14th International Internet & Software Quality Week 2001

Evolutionary Testing of Embedded Systems

Harmen Sthamer, Andre Baresel and Joachim Wegener
DaimlerChrysler AG, Research and Technology

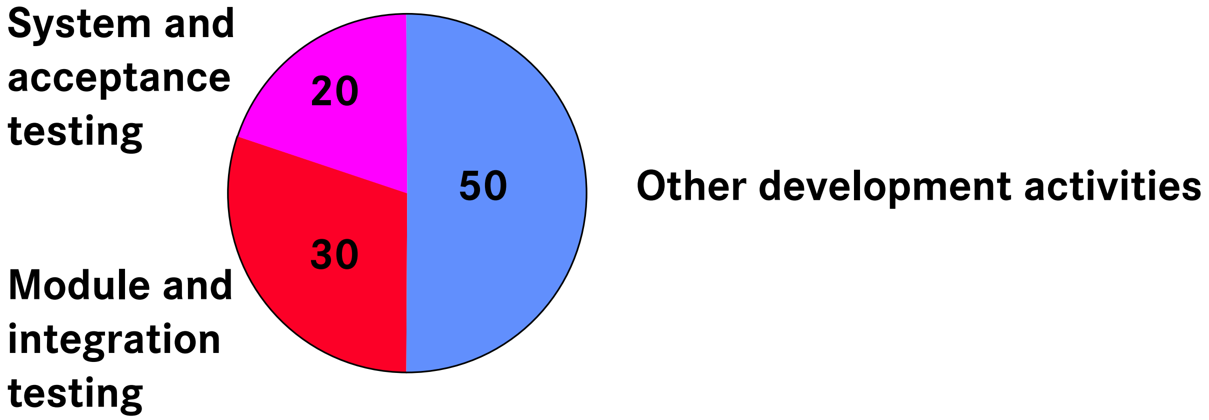
{harmen.sthamer, andre.baresel, joachim.wegener@daimlerchrysler.com}

- Introduction, Motivation
- Evolutionary Algorithms
- Evolutionary Testing (ET) and their Applications
  - Evolutionary Safety Testing
  - Evolutionary Structural Testing
    - ➥ Experiments and Results
  - Evolutionary Temporal-Behavior Testing
    - ➥ Experiments and Results
- Conclusion, Future Work

Testing in Practice

- Testing is the most important analytical quality assurance method
- Testing carries a considerable cost-factor within system development

Average distribution of software development costs for embedded systems



- Testing is too resource intensive
  ➡ high costs
- Testing is not performed systematically
  ➡ low error detection rate

Test automation

Test methods

Test Objectives

Through system execution with selected test data the test aims to

- detect errors in the system under test and
- gain confidence in the correct functioning of the test object

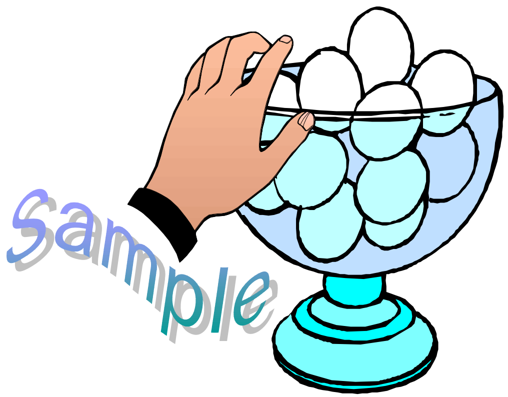
Strong Features

- takes into consideration the real environment (e.g. target computer, compiler) and
- tests the dynamic system behaviour (e.g. run-time behaviour, memory space requirement)

Weak Features

- an exhaustive test is usually impossible

Test data has to be selected according to certain test criteria



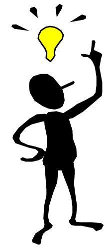
Introduction and Motivation

State of the Art

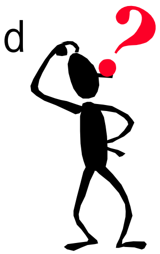
The objectives of testing embedded systems are finding errors and building up confidence in
- **functional behavior** and
- **non-functional behavior**

by executing the test object with selected inputs.

Suitable functional test methods available (e.g. CTE XL)

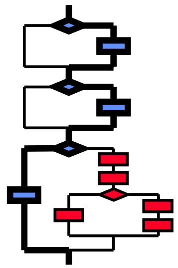


Lack of specialized test methods.



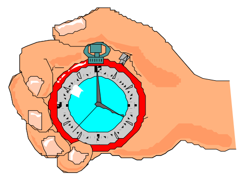
Structutral Testing

- common test approach (included in many standards)
- not possible to check whether all requirements have been implemented
- difficult to automate (limits of symbolic execution) therefore, very expensive and often neglected



Temporal-Behavior Testing

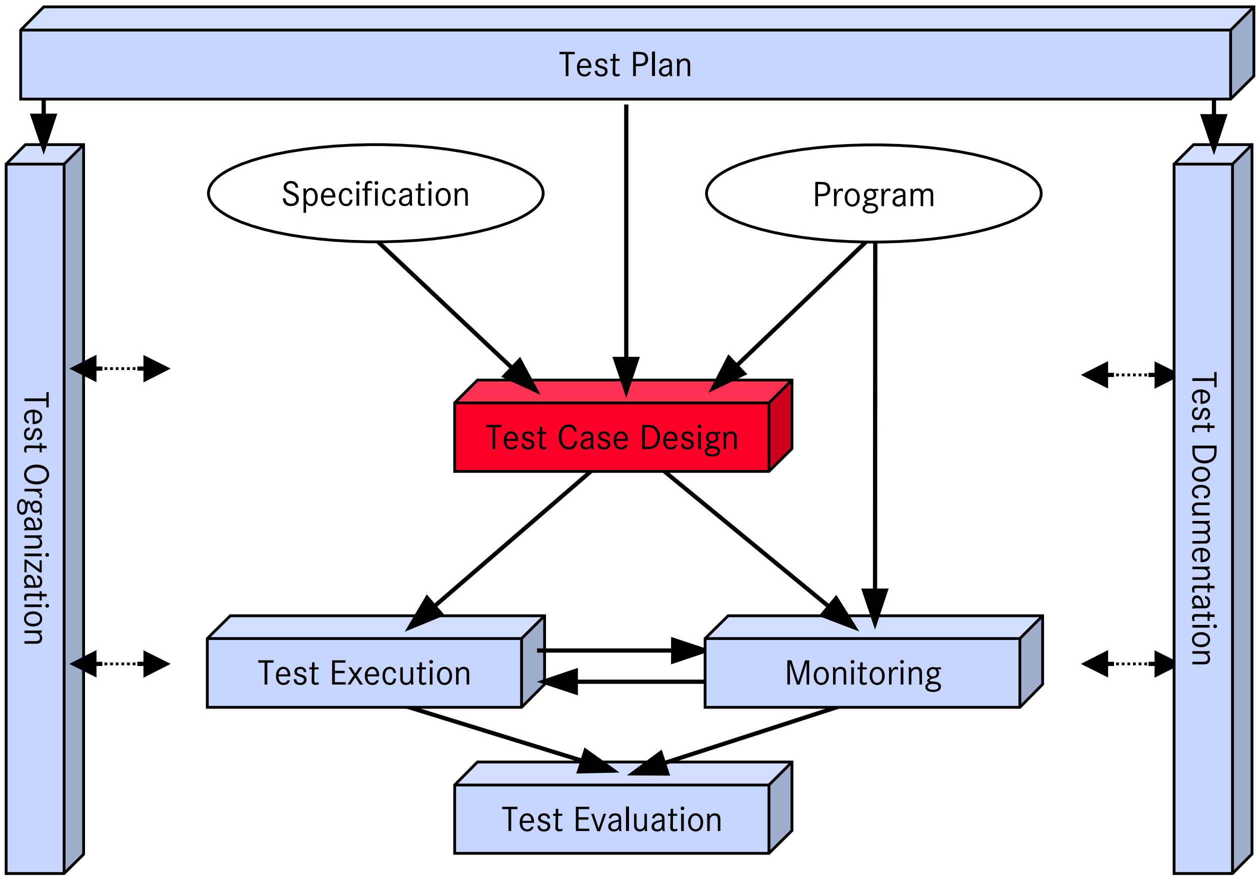
- temporal testing is difficult and very expensive
- no methods or appropriate tools are available



Test Activities

To ensure test quality, test cases need to be defined systematically.

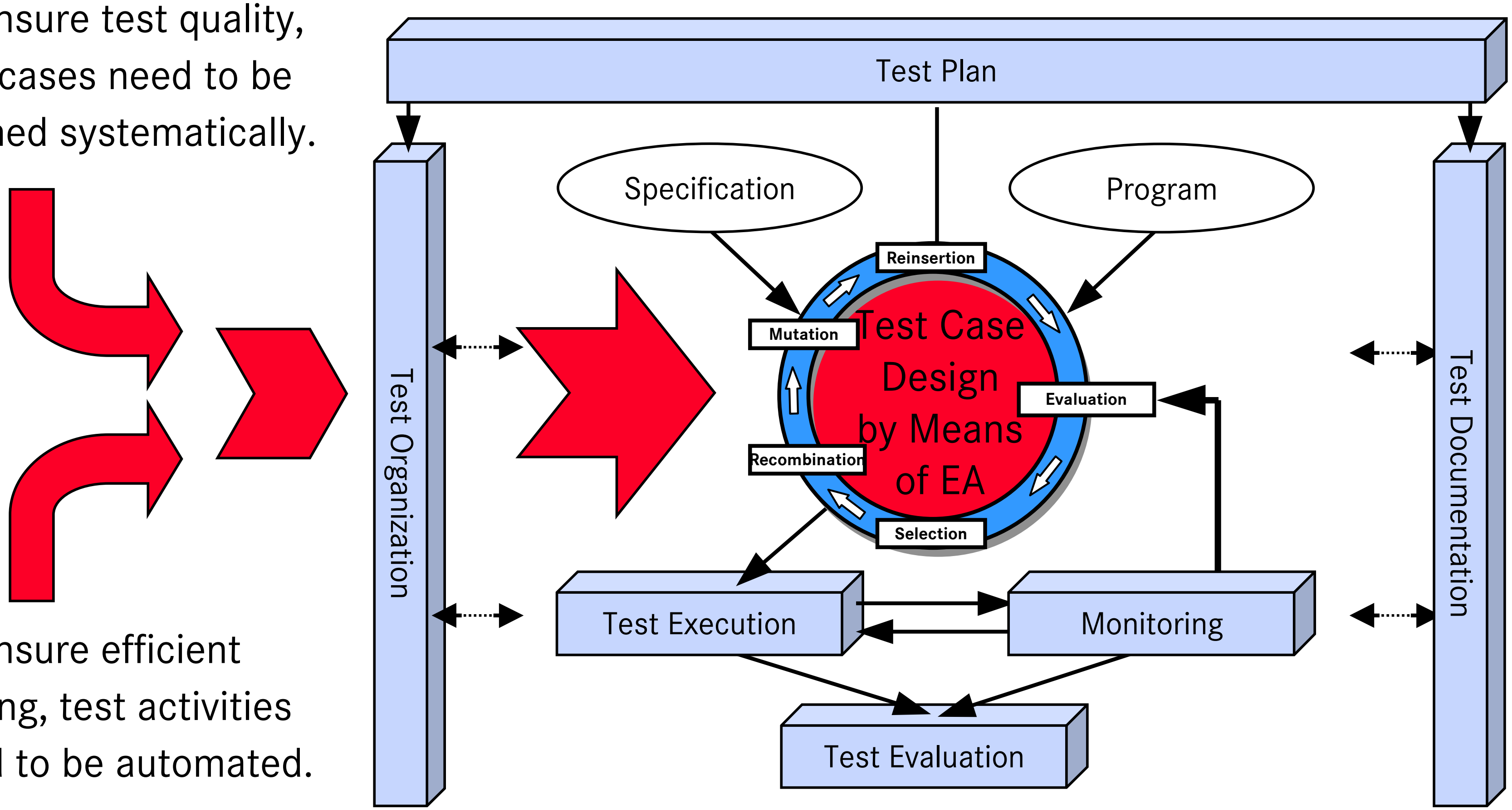
To ensure efficient testing, test activities need to be automated.



- most important for test quality
- usually performed manually

Test Activities

To ensure test quality, test cases need to be defined systematically.



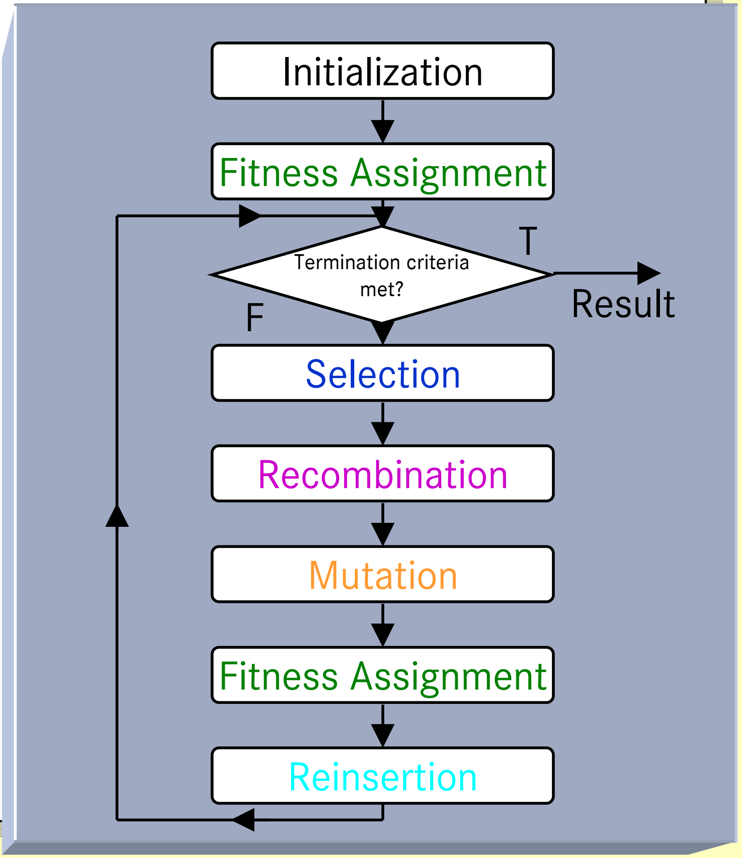
To ensure efficient testing, test activities need to be automated.

Systematic definition and automation promises to reduce testing effort (time and expenses) during the determination of relevant test data

Evolutionary Algorithms

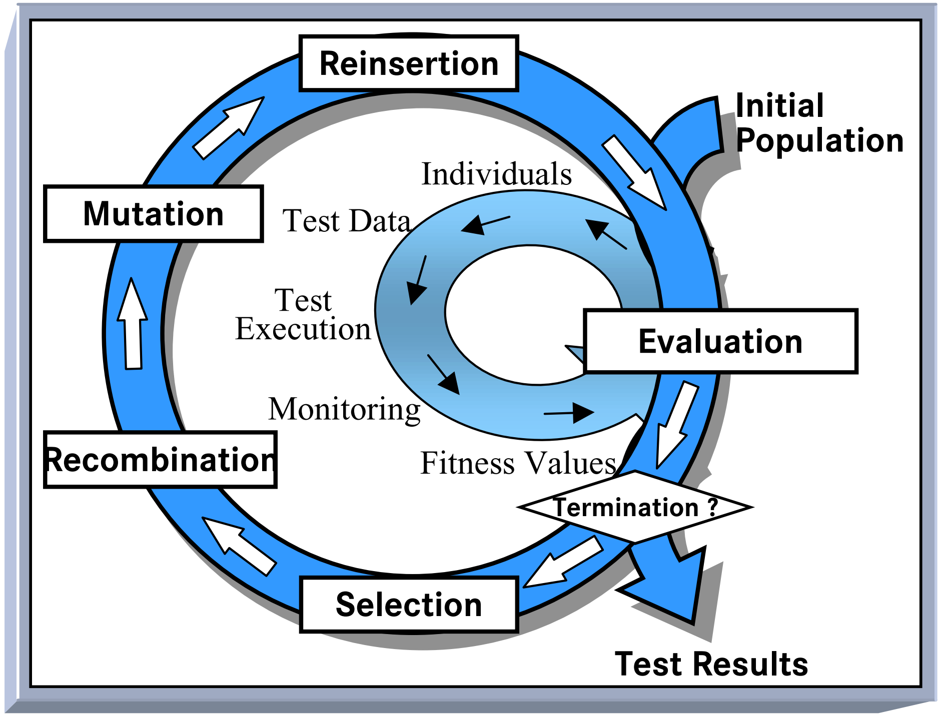
- Iterative optimization method which is based on processes of natural genetics and the theory of evolution.
- In each iteration a new population of individuals (potential problem solution) is generated and evaluated.
- From the current population new populations are generated via
  - selection,
  - recombination,
  - mutation,
  - fitness assignment, and
  - reinsertion of offspring

  until
  - an optimal solution has been found or
  - a predetermined termination criteria is met.
- Important: definition of a suitable objective function



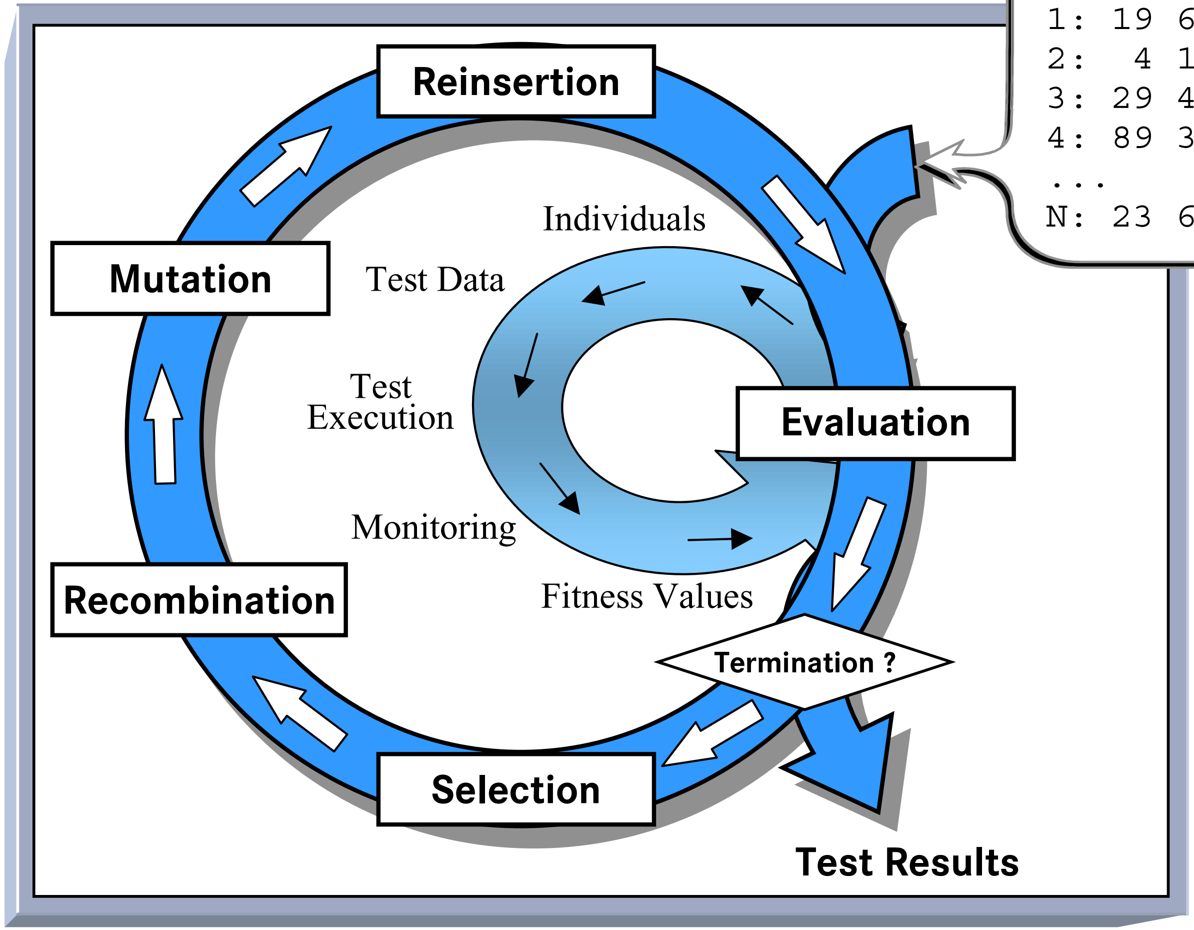
Application of Evolutionary Algorithms to Software Testing

- Input domain of test object forms the search space, in which input situations fulfilling test objectives are searched for
- Each individual represents a test datum for the system under test
- Individual fitness values are based on the monitoring results for the corresponding test datum



- Applications
  - **Testing Temporal Behavior**
  - **Structural Testing**
  - **Safety Tests**
  - Robustness Tests
  - . . .
- **Prerequisite:**
  test objective has to be defined numerically and has to be transformed into an optimization problem (suitable fitness function)

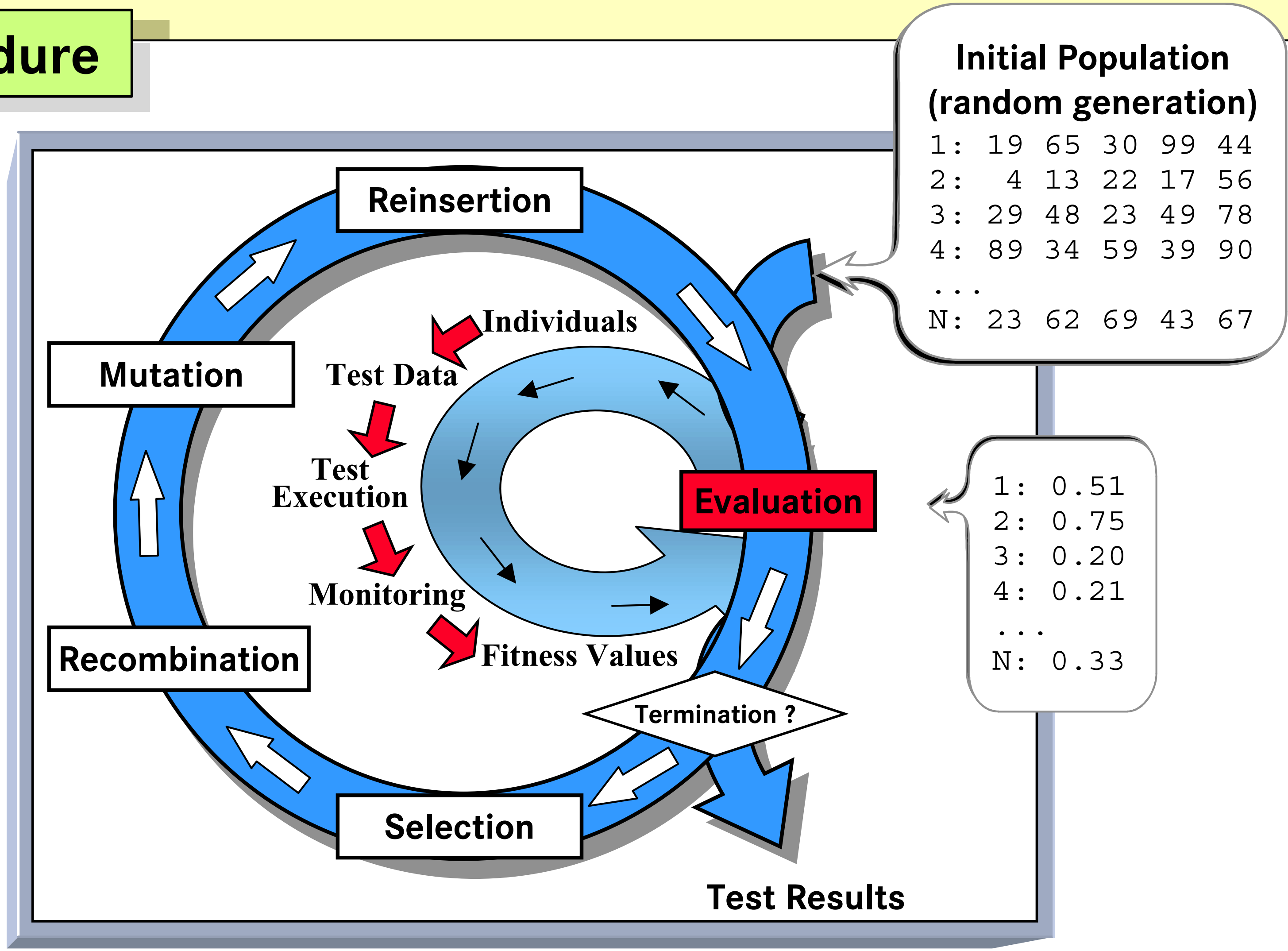
General Procedure



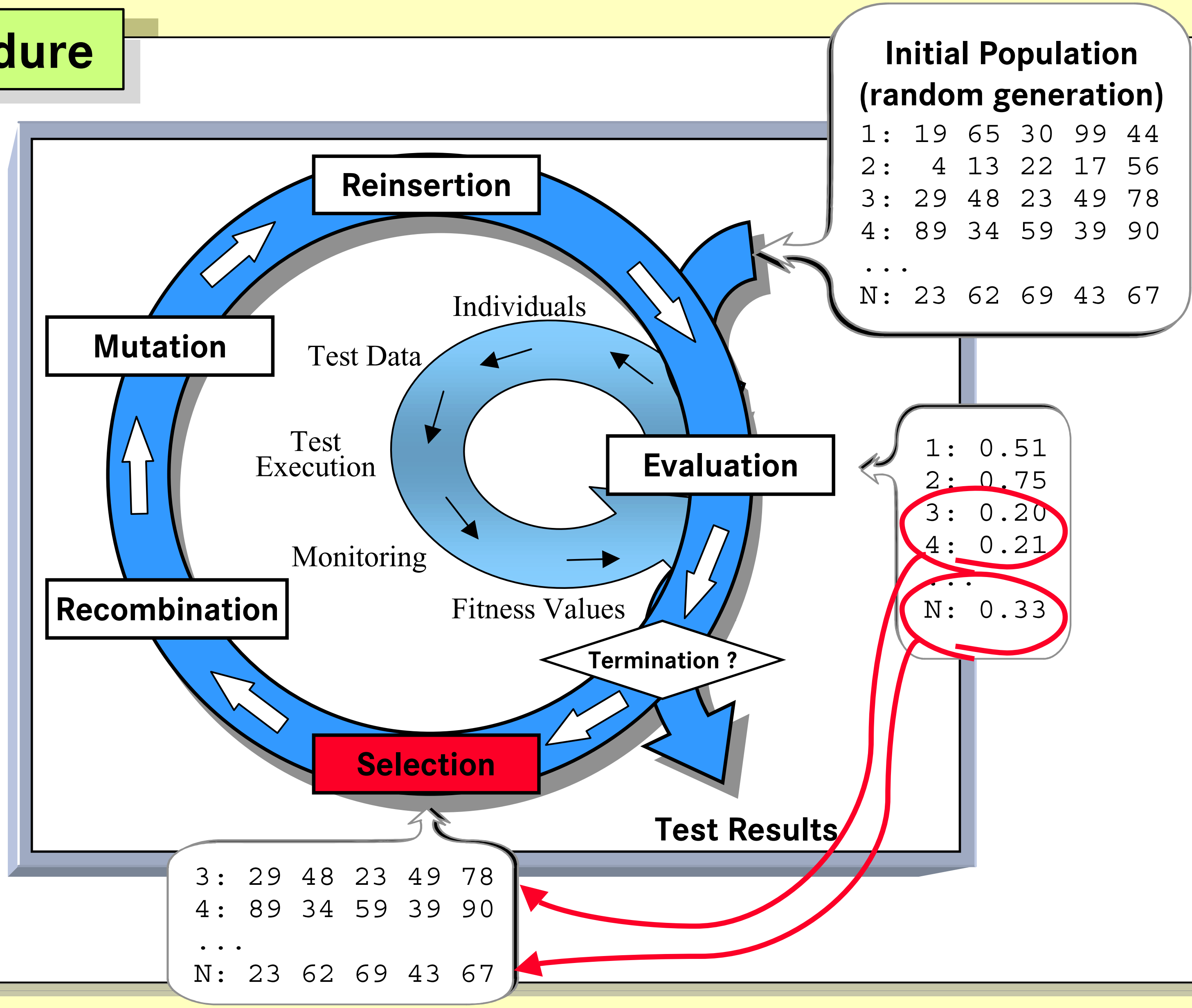
Initial Population (random generation)

```
1:  19 65 30 99 44
2:   4 13 22 17 56
3:  29 48 23 49 78
4:  89 34 59 39 90
...
N:  23 62 69 43 67
```

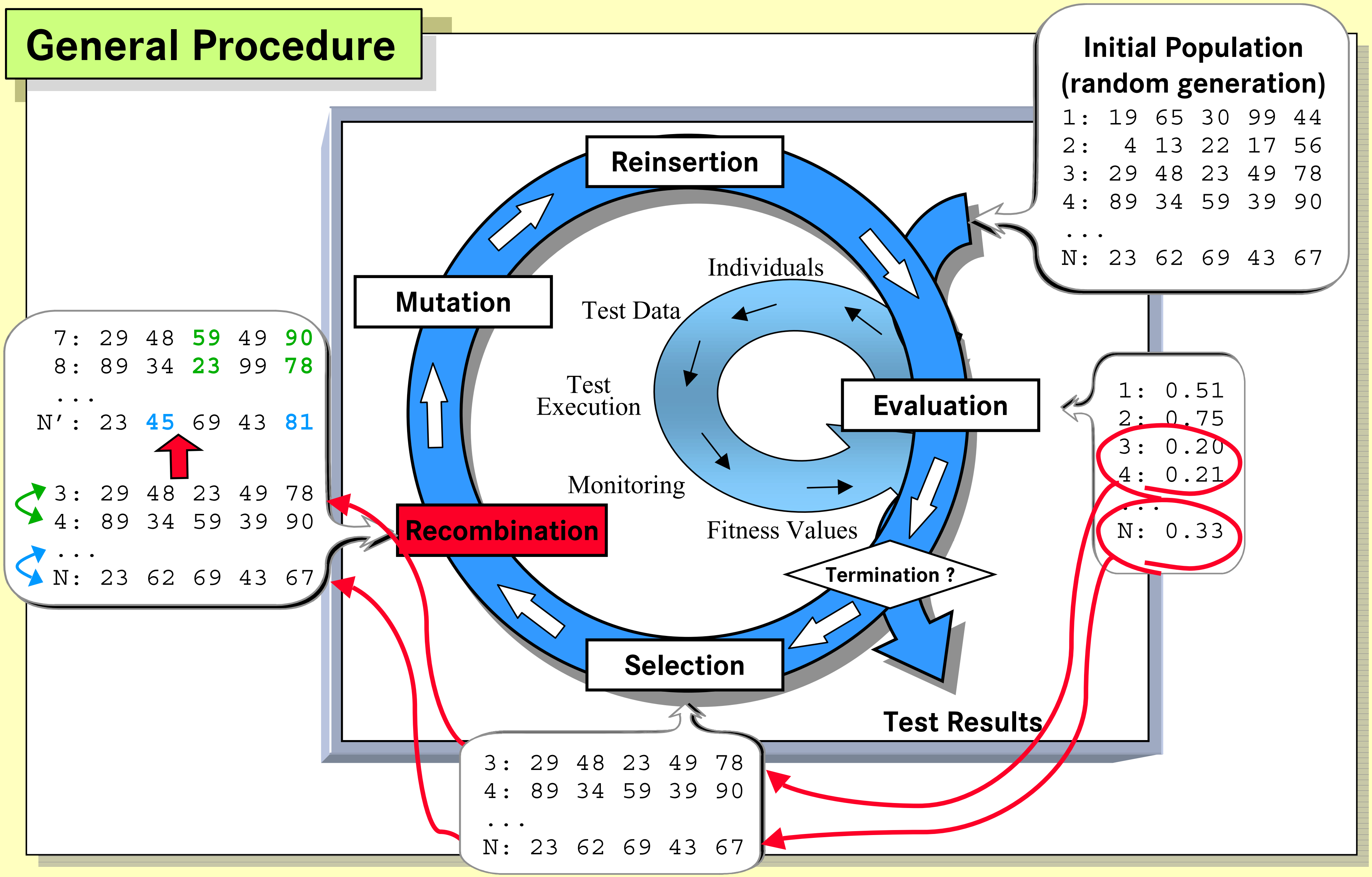
General Procedure



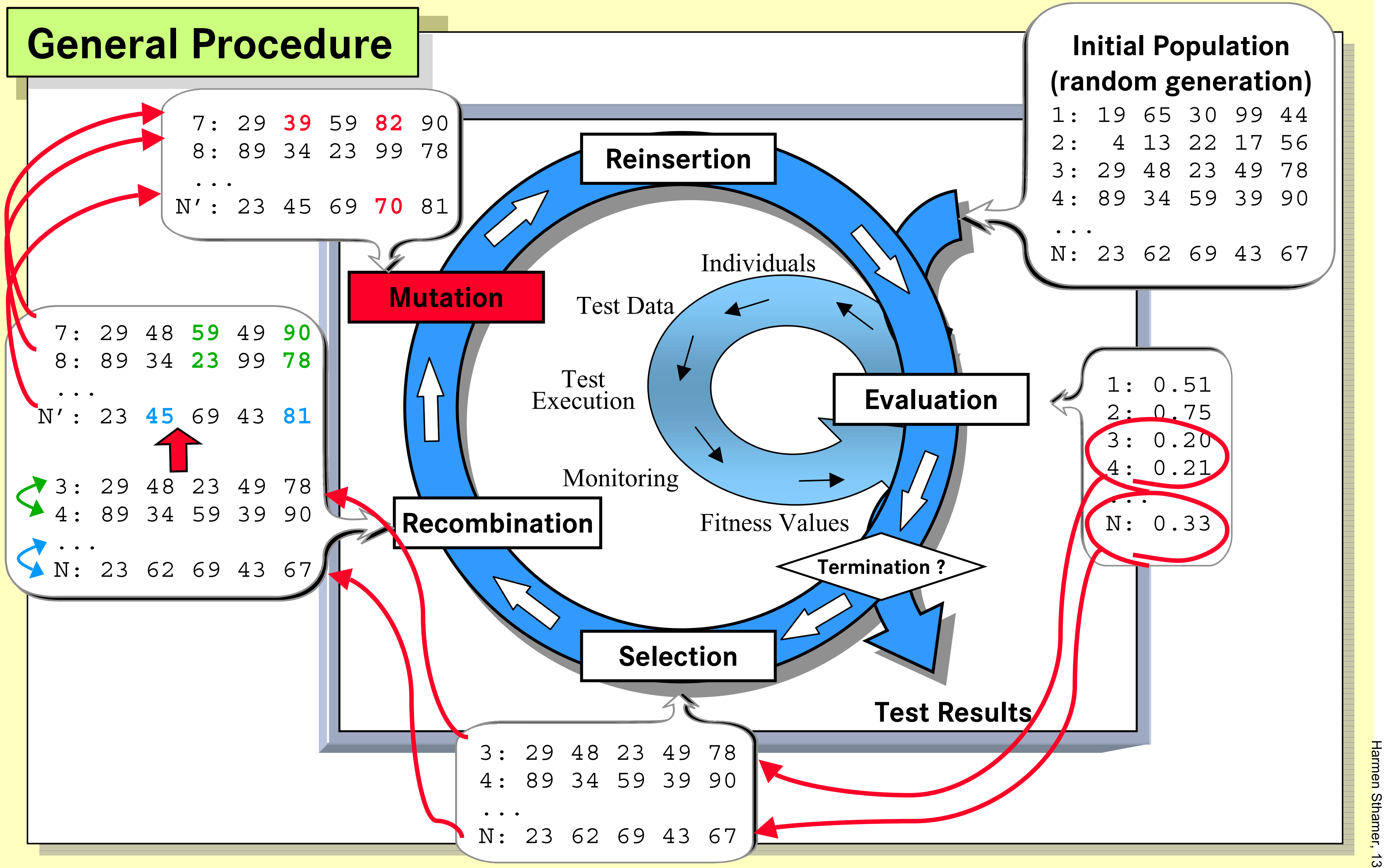
General Procedure



General Procedure

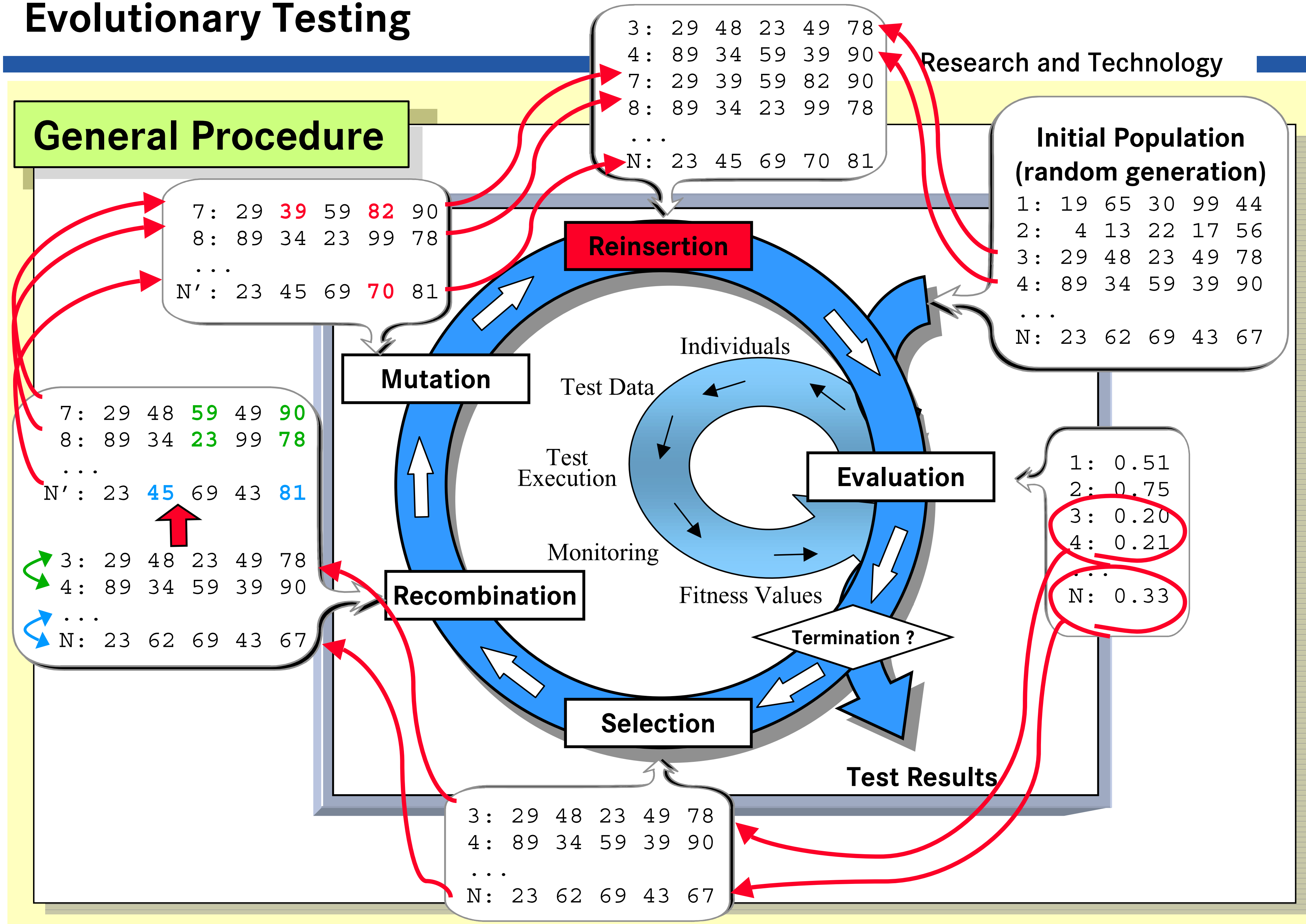


Evolutionary Testing



Evolutionary Testing

General Procedure



Harmen Sthamer, 14

Evolutionary Safety Testing

Safety Testing

Aim

- For safety critical systems, safety constraints are specified, which under no circumstances should be violated. If test data results in a violation of safety constraints
  error

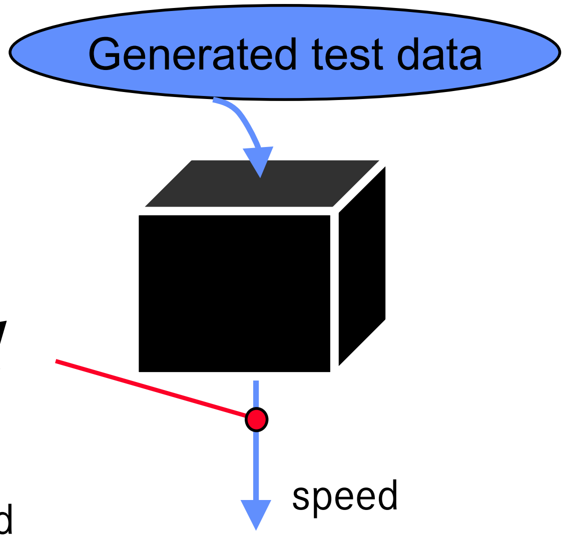
Idea

- Generate test data in order to violate safety constraints
- Fitness function defined as the distance from violating safety condition

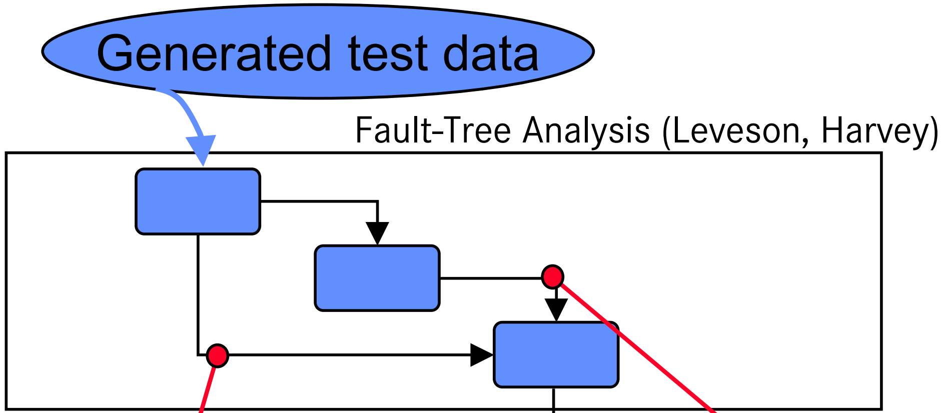
Safety condition: speed < 150 mph

$$F = 150 - speed$$

if F = 0
test successful, safety condition violated



Safety Testing



SC: Gear < 5 || (motor_speed < 7000 rpm)

SC: wheel_speed < 5160 rpm

SC: speed < 150 mph

F = f(5 - Gear) + f(7000 - motor_speed);

F = f(5160 - wheel_speed)

if F = 0 then /* test successful, SC violated

Examples of constructing fitness functions

expression	fitness, if exp. false	fitness, if exp. true
a = b	F = abs(a - b)	F = 0
a ≠ b	F = k	F = 0
a < b	F = (a - b) + k	F = 0
a ≤ b	F = (a - b)	F = 0
a > b	F = (b - a) + k	F = 0
a ≥ b	F = (b - a)	F = 0
a || b	F = min(f(a), f(b))	F = 0
a && b	F = f(a) + f(b)	F = 0

k: smallest step size

Evolutionary Structural Testing

Structural Testing

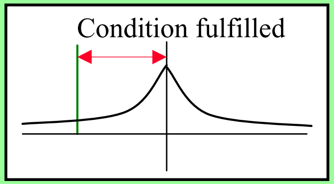
Aim

- Generate test data to cover structural test criteria automatically (statement test, branch test, condition test, path test)

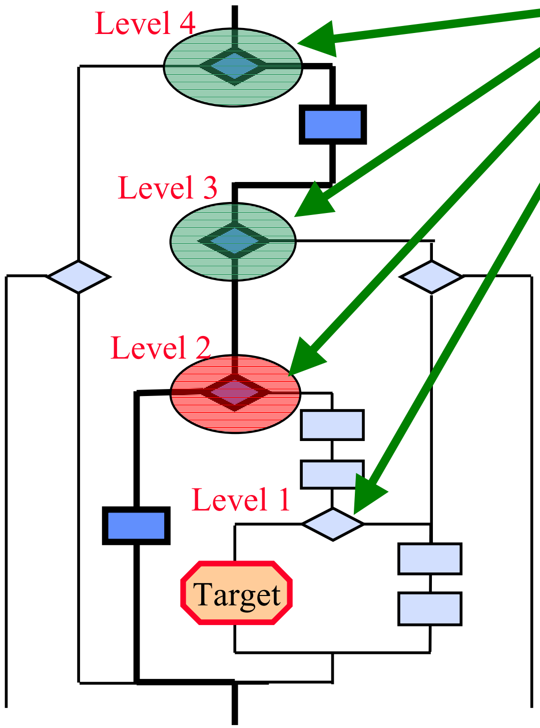
⇨ Each program branch, each condition, each path are considered as a separate independent optimisation problem

Idea

- distance oriented approach
- test case design is performed on the basis of the program structure
- test partitioned into single sub-goals
- separate fitness function for each sub-goal measures distance from fulfilling branch predicates in desired way



Distance Oriented Approaches



1. Approximation level

- Identify relevant branching statements for target node on 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'

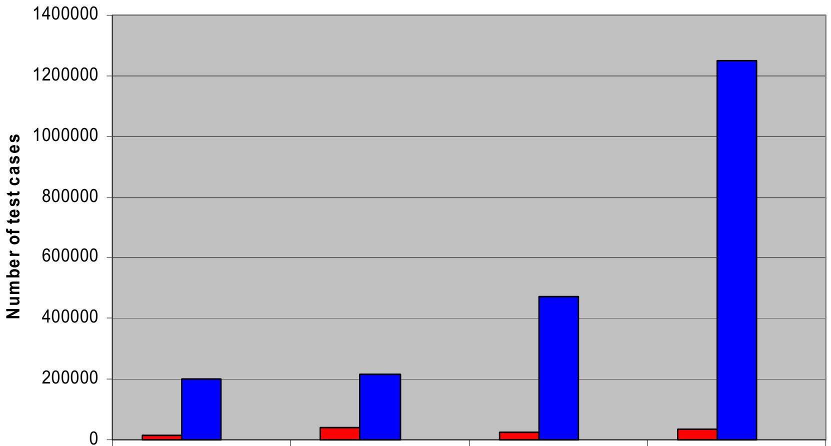
2. Distance measurement in the branching statement with undesired branching

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

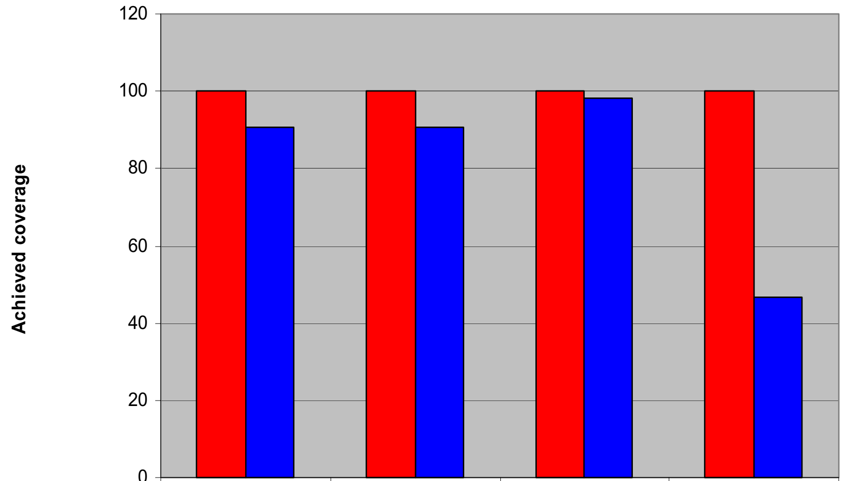
➥ Fitness = Approximation_Level + Distance

Results of Structural Testing

Results achieved with distance oriented approach (Wegener, Baresel, Sthamer)



	Triangle_int	Triangle_float	Complex	My_atof
ET	16915	42086	23633	35263
RT	199743	215834	470931	1251038
RT / ET	11,8	5,1	19,9	35,5



	Triangle_int	Triangle_float	Complex	My_atof
ET coverage	100	100	100	100
RT coverage	90,5 (0%)	90,5 (0%)	98,1 (50%)	46,6 (0%)

Coverage in %

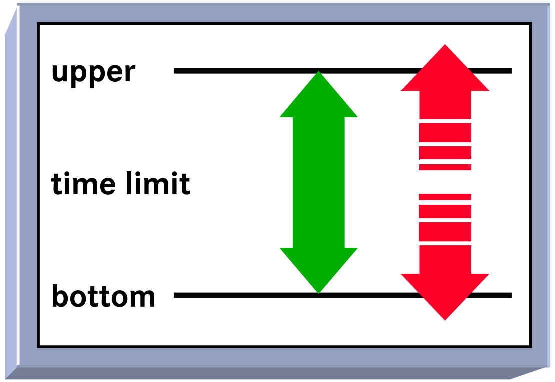
- ET achieves full branch coverage for all test objects, RT achieves only between 46% and 90% branch coverage on average
- ET requires less test cases compared to RT (by a factor of between 5 to 35)

Evolutionary Temporal Behavior Testing

Testing Real-Time Constraints

The temporal behavior of real-time systems is erroneous when input situations exist for which the computation violates the specified timing constraints limits.

Aim



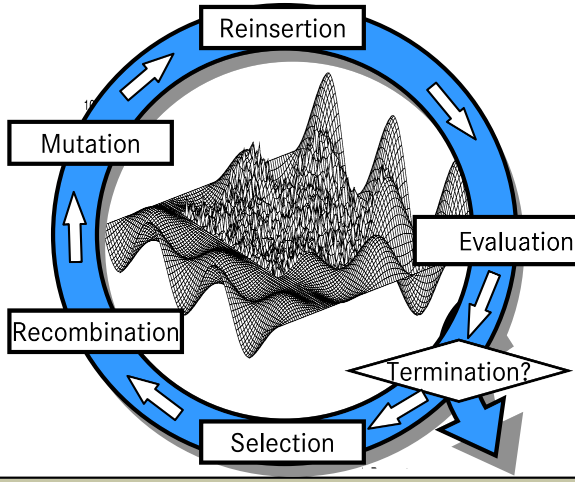
- Find test data with longest and shortest execution times to check whether they cause temporal error

Idea

- The search for input situations with particularly long or short executions times is interpreted as an optimization problem.
- Objective values for individuals based on execution times of corresponding test data

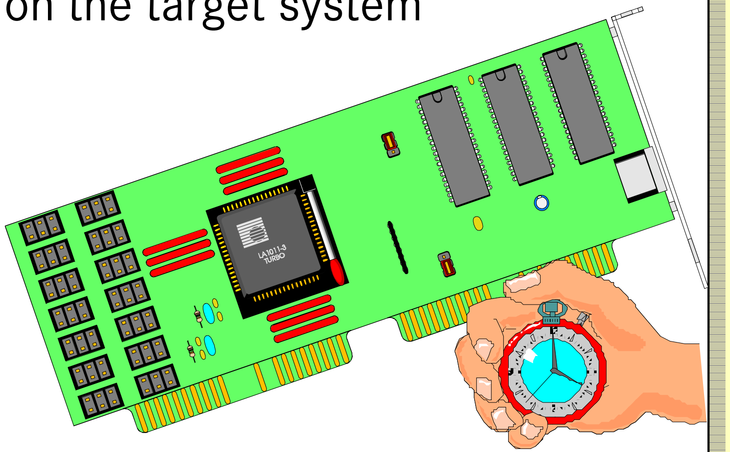
Experiment Environment for Testing Temporal Behaviour

Automatic generation of test data on the host system



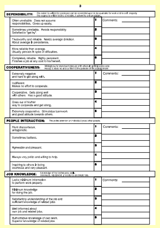
Test Data

Test execution with automatic measurement of execution times on the target system



Execution Times

Test Results

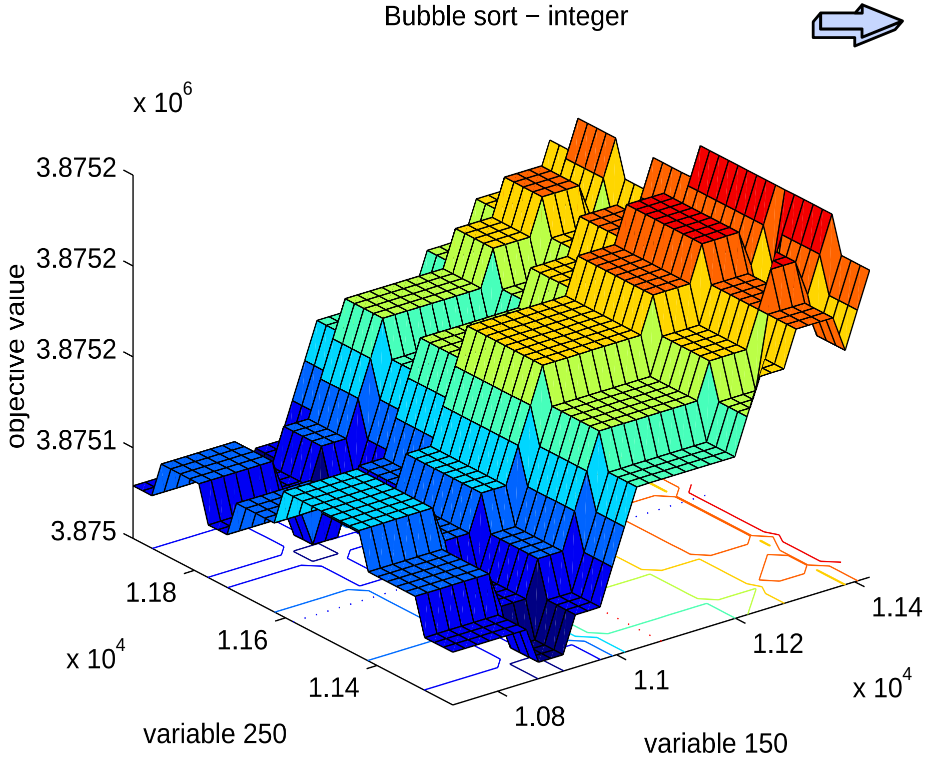


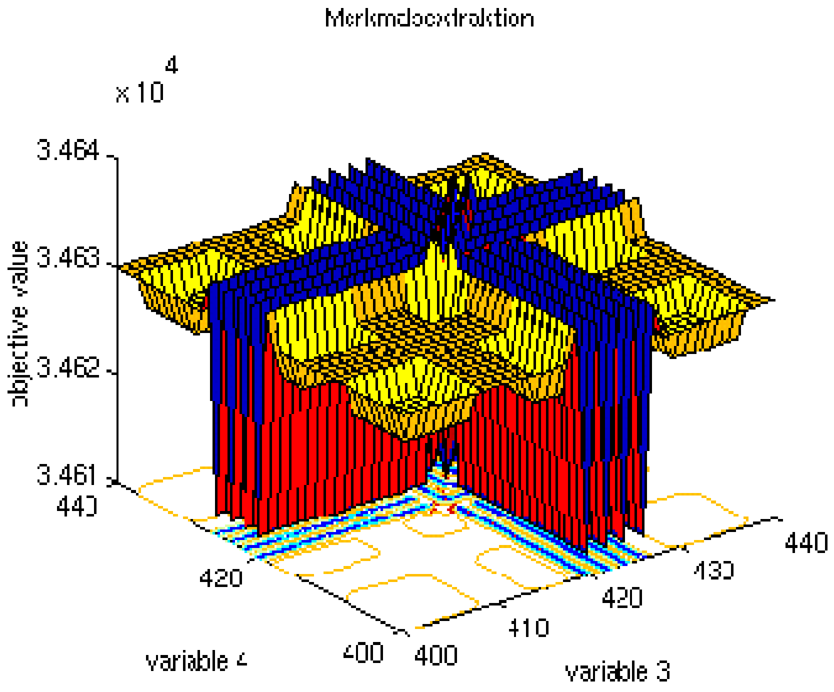
Longest Execution Time

Comparing the execution times with the results of the developer test; possible error elimination

Landscape of Search Space

Very complex
- multidimensional,
- plateaus,
- discontinuities (jumps),
- definition gap,
- local optima
- high dependencies among input variables





Evolutionary Temporal Behavior Testing

Results

- variation between ET and RT results when searching longest and shortest execution times for various examples (in %)

⇓

- for all test objects (except Engine VI) ET results are superior to RT
- for several test objects variances > 50%

⇓

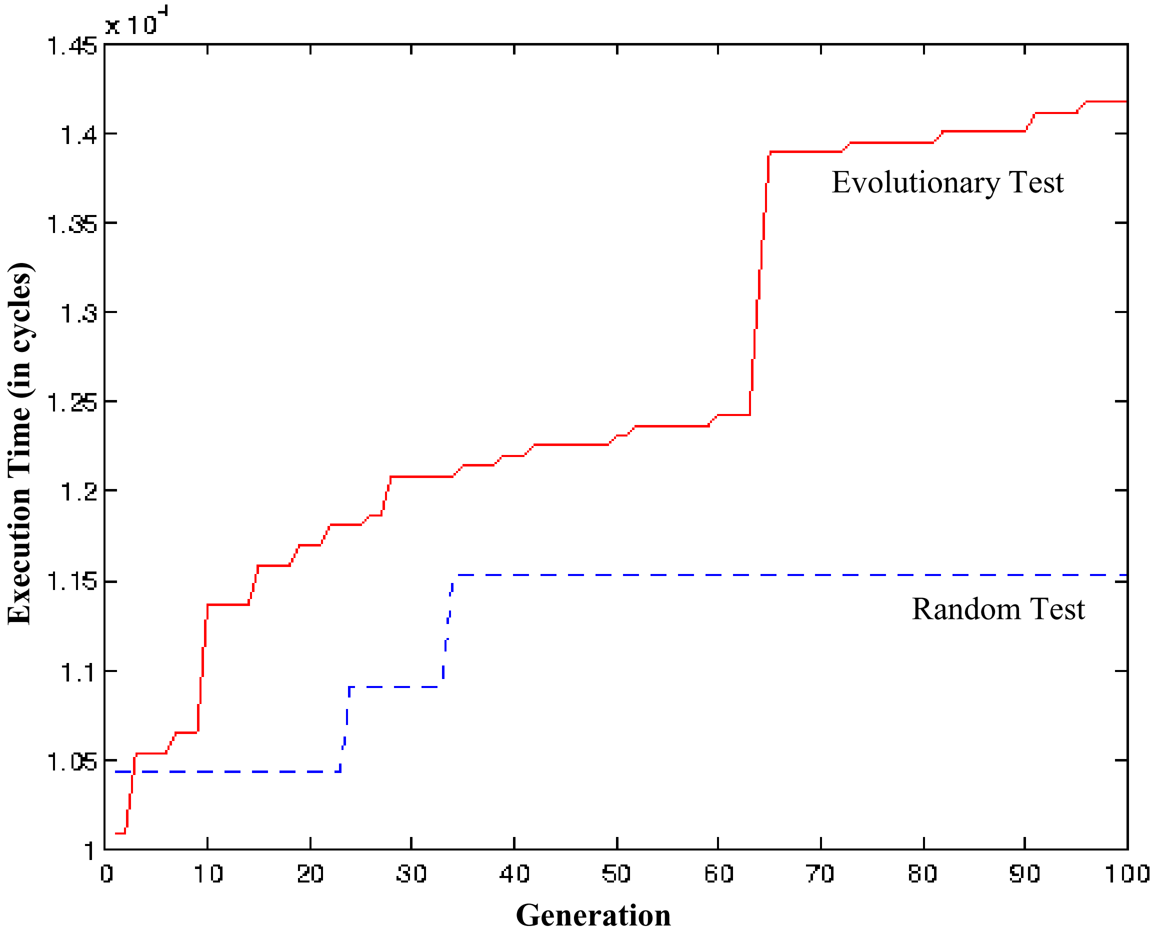
- directed search of ET considerably more powerful than RT

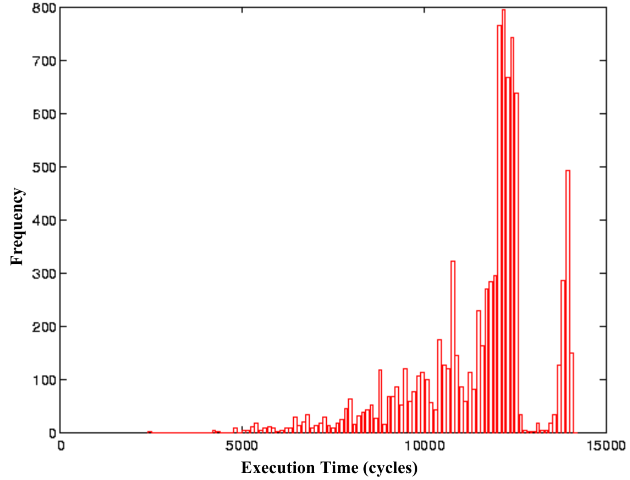
DaimlerChrysler

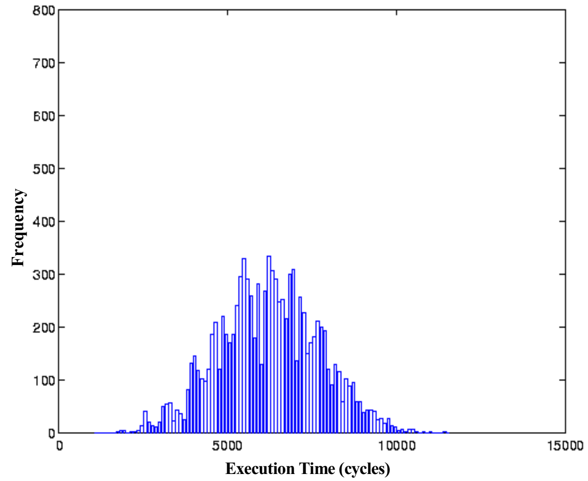
Evolutionary Temporal Behavior Testing

Detailed Analysis of Selected Results

Comparison of test runs for evolutionary testing and random testing when searching the longest execution time for railroad electronics example

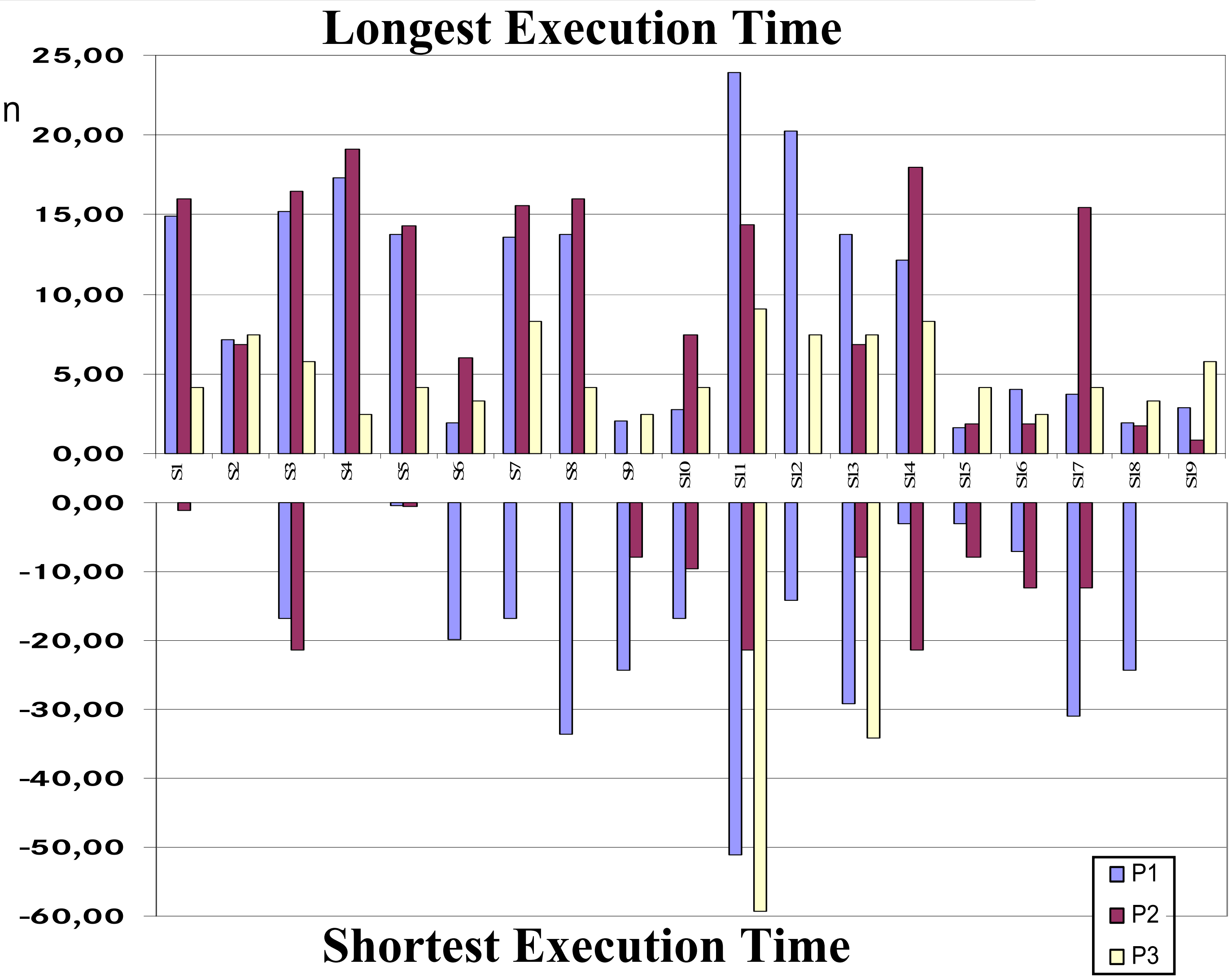






Computer Graphics: ET compared to Functional Testing

- variation (in %) between ET over functional testing (FT) when searching longest and shortest execution times for CG example on various P
- for most results ET is superior to FT
- search for longest execution is more difficult than for shortest
- directed search of ET considerably more powerful than FT



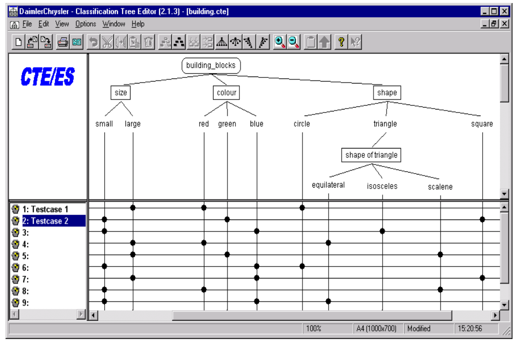
Engine Control

Evolutionary Testing

- Generation of 7.500 or 15.000 test data (50 or 100 generations each with 150 individuals, subject to the number of parameters of the test object)

Functional and Structural Testing

- Test case definitions by the developers of the tasks
  Objective: testing the functional and temporal system behavior
- Test case design on the basis of the specification and program structures
- Functional testing with the Classification-Tree Method
- Branch testing with complete branch coverage

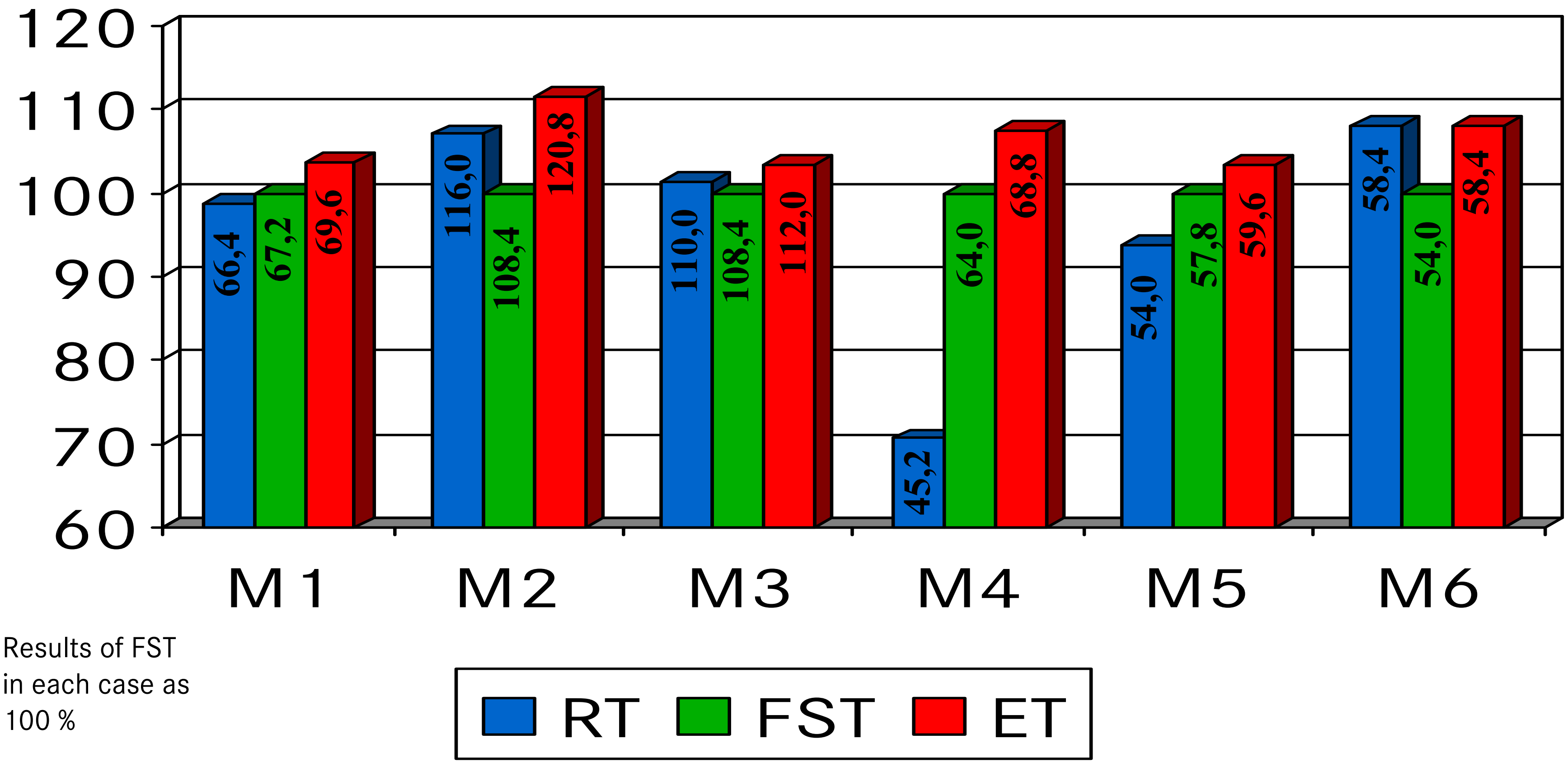


Random Testing

- Generation of 7.500 or 15.000 random test data (corresponding to the predefinitions for the evolutionary test)

Results Engine Control

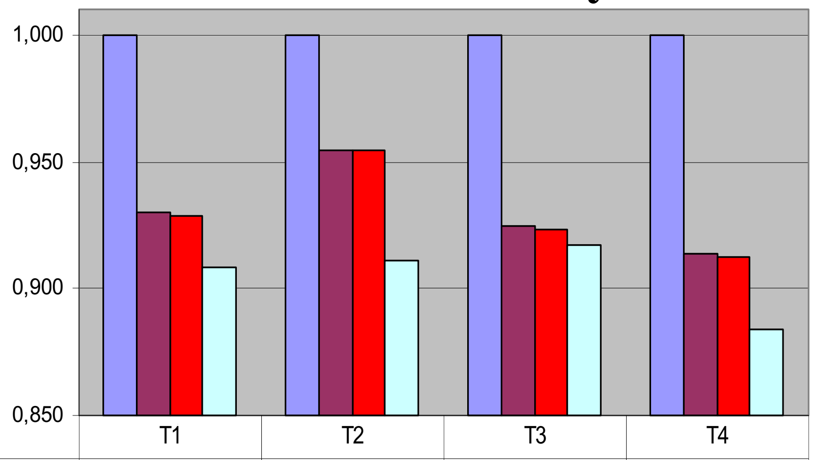
Comparing the longest execution times from **evolutionary testing (ET)**, **functional and structural testing (FST)** as well as **random testing (RT)** for the engine control tasks (execution times in µs)



Results of FST in each case as 100 %

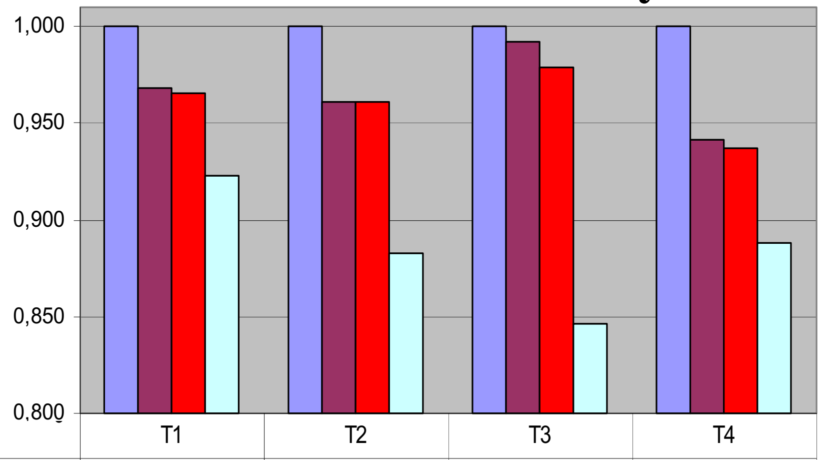
Comparison between Static Analysis and Evolutionary Test

WCET analysis



	T1	T2	T3	T4
WCETmax	2710	742	5154	3604
WCETmeas	2520	708	4764	3292
ET	2516	708	4760	3288
WCETmin	2462	676	4728	3186

BCET analysis



	T1	T2	T3	T4
BCETmax	1992	512	2394	2600
BCETmeas	1928	492	2376	2448
ET	1924	492	2344	2436
BCETmin	1838	452	2026	2308

Results normalized, WCETmax and BCETmax is set to 1

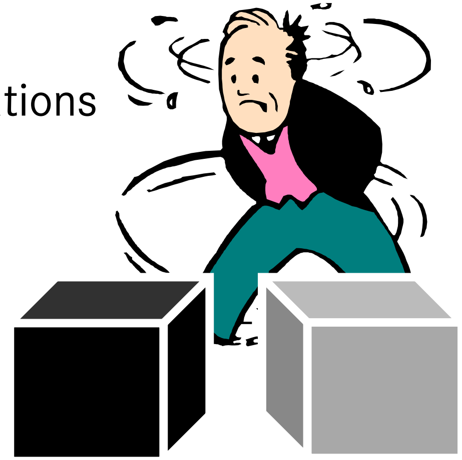
Results

- SA analysis determines upper and lower bounds (*max* and *min*), considers structural as well functional constraints, e.g. depends on various memory access time, caching,
- *meas* means dynamic determined execution time of SA WCET path
- results varified by ET (automatic tool support)

Evolutionary Temporal Behavior Testing

Summary of Temporal Behaviour Testing

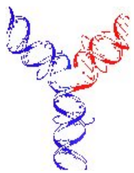
- Evolutionary testing more efficient than random testing.
- No information on function or internal structures are required
- Since ET can adapt to the temporal behavior of the respective test object, it leads to the generation of test data with extreme execution times
- Test object is tested with a large number of different input situations
- Testing is carried out on a Target-System
- The comparison with static analysis shows that the execution times determined by evolutionary testing form a realistic approximation of the extreme execution times.



However

No guarantee that the optimum solution is found, since ET is based:

randomly exchange of **information** among individuals (crossover)



randomly change of **information** within an individual (mutation)

Conclusion, Future Work

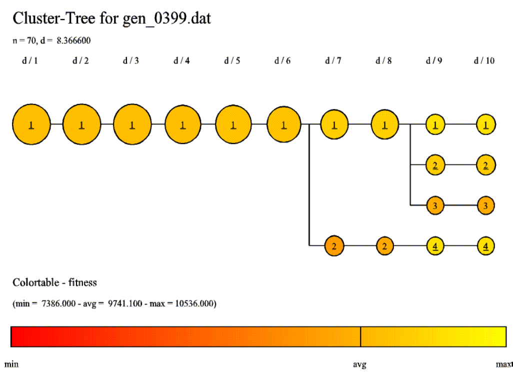
Conclusion

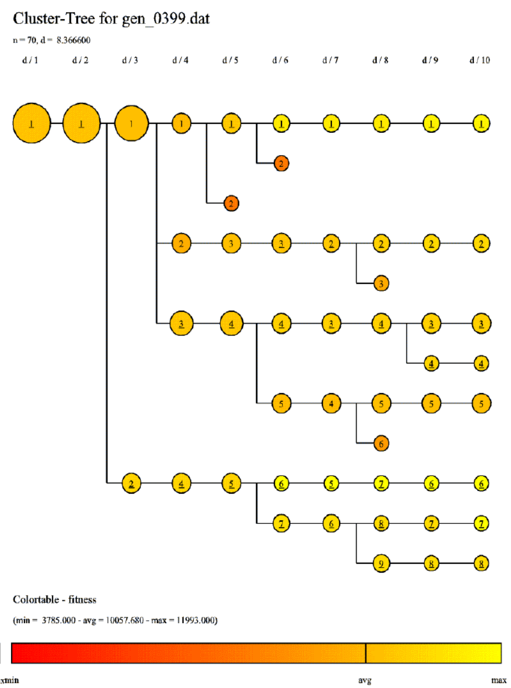
- Evolutionary Testing is a new method for the automation of test case design
- Since the test object can be transformed into an optimization problem, it can be solved with the assistance of metaheuristic search methods
- Dynamic adaptation of evolutionary algorithms
- Due to high level of automation and good results, Evolutionary Testing is well placed to supplement existing test methods. It contributes to better product quality and promotes efficient development
- More research remains to be done to answer outstanding questions
- More papers on Evolutionary Testing, CTE and TESSY can be found on **http://www.systematic-testing.com**
- Further information on Evolutionary Algorithms in SE can be found on **http://www.discbrunel.org.uk/seminal**

Conclusion and Future Work

Future Work

- seeding of test data into initial population, e.g. for structural testing, and temporal behaviour testing
- selection of search technique and configuration of evolutionary operators according to test object metrics
- dynamic configuration of evolutionary operators during test run with respect to test progress
- test termination using cluster analysis
- develop further application fields e.g. regression testing and back-to-back test of control systems, testing interactive systems, testing object-oriented software
- transformation of code in order to increase testability





DAIMLERCHRYSLER