Model-based test generation

Zoltán Micskei, István Majzik,
Oszkár Semeráth

Critical Systems Research Group

Learning outcomes

- Illustrate how models can be used in testing (K2)
- Explain the typical model-based test generation process (K2)
- Apply different selection criteria to finite state machines to select test cases (K3)
- Use an MBT tool to generate test cases (K3)

What is model-based testing?

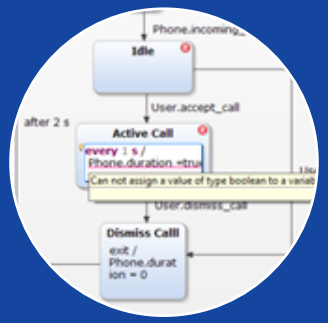
“Testing based on or involving models” [ISTQB]

- Not just test generation
- Not just automatic execution
- Not just for model-driven engineering

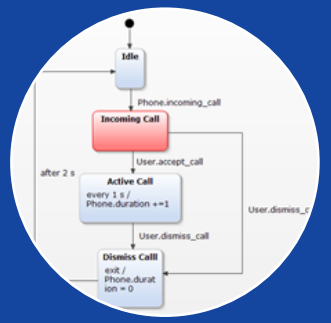
Source of definition: ISTQB. “Foundation Level Certified Model-Based Tester Syllabus”, Version 2015

Landscape of MBT goals

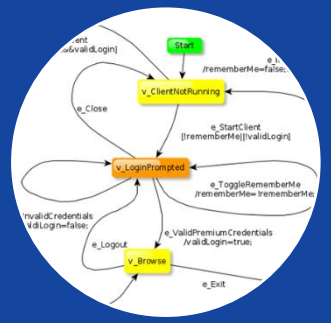
Shared under-standing



Checking specifications



Simulation



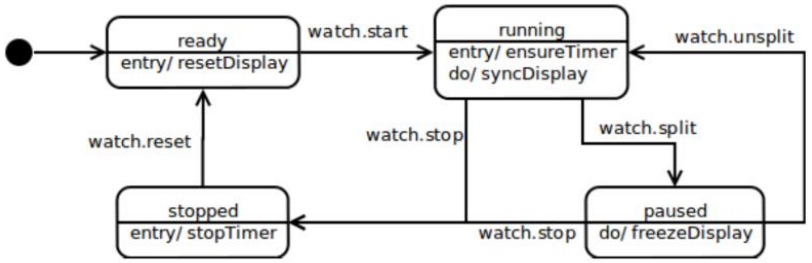
Test data creation

Executable tests

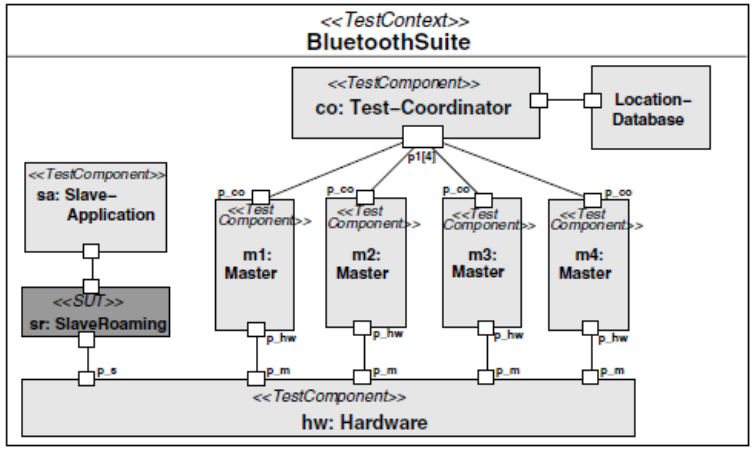
more informal

more formal

Using models in testing (examples)



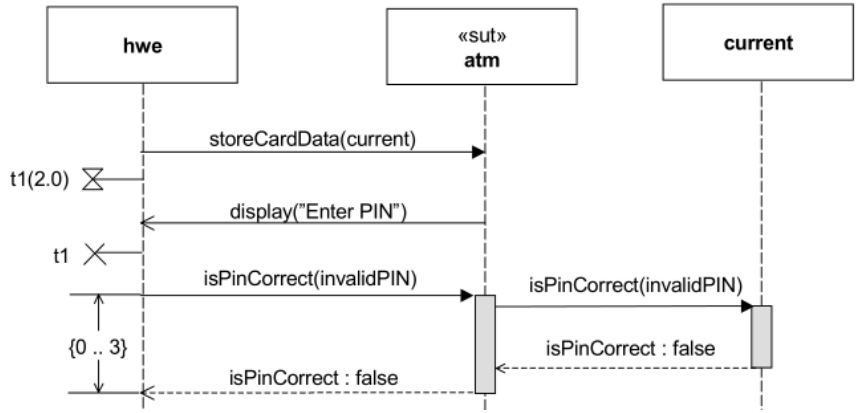
Behavior of SUT



Test configuration

```
timer t;
t.start(5.0);
alt {
 [] i.receive("coffee") {
  Count := Count+1; }
 [] t.timeout { }
}
```

Test sequences



Test sequences

Source: OMG UTP

Benefits of using models

- Close communication with stakeholders
  - Understanding of domain and requirements
- Early testing: modeling/simulation/generation
- Higher abstraction level (manage complexity)
- Automation (different artefacts)

More specific meaning: Test generation

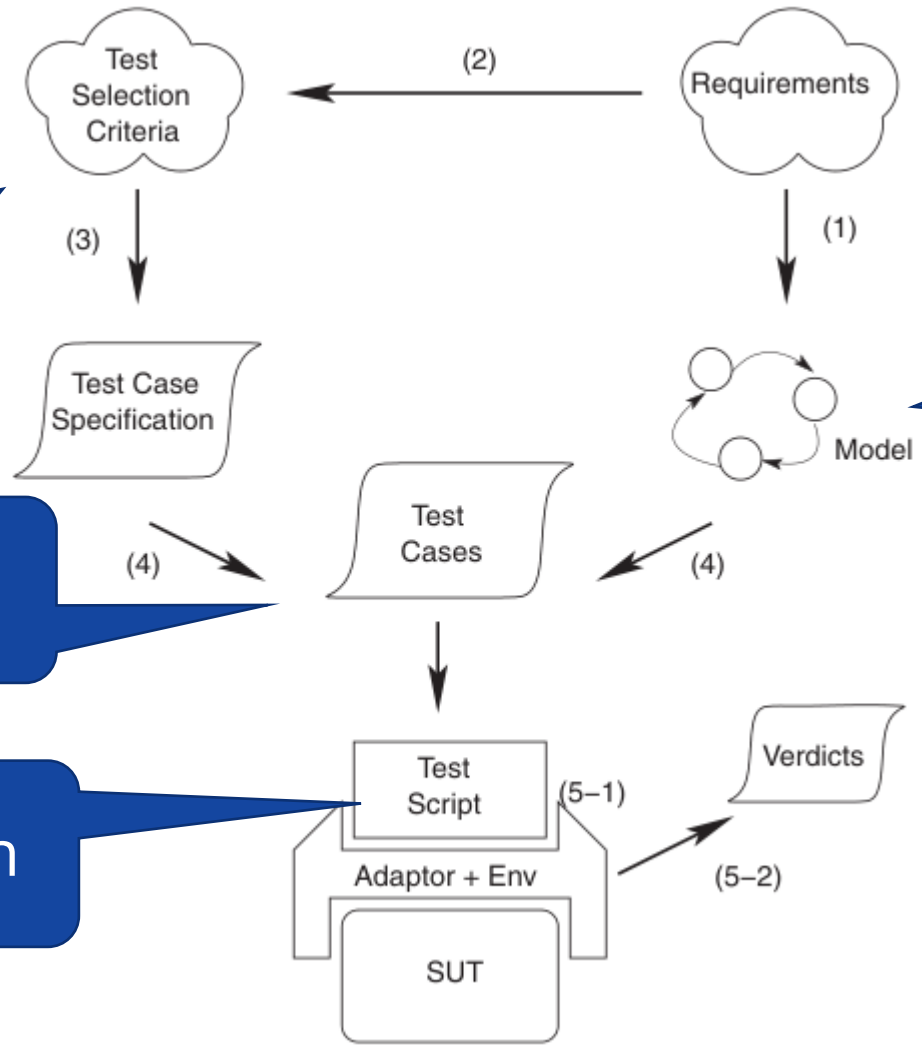
„MBT encompasses the processes and techniques for

- the automatic derivation of abstract test cases from abstract models,
- the generation of concrete tests from abstract tests,
- the manual or automated execution of the resulting concrete test cases”

Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

Typical MBT process

State, path, requirement coverage...



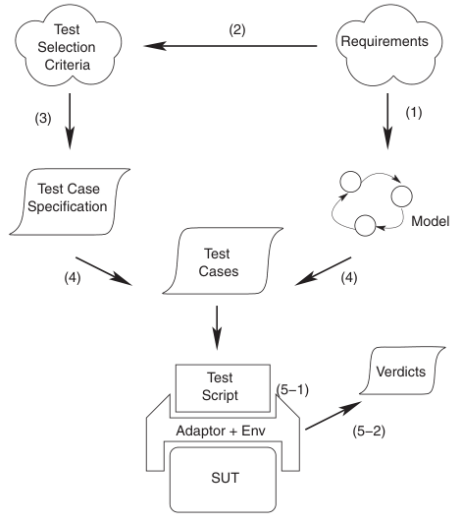
Test model

Abstract test case

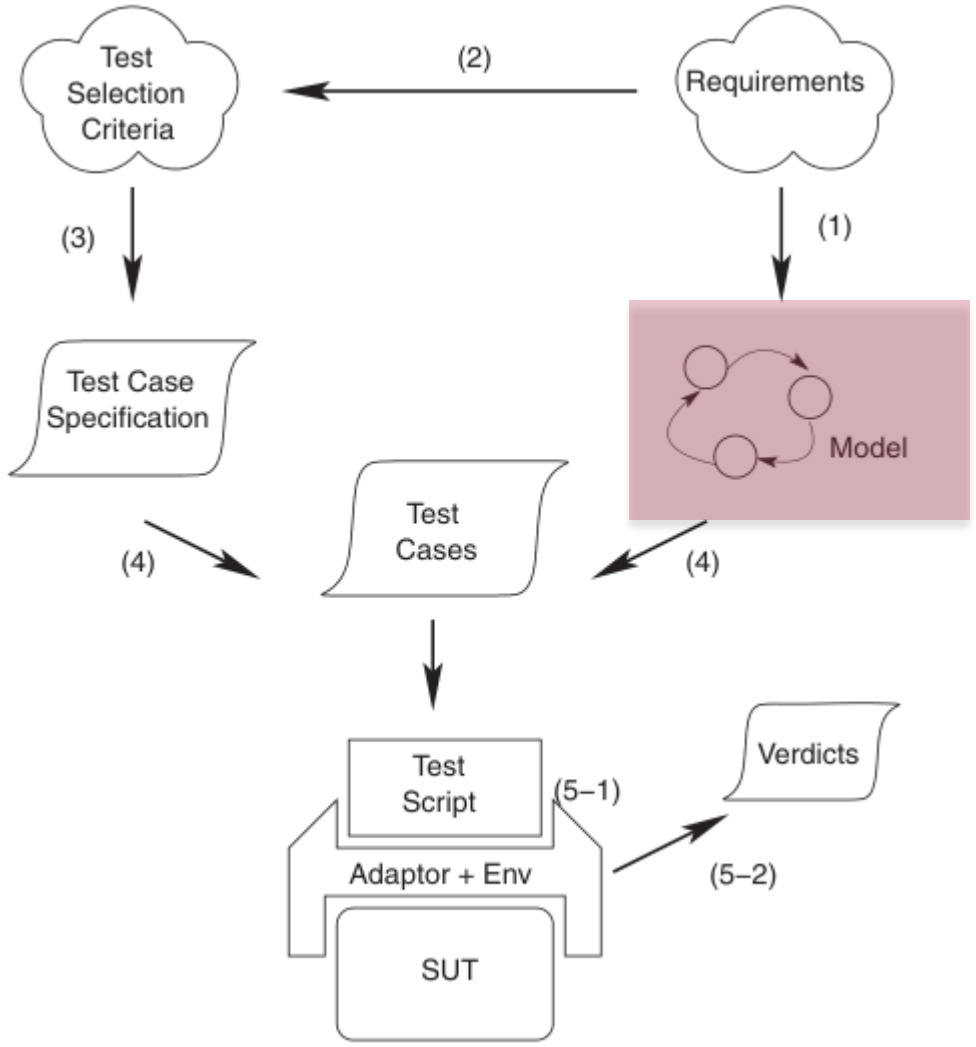
Concretization

Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

MBT process


Test Selection Criteria
(2)
Requirements
(3)
(1)
Test Case Specification
Model
Test Cases
(4)
(4)
Test Script
(5–1)
Verdicts
Adaptor + Env
(5–2)
SUT

Typical MBT process



Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

Questions for modeling

- What to model?
  - What is the test object?
  - Functionality / performance factors / …
- What abstraction level to use?
  - Too many or too few details
  - Separate models for different test objectives
- What modeling language to use?
  - Structural, behavioral

Focus of the model

System

- System as intended to be
- Conformance of model-SUT

Usage

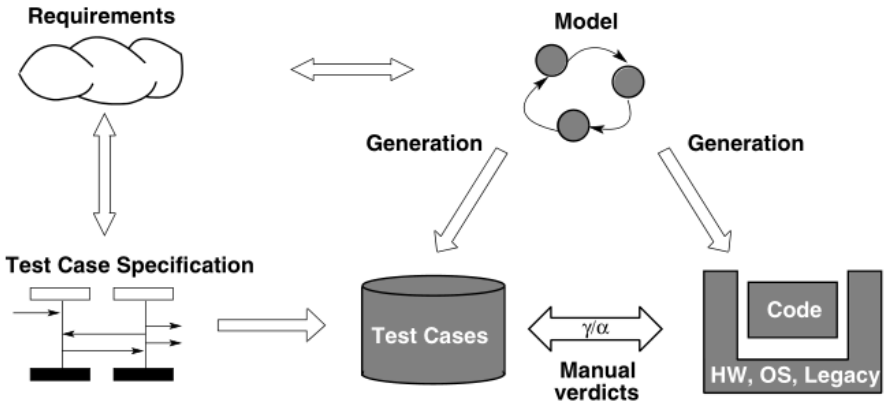
- Model environment/users
- Inputs to the system

Test

- Model one or more test case
- E.g. sequences + evaluation

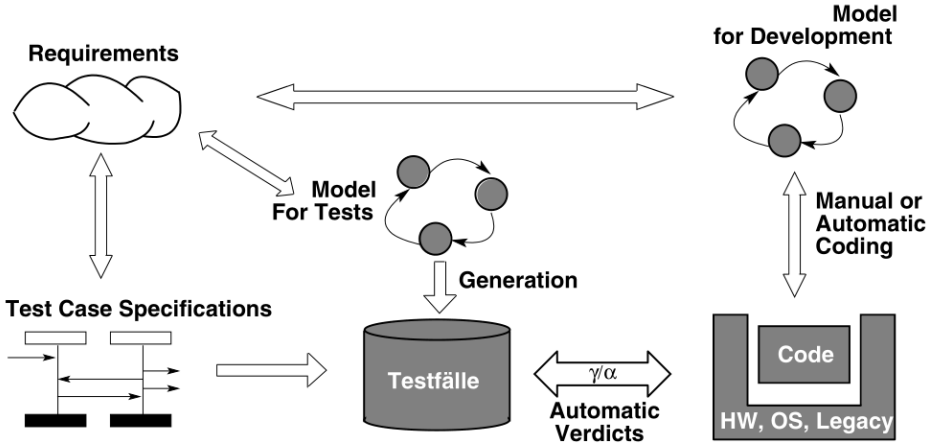
Reuse: Development and Test modeling

What if I have existing design models?



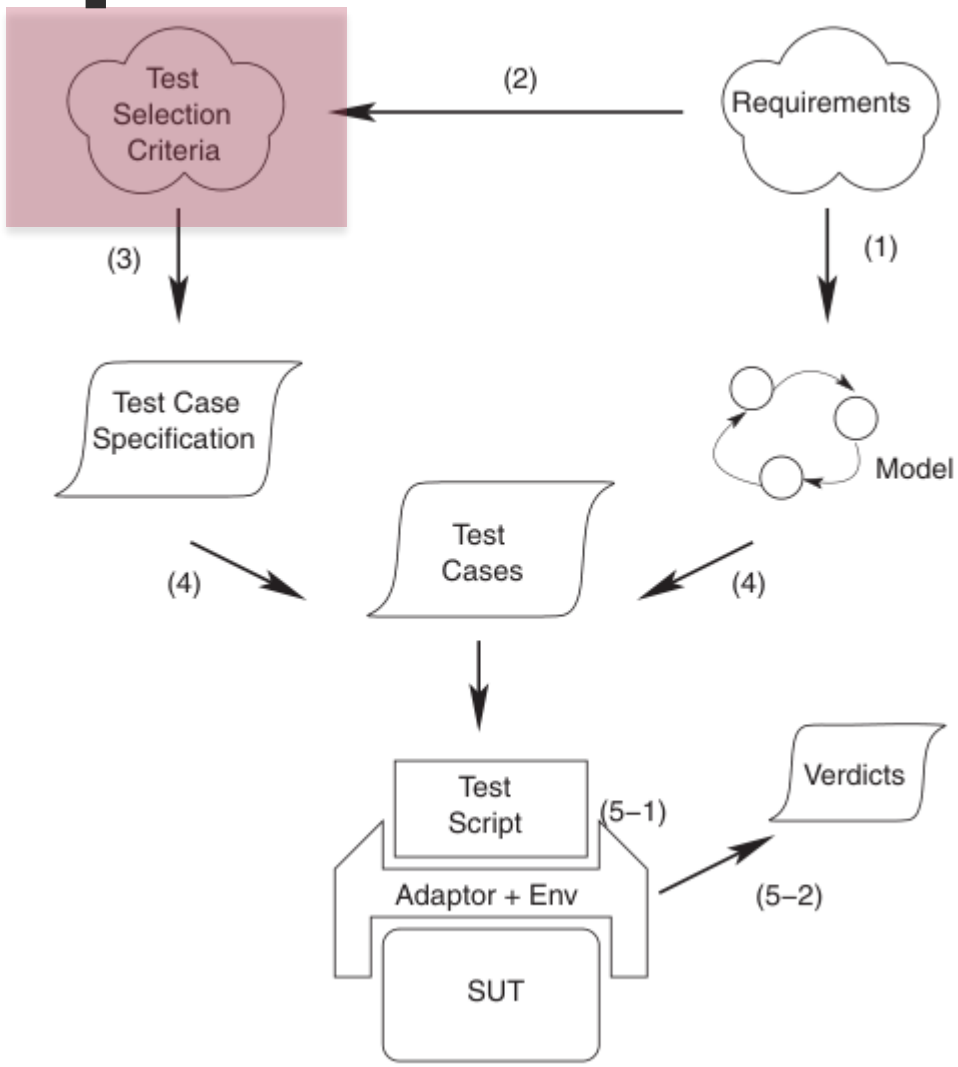
Problem: what do we test here?

Approach: separate dev. and test models



A. Pretschner, J. Philipps. „Methodological Issues in Model-Based Testing", Model-Based Testing of Reactive Systems, 2005.

Typical MBT process

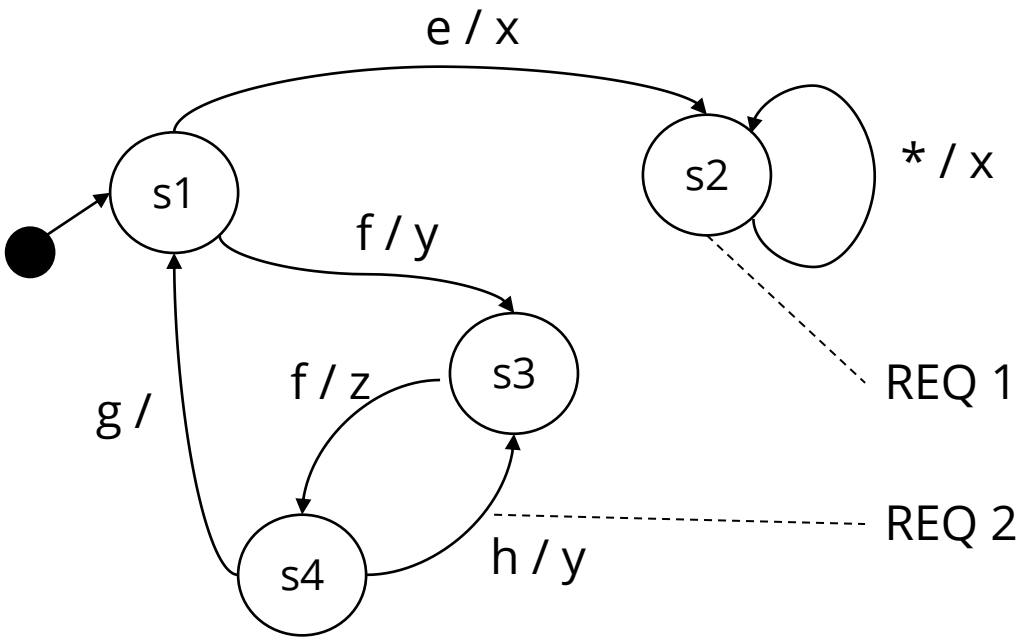


Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

Typical test selection criteria

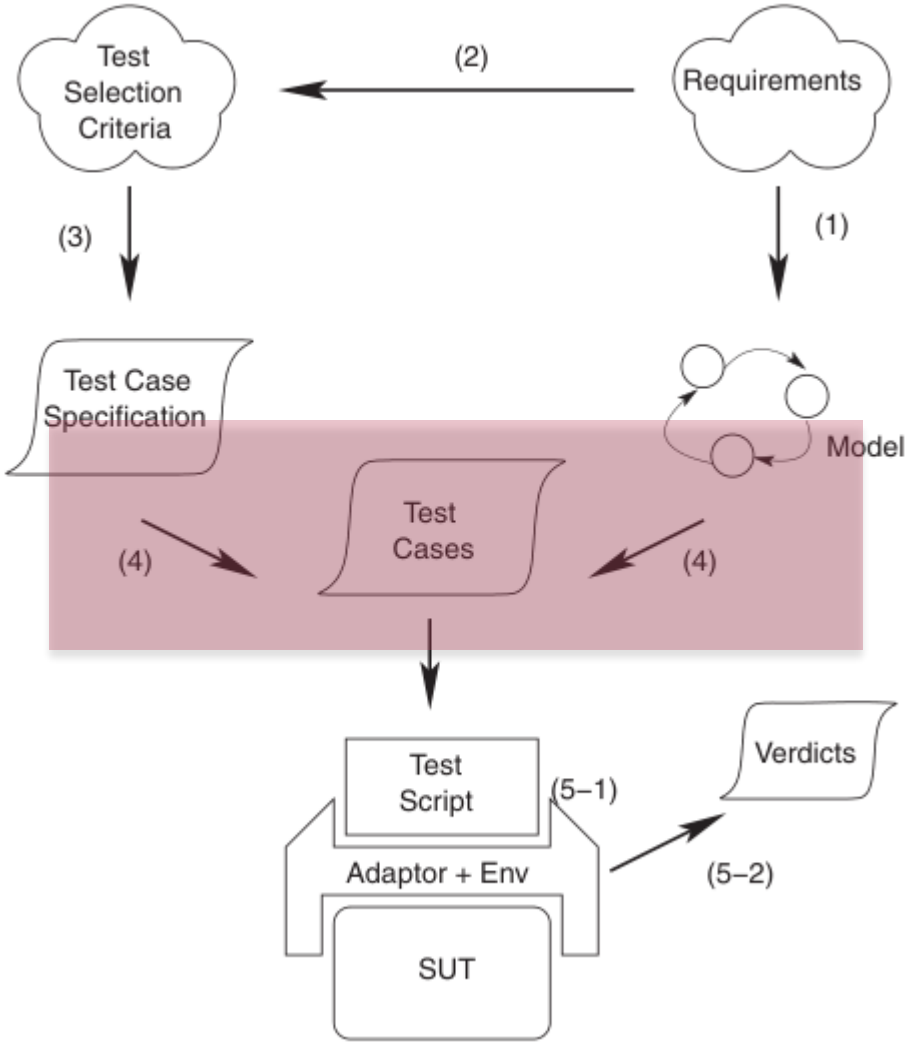
- Coverage-based
  - Requirements linked to the model
  - MBT model elements (state, transition, decision…)
  - Data-related (see spec. test design techniques)
  - Behaviour-related (component interaction)
- Random / stochastic
- Scenario- and pattern-based (use case…)
- Project-driven (risk, effort, resources…)

EXAMPLE Test selection for state models



- Select test cases for full
  - requirement coverage
  - state coverage
  - transition coverage

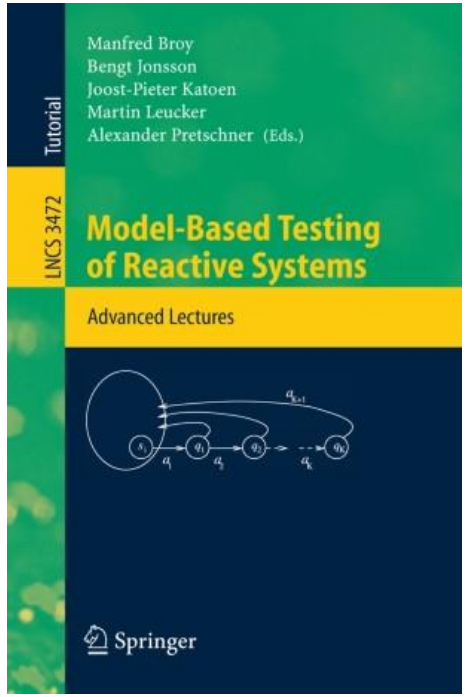
Typical MBT process



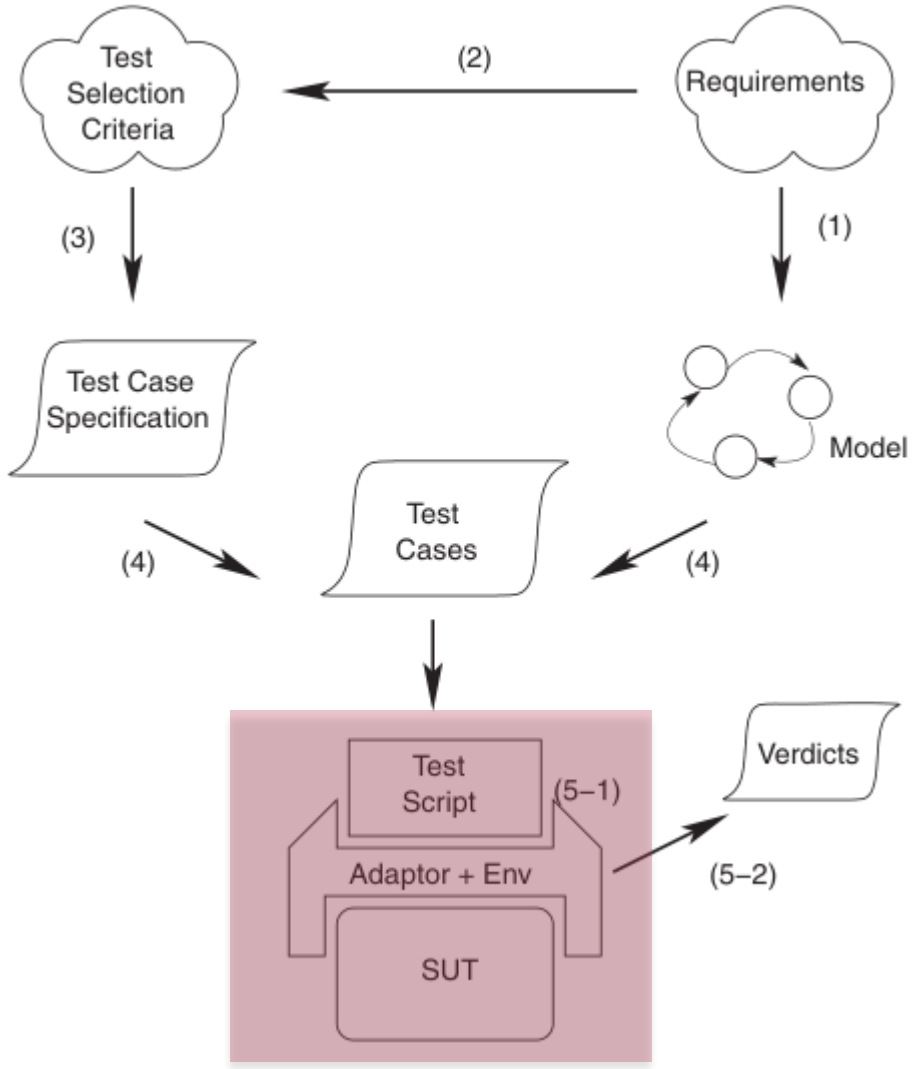
Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

Test generation methods (sample)

- Direct graph algorithms
  - Transition coverage →
    "New York Street Sweeper problem"
- Finite State Machine (FSM) testing
  - Homing and synchronizing sequences, state identification and verification, conformance…
- Labeled Transition System (LTS) testing
  - Equivalence and preorder relations, ioco
- Using model checkers
- Fault-based (mutation)



Typical MBT process



Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

Abstract and concrete test cases

- Abstract test case
  - Logical predicate instead of values
    (e.g., SLOW/FAST instead of 122.35)
  - High-level events and actions

Abstraction gap!

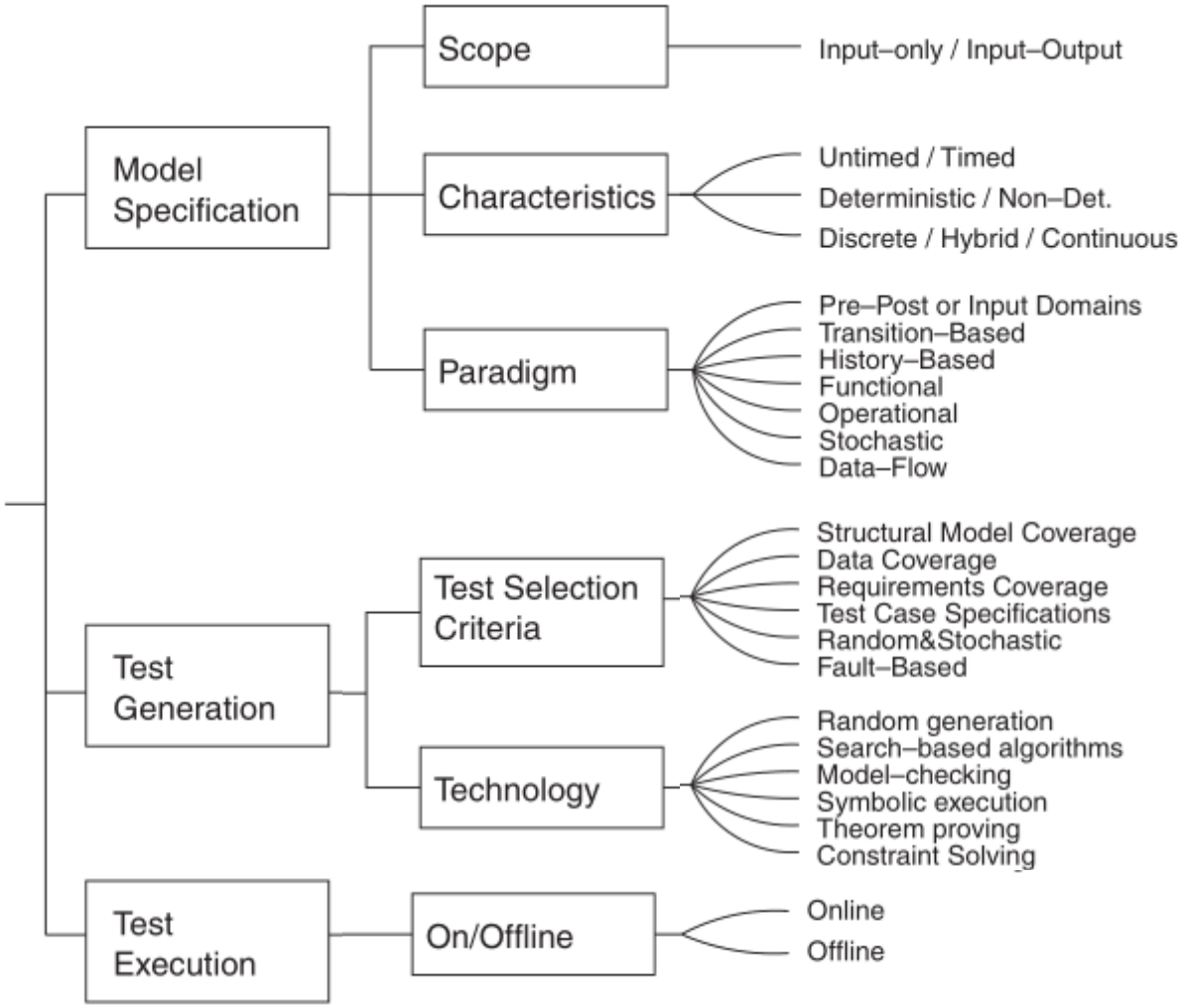
- Concrete test case
  - Concrete input data
  - Detailed test procedure (manual or automatic)

Adaptation (automatic execution)

- Adaptation layer
  - Code blocks for each model-level event and action
  - Wrapper around the SUT

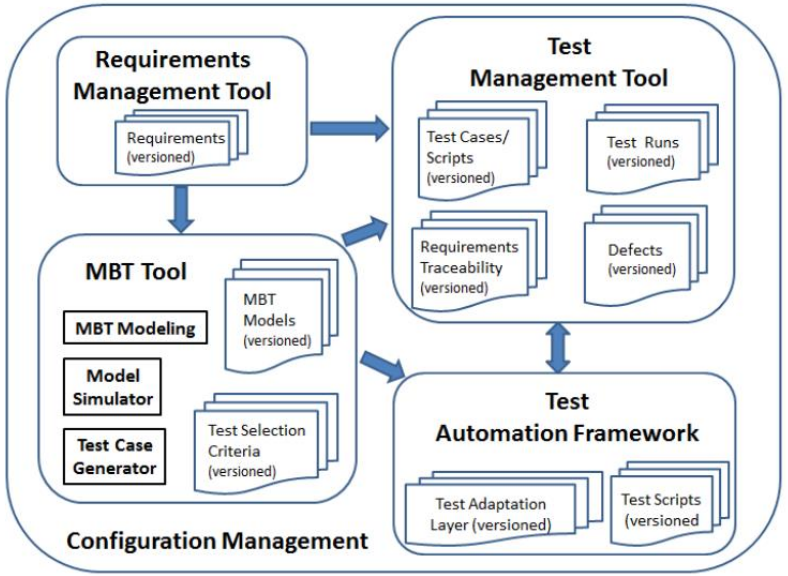
- See: Keyword-driven testing

Summary: Taxonomy of MBT approaches



Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches", STVR 2012; 22:297–312

Tools and Case Studies



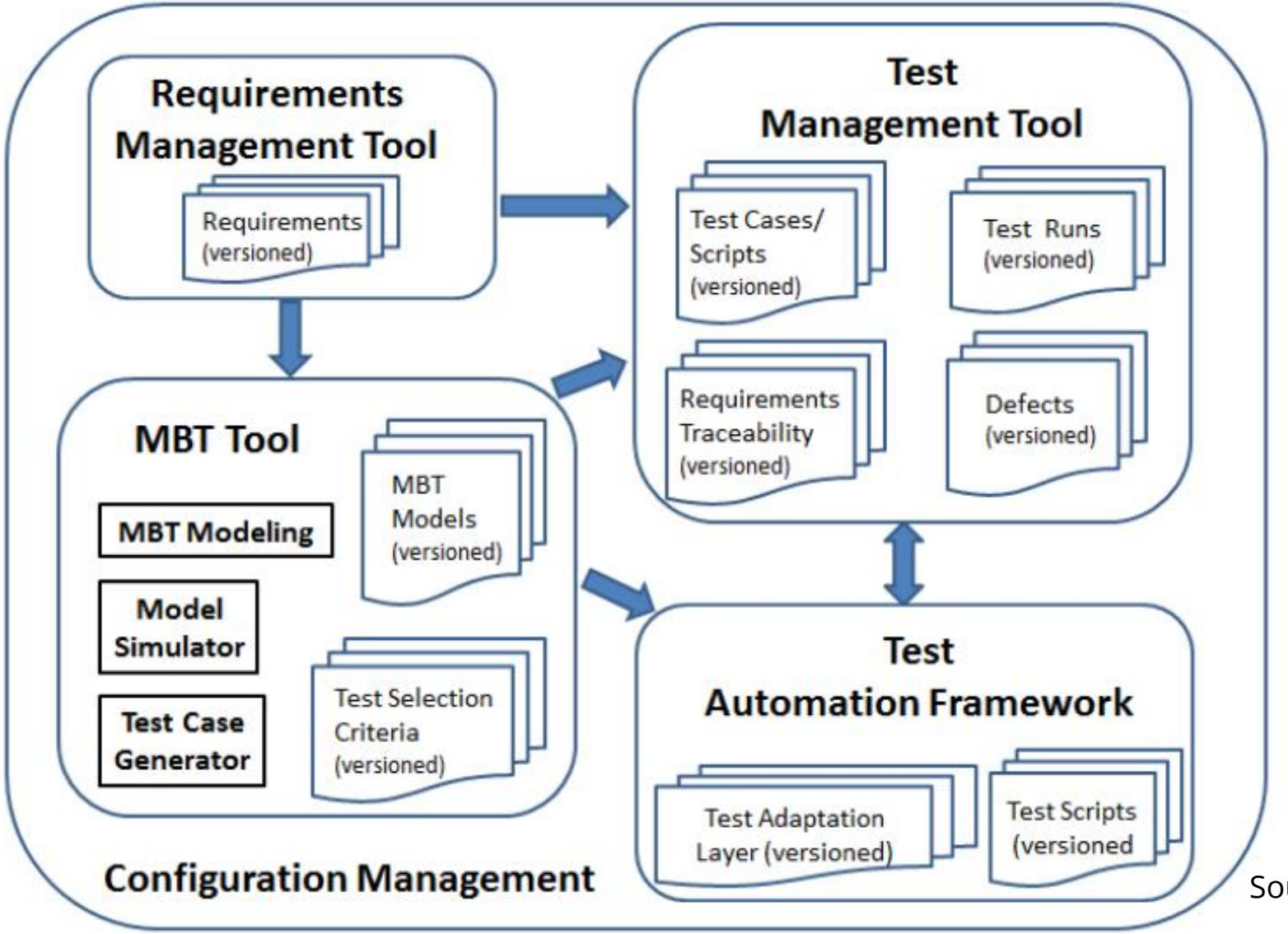
Typical use cases

- **Fast & easy**
  - Simple modeling
  - Using open tools

- **Full fledged**
  - Complex, commercial tool
  - Full lifecycle support

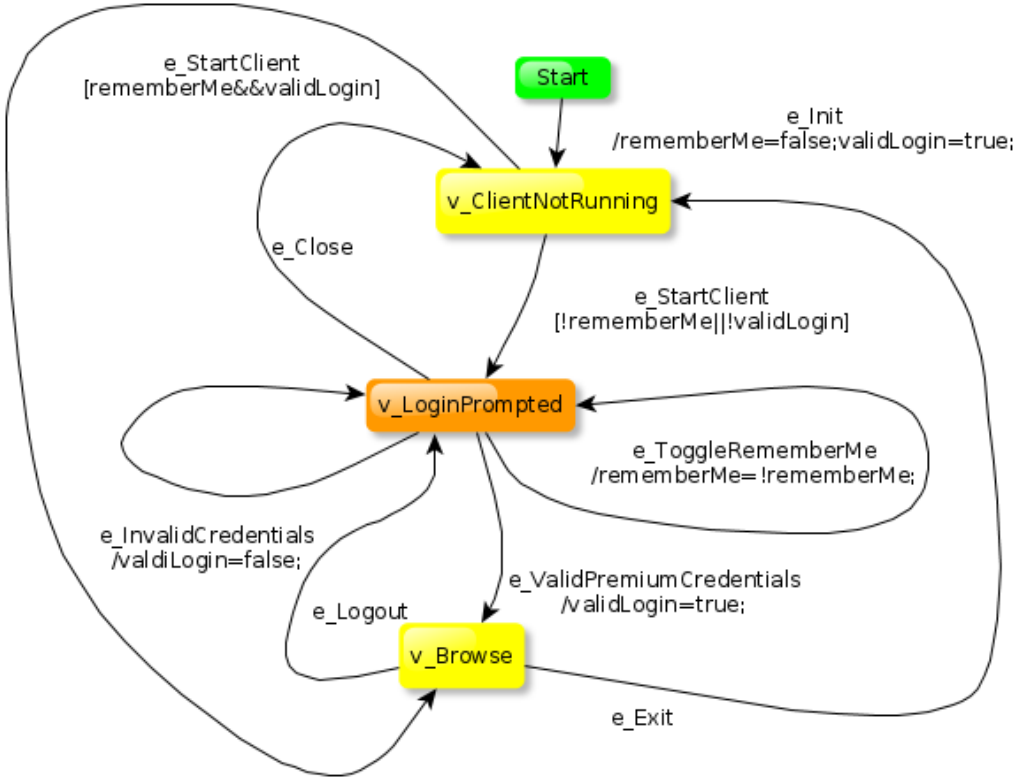
- **Advanced**
  - Custom modeling languages/tools

MBT tool chain



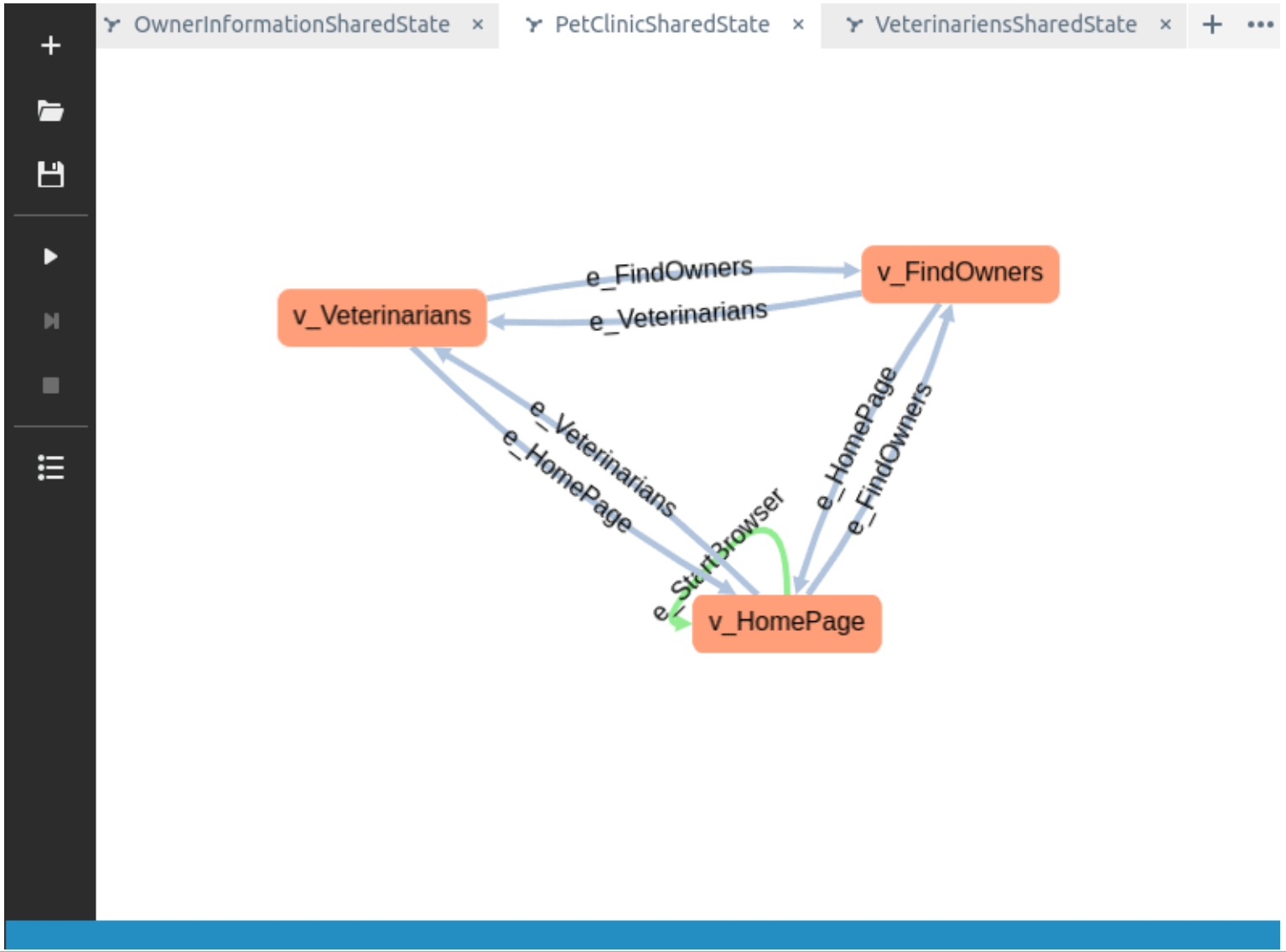
Source: ISTQB syllabus

Open source tool: GraphWalker



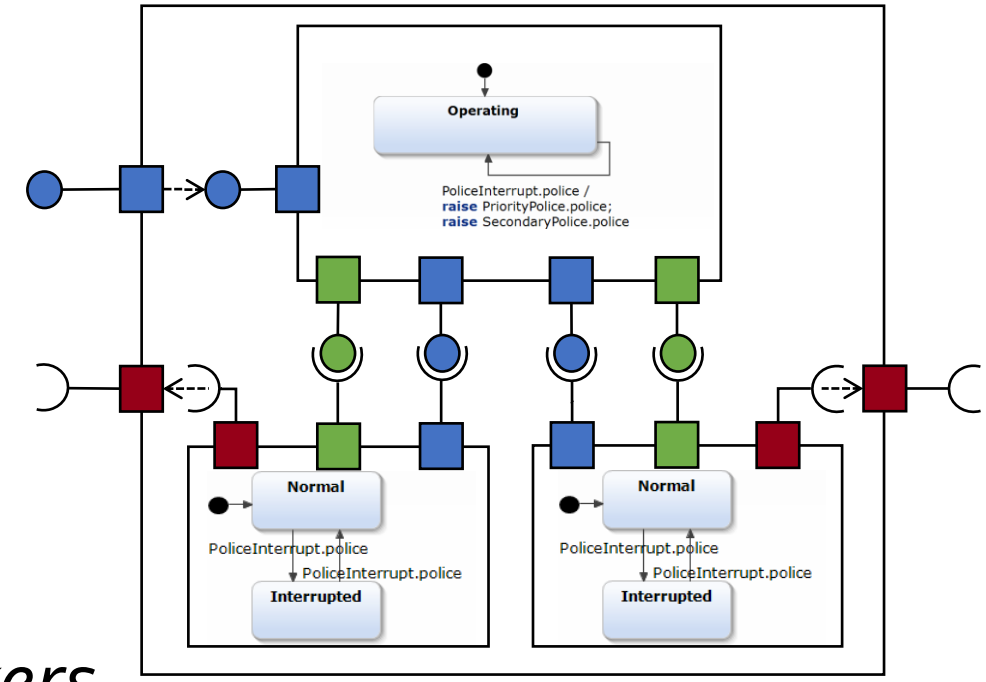
Source: GraphWalker

- FSM model + simple guards
- Coverage: state, transition, time limit (random walk)
- Traversing the graph: random, A*, shortest path
- Generating JUnit test stubs (adapter)


OwnerInformationSharedState
PetClinicSharedState
VeterinariensSharedState
v_Veterinarians
e_FindOwners
e_Veterinarians
v_FindOwners
e_Veterinarians
e_HomePage
e_HomePage
e_FindOwners
v_HomePage

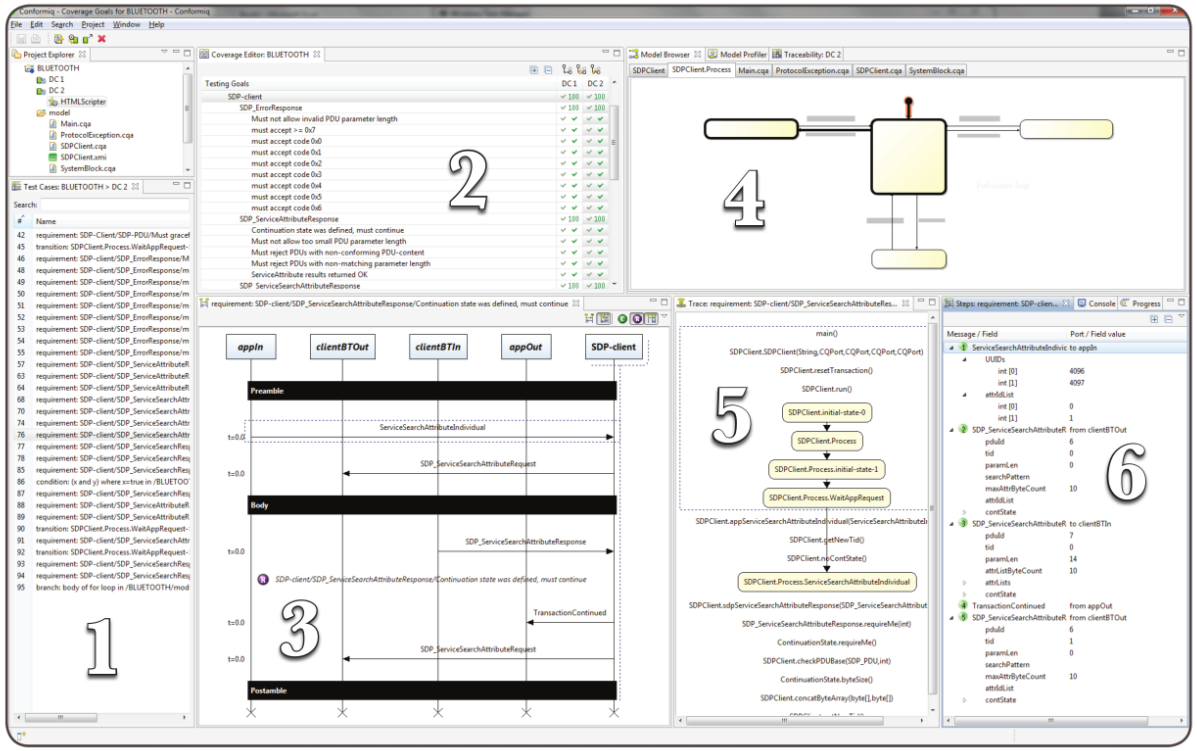
Open source tool: Gamma framework

- UML/SysML-based *statecharts* (*GSL*) + *topology* (SysML *ibd*) *descriptions* (*GCL*)
- Test coverage criteria
  - State, transition, transition-pair
  - Interaction
  - Dataflow
- Traversing the model using *model checkers*
  - UPPAAL
  - Theta
- Generating abstract test cases (GTL) and concrete JUnit tests
  - Implementation adapter: Reflective Java API



https://github.com/ftsrg/gamma

Industrial MBT tool – Conformiq

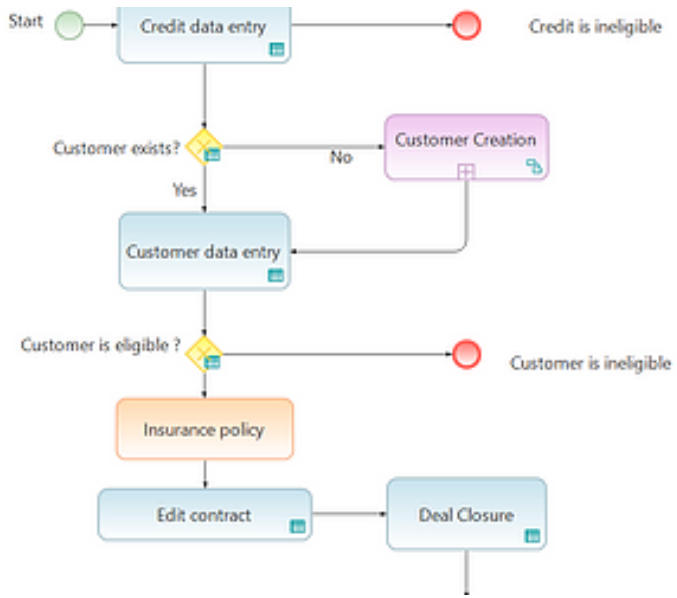


Conformiq Designer IDE for automatic test case generation

Source: Conformiq. „Testing Bluetooth Protocol Stacks with Computer-Generated Tests". Technology brief. 2010

- State machine models + Java action code
- Coverage: requirement, state, transition…
- Integration with numerous other tools

Industrial MBT tool – Smartesting Yest



	Actions	Expected results
1	Check Credit data : Amount : **599** Duration : **btwn 10 and 36 moonths** Type of Goods : **white_goods**	Check message: **Valide**
2	Check customer credentials: Age : **35** Profession : **employee**	check that client is : **Valide**
3	Edit contract	
4	Signature of contract	Check that contract is correcly saved

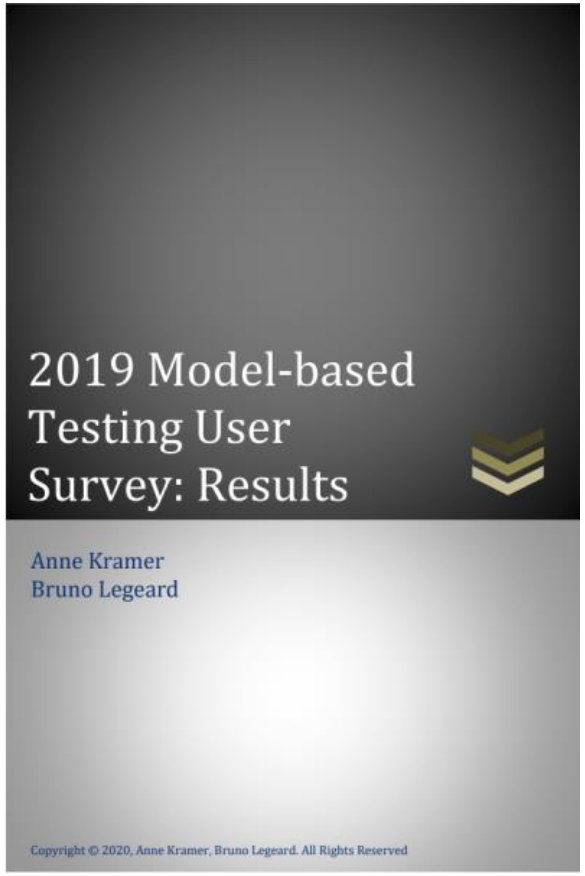
- Workflow-based model + decision tables
- Select important test cases from combinations

Tools (cont'd)

- **MoMuT::UML (academic)**
  - UML state machines, mutation testing
- **Harmony (harmony.ac)**
  - Gherkin-like syntax for partitions/constraints

List of tools:
http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html

MBT User Survey



Answer Options	2019
Acceptance testing	51,7%
System testing	79,3%
Integration testing	51,7%
Component (or unit) testing	10,3%

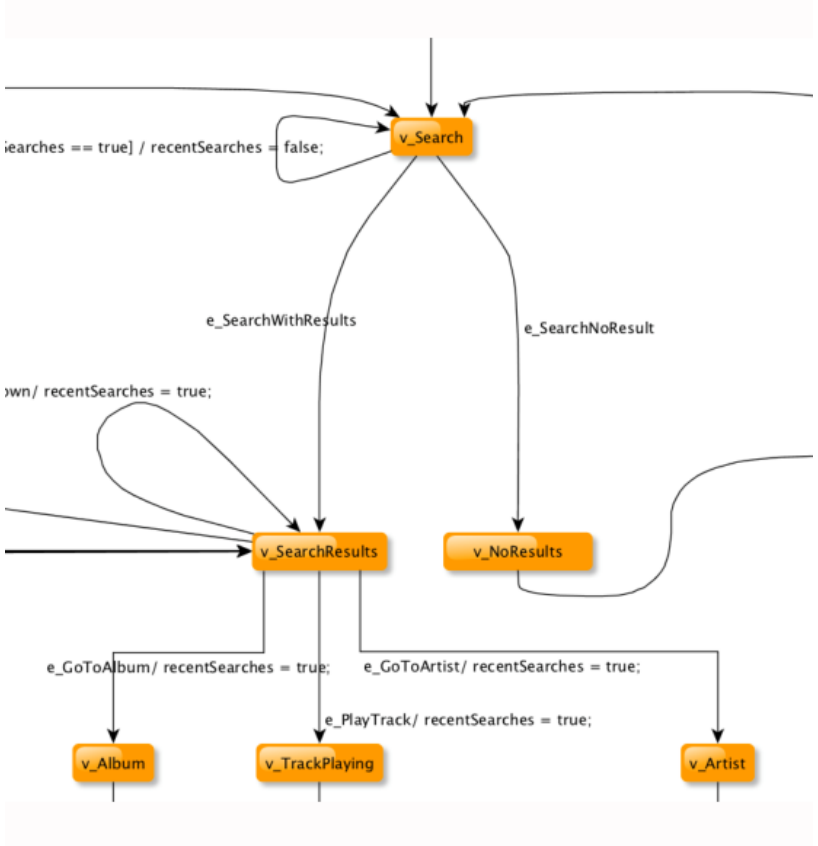
Answer Options	2019
Test cases (for manual test execution)	66,7%
Test scripts (for automated test execution)	70,8%
Test data	12,5%
Other artifacts (documentation, test suites,...)	20,8%

- “approx. 80h needed to become proficient”
- MBT is effective
- Lots of other details!

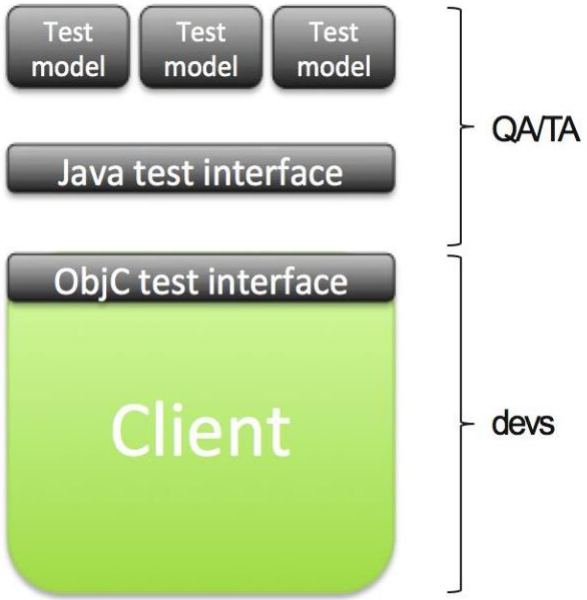
Source: https://www.cftl.fr/wp-content/uploads/2020/02/2019-MBT-User-Survey-Results.pdf

Case study: Spotify

Model + GraphWalker



MBT + test automation

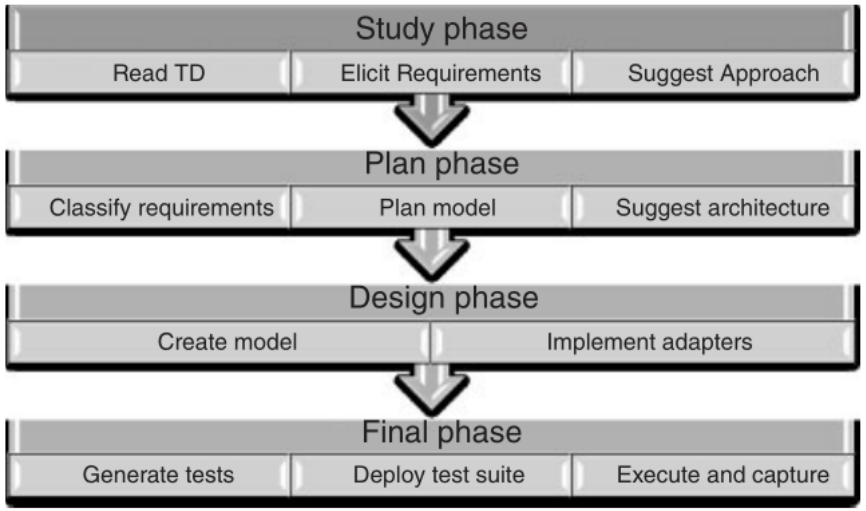


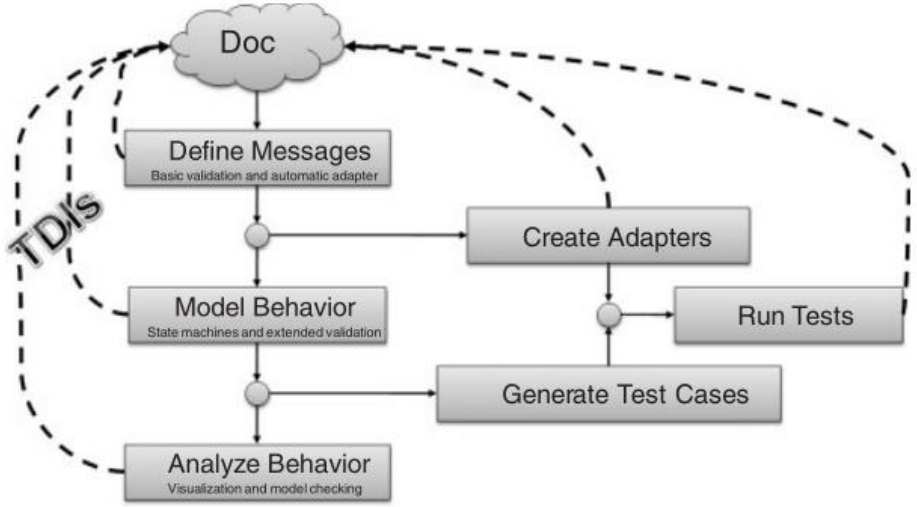
Spotify

Test automation and Model-Based Testing in agile dev cycle @ Spotify, UCAAT 2013

Case study: MS protocol documentation

- 250+ protocol, 25.000+ pages documentation
- 250+ man year, 350+ engineer
- Tool: SpecExplorer



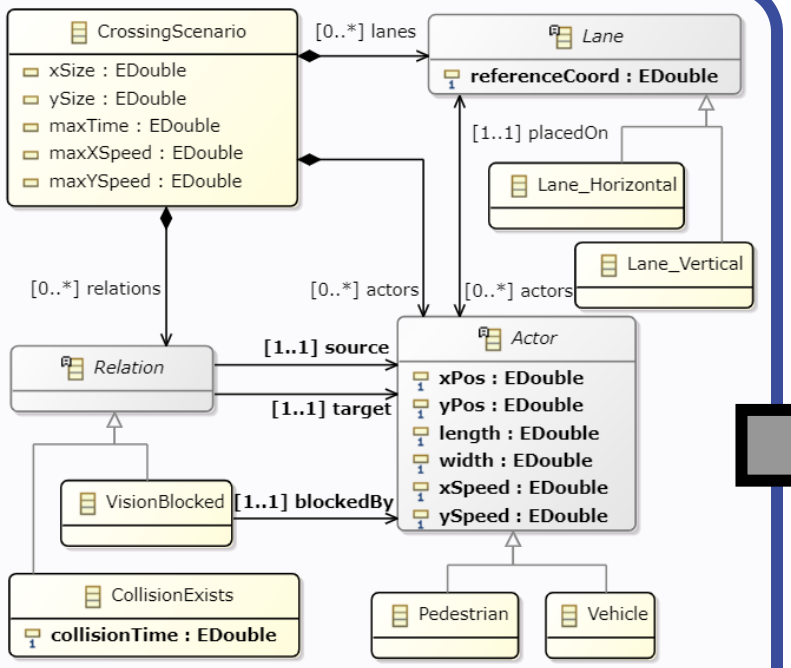


Details: http://queue.acm.org/detail.cfm?id=1996412

Source: W. Grieskamp et al. „Model-based quality assurance of protocol documentation: tools and methodology," STVR, 21:55-71, 2011

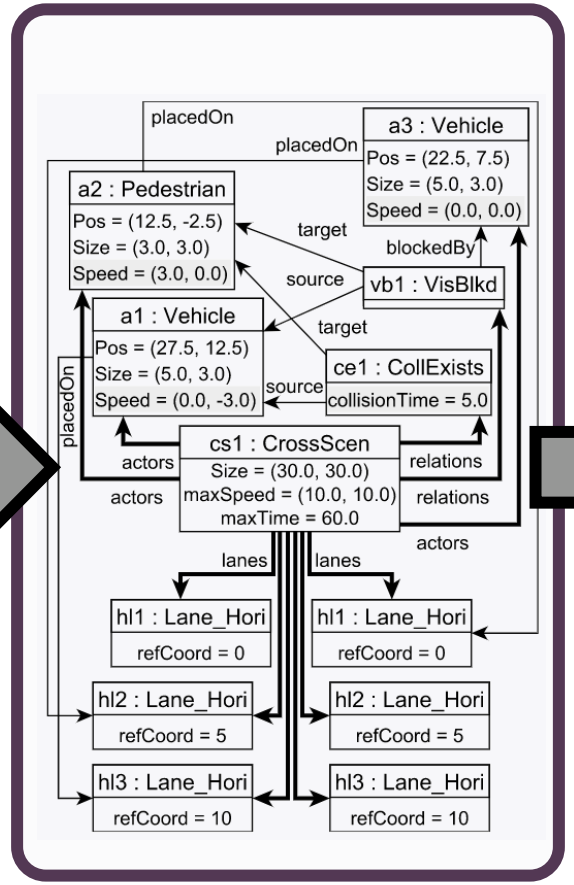
Case study: AV testing

Modeling Dangerous Situations

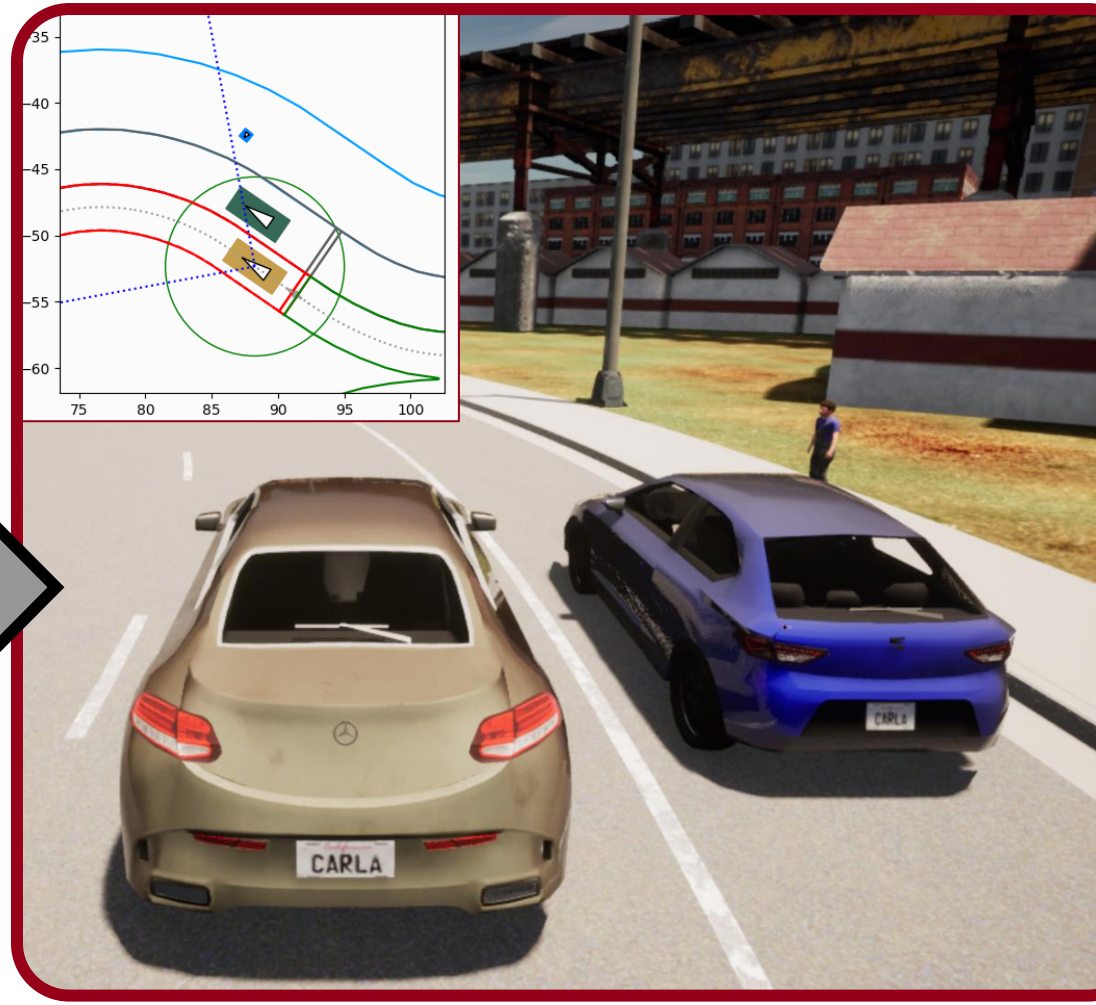



```
pattern minimumDistance(a1: Actor, a2: Actor) {
  Actor.xPos(a1, x1); Actor.yPos(a1, y1);
  Actor.xPos(a2, x2); Actor.yPos(a2, y2);
  a1 != a2;
  check((x1-x2)^2 + (y1-y2)^2 < 5^2);
```

Instantiating situations



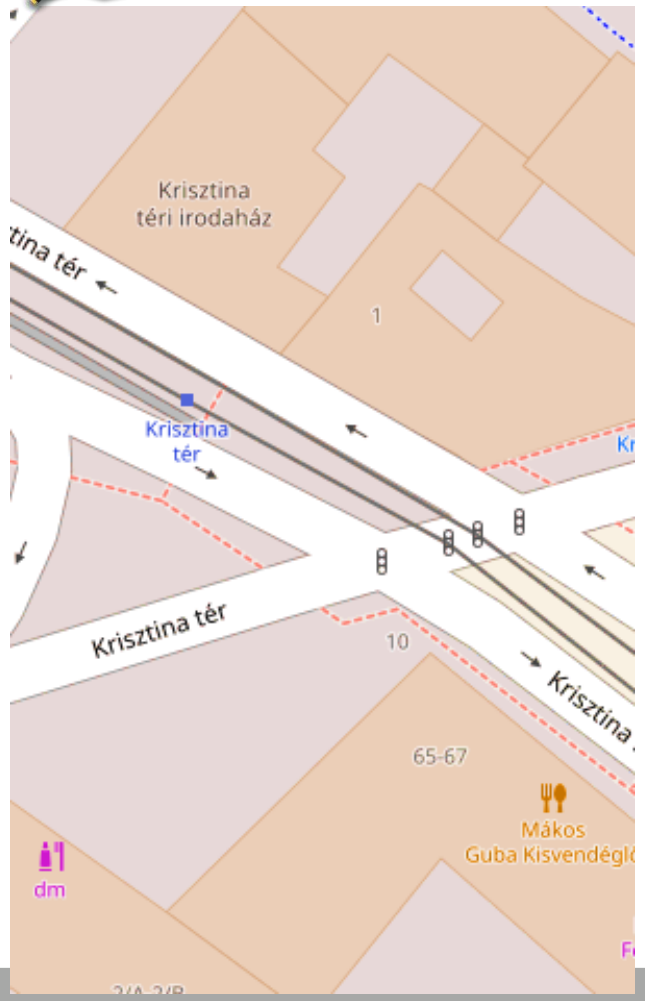
Running the scenarios



Real-world location

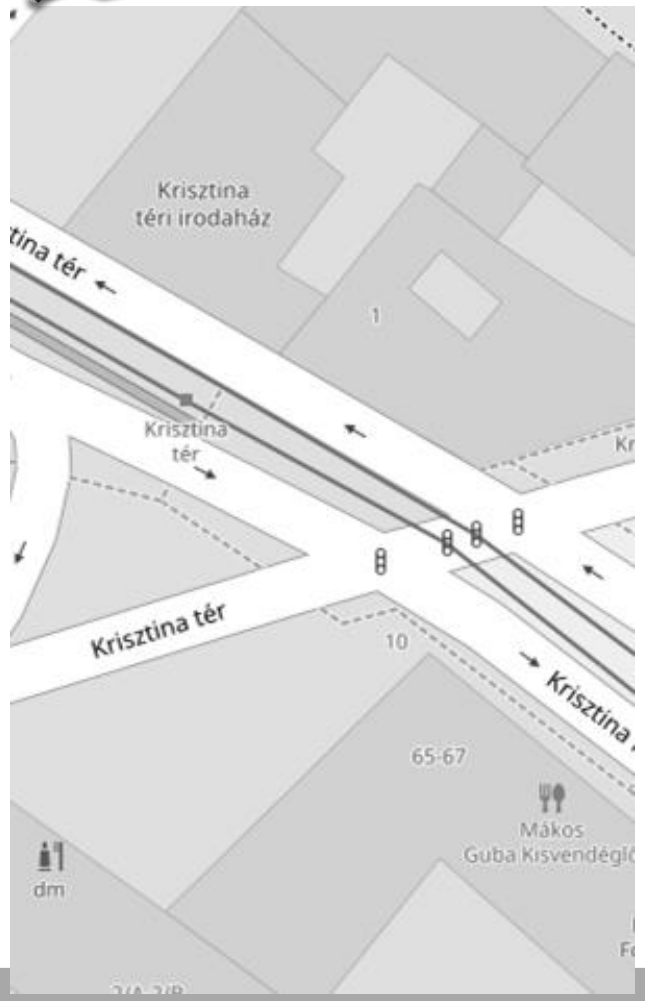
Real-world location

Map Import
OpenStreetMap

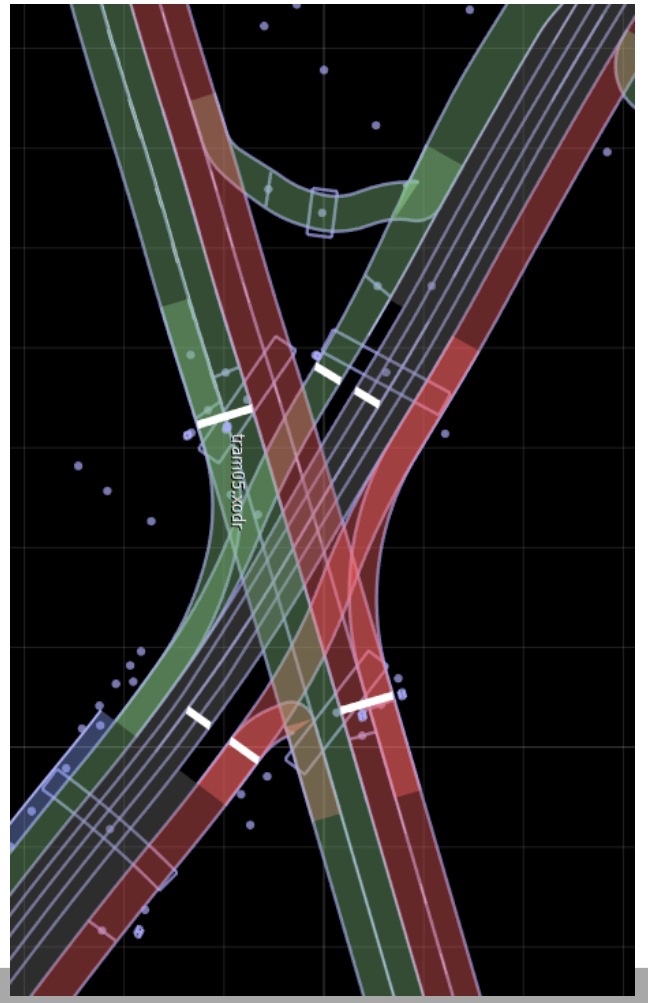


Real-world location

Map Import
OpenStreetMap

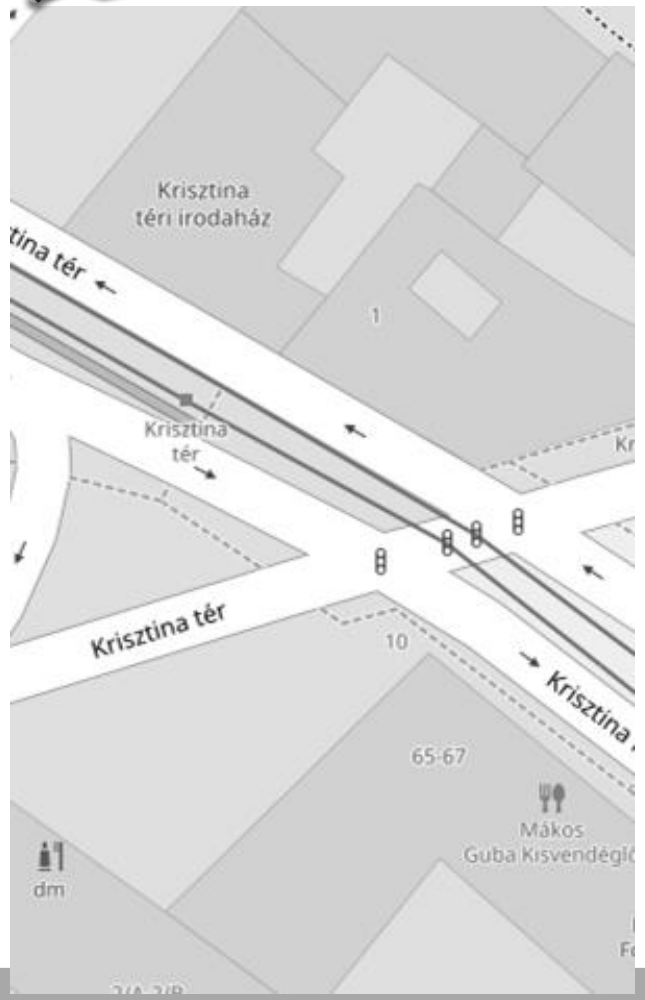


Road topology import
OpenDrive

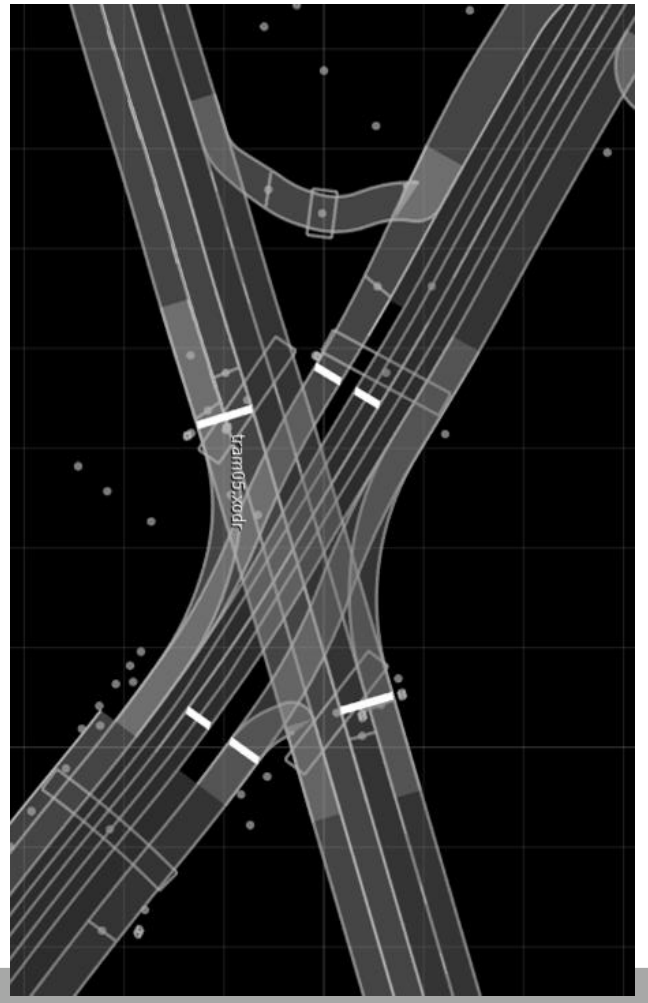


Real-world location

Map Import
OpenStreetMap



Road topology import
OpenDRIVE



Adding buildings, signs
Google Maps, Roadrunner

VECTORZERO

Real-world location

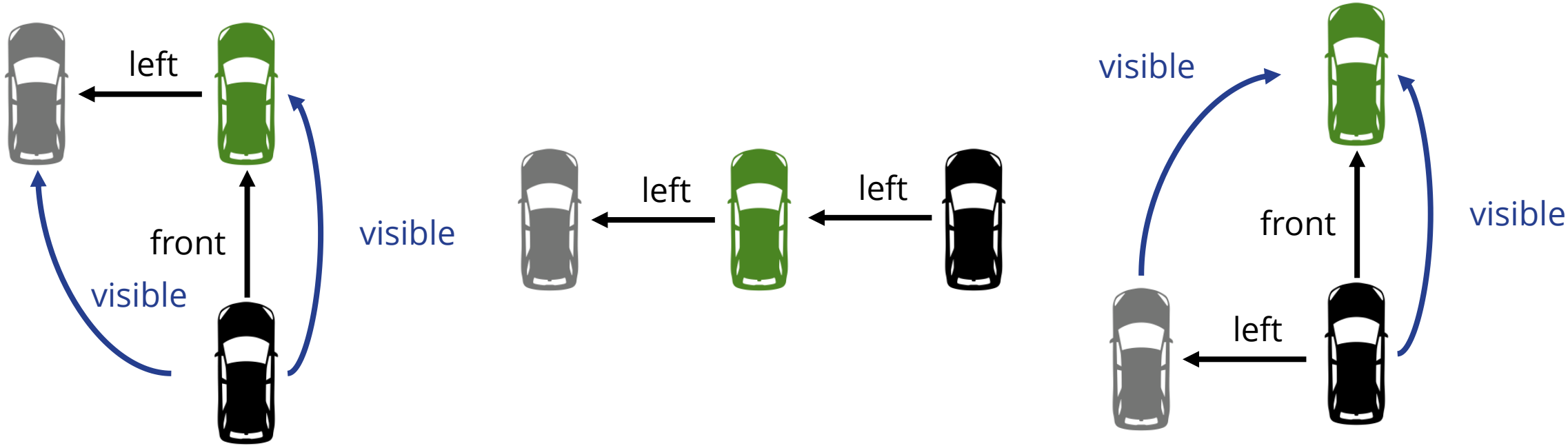
Map Import
OpenStreetMap

Road topology import
OpenDRIVE

Adding buildings, signs
Google Maps, Roadrunner

Import existing test track

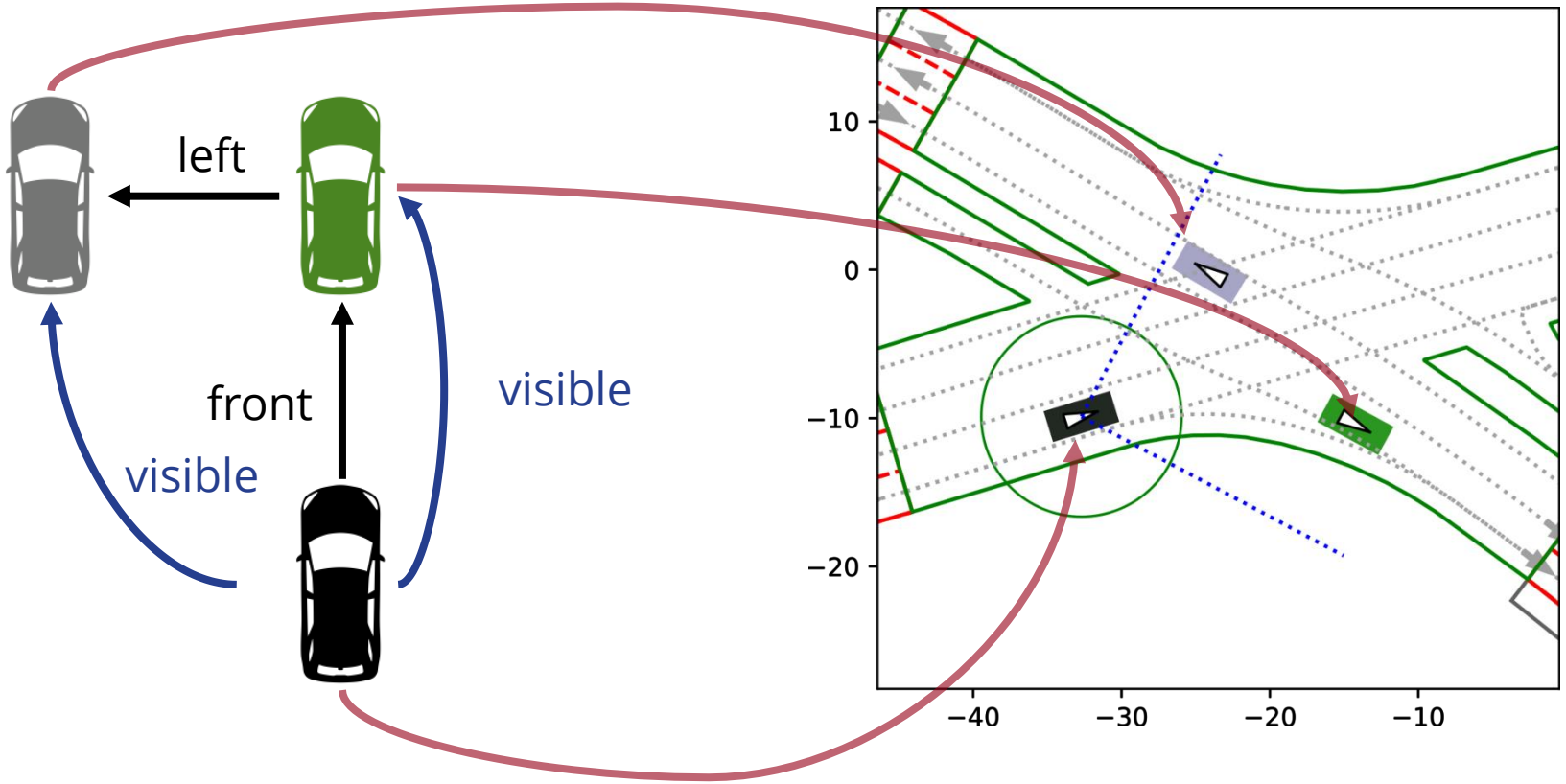
- **Goal:** synthetize different valid traffic situations as test inputs



Structurally different situations ⇒ Semantically different scenes

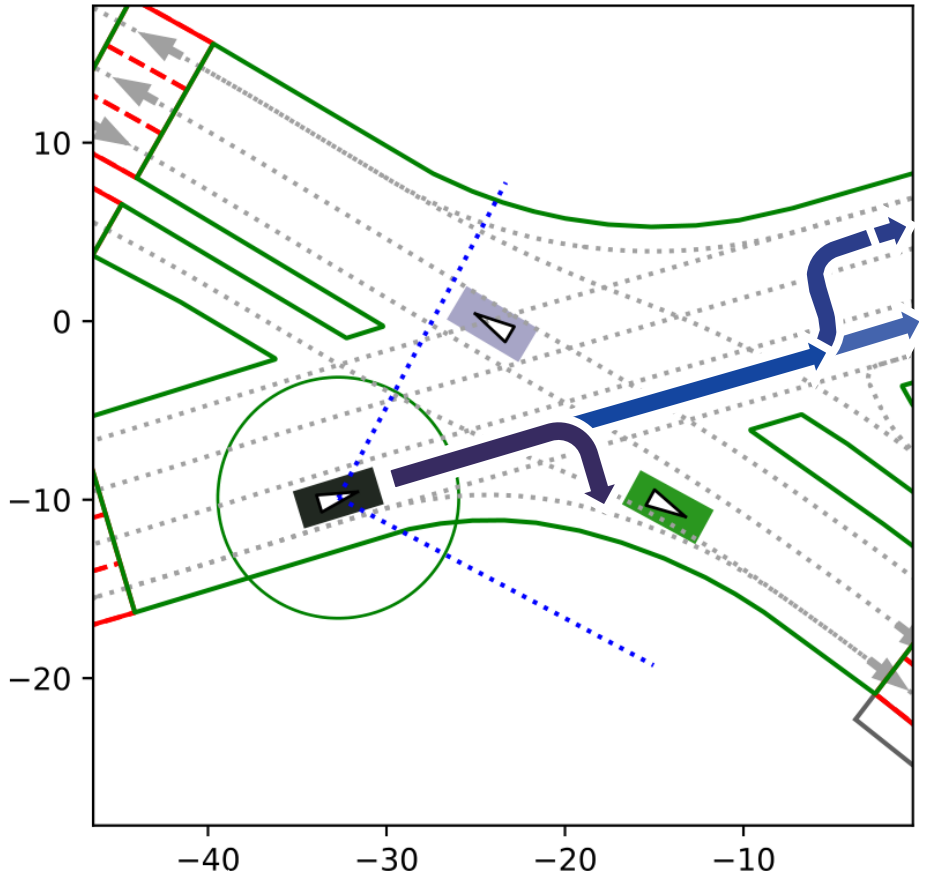
- **Solution:** graph generation with VIATRA Solver / Refinery

- **Goal:** allocate the situation on the map



- **Solution:** Scenic probabilistic scenario specification language

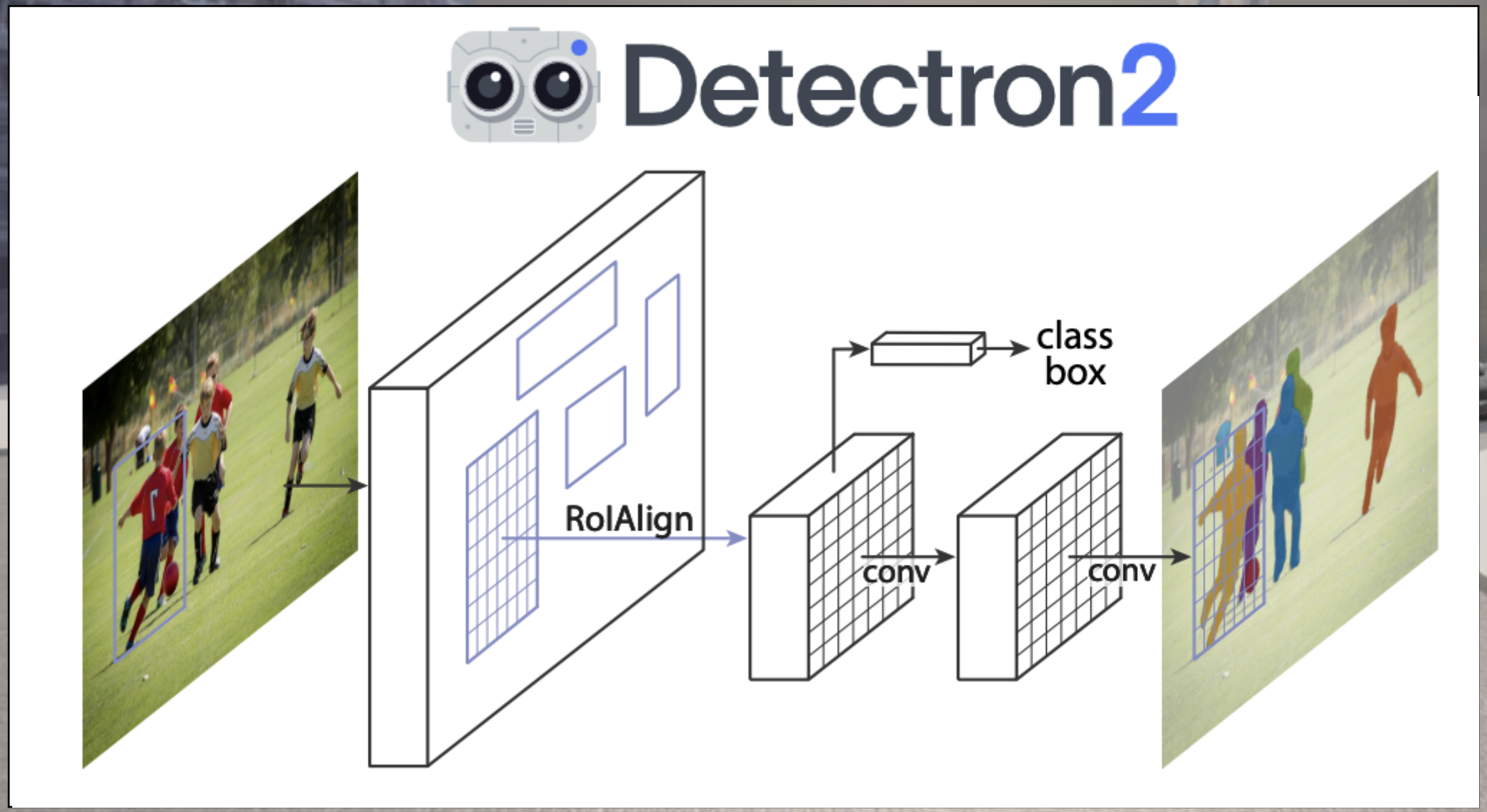
- **Goal:** assign tasks to the actors to get dynamic scenarios



- Placing random scenery to perturb the image

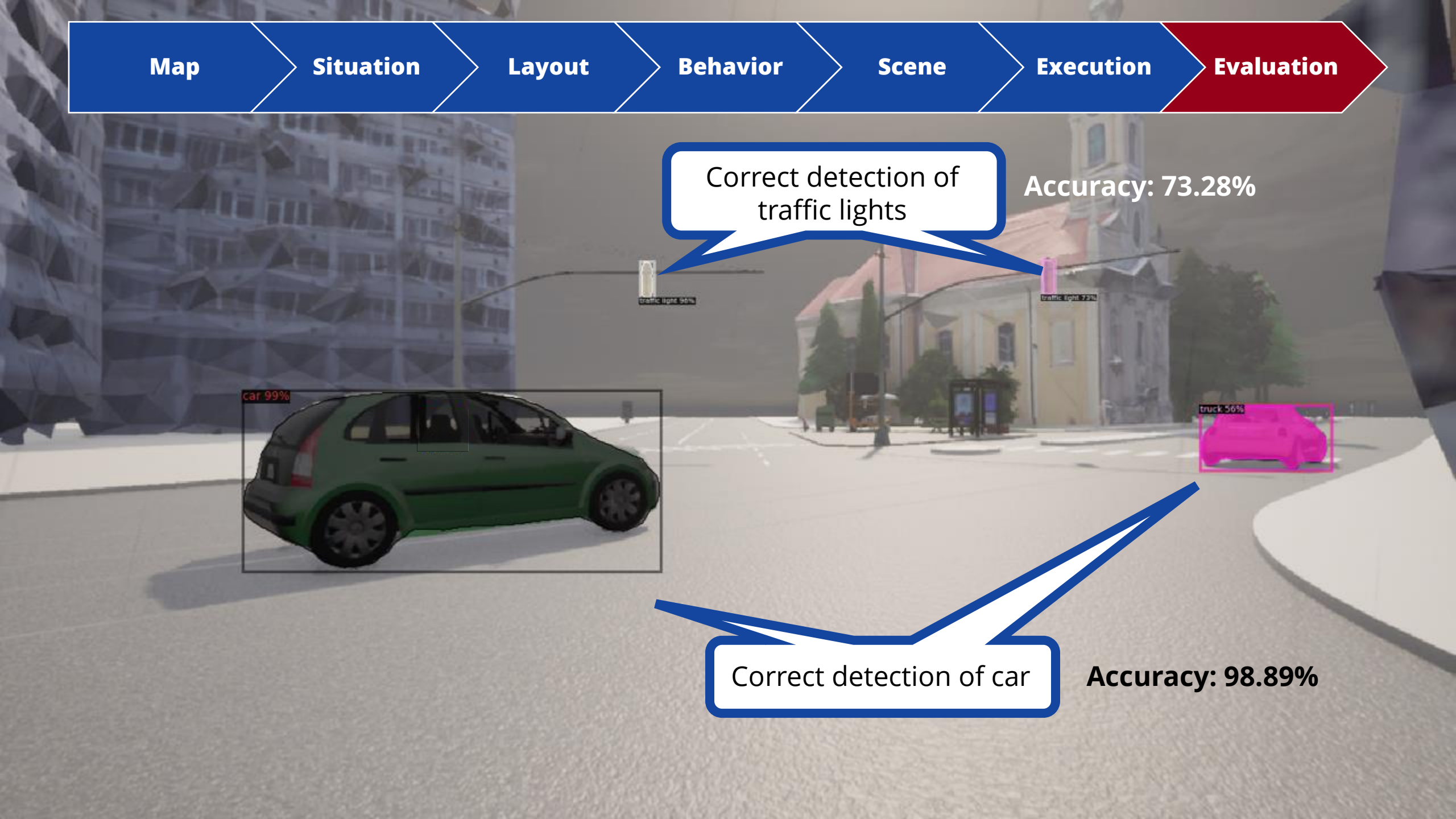
- Generate different weather conditions


CARLA



He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. 2017.

Map | Situation | Layout | Behavior | Scene | Execution | Evaluation



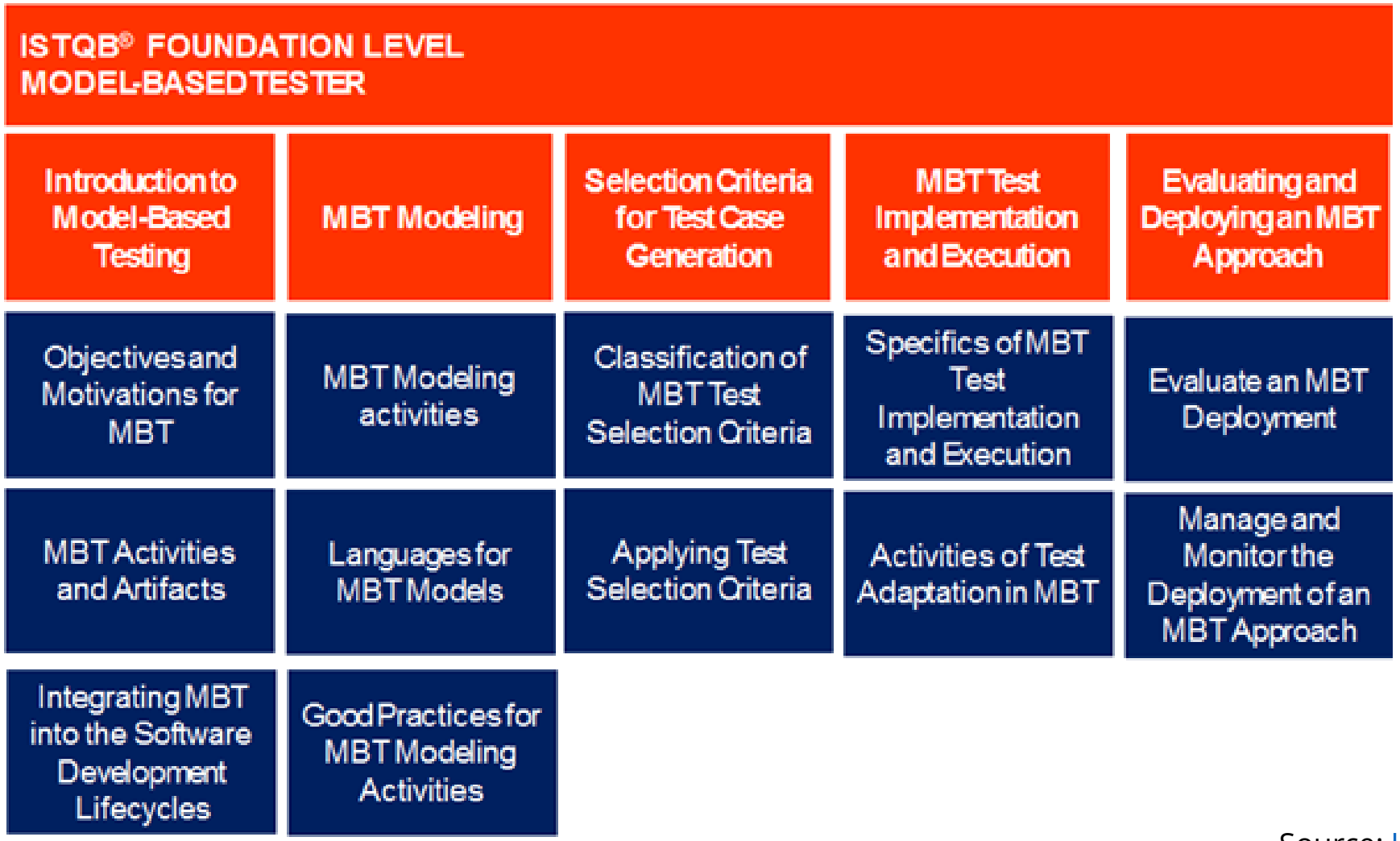
“Cheat sheet” for introducing MBT

From Robert V. Binder (http://robertvbinder.com/)

Recommended	Not recommended
Complex SUT behavior	Simple functionality
Abstractable requirements	Subjective evaluation
Testable interfaces	Monolithic GUI
Must to regression testing	Low-value, deprecated GUI
Sophisticated test engineers	Little or no established testing
	Non-technical QA team

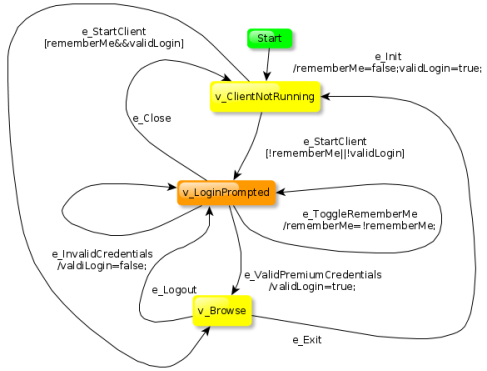
See also: „Model-Based Testing: Why, What, How,” http://www.slideshare.net/robertvbinder/model-basedtestingignite

ISTQB CTFL-MBT training + exam

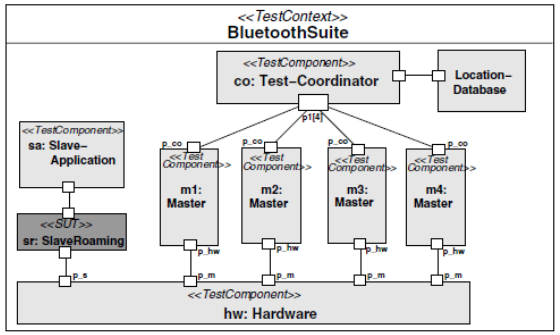


Source: ISTQB

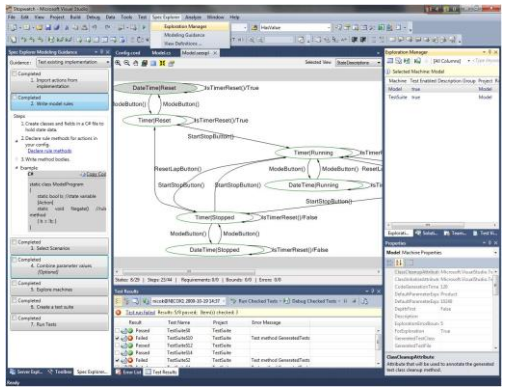
Summary



Many models,
test goals and tools



MBT = using models in testing



Scaling from
brainstorming to
fully automatic
test case generation