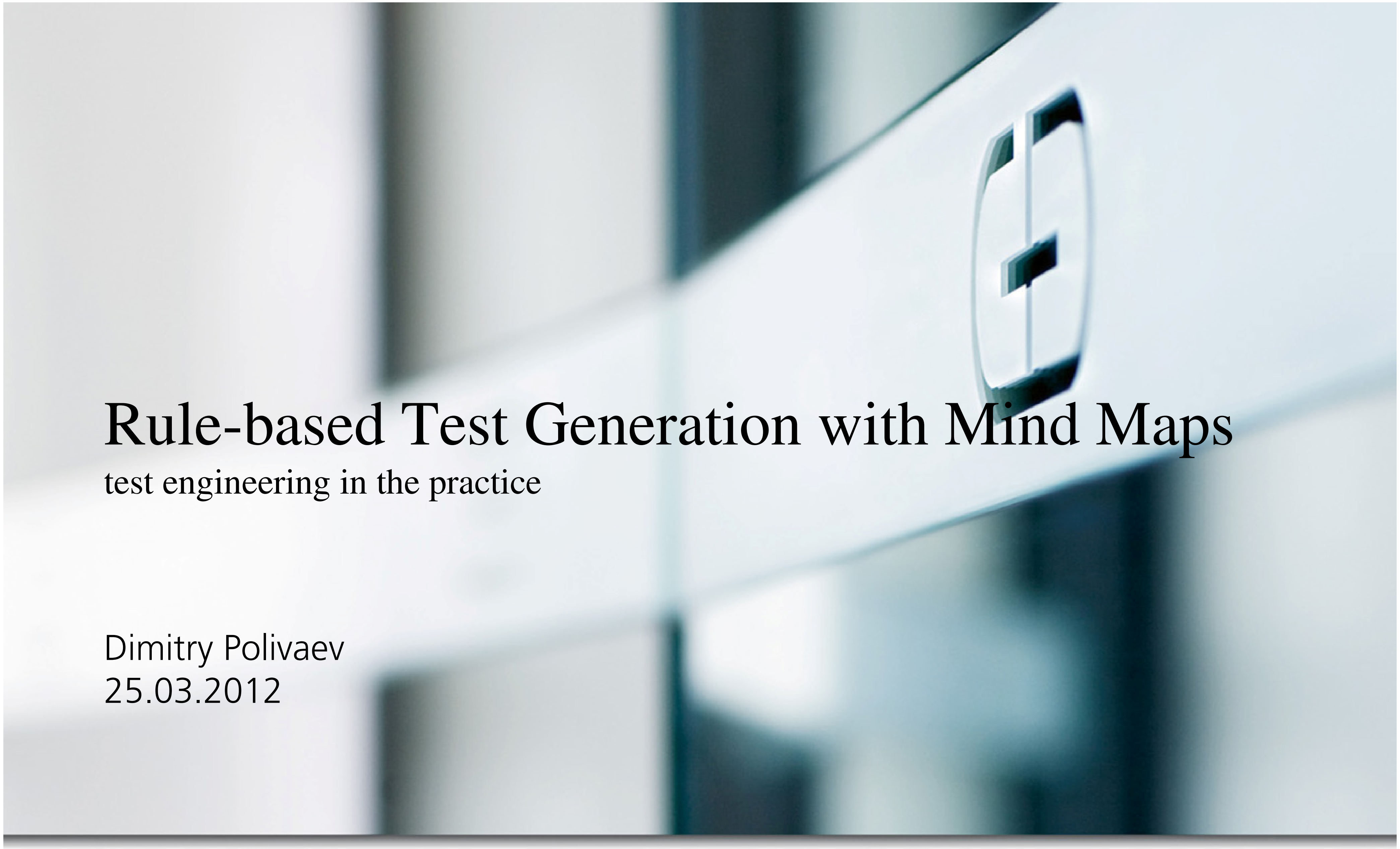


Rule-based Test Generation with Mind Maps

test engineering in the practice

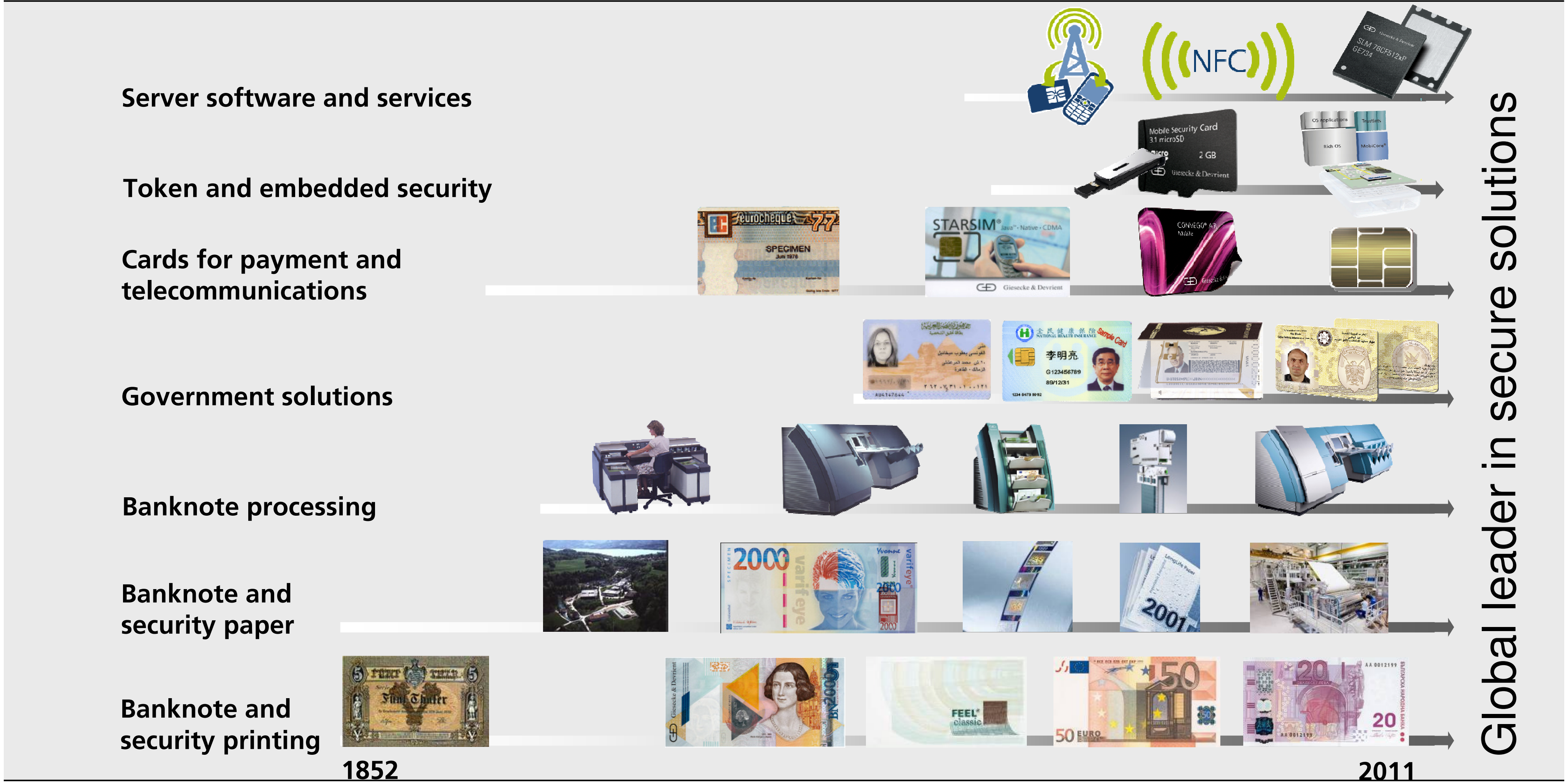
Dimitry Polivaev

25.03.2012

Giesecke & Devrient

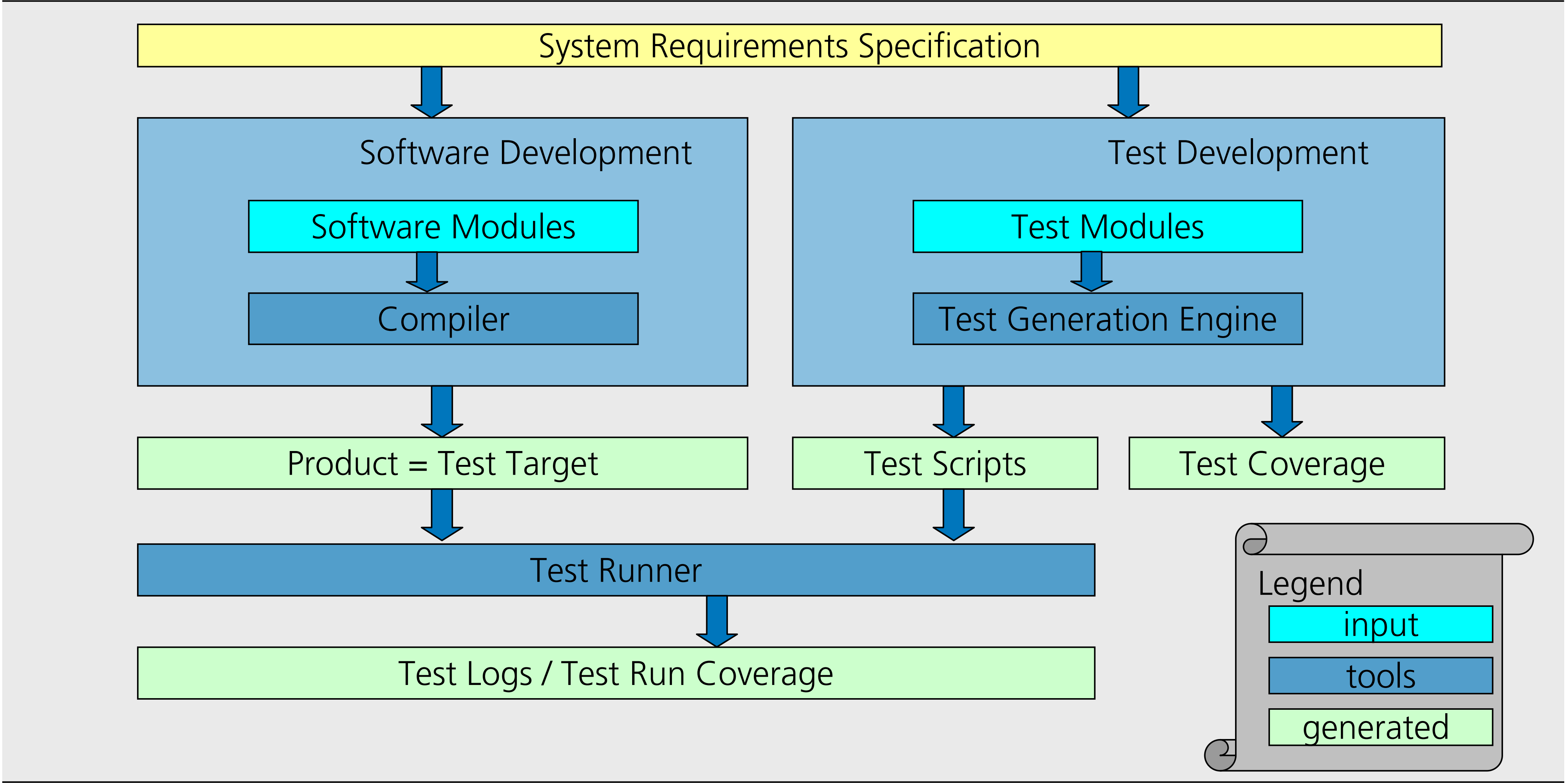
Creating Confidence.

G&D has been growing through continuous innovation


Server software and services
Token and embedded security
Cards for payment and telecommunications
Government solutions
Banknote processing
Banknote and security paper
Banknote and security printing
1852
2011
Global leader in secure solutions

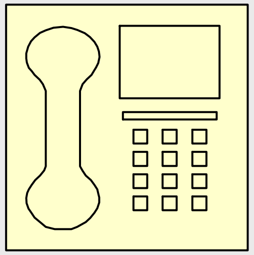
Giesecke & Devrient

Test engineering is similar to product development



Giesecke & Devrient

SUT Example: calculation of phone call costs



Destination	Standard Tariff	CheapCall Tariff	At night, on the weekend	Time unit (seconds)
National	0.10$	0.07$	0.03$	1
International	1.00$	0.50$	0.80$	20

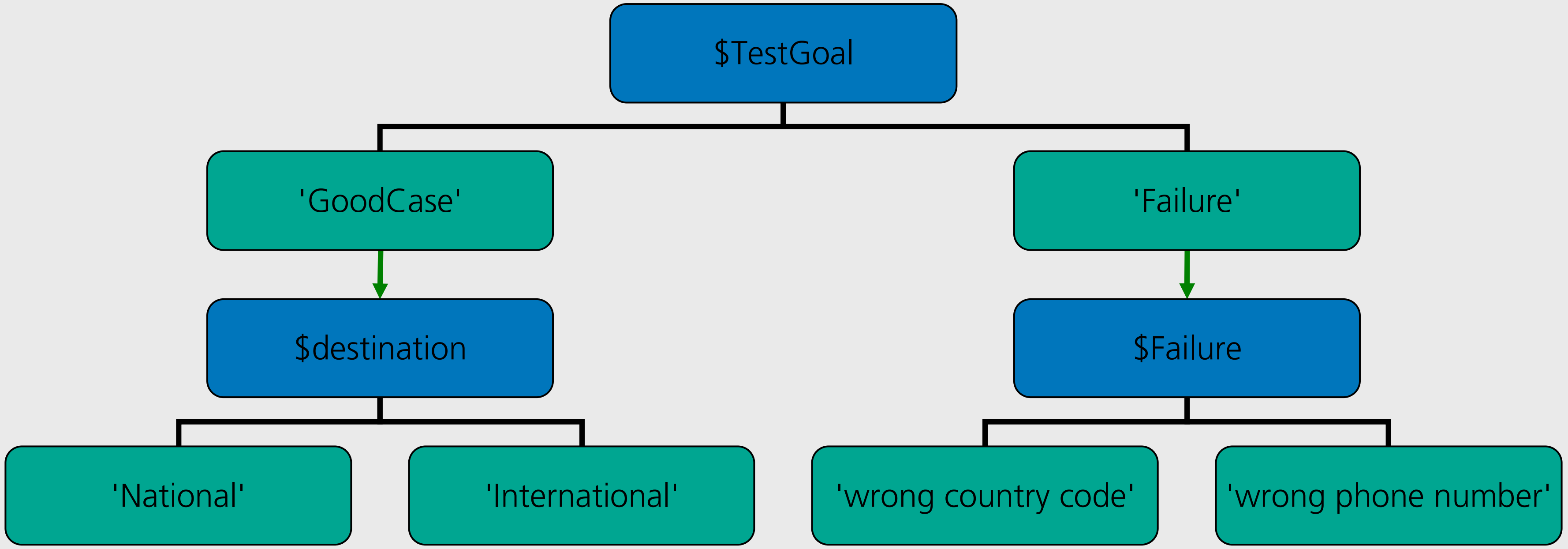
1. if country is empty or 'National' the destination is set to National ,
2. if country is 'Greenland', 'Blueland' or 'Neverland' the destination is set to 'International',
3. All other country values are invalid.
4. Tariff "At night" applies if the call begins between 8 pm and 6 am.
5. Tariff "On the weekend" applies if the call begins on Saturday or on Sunday.
6. Call duration given in seconds is rounded up to units specified in column "Time unit".
7. Maximal call duration is limited to 24 hours.

Giesecke & Devrient

Test case systematic

Each test case is defined by all its properties e.g. test case name, test goal ("good case", "semantic error XY",) test commands, their input data, expected results.

Systems of test cases can be organized by classification tree like systems of test case properties.



(Each blue element is a test case property, green elements represent possible values.)

Giesecke & Devrient

Test generation process

- Test generation is a process of selection of relevant test cases.
- It finds all relevant value combinations of different test properties, considering dependencies between them.
- It builds a test suite consisting of executable test scripts and test coverage information.

Basic test generation modules

Test generation modules are responsible for test property calculation, test script generation and test coverage evaluation.

- Different tasks are handled by different modules.

There are modules **Strategy, SUT Model, Writer, Solver** and **Goals**.

- **Strategy** (mandatory) describes **WHAT** is being tested. It defines an exploration of test case property space, for example using the Cartesian product, random combinations or defining functional dependencies between the property values.
- **SUT Model** (optional) can calculate behavior related values namely expected results, statement coverage, path coverage, modified condition / decision coverage, internal states at given statements for given input parameters and data state before call.

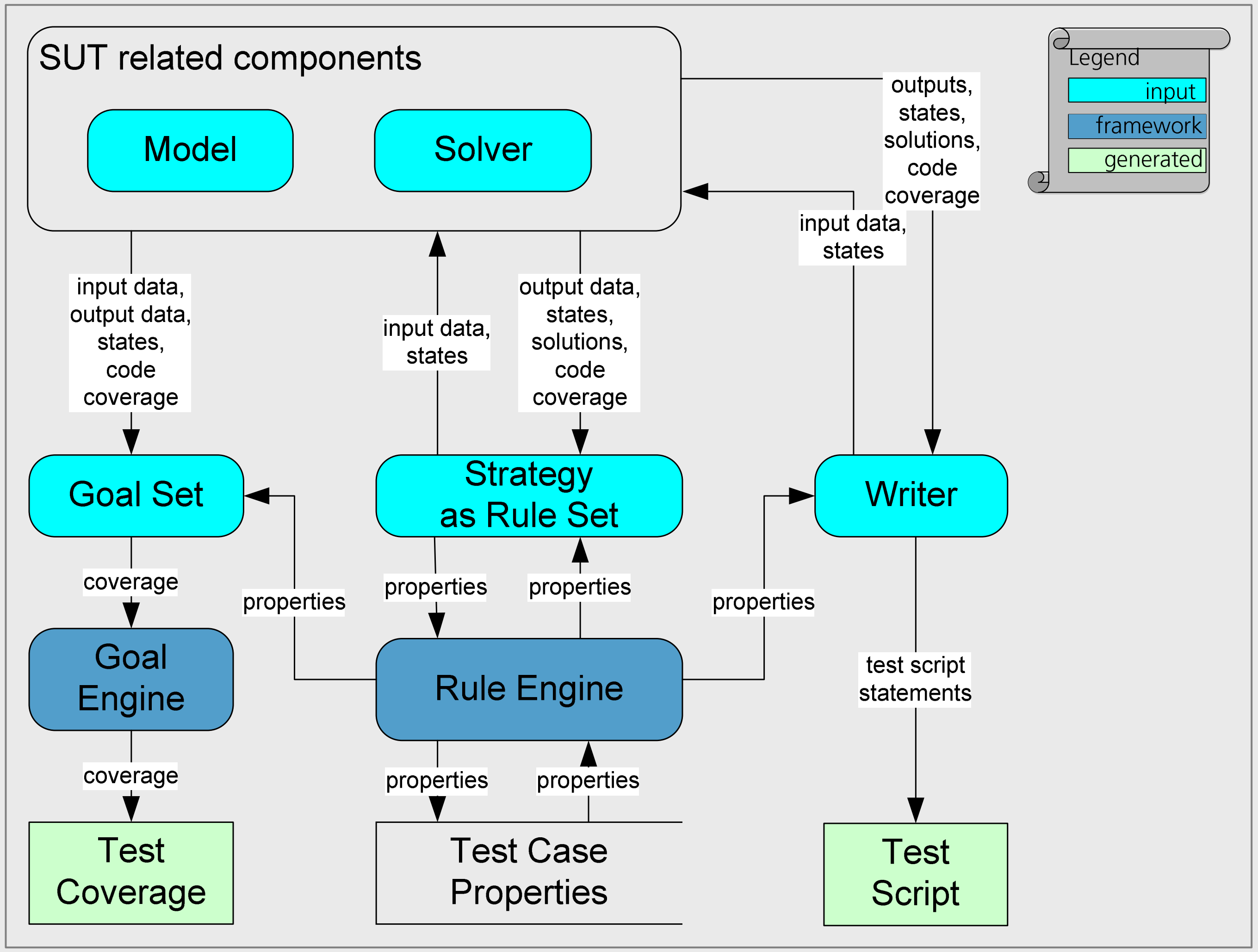
Giesecke & Devrient

Basic test generation modules

- **Writer** (mandatory) creates test script fragments in the target language e.g. Test header, Test name, Test description, Test script commands, Comments
- **Solver** (optional) specifies **HOW** the SUT is tested. It can be used to calculate test commands in preconditions, post processing or verification so that test property values already set by the strategy are satisfied.
- **Goals** (optional) describes **WHY** the tests are needed. It defines combinations of property values to be covered by a complete test suite. The module can be used for checking the coverage and for selecting a sufficient subset of generated test cases.
  - Examples: status word coverage, statement coverage, path coverage, state coverage.

Giesecke & Devrient

Test generator architecture



Giesecke & Devrient

Modular design makes changes easier

Test module implementations can be changed independently:

- Strategies can be exchanged for testing of different aspects. For instance it is possible to have a small amount of test cases, with a high coverage of the requirements, available quickly. During the bug fixing of the found errors the strategies can be extended.
- SUT models can be exchanged for testing of different products / product variations.
- Different solvers can be applied for different ways or prepare initial state or reset test target after test.
- Different writers can be used for generating scripts for different script languages
- Different test goals can be defined for test case selection for test depth variation, e.g. for smoke tests, regression tests etc.

Giesecke & Devrient

Strategy as a rule set. Iteration rules.

The strategy can be compactly represented by a **set of rules** specifying property dependencies.

There are two kinds or rules:

Iteration rules defines iterations over property values.

They are evaluated when some another property is assigned by another iteration rule (so-called **forward chaining**).

Each iteration rule contains three parts:

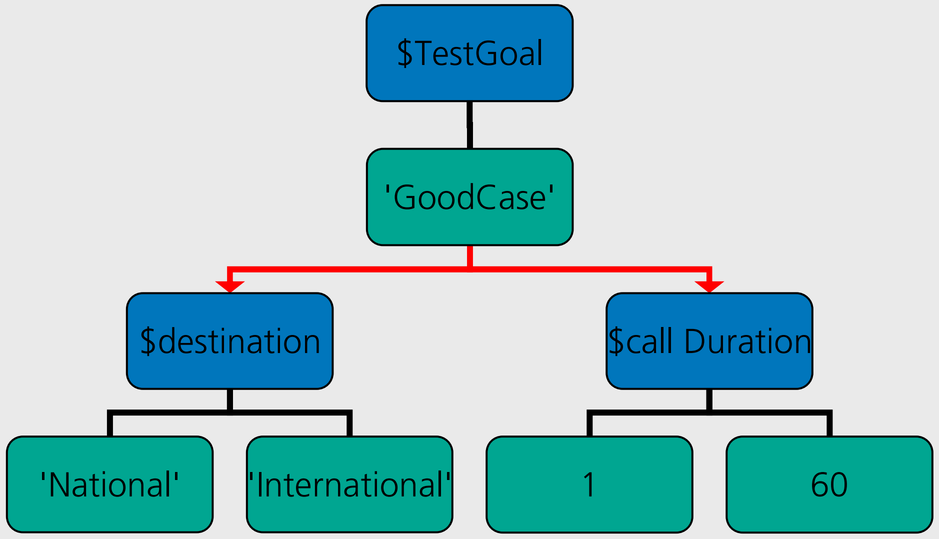
- the WHEN-part describes when the rule is applied,
- the optional IF-part describes an additional condition, and
- the THEN-part describes the action setting the list of property values and sometimes adding new rules to the set.

Examples of iteration rules

1. *WHEN* test generation is started
   IF (empty)
   THEN property TestGoal is sequentially assigned values 'GoodCase', 'BadCase'

2. *WHEN* property TestGoal is assigned
   IF it has value 'GoodCase'
   THEN property destination is sequentially assigned values 'National' and 'International'

3. *WHEN* property TestGoal is assigned
   IF it has value 'GoodCase'
   THEN property call- Duration is sequentially assigned values 1 and 60



Giesecke & Devrient

Default rules. Rules with extra conditions

Default rules assign one value to a property not assigned by the iteration rules. They are evaluated if its target property value was requested by another rule or by the test generation algorithm (so-called **backward chaining**).

They can contain only an optional IF - part with a condition and a THEN - part describing an action. Default rules are not designed for producing iterations, they always assign single values.

Example:

1. *IF* destination = 'International'
   THEN property country is assigned a value 'Yellowland'

Giesecke & Devrient

Rule Stacks

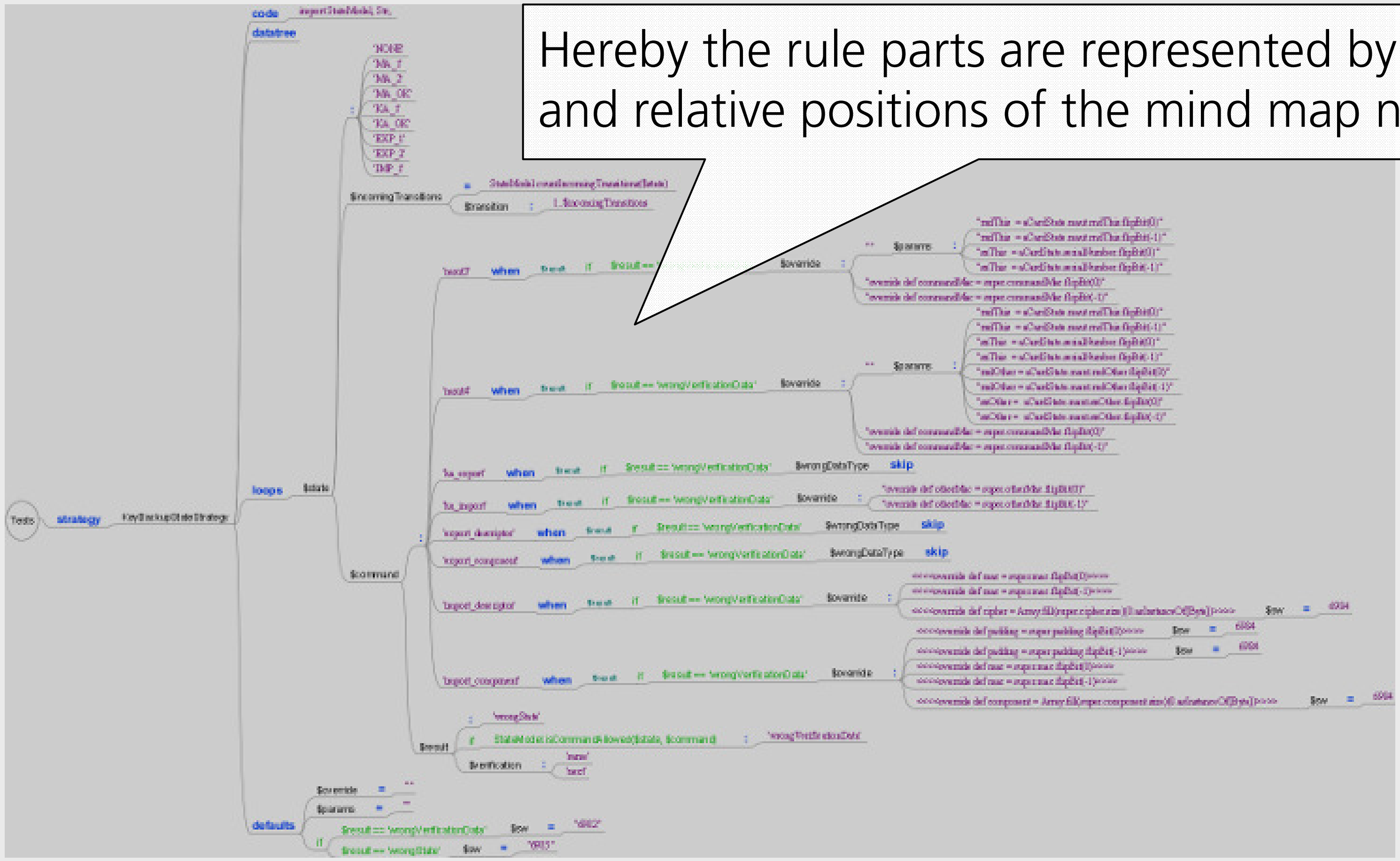
Iteration rules with the same properties in the WHEN part and the same target property build a so-called rule stack. They are processed in the opposite order to their definition. There are also rule stacks with default rules.

If some rule with empty condition or with satisfied condition was found, no other rules from the stack are executed.

Rule sets must be self-consistent: if a property has already been assigned, no other iteration rule may try to reassign it.

Rules added to the stack later can override previously defined rules. This can be used to define strategy variations overloading some default strategy.   For example there can be some common and product dependent subsets for testing of a product line.

Rules can be defined in mind maps



Hereby the rule parts are represented by contents and relative positions of the mind map nodes.

Giesecke & Devrient

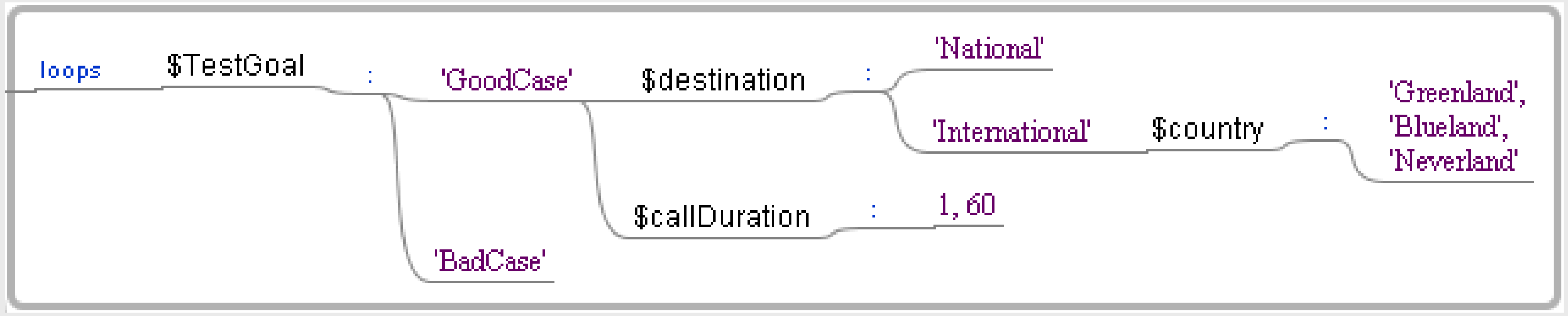
Mind Map representation of Rule based Test Strategy

The map based rule set looks similar to a classification tree. It is substantially more powerful because it supports concepts of rule stacks, functional conditions and simultaneous iterations over different property values.

Mind maps as a representation of generation rules are created with open source tool Freeplane. It offers good visualization, automatic context dependent node formatting, search and filtering of the rule sets.

Use of mind maps for test strategy implementation simplifies development, reviewing and improvement of the strategy.

Example strategy as a mind map



The strategy produces following property combinations:

```
1.  $TestGoal:'GoodCase'/$destination:National/$callDuration:1
2.  $TestGoal:'GoodCase'/$destination:International/$country:Greenland/$callDuration:60
3.  $TestGoal:'GoodCase'/$destination:International/$country:Blueland/$callDuration:1
4.  $TestGoal:'GoodCase'/$destination:International/$country:Neverland/$callDuration:60
5.  $TestGoal:'BadCase'
```

It demonstrates simultaneous iteration of different property values

Properties requires for test output but not set by iteration rules can be set using default rules also defined in a mind map.

Giesecke & Devrient

Test Goals

Test selection and test coverage can be expressed as a set of test goals. There are finite and infinite goals:

- A finite test goal is a check list with limited number of values.
- In the infinite goal the number of possible values is unlimited or not known before the test generation.

Based on the goals the generator can decide which test cases should be kept and generate test suite coverage statistics.

Finite test goal

A finite test goal is exactly like a check list. Such check lists can include

- all or selected statements within the model code (code coverage),
- all values of input parameters,
- output values or model interim data,
- any combinations between the above data like execution paths in the model,
- other model coverage information like modified condition/decision coverage.

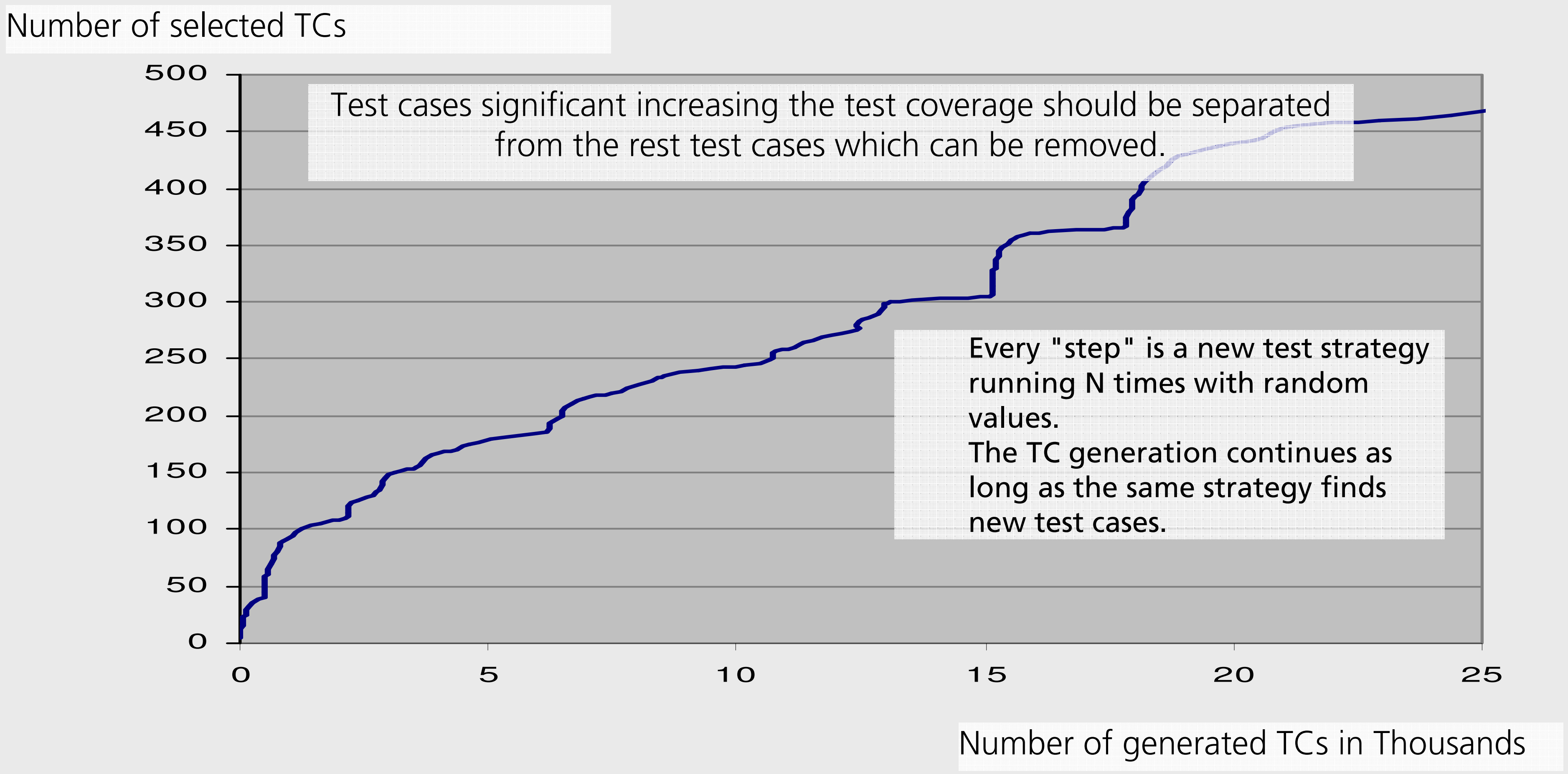
So a finite goal can be defined as a pair of the complete check list given as e.g. set of strings and a function from the any data available to the strategy to the values to be compared to the check list values.

Infinite test goal

The infinite goals consists only of the single function of test properties, there is no predefined check list.

The test is significant for an infinite goal if the function returns a result not returned before, and the check lists is automatically extended when a new value is calculated

Test Case Selection using Test Goals



Test cases significant increasing the test coverage should be separated from the rest test cases which can be removed.

Every "step" is a new test strategy running N times with random values.
The TC generation continues as long as the same strategy finds new test cases.

Giesecke & Devrient

Test Coverage

Given the test goals the framework collects test coverage statistics during the test generation.

They describe how often any particular value from the check lists was met by all of the saved test cases.

```
GOAL: 'Flowchart coverage' (achieved)
* Invalid Number                (455/1)|=========================
* National Code                 (204/1)|============
* International1 Code           (147/1)|=========
* OnTheWeekend                   (50/1)|===
* At night                       (34/1)|==
* National                       (83/1)|=====
* National+Standard              (20/1)|==
* National+Weekend                (9/1)|=
* International1+Standard        (34/1)|==
* International1+Weekend         (23/1)|==
* International2+Standard        (29/1)|==
* International2+Weekend         (12/1)|=
```

Giesecke & Devrient

Project Experiences

Project experiences demonstrated advantages of Rule-based Test Generation:

- Increase of test maintainability, reduced test adaptation efforts on specification updates.
- Better test systematic through
  - Classification of test cases (requirements, test ideas)
  - Combination and selection determined by formal description of test strategy
  - Combination with random based combinations increases test coverage and finds unexpected bugs
- Systematic use of with random elements in the test strategy increases test coverage and finds unexpected bugs.

Giesecke & Devrient

Summary

Systems of test cases can be organized by test case properties.

Modular designed test generators can be used to generate them. Basic test generation modules are **Strategy, Model, Solver, Writer** and **Goals**.

Language matters. There is a test generation language specialized on implementing the modules.

Writing of test strategies as **mind maps based rule sets** helps to achieve an additional productivity increase.

Replacement of "test handicraft" by test engineering allows to achieve better Test Coverage and Test Depth and decrease test development and maintenance costs.