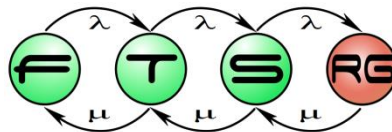


Model-based test generation

Zoltan Micskei, Istvan Majzik

**Budapest University of Technology and Economics
Fault Tolerant Systems Research Group**



Main topics of the course

- **Overview (1)**
 - V&V techniques, Critical systems
- **Static techniques (2)**
 - Verifying specifications
 - Verifying source code
- **Dynamic techniques: Testing (7)**
 - Developer testing, Test design techniques
 - Testing process and levels, **Test generation**, Automation
- **System-level verification (3)**
 - Verifying architecture, Dependability analysis
 - Runtime verification

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)

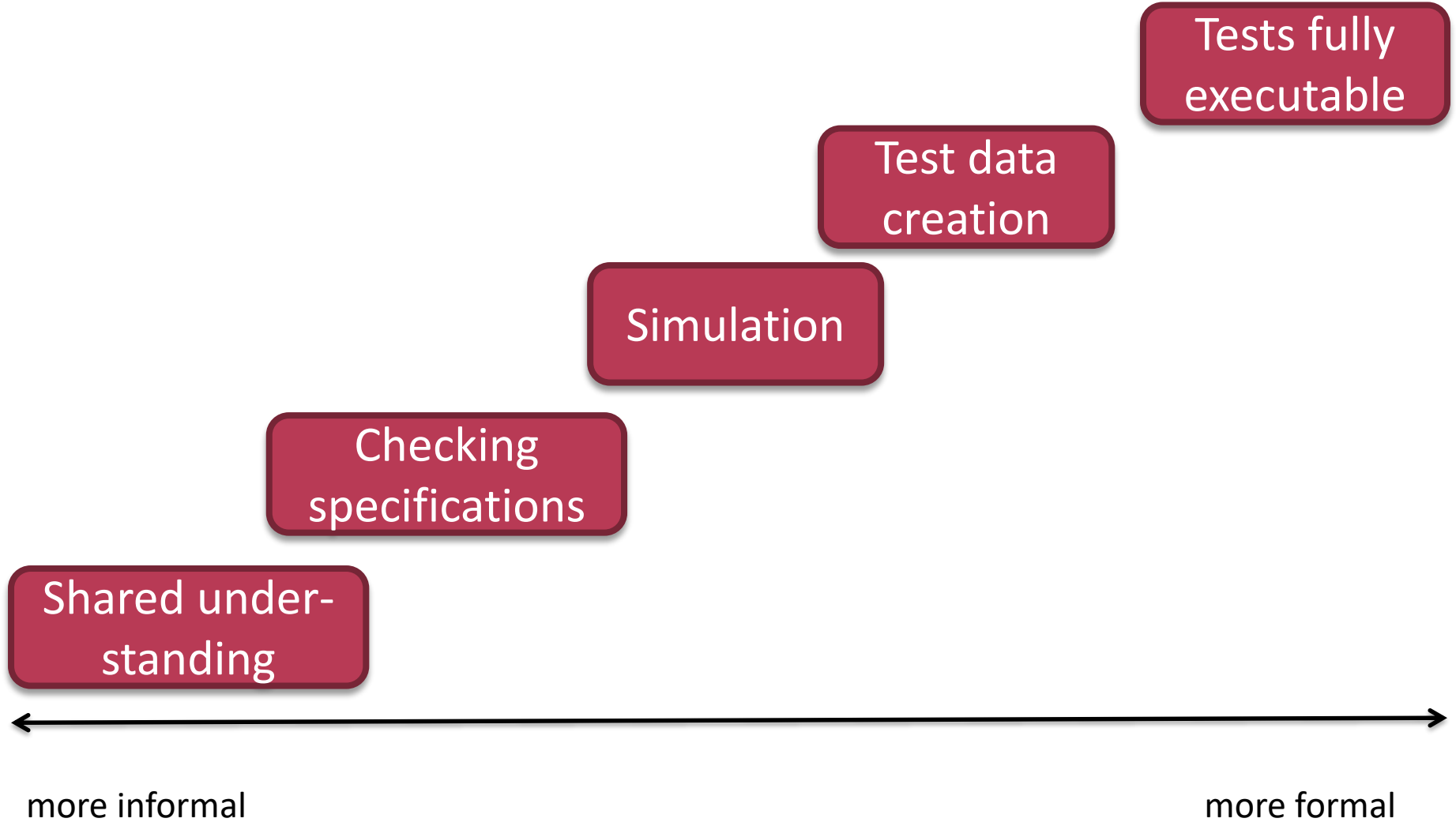
What is model-based testing?

“Testing based on or involving models” [ISTQB]

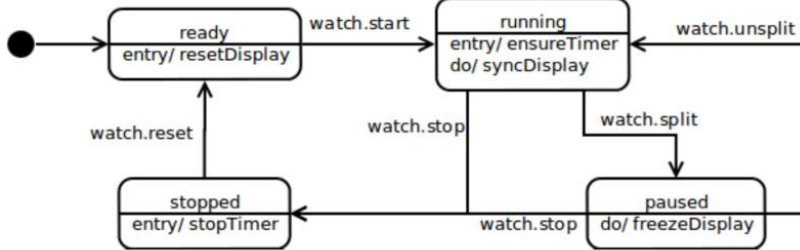
- Not just test generation
- Not just automatic execution
- Not just for model-driven engineering

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

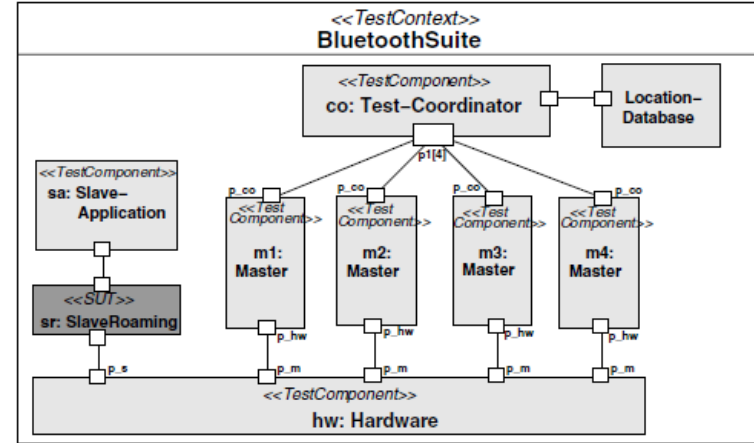
Landscape of MBT goals



Using models in testing (examples)



Behavior of SUT

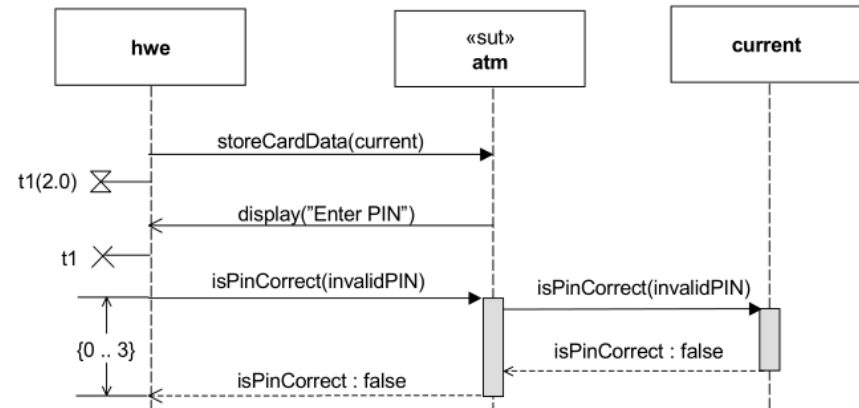


Test configuration

```

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

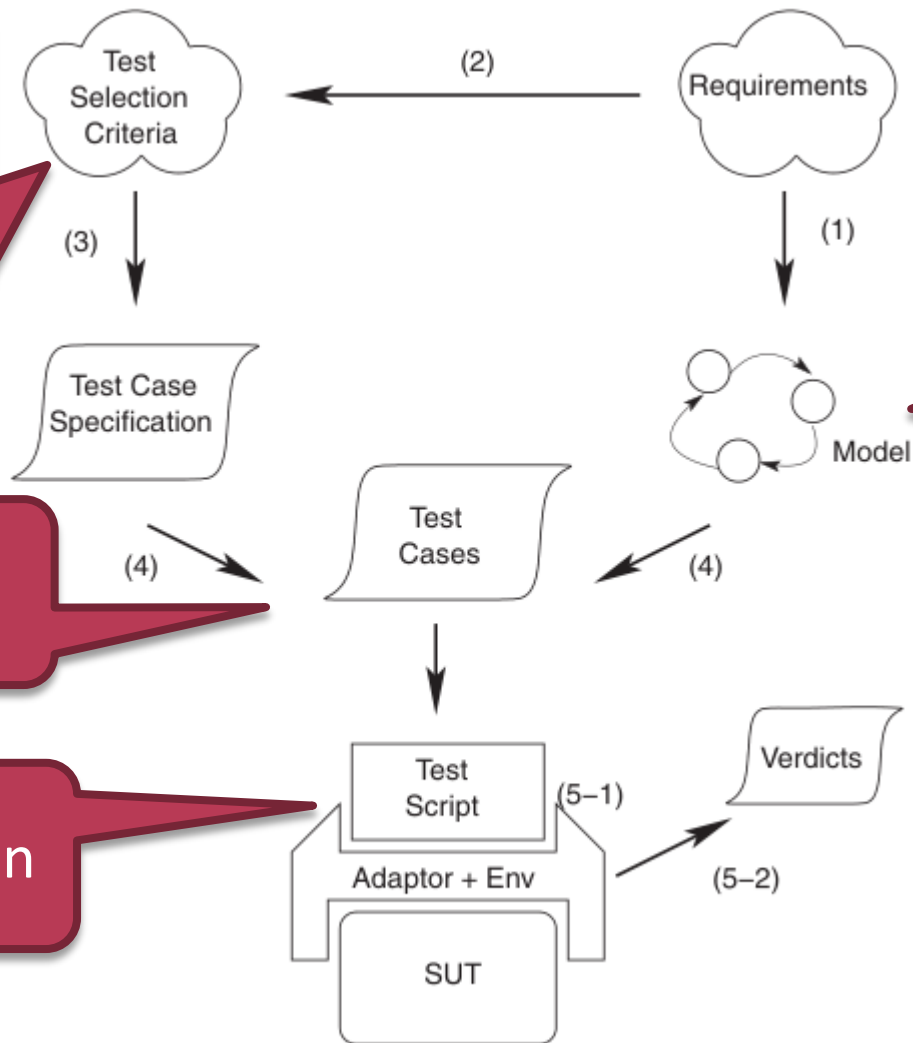
Typical MBT process

State, path, requirement coverage...

Test model

Abstract test case

Concretization

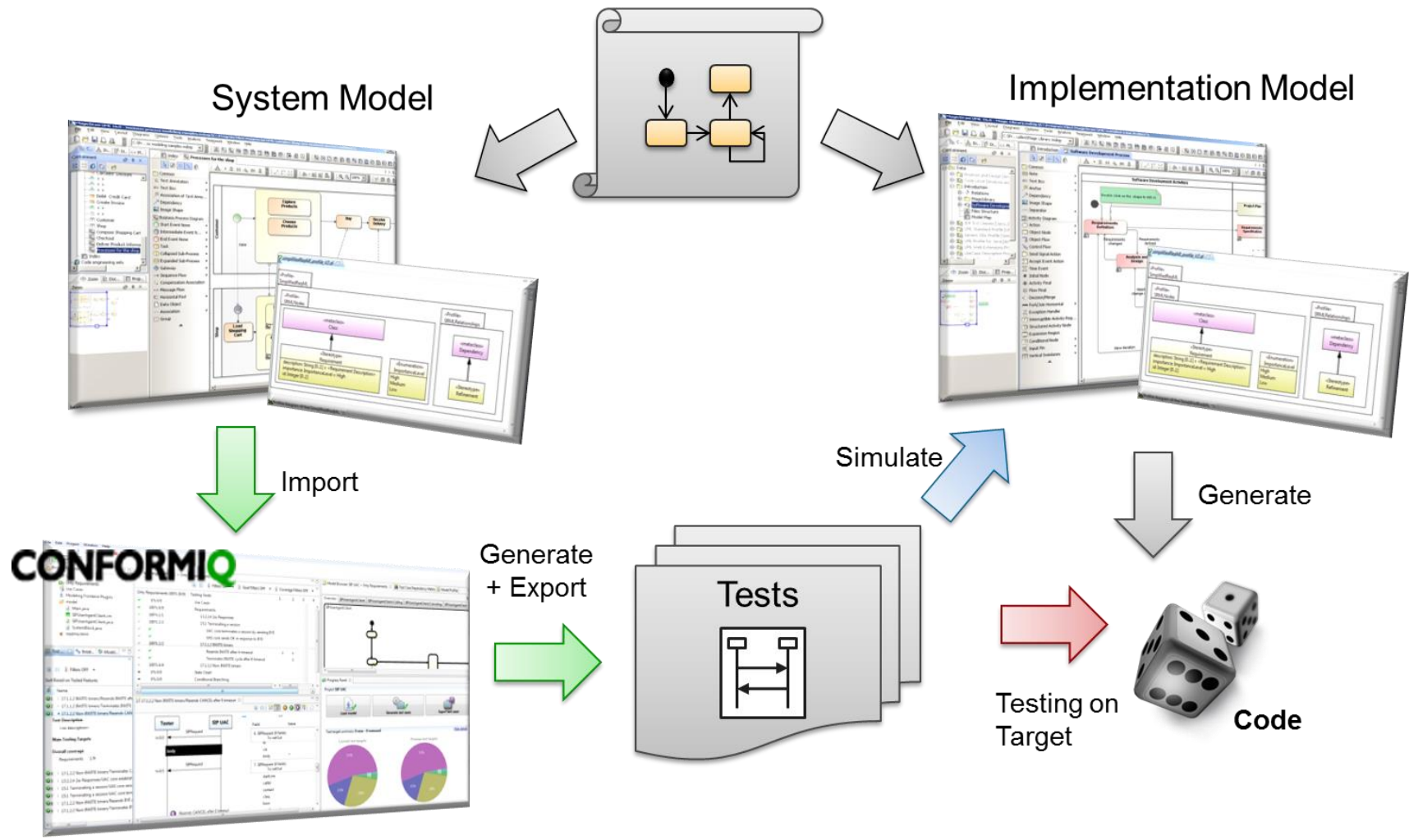


Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

- Create test model using FSMs
- Use GraphWalker to generate test sequences
- Write adaptation to connect to Java code

Example: Model driven workflow

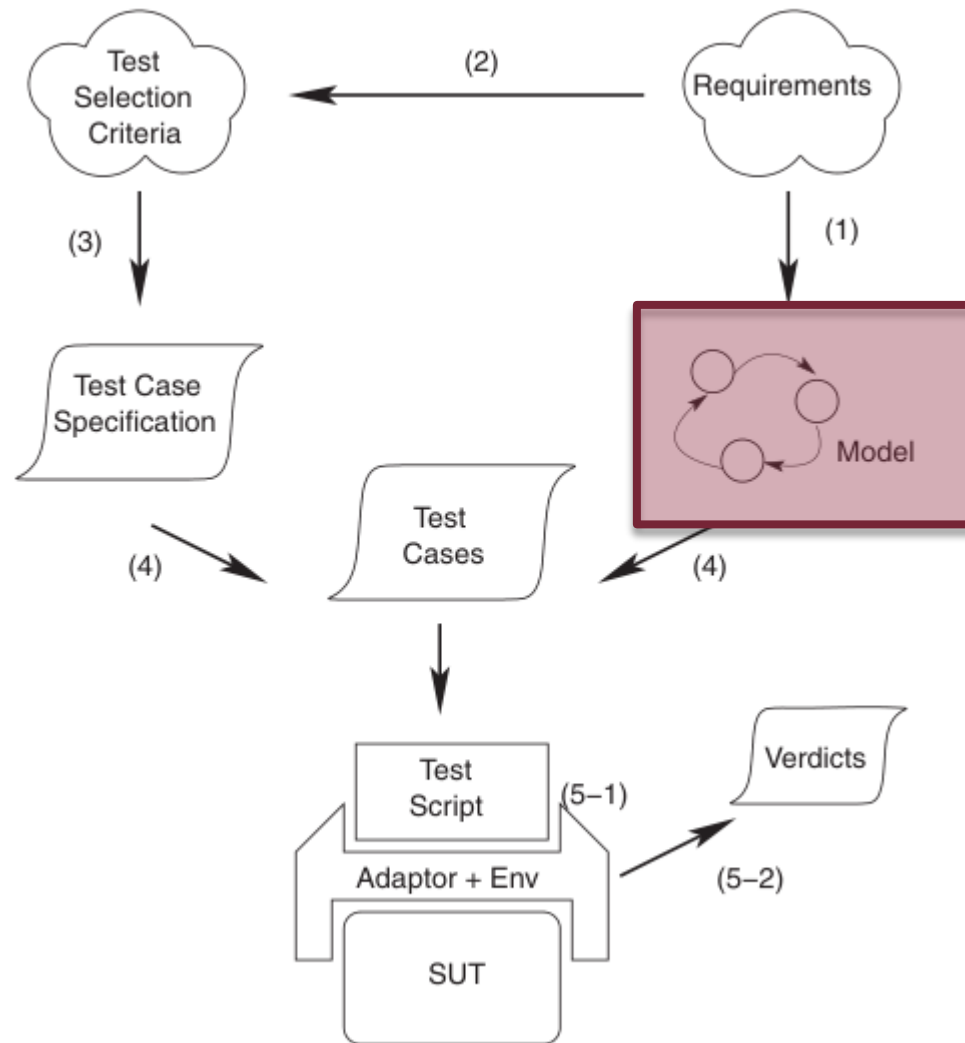
Functional Specification / Design Model



Source: Kimmo Nupponen. "[Model driven workflow](#)", 2016.

MBT PROCESS

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

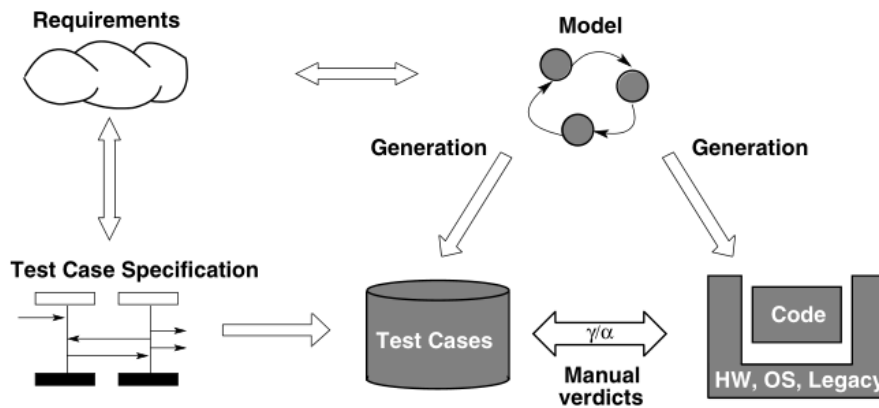
- Model environment/users
- Inputs to the system

Test

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

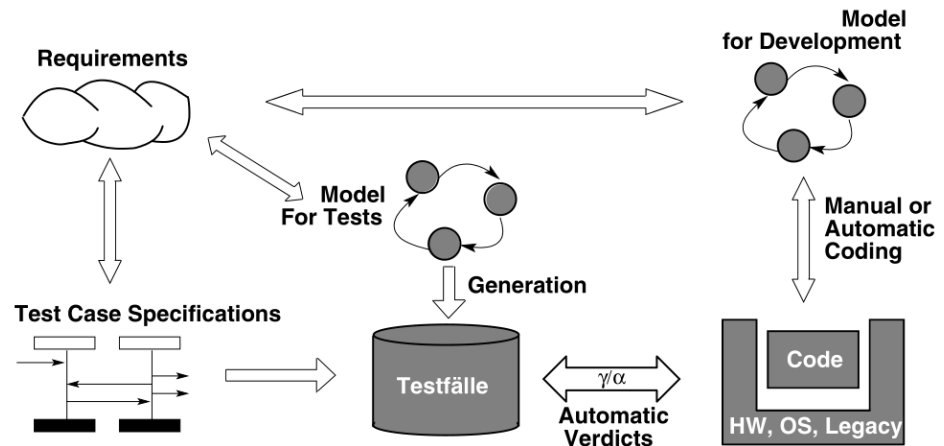
Reuse: Development and Test modeling

What if I have existing design models?



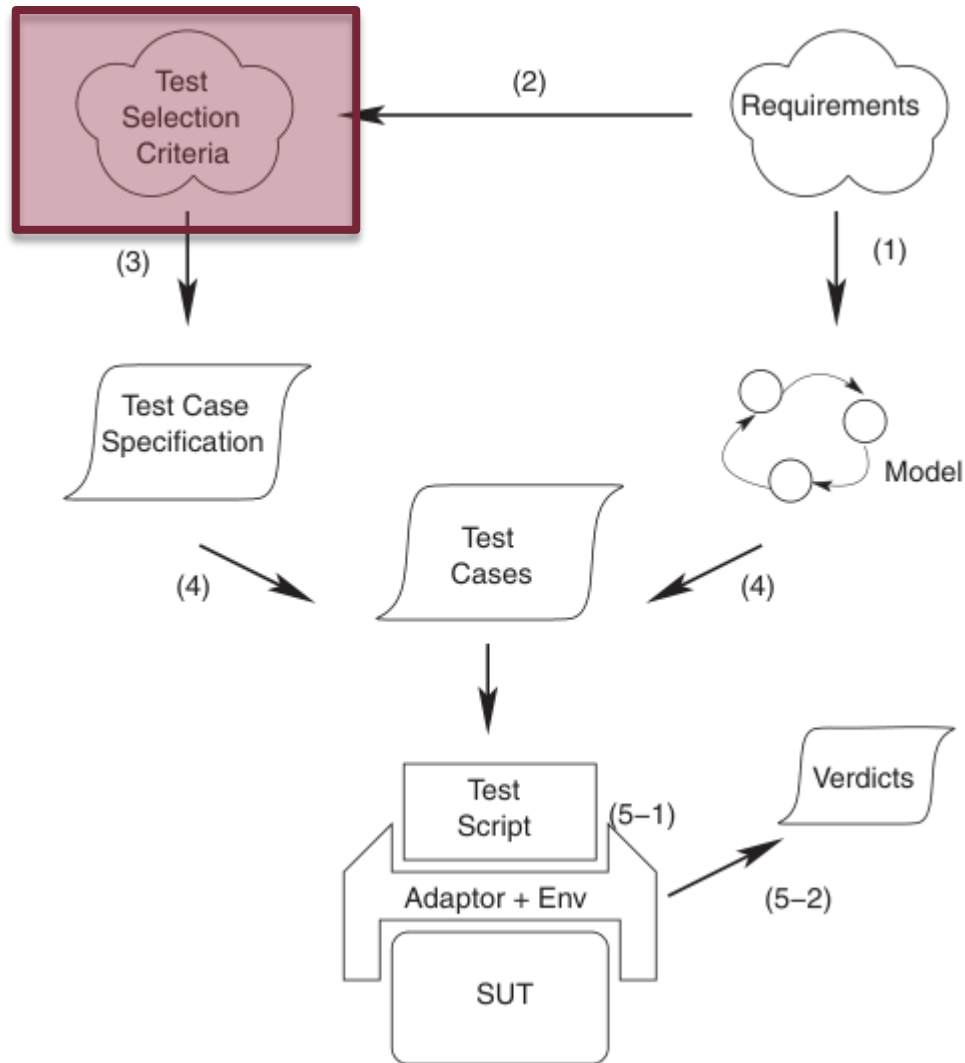
Problem: what do we test here?

Approach: separate dev. and test models



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

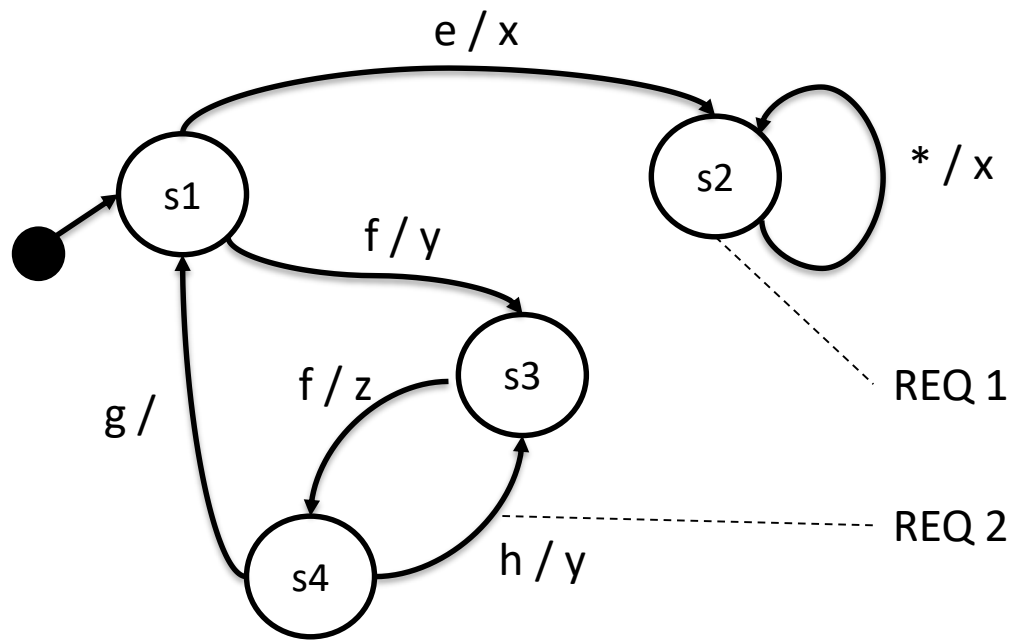
Typical MBT process



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

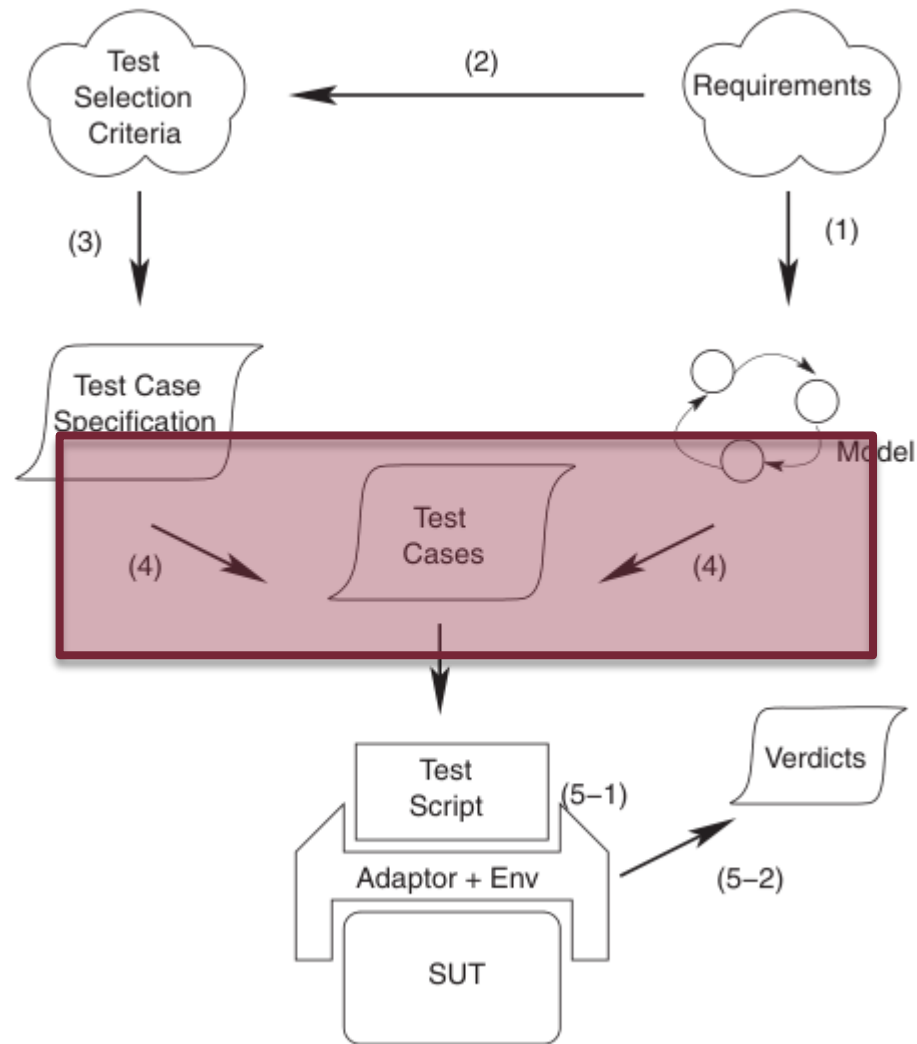
Typical test selection criteria

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



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

Typical MBT process

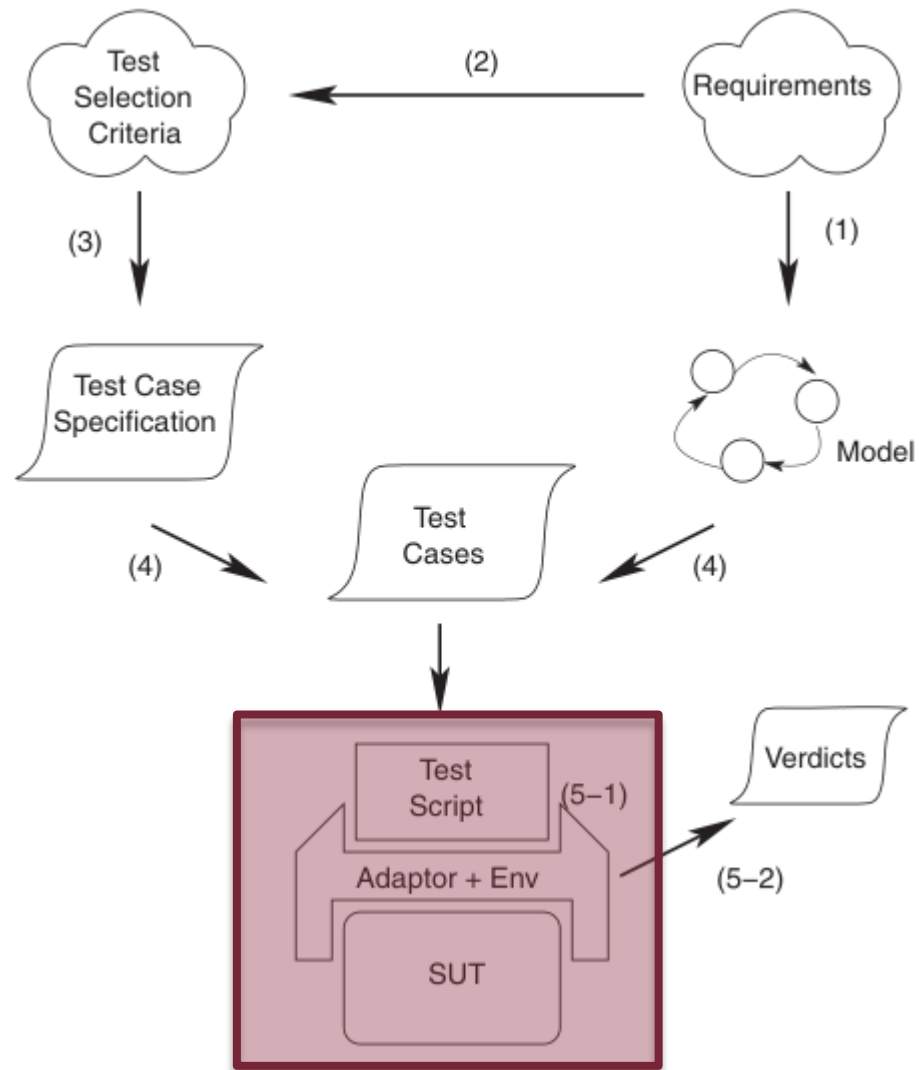


Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

Test generation methods (sample)

- **Direct graph algorithms**
 - Transition coverage → “New York Street Sweeper problem”
- **FSM testing**
 - Homing and synchronizing sequences, state identification and verification, conformance...
- **LTS testing**
 - Equivalence and preorder relations, ioco
- **Using model checkers**
- **Fault-based (mutation)**

Typical MBT process

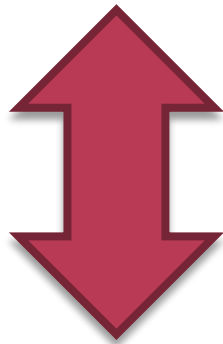


Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

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

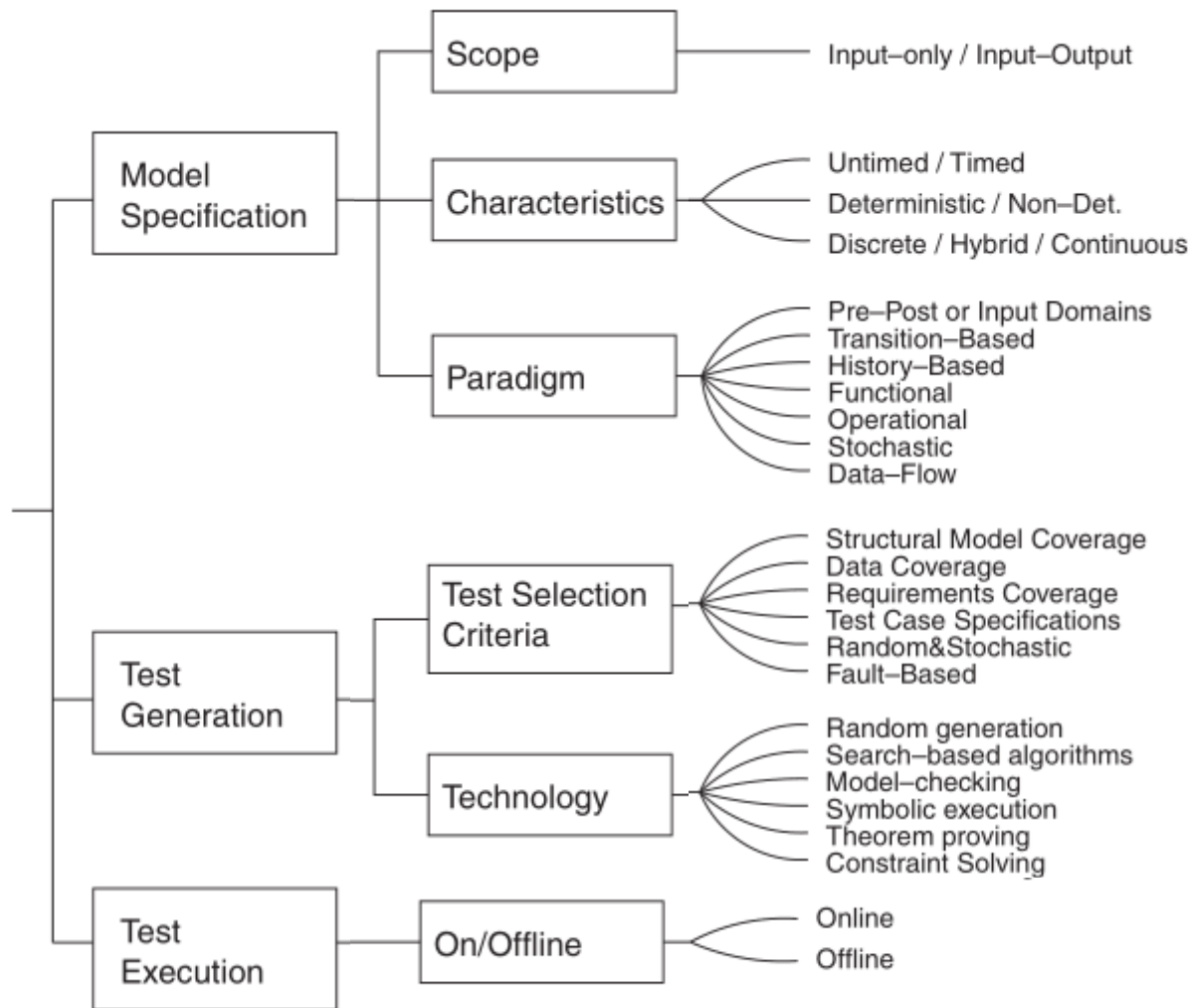
- Concrete input data
- Detailed test procedure (manual or automatic)

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



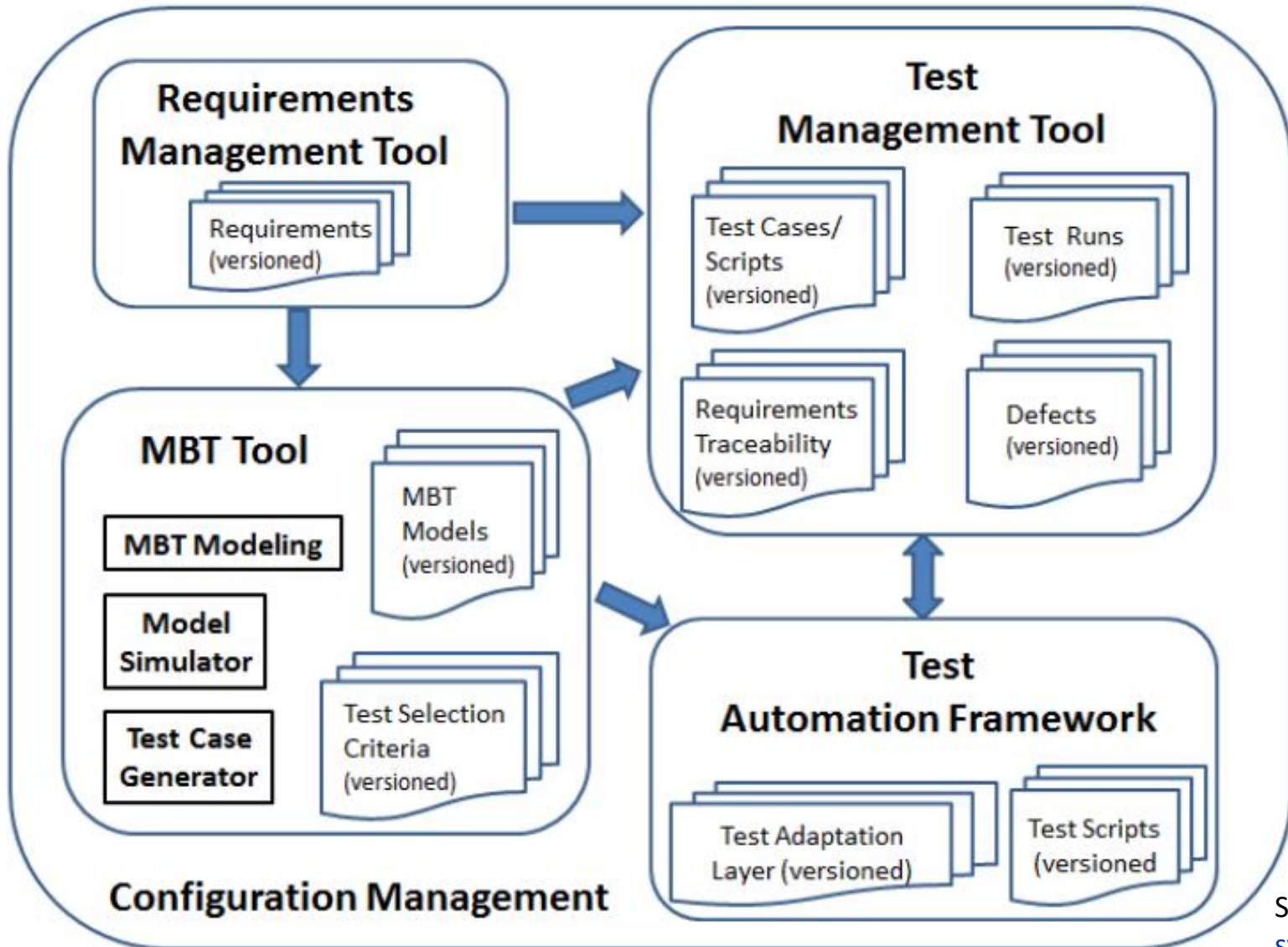
Source: M. Utting, A. Pretschner, B. Legeard. „A taxonomy of model-based testing approaches”, STVR 2012; 22:297–312

TOOLS AND CASE STUDIES

Typical use cases

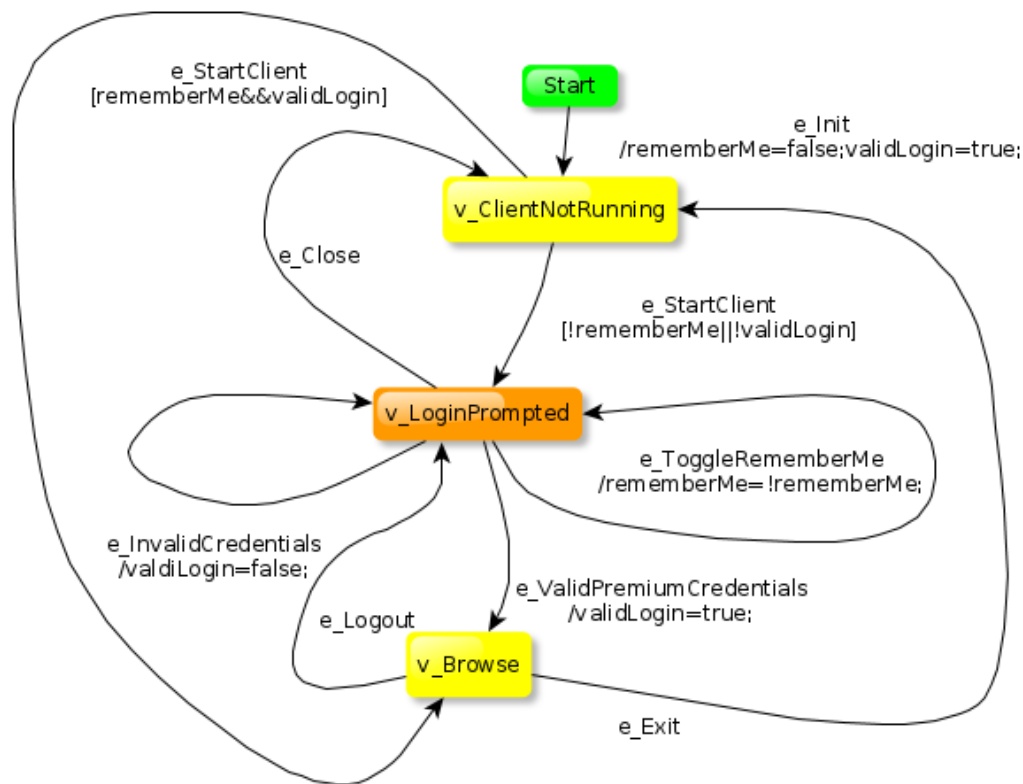
- **Fast & easy**
 - Simple modeling
 - Using open tools
- **Full fledged**
 - Complex, commercial tool
 - Full lifecycle support
- **Advanced**
 - Custom modeling languages/tools

MBT tool chain



Source: [ISTQB syllabus](#)

Open source tool: GraphWalker



Source: [GraphWalker](#)

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

Industrial MBT tool – Conformiq

The screenshot displays the Conformiq Designer IDE interface. It includes several panes:
1. Project Explorer on the left showing the project structure.
2. Coverage Editor in the top center showing a list of testing goals with their coverage percentages.
3. Model Browser on the top right showing a block diagram of the system components.
4. Test Cases pane on the bottom left showing a list of test cases with their names and IDs.
5. Sequence Diagram in the bottom center showing interactions between components like appIn, clientBTOut, clientBTIn, appOut, and SDP-client.
6. State Machine in the bottom right showing the state transitions for the SDP-client component.
7. Console on the far right showing the execution output of the test cases.

Conformiq Designer IDE for automatic test case generation

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

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

Industrial MBT tool – SpecExplorer

The screenshot shows the SpecExplorer interface in Visual Studio. The main window displays a state machine model with states like `DateTime|Reset`, `Timer|Reset`, `Timer|Running`, `Timer|Stopped`, `DateTime|Running`, and `DateTime|Stopped`. Transitions are labeled with actions like `StartStopButton()` and `ResetLapButton()`. The left sidebar shows a 'Spec Explorer Modeling Guidance' pane with steps for creating classes, writing rules, and running tests. The bottom pane shows 'Test Results' with a table of test outcomes.

Result	Test Name	Project	Error Message
Passed	TestSuiteS0	TestSuite	
Failed	TestSuiteS10	TestSuite	Test method GeneratedTests
Passed	TestSuiteS12	TestSuite	
Passed	TestSuiteS14	TestSuite	
Failed	TestSuiteS2	TestSuite	Test method GeneratedTests

The screenshot shows two code files in Visual Studio. The top file is `Config.cord` with configuration settings for the SpecExplorer project. The bottom file is `Calculator.cs` showing the implementation of the calculator service, including namespace declarations and implementation details.

```
using SpecExplorerProject.JointImplementation; // Using this name space for all implementation actions
using Microsoft.Modeling; // Include for use of modeling types (sets in this case)
// Bundle Switch Option values in one config
config MainSwitches
{
    switch testClassbase = "vs";
    switch generatedtestpath = ".\\TestSuite";
    switch generatedtestnamespace = "SpecExplorerProject.TestSuite";
    switch StackDepth=2048;
    switch PathDepthBound=1024;
    switch StepBound=5000;
    switch StateBound=5000;
    switch ForExploration=false;
    switch TestEnabled=false;
}
```

Source: <https://visualstudiogallery.msdn.microsoft.com/271d0904-f178-4ce9-956b-d9bfa4902745>

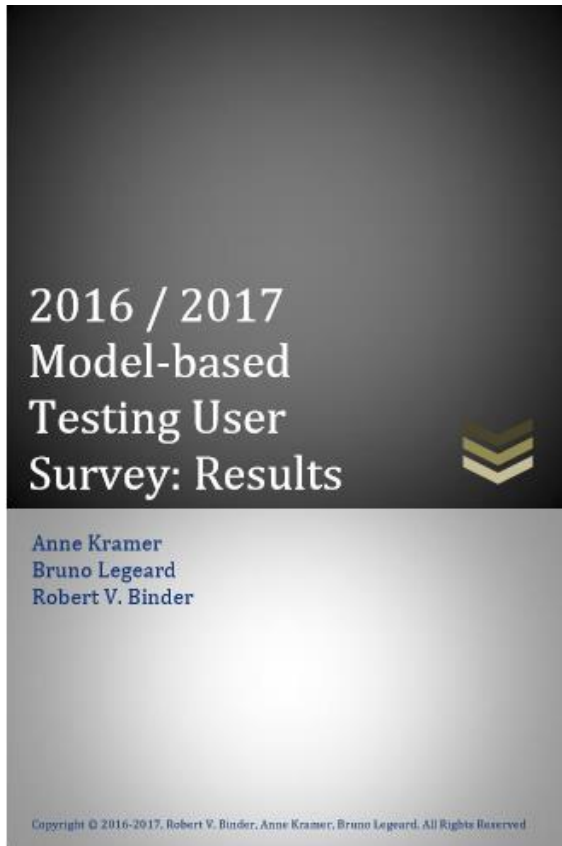
- C# model program + adapter code
- Slicing: scenarios, action patterns

Tools (cont'd)

- **Certiflyt (Smartesting)**
 - UML + OCL models
- **MoMuT::UML (academic)**
 - UML state machines, mutation testing
- **4Test-Plus (4test.io)**
 - Gherkin-like syntax for partitions/constraints

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

MBT User Survey



~100 participants
32 questions

Testing levels

System testing	77,4%
Integration testing	49,5%
Acceptance testing	40,9%
Component testing	31,2%

Generated artifacts

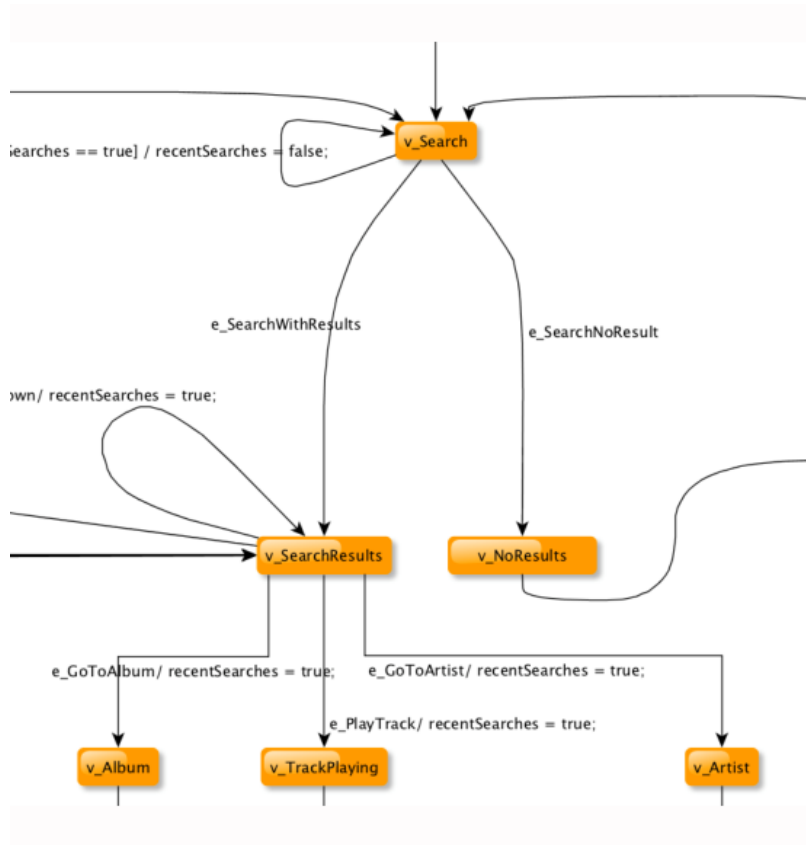
Automated test scripts	84,2%
Manual test cases	56,6%
Test data	39,5%
Others (docs, test suites...)	28,9%

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

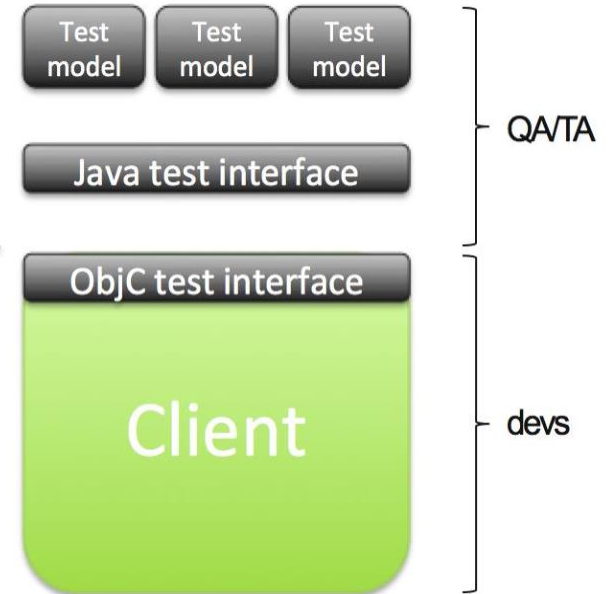
Source: <http://www.cftl.fr/wp-content/uploads/2017/02/2016-MBT-User-Survey-Results.pdf>

Case study: Spotify

Modell + GraphWalker



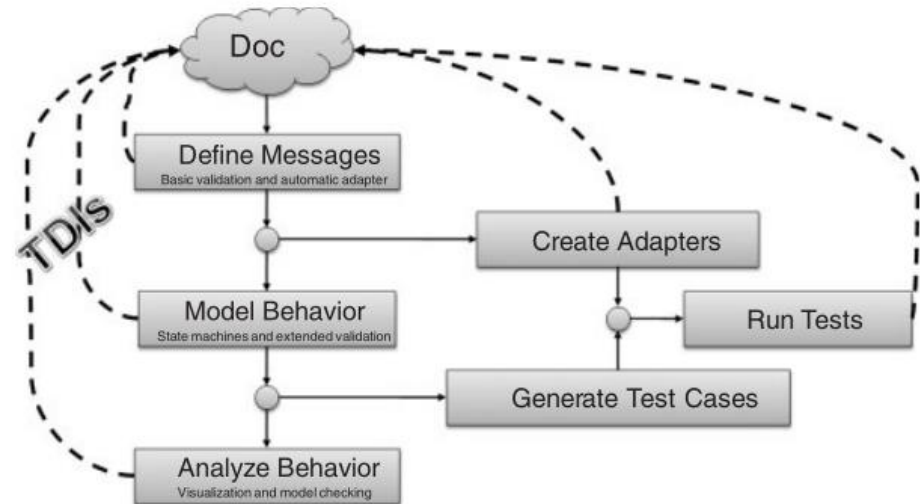
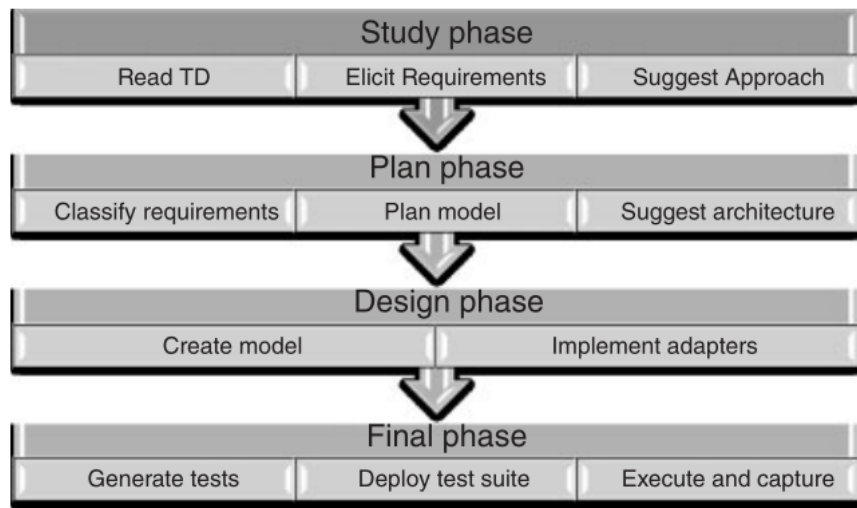
MBT + test automation



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

“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>

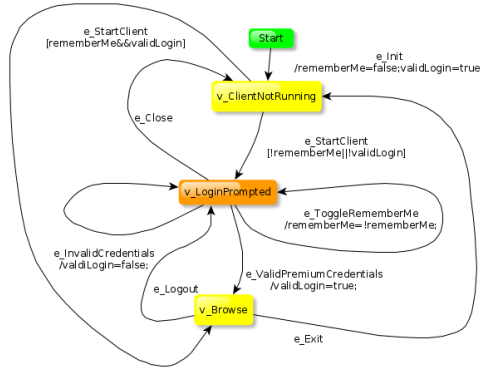
ISTQB CTFL-MBT training + exam

ISTQB® FOUNDATION LEVEL MODEL-BASED TESTER

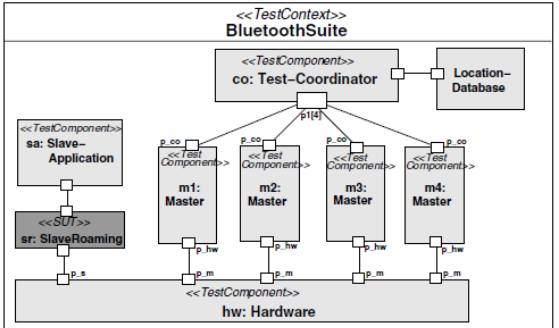
Introduction to Model-Based Testing	MBT Modeling	Selection Criteria for Test Case Generation	MBT Test Implementation and Execution	Evaluating and Deploying an MBT Approach
Objectives and Motivations for MBT	MBT Modeling activities	Classification of MBT Test Selection Criteria	Specifics of MBT Test Implementation and Execution	Evaluate an MBT Deployment
MBT Activities and Artifacts	Languages for MBT Models	Applying Test Selection Criteria	Activities of Test Adaptation in MBT	Manage and Monitor the Deployment of an MBT Approach
Integrating MBT into the Software Development Lifecycles	Good Practices for MBT Modeling Activities			

Source: [ISTQB](https://www.istqb.org/)

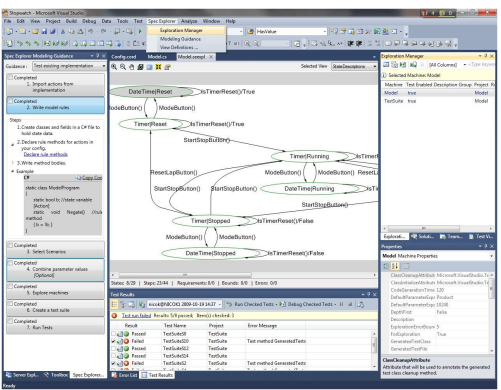
Summary



Many models,
test goals and tools



MBT = using models in testing



Scaling from
brainstorming to
fully automatic
test case generation

