Budapest University of Technology and Economics
Department of Measurement and Information Systems

# Model-Based Automatic Test Generation for Event-Driven Embedded Systems Using Model Checkers

Zoltan Micskei and Istvan Majzik
{micskeiz, majzik}@mit.bme.hu

DepCos-RELCOMEX ‘06

2006. 05. 27.

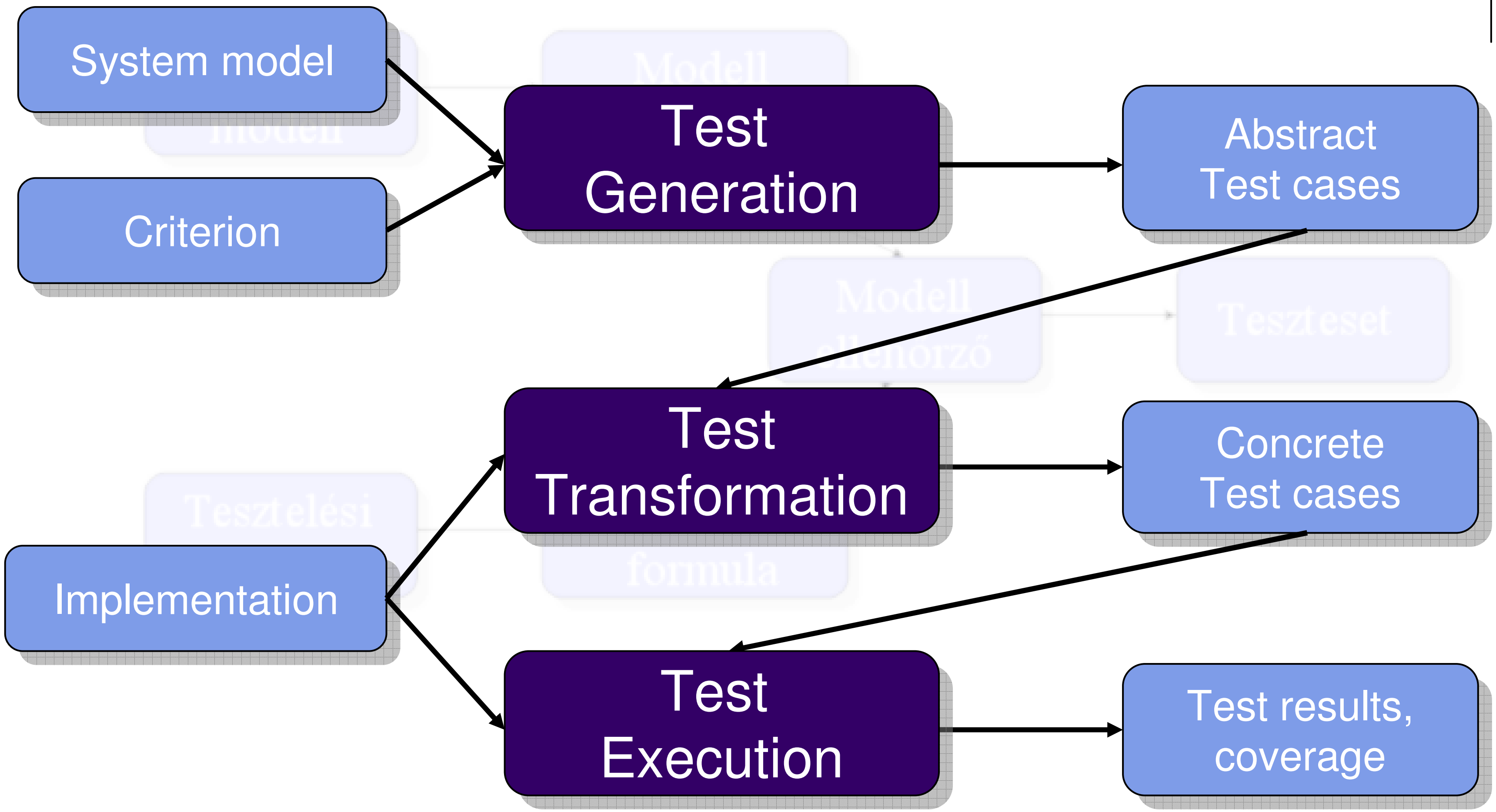
# Testing concepts

- Goal: improving the quality of the system
- Test case: input events and *expected output actions* representing a typical paths
- Test suite: set of test cases
- Test requirement: a specific sub goal for testing, e.g. call a function
- Coverage criterion: determines a set of test requirements, e.g. cover all statements

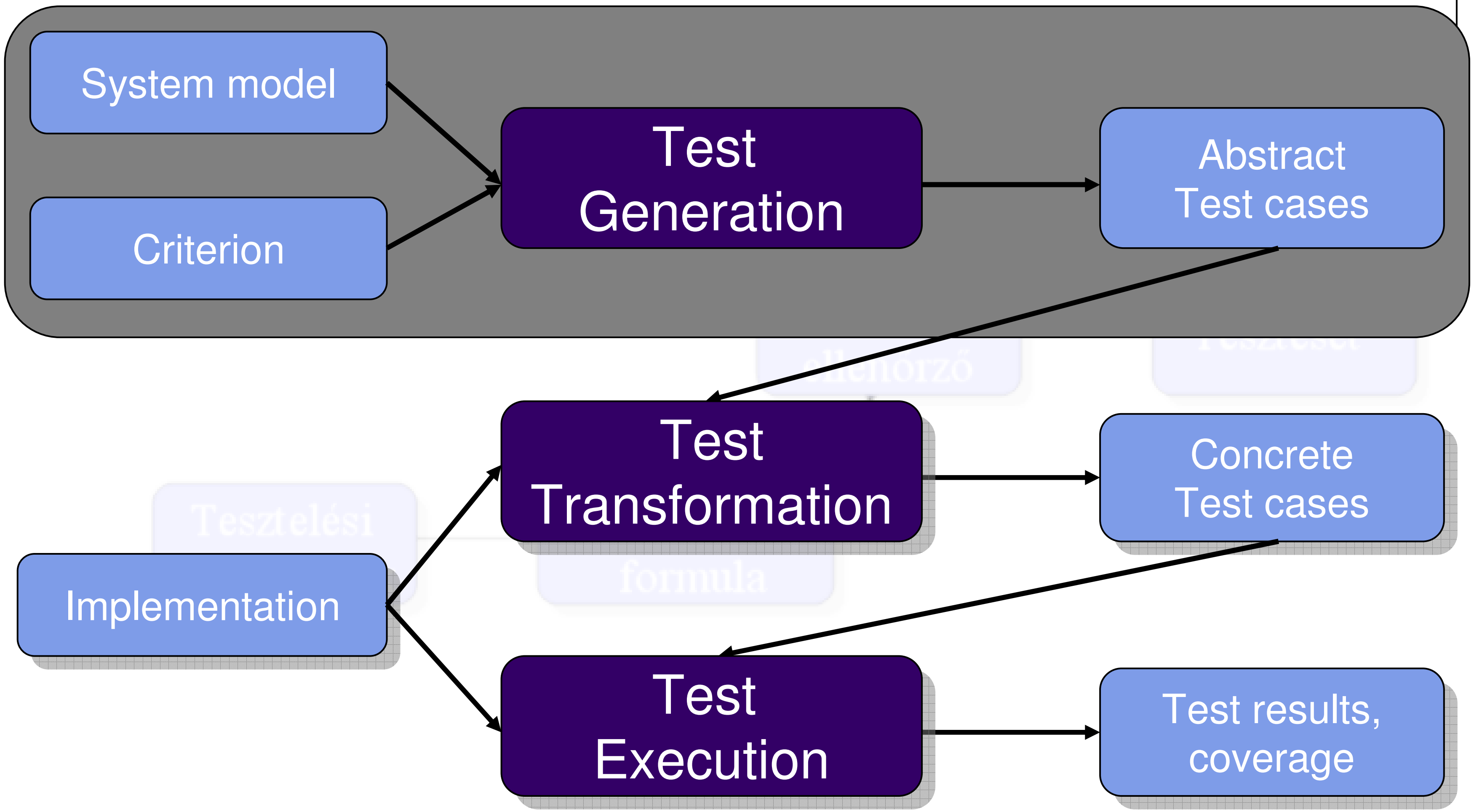
# Problems in testing

- Test cases are written manually
  - Time and resource consuming
- If the specification/code changes all the test cases should be modified
- Detailed behavioural model can help
  - Parts of the testing tasks can be automated

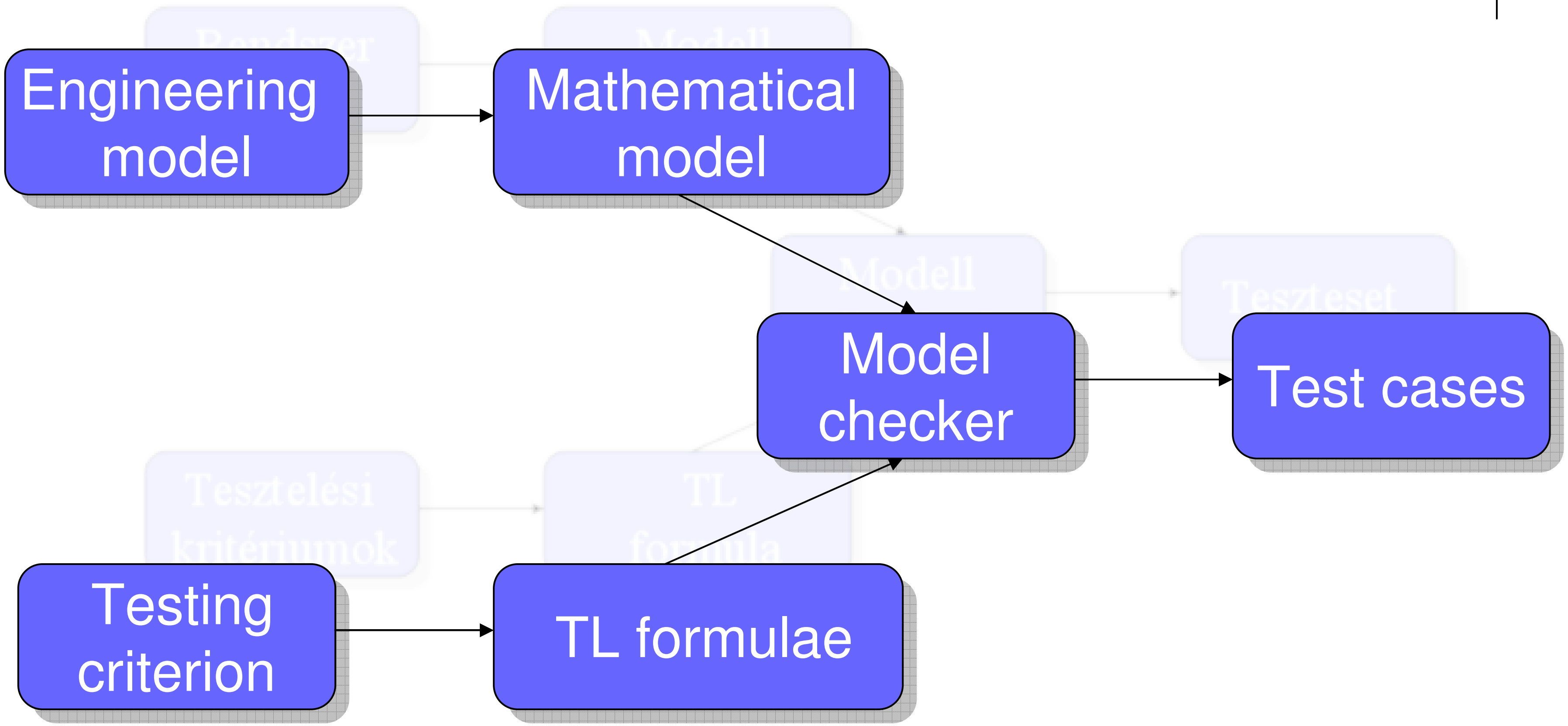
# Testing framework



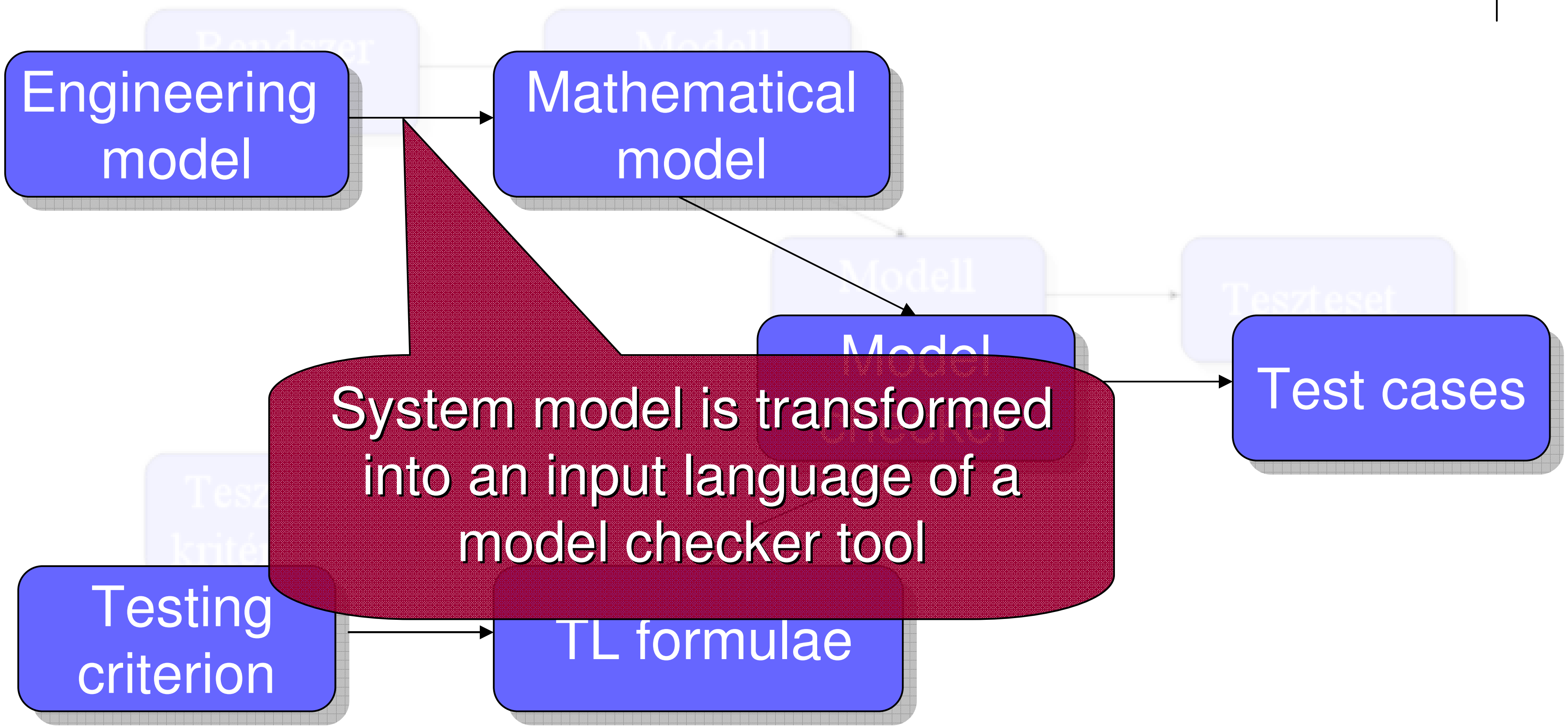
# Testing framework



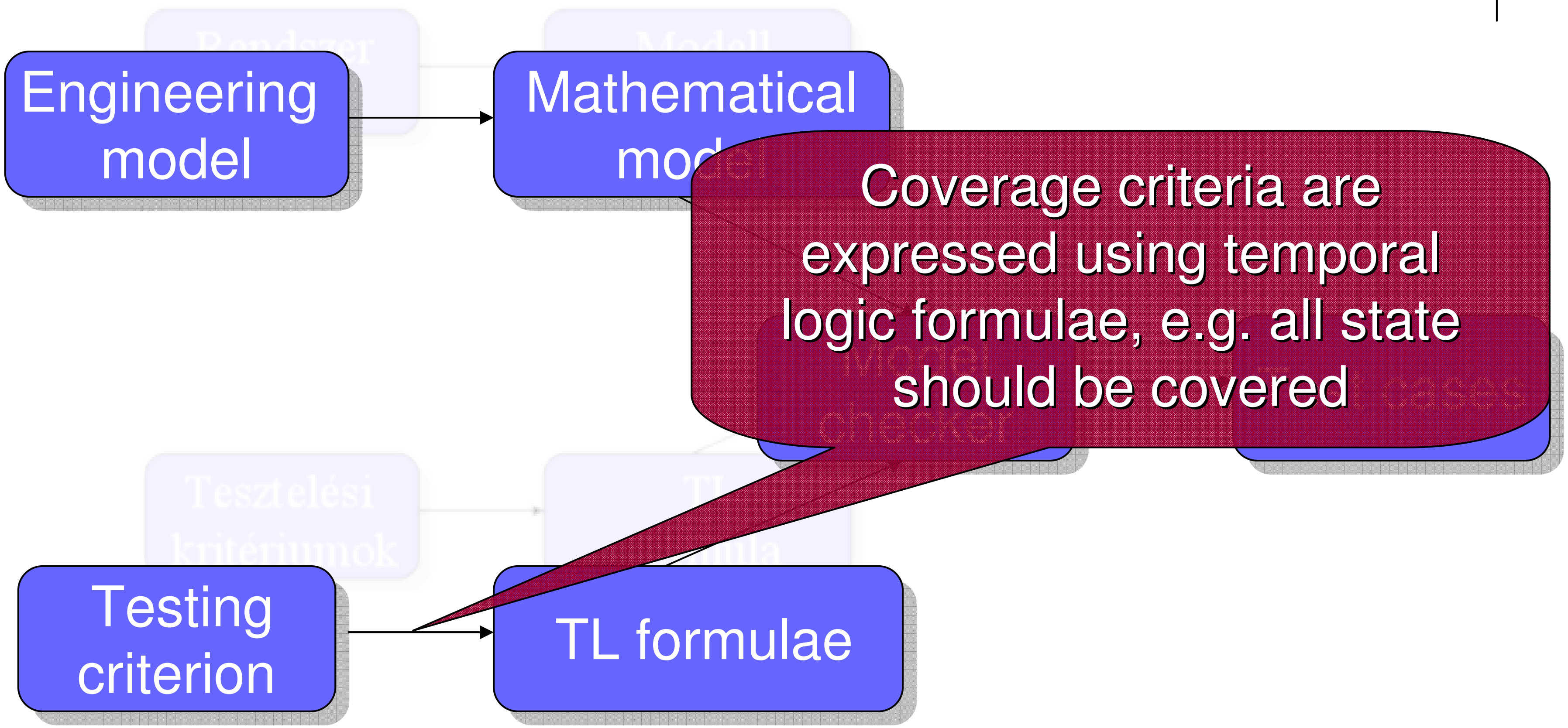
# Test generation using a model checker



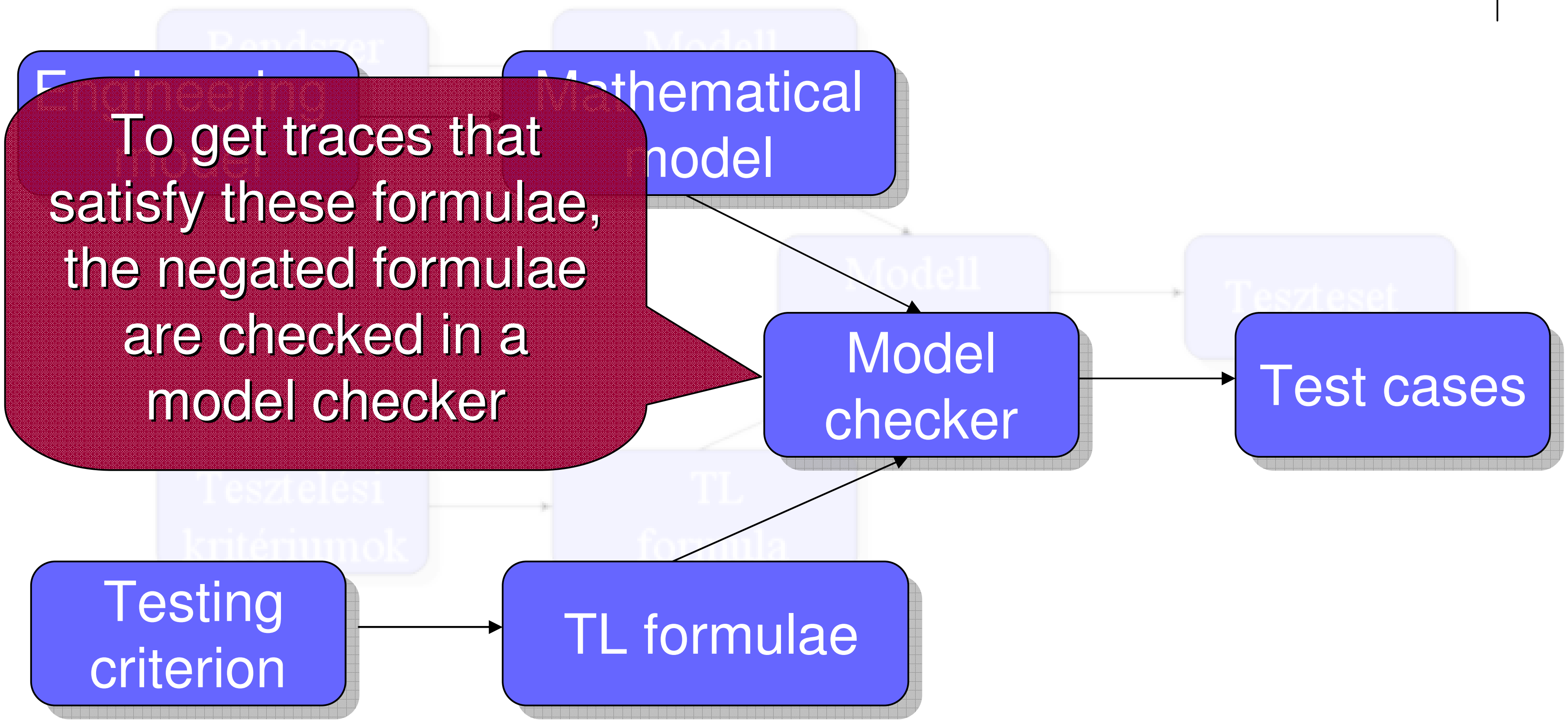
# Test generation using a model checker



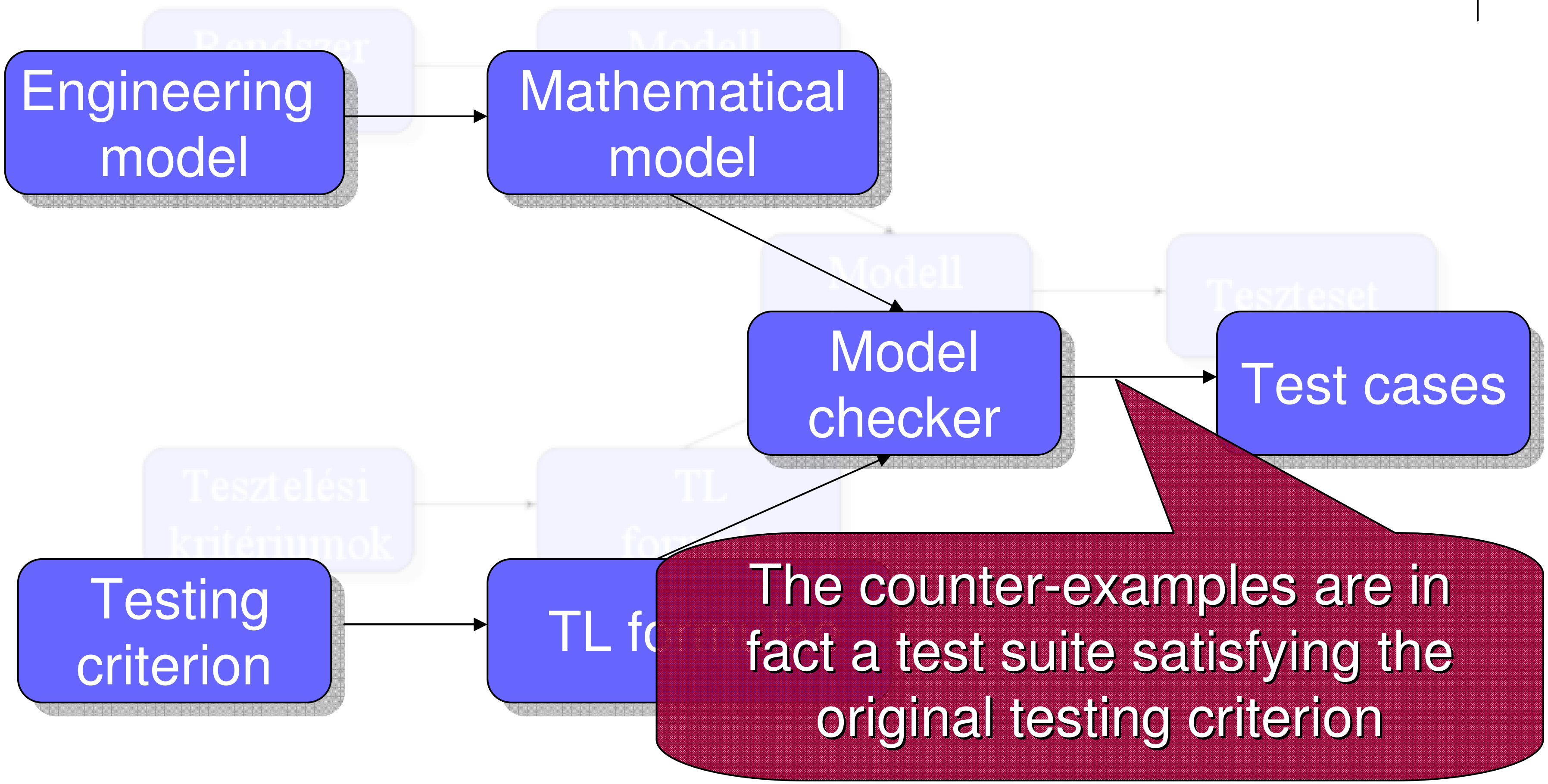
# Test generation using a model checker



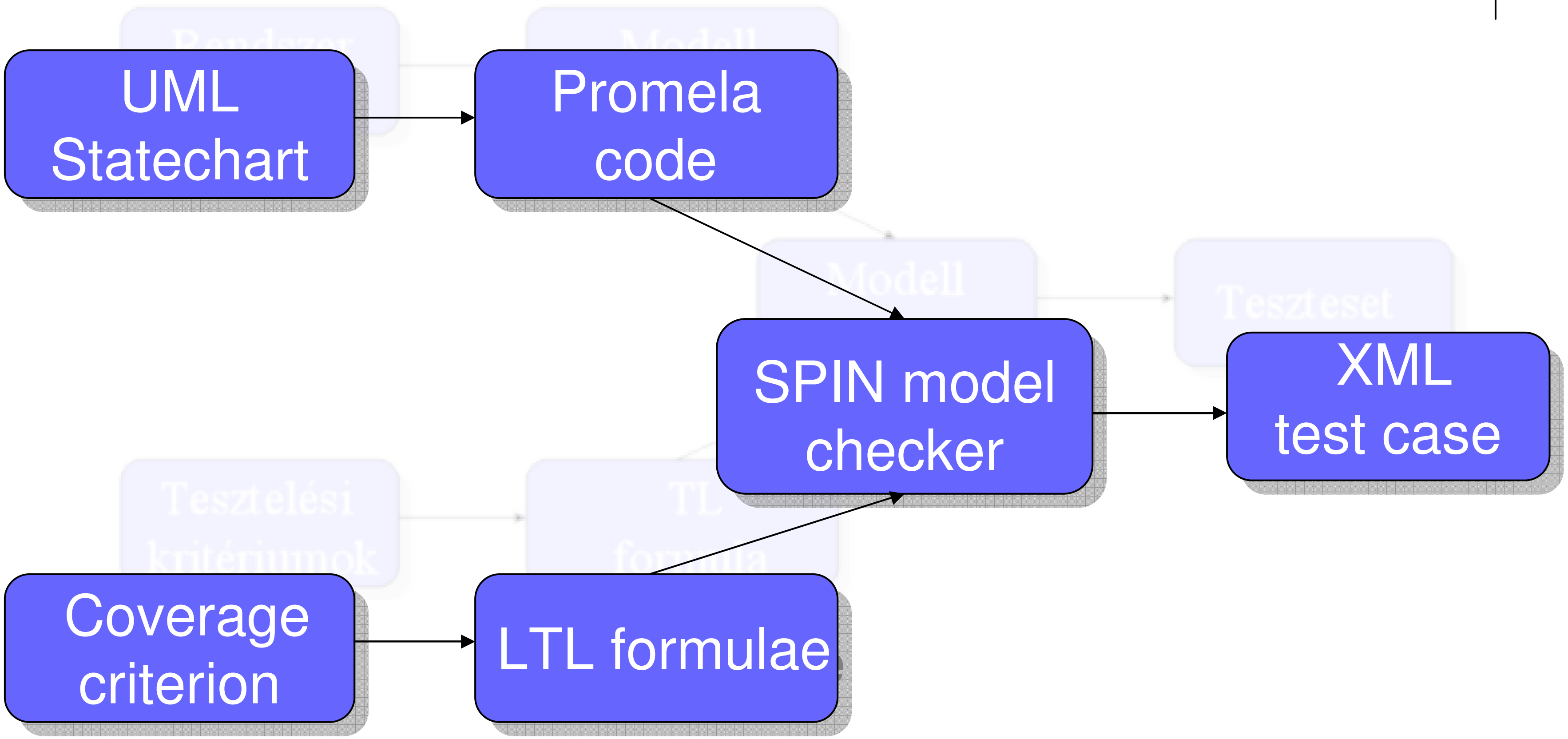
# Test generation using a model checker



# Test generation using a model checker



# Implemented test generator



# Implementation details

- Model transformation
  - UML → *Extended Hierarchical Automata* → Promela
- Test generation input
  - Model: UML Statechart representing the behaviour
  - Criterion: coverage criterion on the model
    - *Currently*: All or selected states/transition covered, custom temporal logic formulae
- Output
  - Events collected from the SPIN detailed trace

# Efficiency of test generation

- Classical model checking: exhaustive verification of the full state space
- Test generation: finding a minimal length counter-example quickly
  → special configuration is necessary.
- Measurements:
  - Effects of SPIN's 10 parameters on runtime
  - Goal: minimize time needed for test generation while test suite is minimal in length

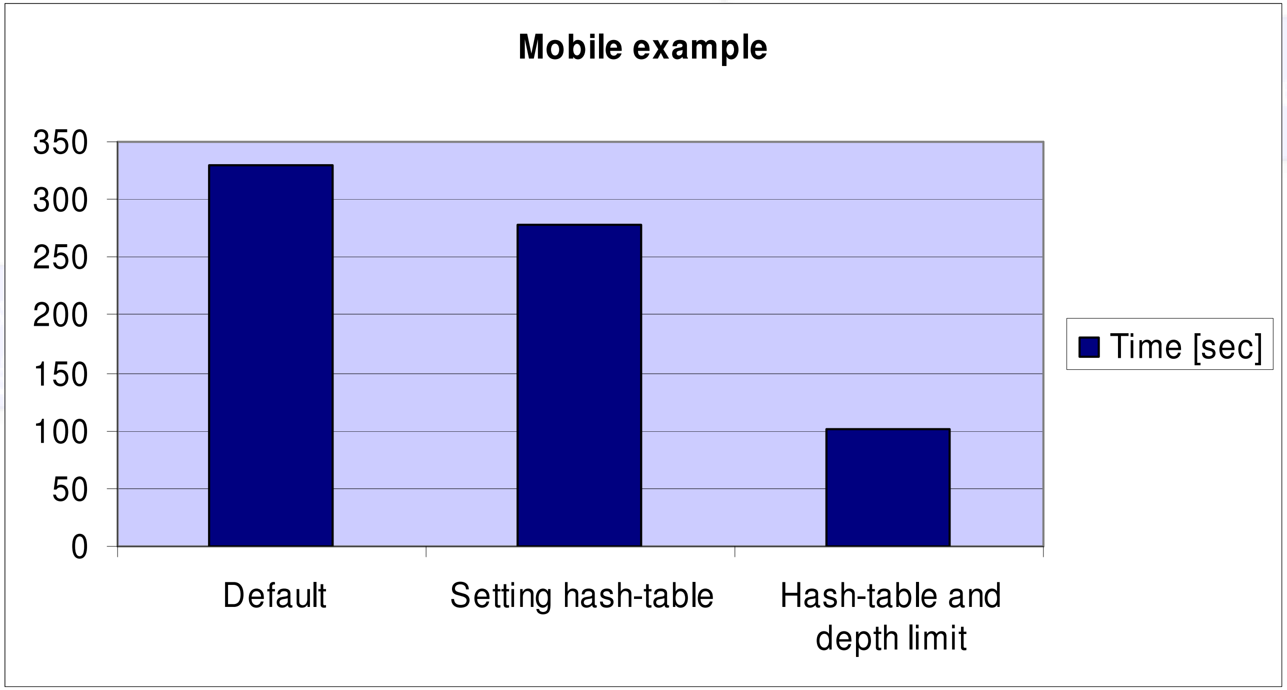
# Analyzed SPIN parameters

- -dBFS: Breadth First Search
- -m: depth limit in Depth First Search
- -i and –I: minimal counter-example iteratively
- -w: size of the hash table to store states

- Other parameters (e.g. –dNOFAIR, -dSAFETY) did not have significant effect

# Experiment 1: Mobile model

- Statechart describing behaviour of a mobile phone
- 10 states, 21 transitions



# Detailed results

| Options | Duration | Sum length | Shortest test |
|---|---:|---:|---:|
| -i | 22m 32.46s | 17 | 3 |
| -dBFS | 11m 48.83s | 17 | 3 |
| -i -m1000 | 4m 47.23s | 17 | 3 |
| -I | 2m 48.78s | 25 | 6 |
| default | 2m 04.86s | 385 | 94 |
| -I -m1000 | 1m 46.64s | 22 | 4 |
| -m1000 | 1m 25.48s | 97 | 16 |
| -m200 -w24 | 46.7s | 17 | 3 |

# Detailed results

| Options | Duration | [illegible] length | Shortest test |
|---|---|---|---|
| -i | 22m 32.46s | 17 | 3 |
| -dBFS | 11m 48.83s | 17 | 3 |
| -i -m1000 | 4m 47.23s | 17 | 3 |
| -l | 2m 48.78s | [illegible] | [illegible] |
| default | 2m 04.86s | 385 | 94 |
| -l -m1000 | 1m 46.64s | 22 | 4 |
| -m1000 | 1m 25.48s | [illegible] | [illegible] |
| -m200 -w24 | 46.7s | [illegible] | [illegible] |

Iterative search: short test cases but long duration

BFS: good results, but ran out of memory in more complex models

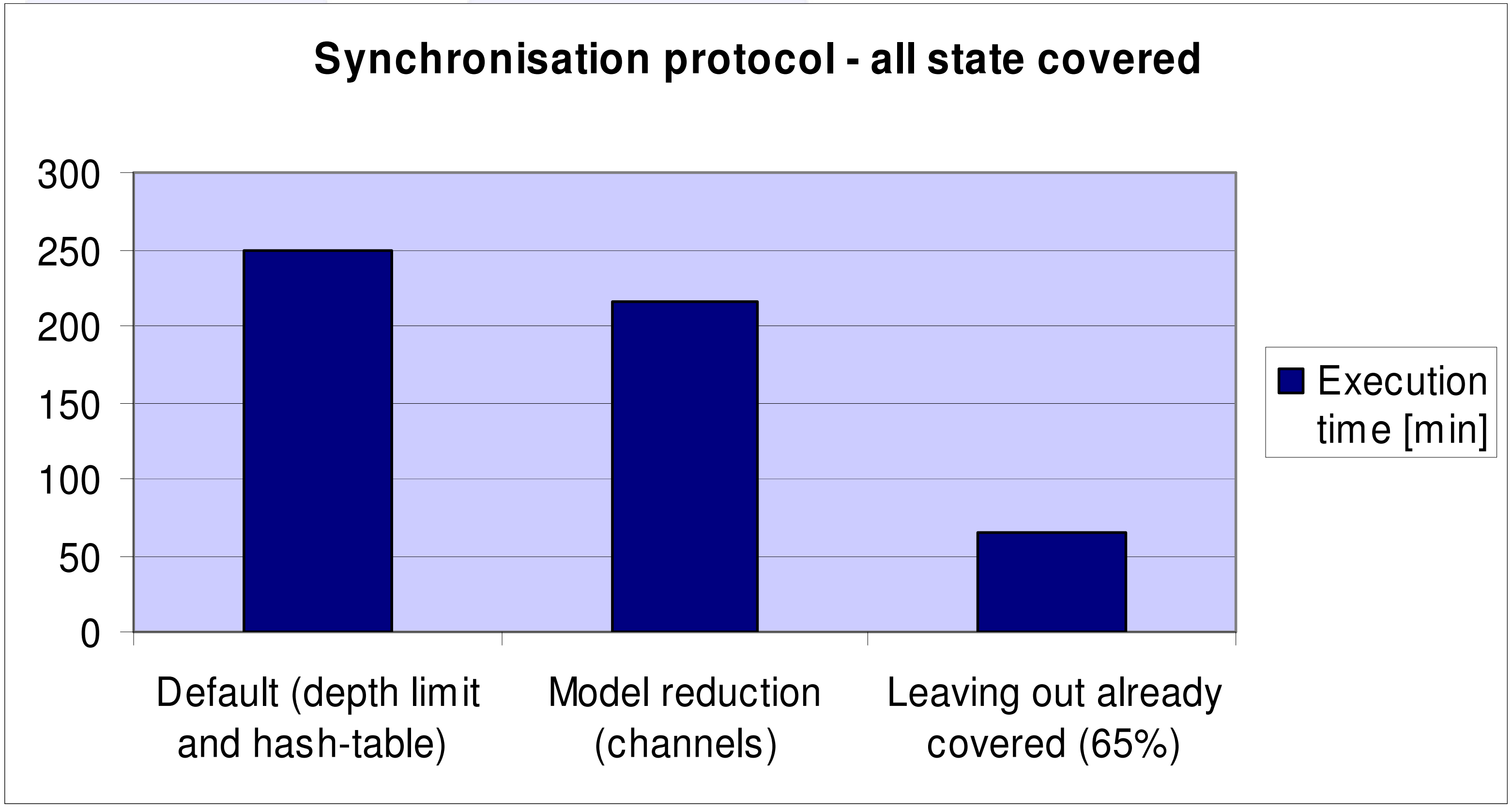
Limiting the depth resulted always in shorter execution

Limiting the depth and setting the hash table was the optimal setting

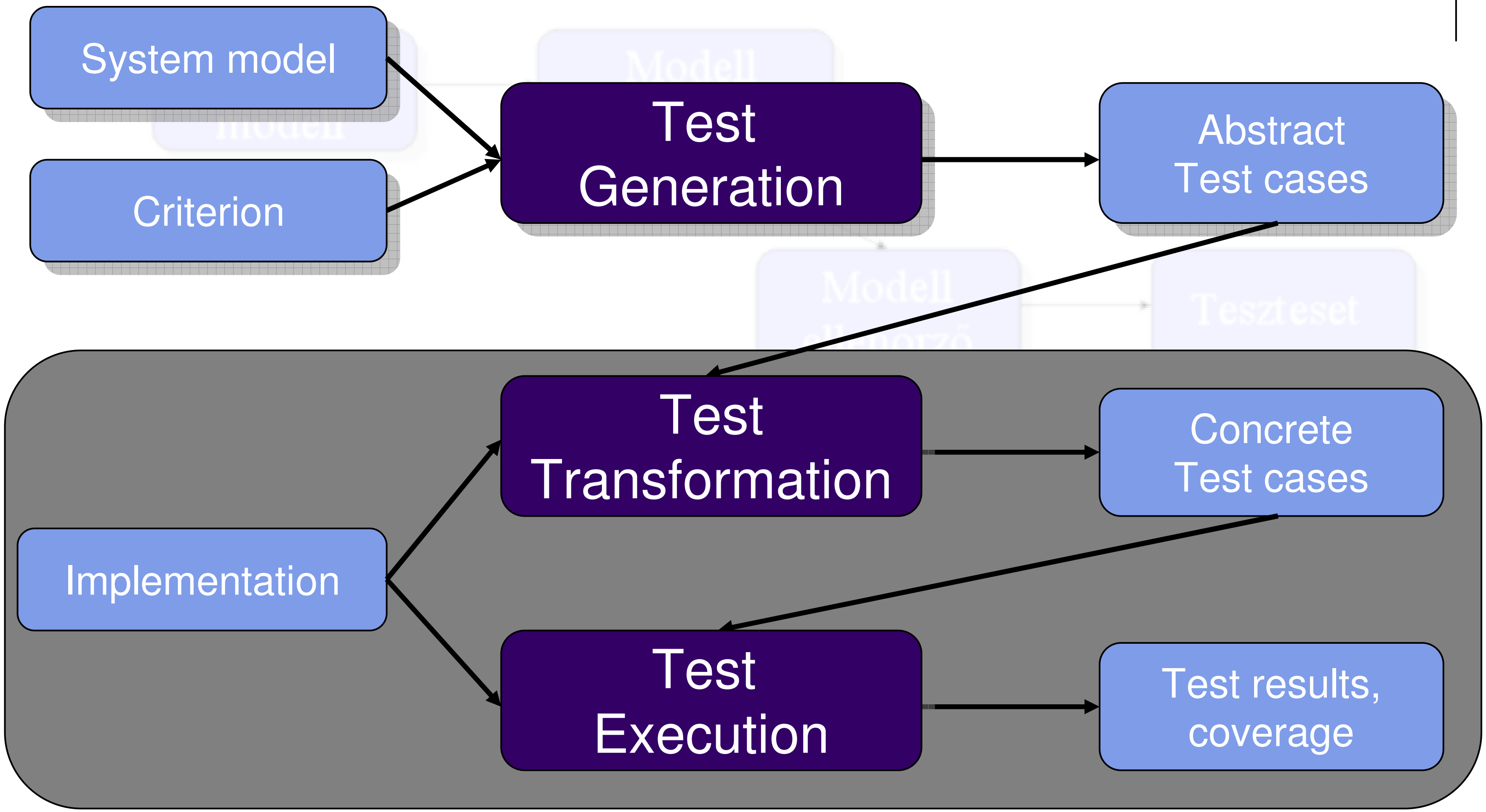
# Real-life case study

- Industrial partner's synchronization protocol
- 5 objects, 31 states, *174 transitions*
- 2e+08 states visited
- Further techniques needed:
  - State space compression – bit-state hashing
  - Reduction of the model in a *conservative manner*
  - Leaving out requirements already covered
- Finding minimal length/size test suite is NP-complete

# Synchronisation protocol - results



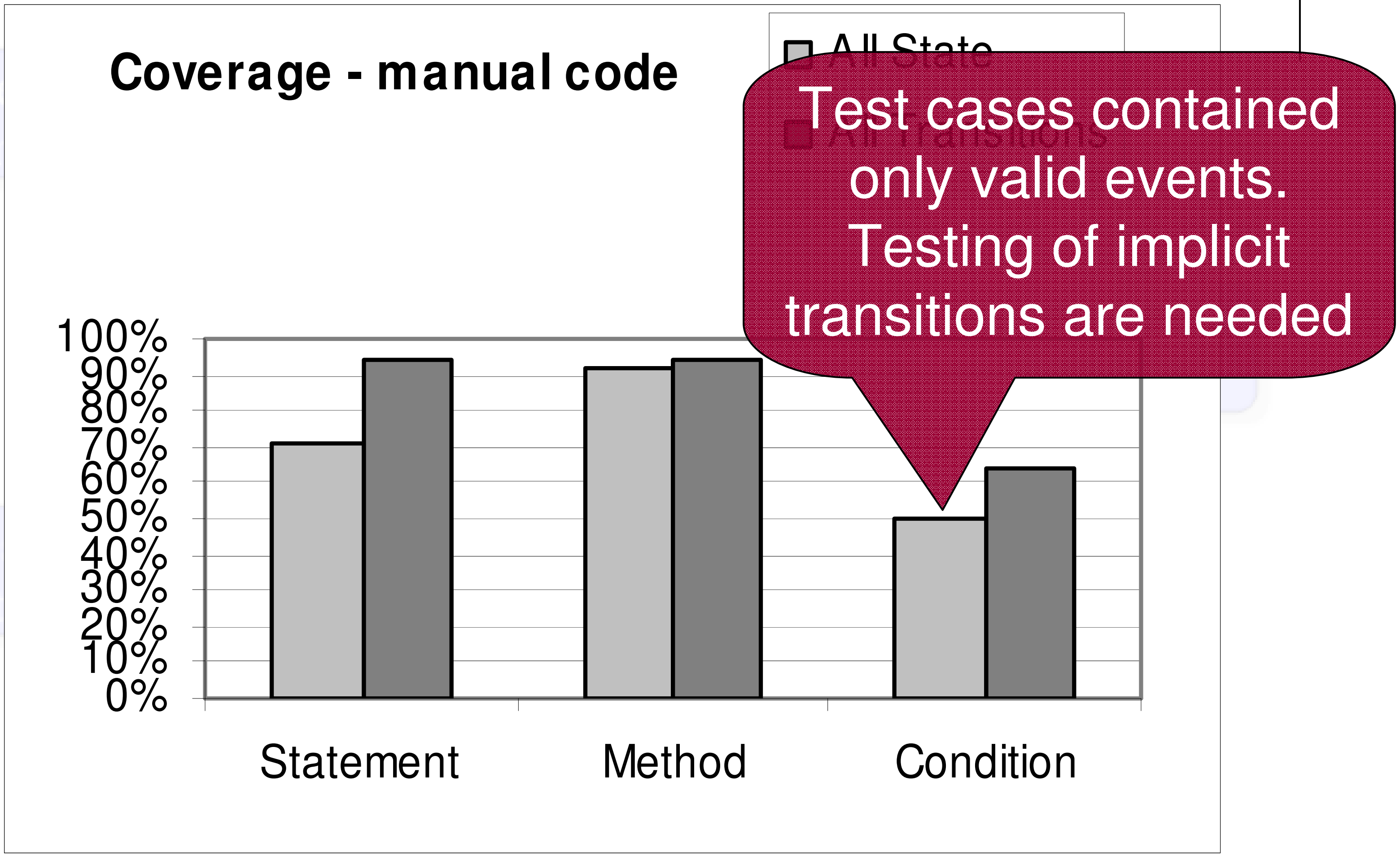
# Testing framework



# Testing implementations

- Use the abstract test cases to test conformance of an implementation
- Definition of a test interface (mapping)
- Conducted experiments
  - Mobile model, with two implementation:
    - manually coded using nested switch method
    - Statechart Java code generator
  - Two test execution framework:
    - JUnit, Rational Robot

# Code coverage results



# Conclusion

- Tool chain for automatic test generation of event-driven systems
- Multiple coverage criteria on the model
- Optimizing model checker for test generation
- Applying tests to implementation
- Code coverage on implementation
- Real-time systems: generating also timing information