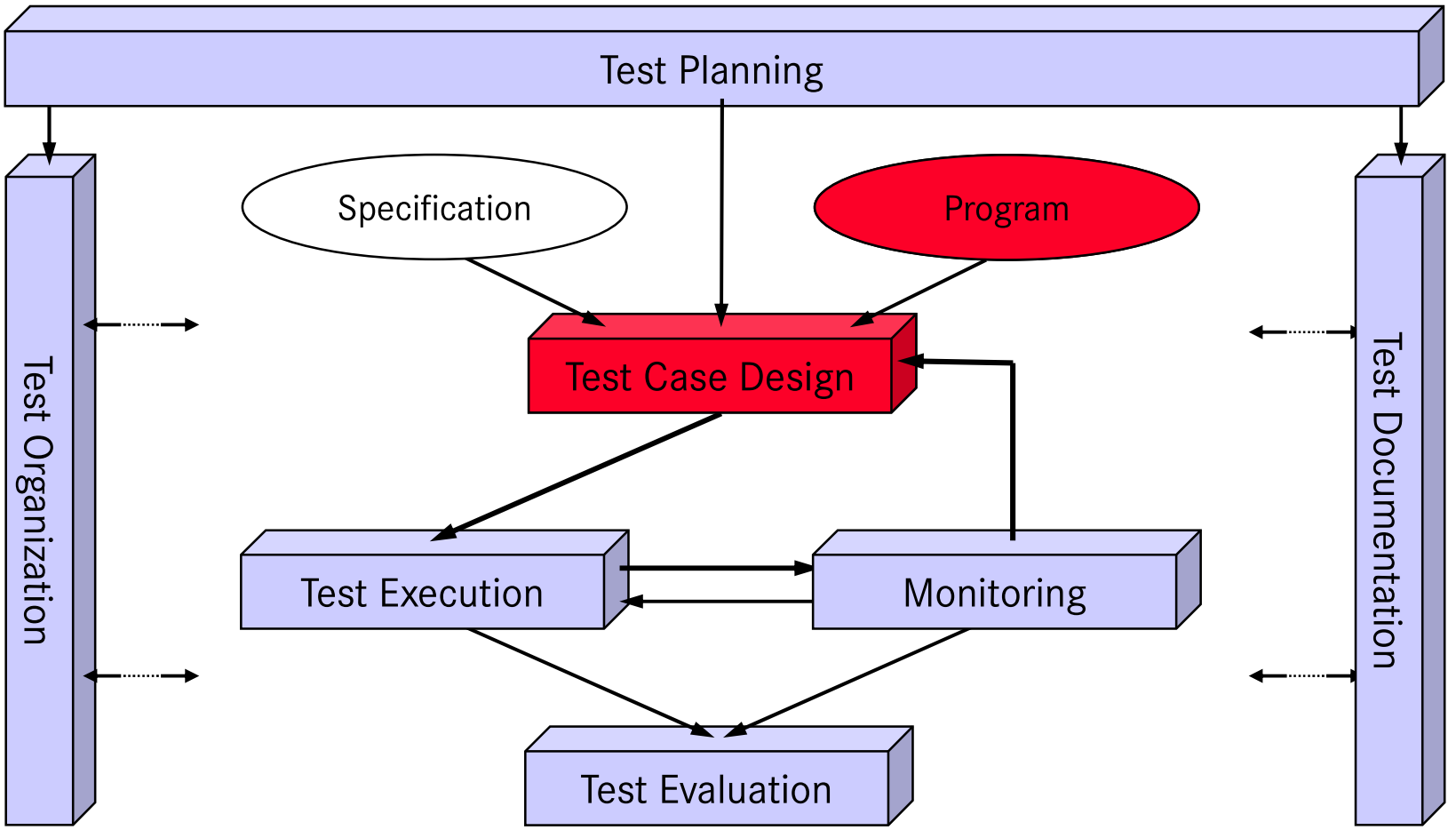
Def. Testability: the degree to which a system facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE Standard Computer Dictionary]

Def. Evolutionary Testability: the degree to which a system facilitates the establishment of test criteria and the performance of evolutionary tests to meet those criteria

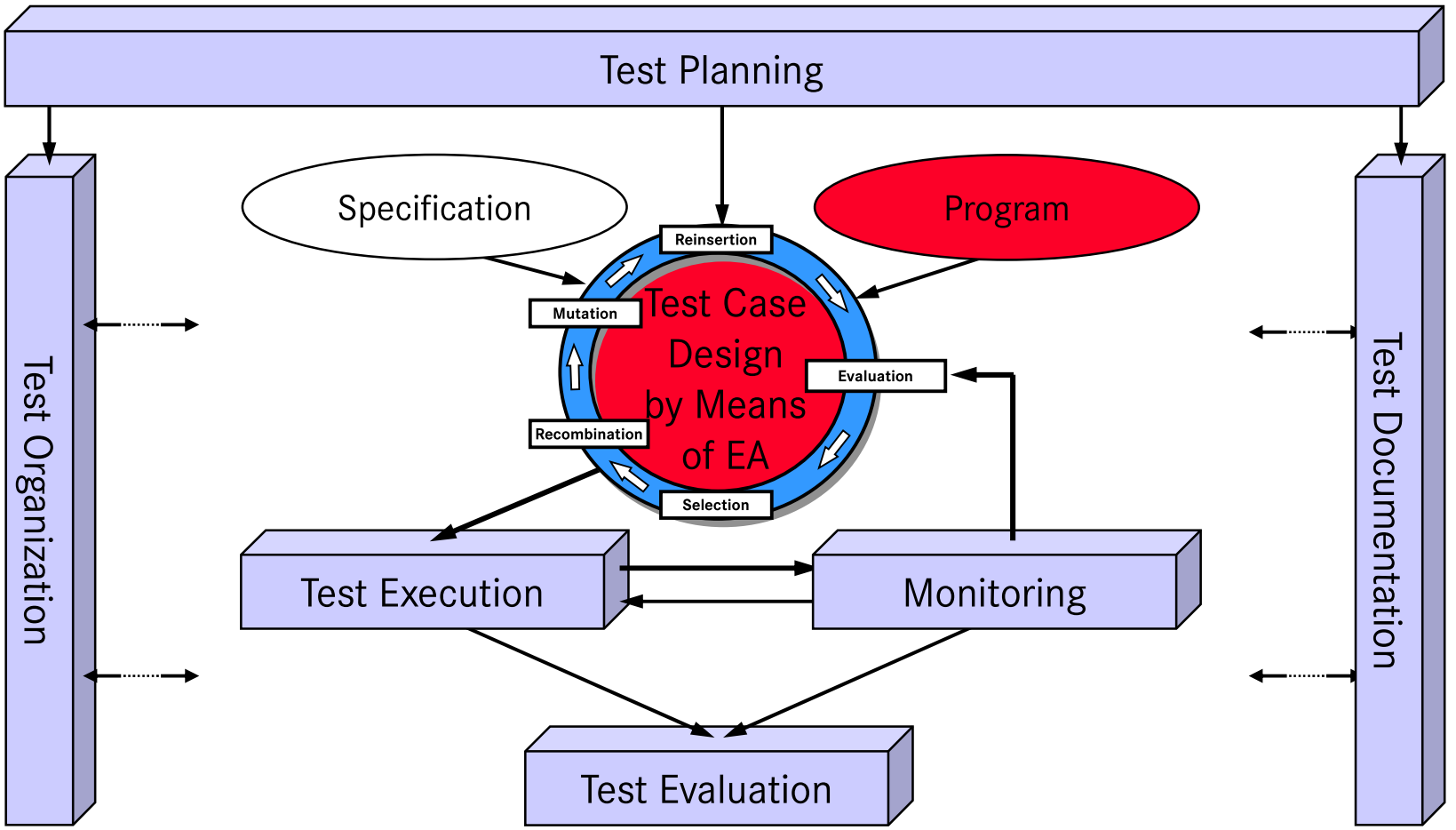
Def. Evolutionary Structural Testability: the degree to which a system facilitates the performance of evolutionary structural tests to meet structure-oriented test criteria

- Questions:
  - Is it appropriate to apply evolutionary testing to automate structural test case design for a given test object?
  - Which coverage seems to be achievable by the evolutionary structural test?
  - Should we apply transformations? Can we verify the improvement of evolutionary testability?

Test Case Design by Means of Evolutionary Testing



Test Case Design by Means of Evolutionary Testing

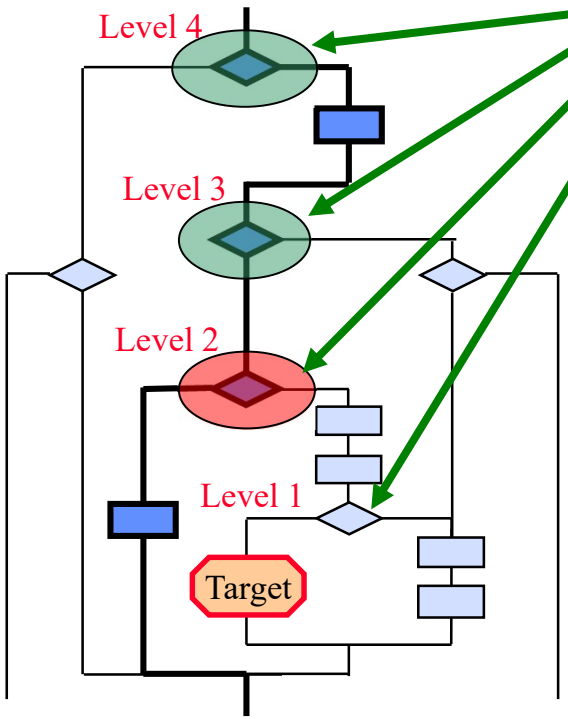


Structure-oriented Test Case Design

```
#define UNDEFINDED 0
#define CLIENT 1
#define SERVER 2

char result[100];

int encode_number(char * input)
{
            int i=0,pp=-1,j,len,l2;
            int mode=-1;
            char c;
            char s[40];

            while ((c=input[i++])!=0 && (c=='.' || (c-='0')>0 && c<10)) if (pp==-1 && c+'0'=='.') pp=i-1;
            if (c!=0 && input[--i]!='d' && input[i++]!='h') { printf("Input sequence wrong.\n"); return -1; }
            if (c==0) mode++;       /* decimal without name */
            c+='0';
            if (c=='d') { i++; mode += 2; }
            if (c=='h') mode += 3;
            if (pp==-1 || input[--i]=='.') { mode =1; i++; } /* float without a value after point */

            switch (mode)
            {
              case 1: /* decimal */
                        l2=i;
                        if (input[i++]=='n')
                        {
                                    l2=i;
                                    strcpy(&s[0],&input[l2]) ; /* decimal with name */
                                    strcat(s,"_");

                        } else /* wout name: */
                        . . .
```

Analysis of program code in order to determine the test relevant input situations that errors can be detected with?

Fitness Function for Structure-oriented Testing



1. Approximation level

- Identify relevant branching statements for target node on the basis of control-flow graph
- Relevant branching statements can lead to a miss of the desired target
- In this sense approximation-level corresponds to 'distance from target'

2. Local distance calculation in the branching statements with undesired branching

- Evaluation of predicate in a branching condition in the same manner as described for safety testing, e.g.
  if A = B ⇨ Local_Distance = | A - B |

↪ Fitness = Approximation_Level + Local_Distance

Structure-based Complexity Measures

- Complexity measures are often used to assess understandability and testability of programs
- Gross has shown in his work that standard complexity measures do not seem to be appropriate to assess the testability of programs for temporal behavior testing
- Nevertheless, since most of the existing complexity measures are structure-based
  - →first impression: high values of complexity measures lead to lower evolutionary testability for automation of structural testing
  - →Investigate the suitability of structure-based complexity measures to assess evolutionary structural testability of test objects
    - Executable Lines of Code
    - Myers Interval
    - Cyclomatic Complexity
    - Nesting Level Complexity
    - Halstead's Vocabulary
    - Halstead's Length
    - Number of Test Aims

Experiments

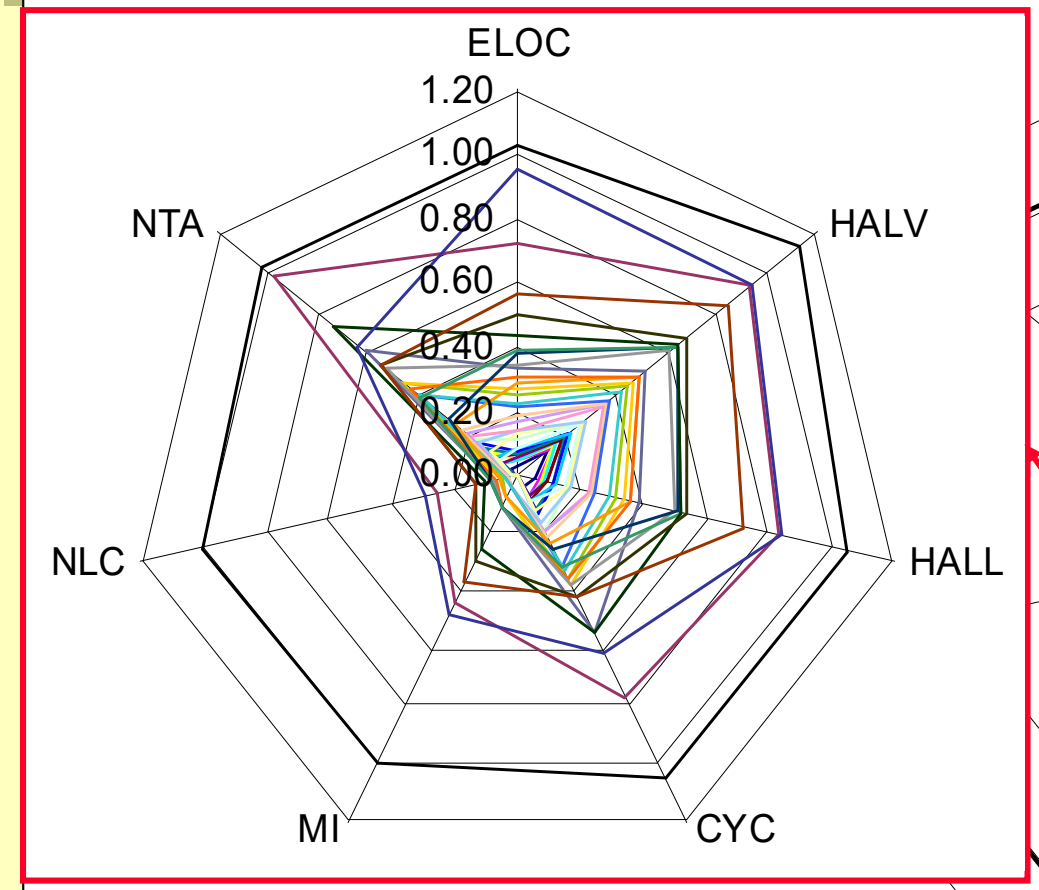
Experiment Settings

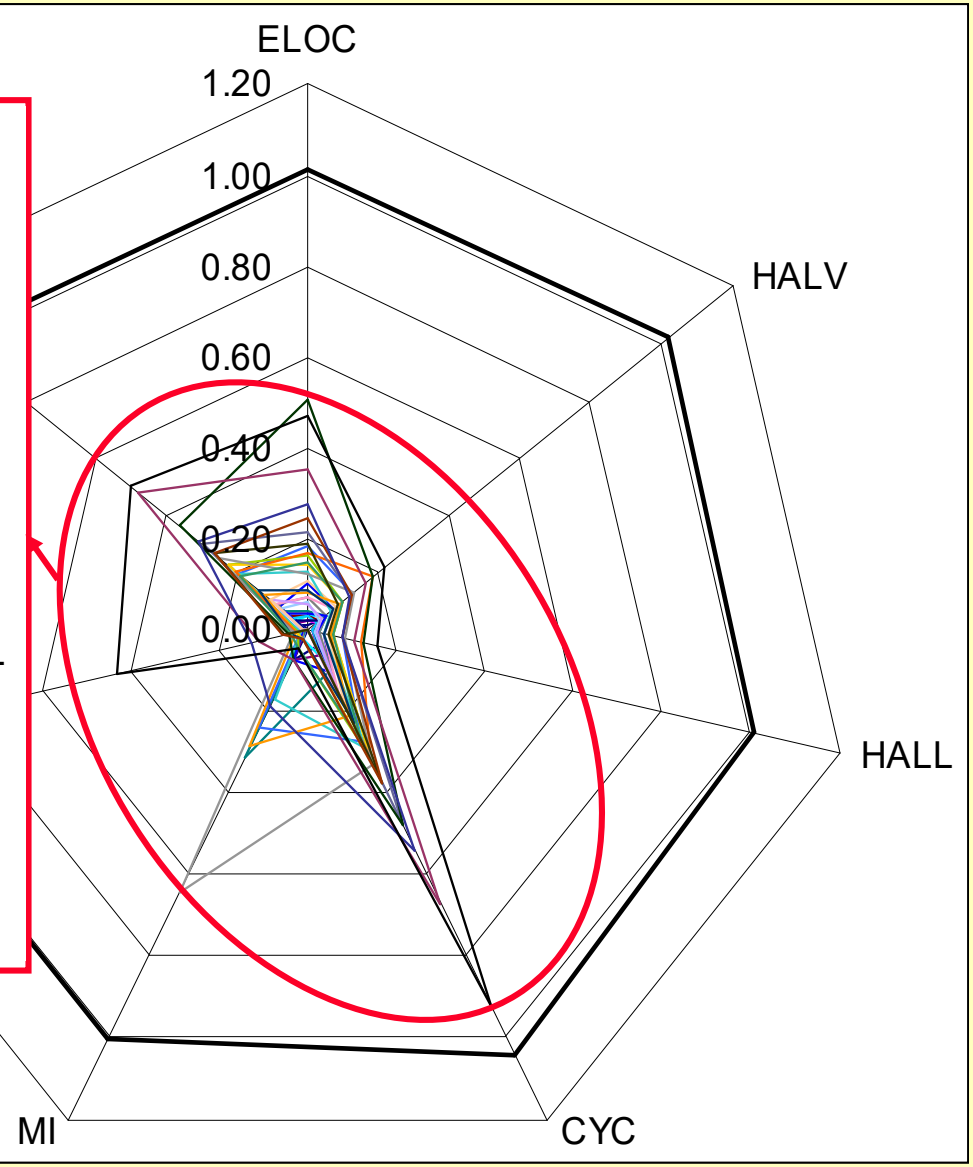
- 33 test objects (C functions)
  - from different application fields (e.g. functions for mathematical calculations, string manipulation, engine electronics, vehicle dynamics, and interior systems)
  - with large value spectrum for the different complexity measures

- branch tests performed with the DaimlerChrysler Evolutionary Test System
  - 10 test repetitions for each test object
  - population size (9 * 100 individuals), rank-based fitness assignment, generation gap 90%, migration, competition
  - limit of 200 generations for each single test aim (~162.000 individuals)

Experiments

Test Object Complexities

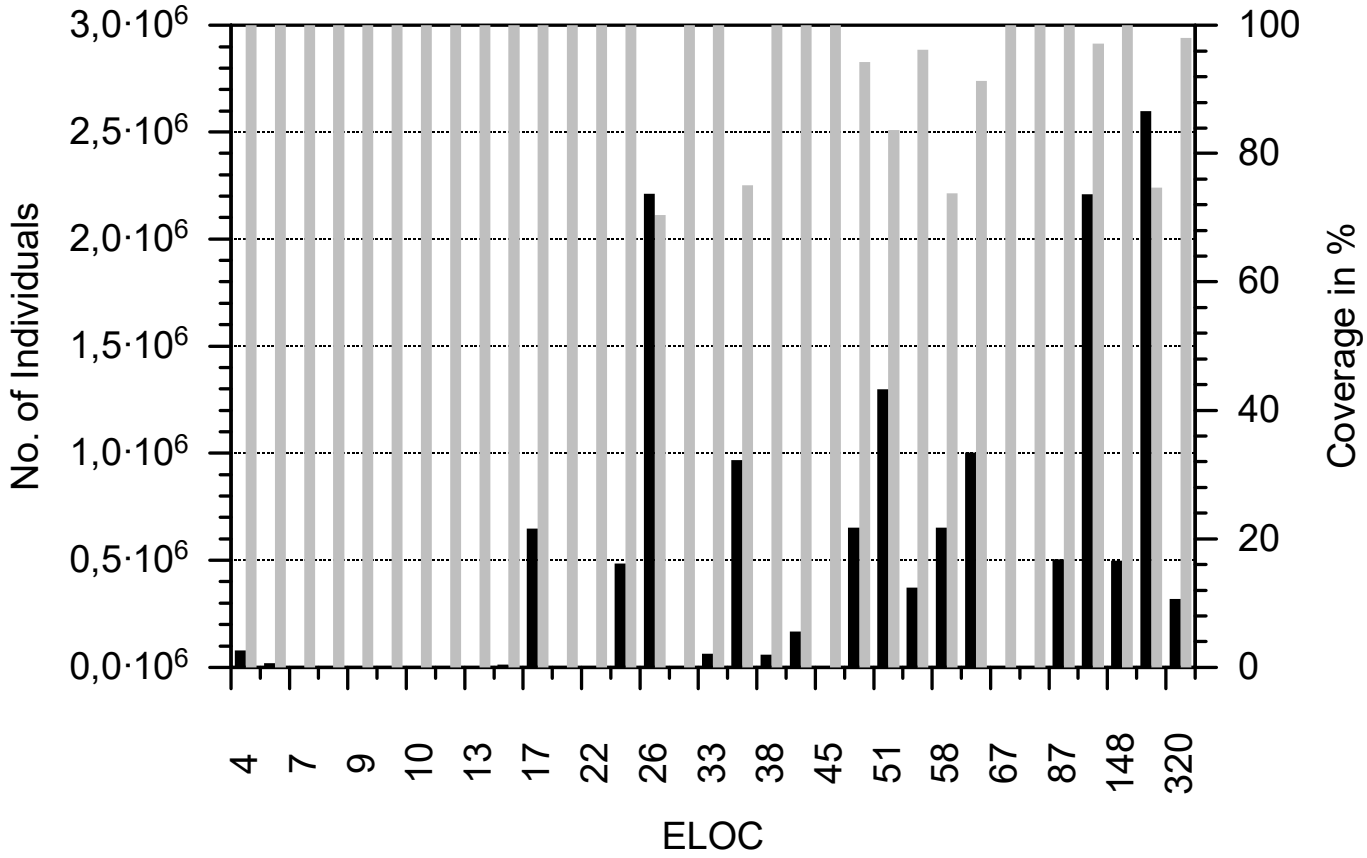




"Normalized" representation of complexity measures

Executable Lines of Code

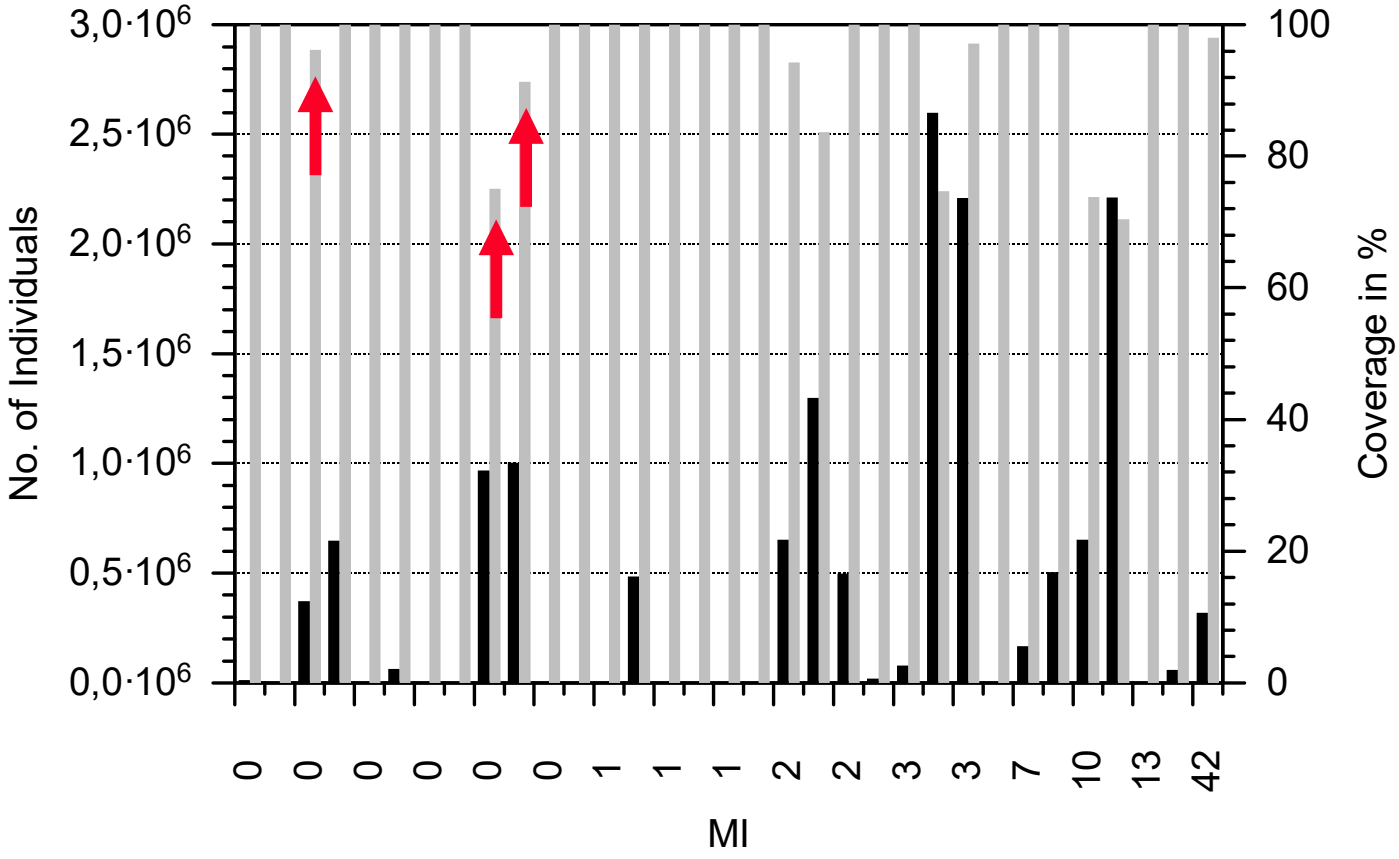
ELOC = lines of code - blank lines - comment lines



Myers Interval

MI = sum of logical operators in the conditional expressions of the program under test

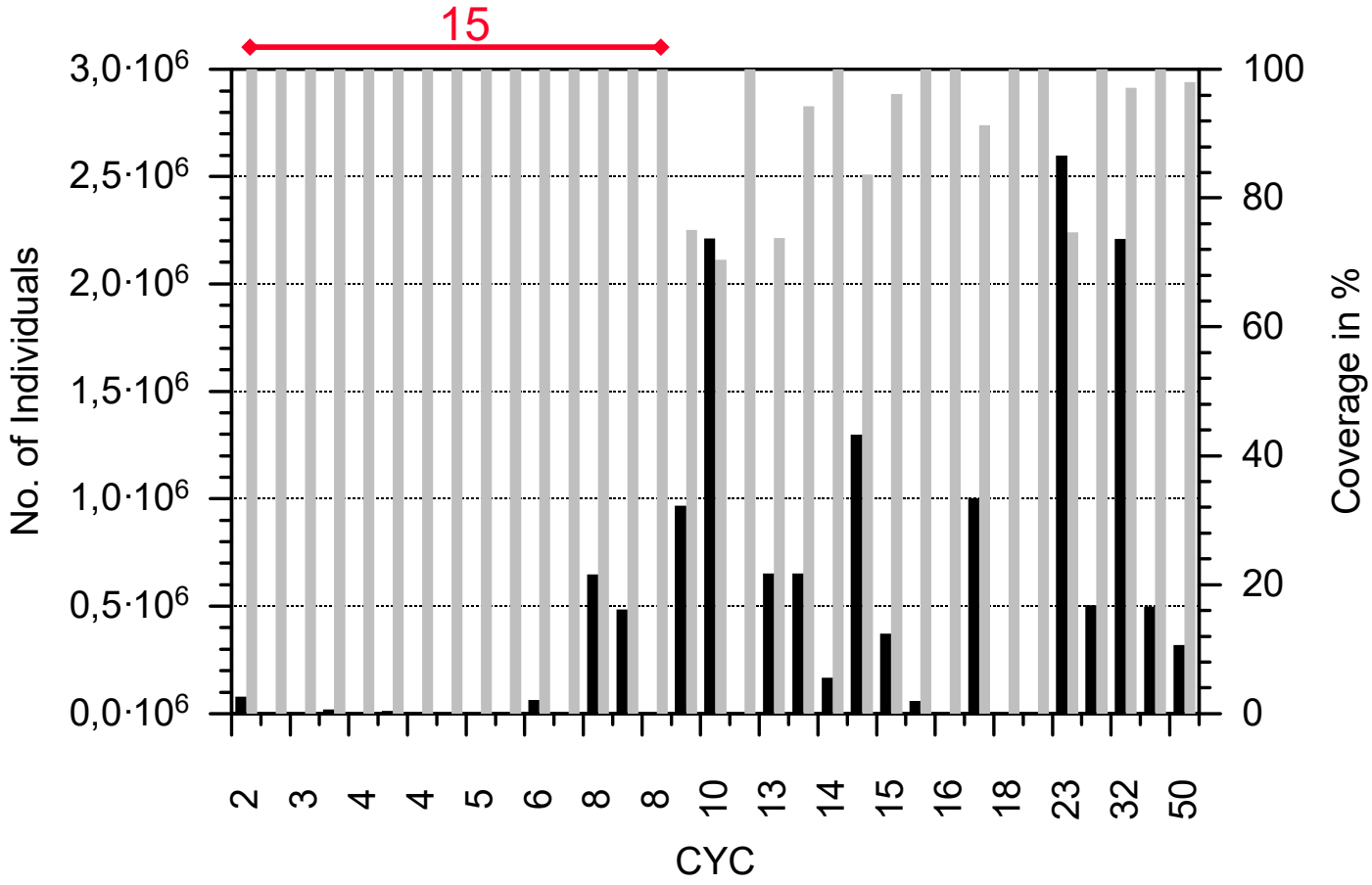
Hypothesis: the more complex the conditional expressions, the more difficult to reach certain program branches



Cyclomatic Complexity

CYC = number of cfg edges - number of cfg nodes + 1

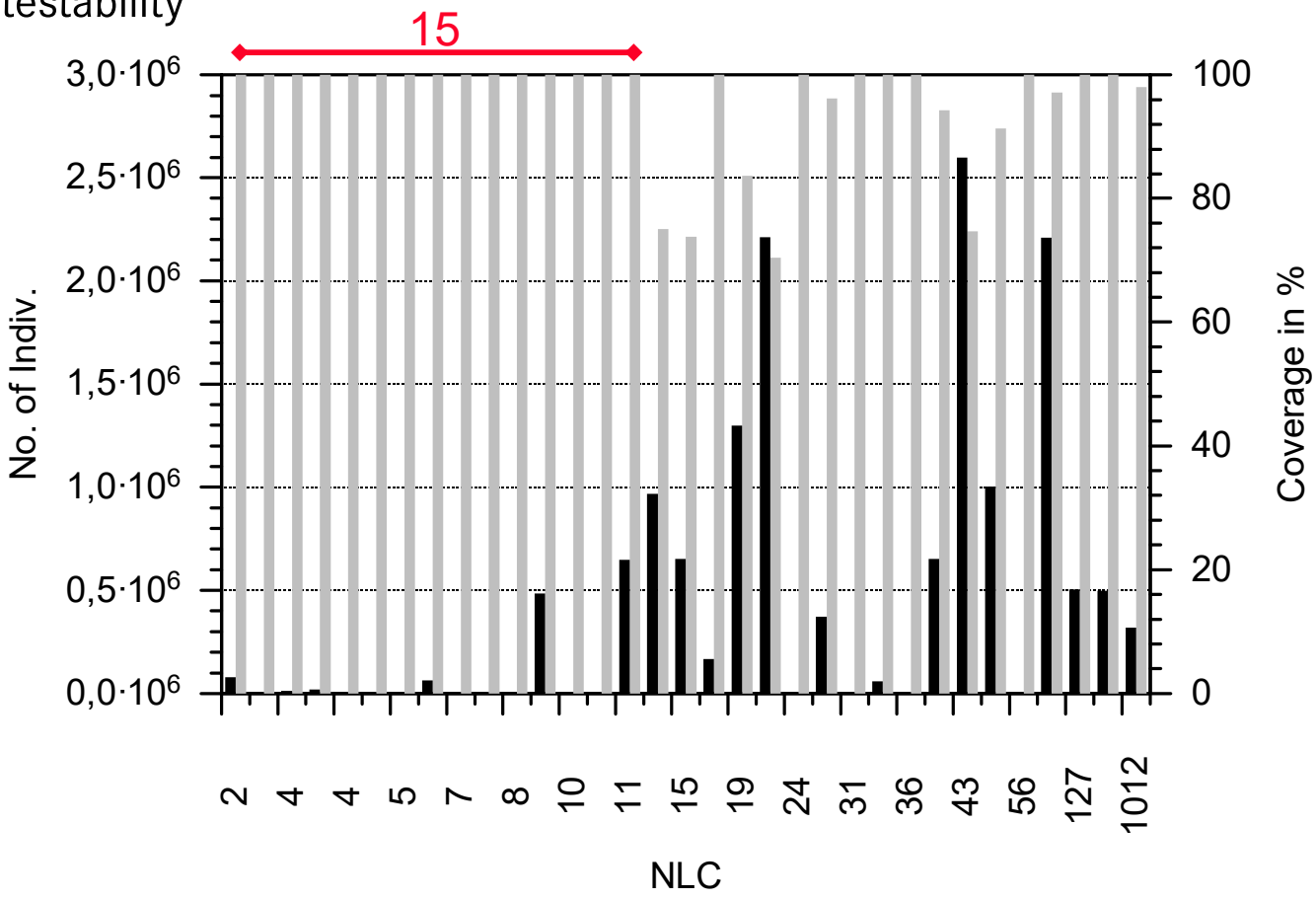
Hypothesis: a high CYC indicates a complex control flow which leads to a decreased testability



Nesting Level Complexity

NLC = sum of the nesting levels of the program's nodes

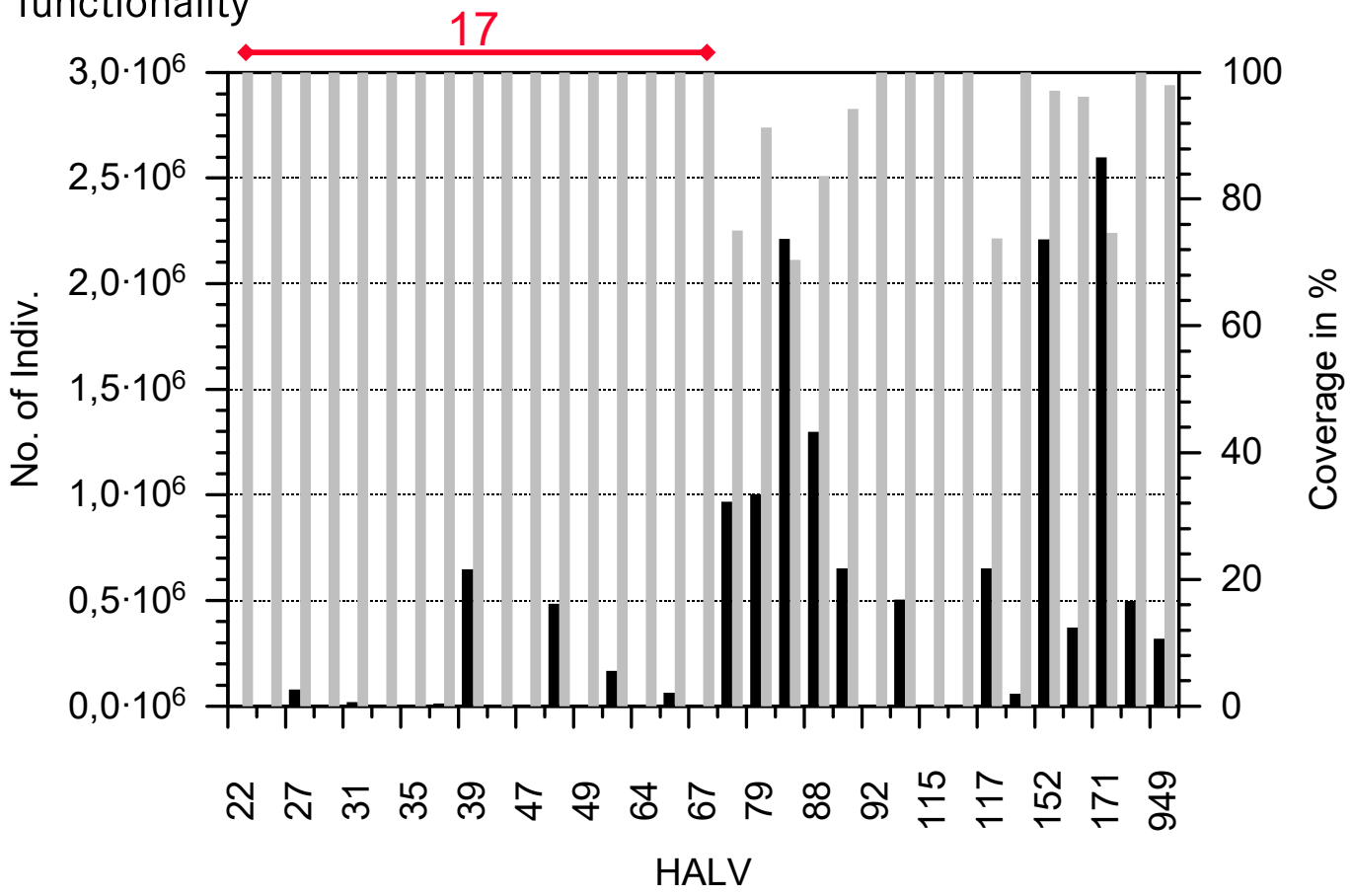
Hypothesis: a high value of NLC indicates many nested nodes which leads to a decreased testability



Halstead's Vocabulary

HALV = number of operators + number of operands

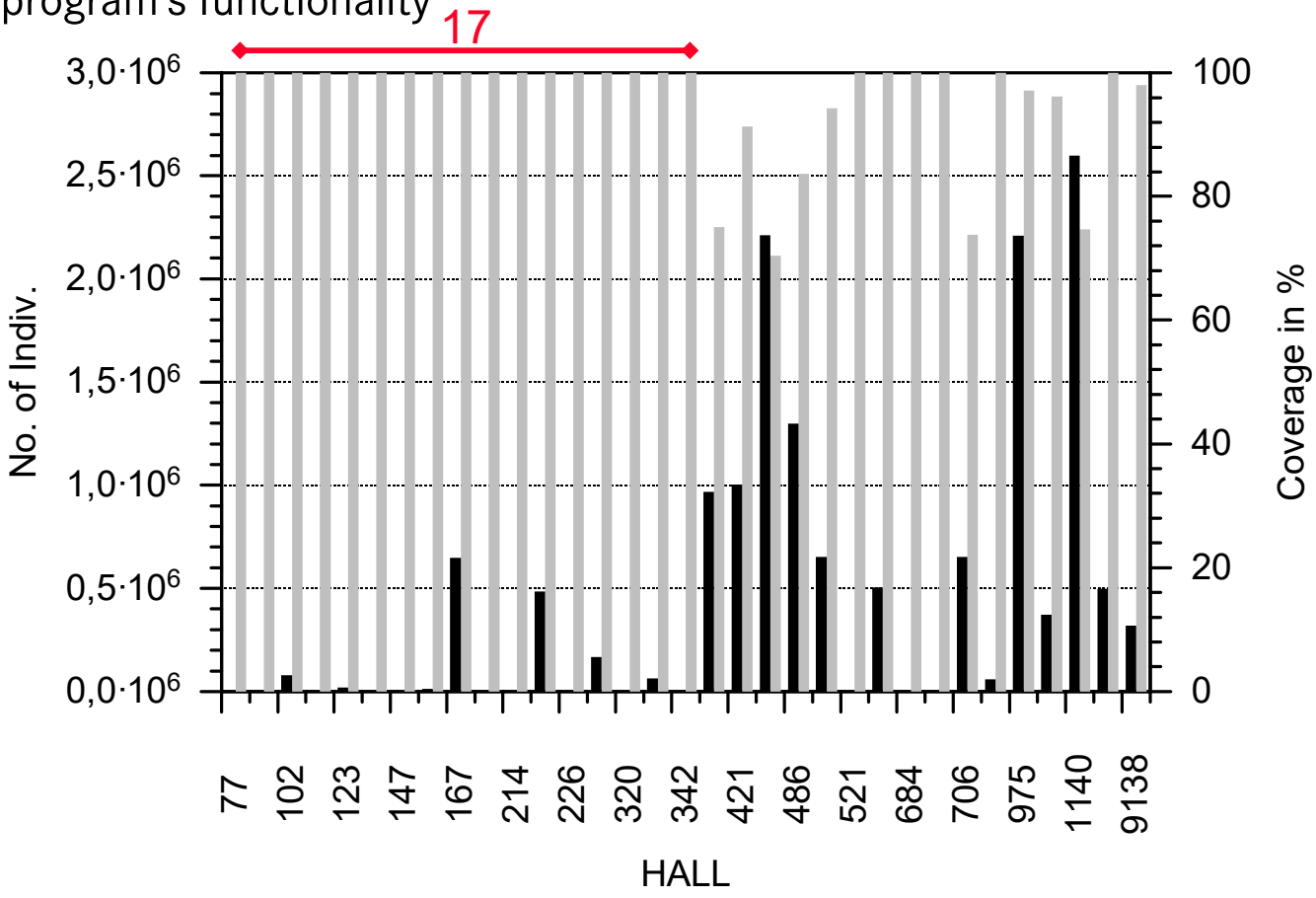
Hypothesis: the more operators and operands a program has, the more complex is its functionality



Halstead's Length

HALL = number of uses of operators + number of occurrences of operands

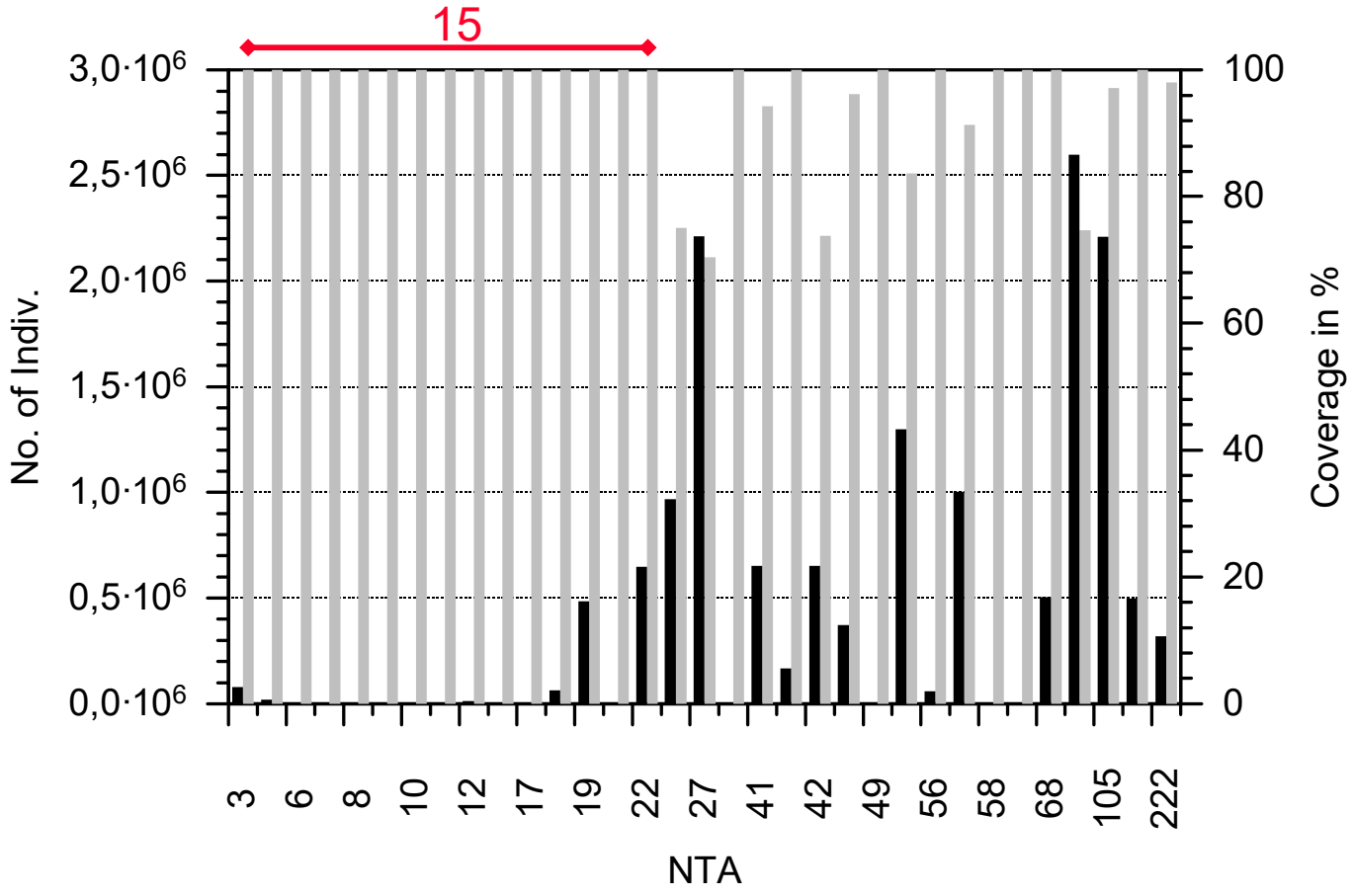
Hypothesis: the more frequently operators and operands are used, the more complex is the program's functionality



Number of Test Aims

NTA = number of program branches

Hypothesis: the more test aims a program has, the more complex is its control-flow



Conclusion

- for 23 out of 33 test objects full branch coverage was achieved, lowest branch coverage obtained was 70%
- full coverage always achieved for low program complexities, exception Myers Interval
- more inconsistent results for higher program complexities:
  - in several cases even for high complexities full coverage is achieved, but
  - in several cases for mid complexities no full coverage is achieved

→no clear relationship between established structure-based complexity measures and evolutionary testability wrt structural testing

- best results achieved for Halstead's Length and Halstead's Vocabulary