femto-st SCIENCES & TECHNOLOGIES

Model-Based Testing in Practice

Frédéric Dadeau – FEMTO-ST/DISC

Ecole de Jeunes Chercheurs en Programmation

Lille – June 30th, 2016

This PDF and additional material can be found at: http://bit.ly/294MsPR

Who am I?

Assistant professor at FEMTO-ST / DISC (Computer Science Department)

Former student of the EJCP'04 (12 years ago…)

Research interests: Model-Based Testing

- using symbolic execution
- using test scenarios
- using temporal properties
- using model mutations

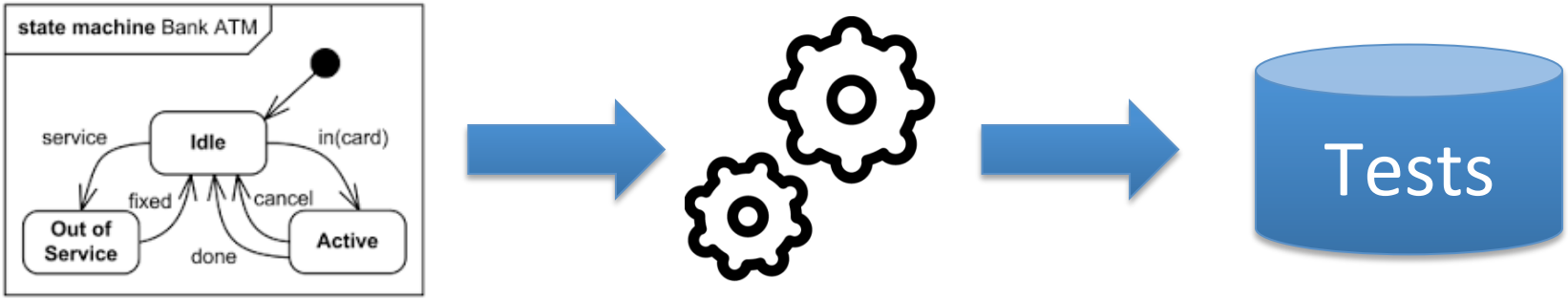
applied to embedded systems, security, and all recent buzzwords (CPS, IoT, etc.) that (used to) get projects funded...

Close collaboration with Smartesting (spin-off the DISC) and the CertifyIt MBT tool

Agenda

1. **What is Model-Based Testing?**
2. Taxonomy of Model-Based Testing approaches
3. Test execution & conformance relationships
4. Practical session – MBT of a web application with ModelJUnit
5. Demonstration – MBT with Smartesting CertifyIt
6. Summary: benefits/drawbacks of the MBT approaches

1. What is Model-Based Testing?



Model-Based Testing: testing based on or involving models

Test case: *A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.*

ISTQB Glossary 2015 – International Software Testing Qualifications Board - http://www.istqb.org – MBT extension

1. What is Model-Based Testing?

In MBT approaches, models are used to:

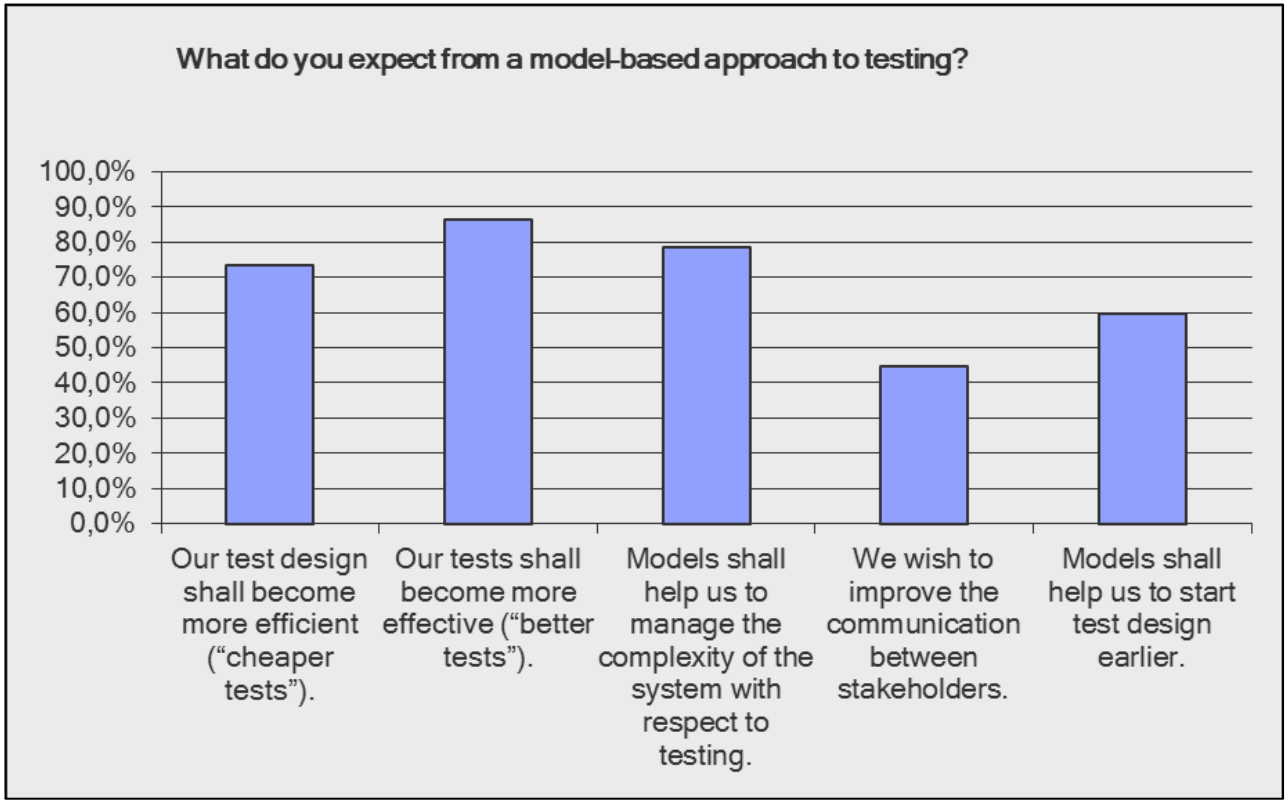
- compute test cases: test data and/or operation sequences
- predict the test oracle, and thus establish the test verdict

→ automate test generation & execution

Scientific challenges:

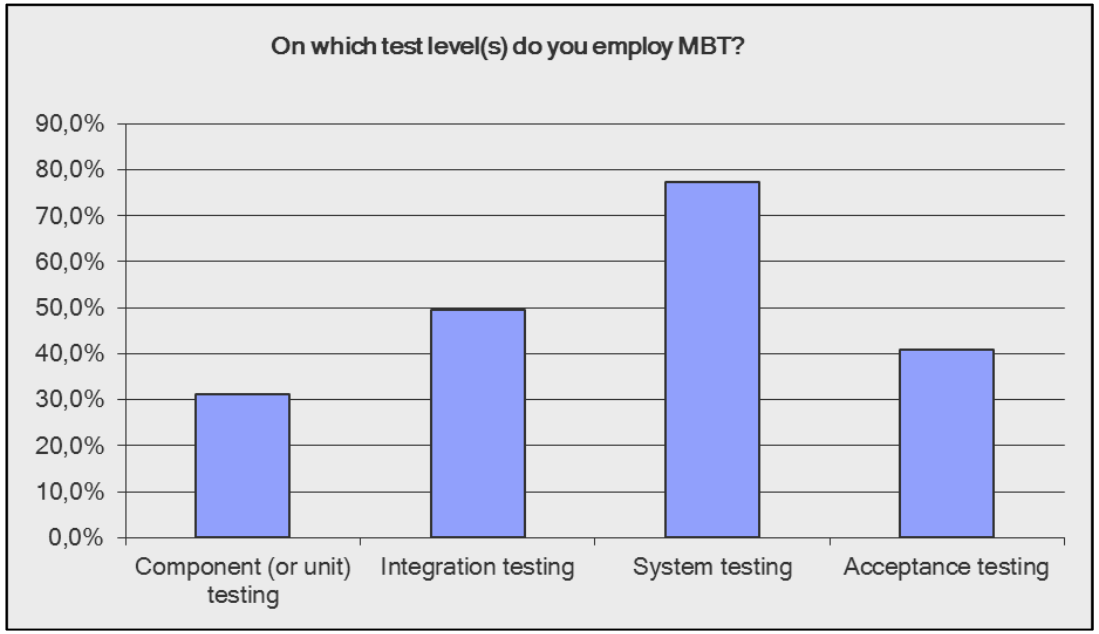
- how to model the SUT?
- how to exploit the model to derive the tests?
- how to bridge the gap between the (abstract) model and the (concrete) SUT?
- how to establish the conformance between the model and the SUT?

1. What is Model-Based Testing?



Source – MBT User Survey 2014
Model-Based Testing – Where do we stand? – CACM – 2/15 – Binder, Legeard, Kramer

1. What is Model-Based Testing?



Source – MBT User Survey 2014

Component testing:

Testing of individual software component

Integration testing:

Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems

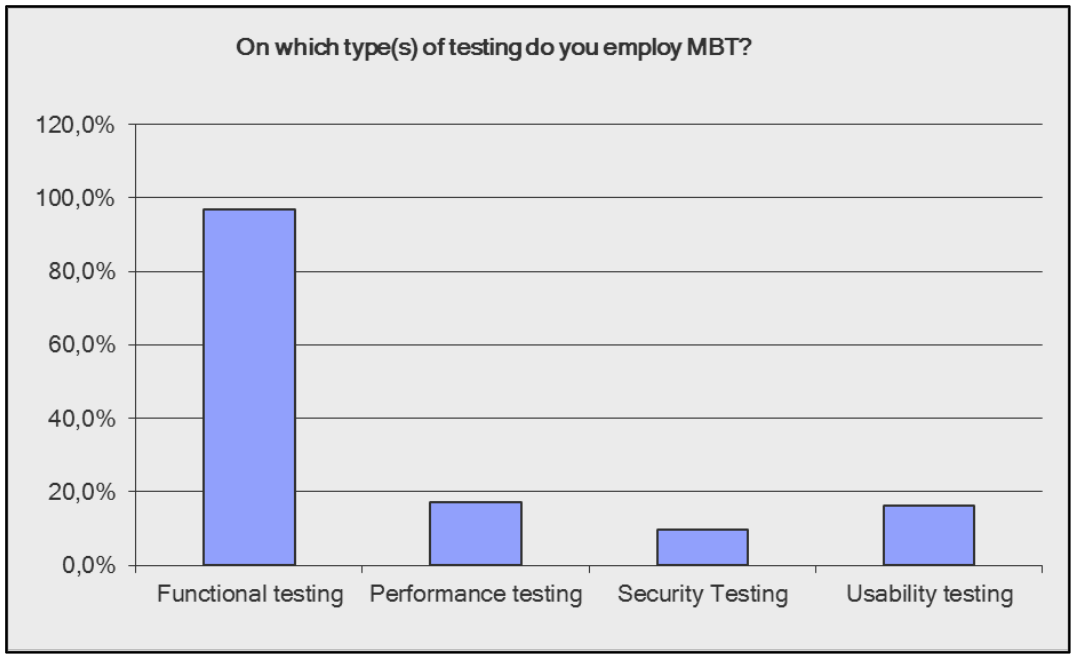
System testing:

Testing an integrated system to verify that it meets specified requirements

Acceptance testing:

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to unable the user/customer/other authorized entity to determine whether or not to accept the system

1. What is Model-Based Testing?



Source – MBT User Survey 2014

Functional testing:

Testing based on an analysis of the specification of the functionality of a component or a system.

Performance testing:

Testing to determine the performance (degree to which a system or a component accomplishes its desginated functions within given constraints regarding processing time and throughput rate) of a software product.

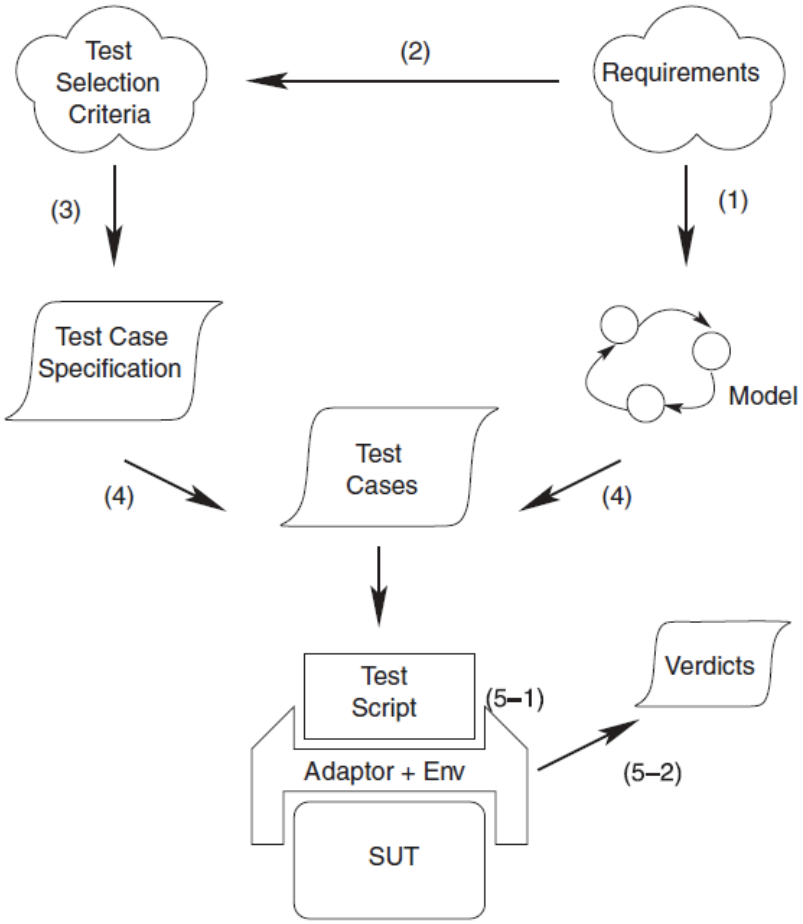
Security testing:

Testing to determine the security (ability to prevent unauthorized access, whether accidental or deliberate, to programs and data) of a software product.

Usability testing:

Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.

1. What is Model-Based Testing?



(1) Model design

(2) Choice of the appropriate test selection criteria

(3) Transformation of TS criteria into an « operational form » (algorithms)

(4) Test cases generation

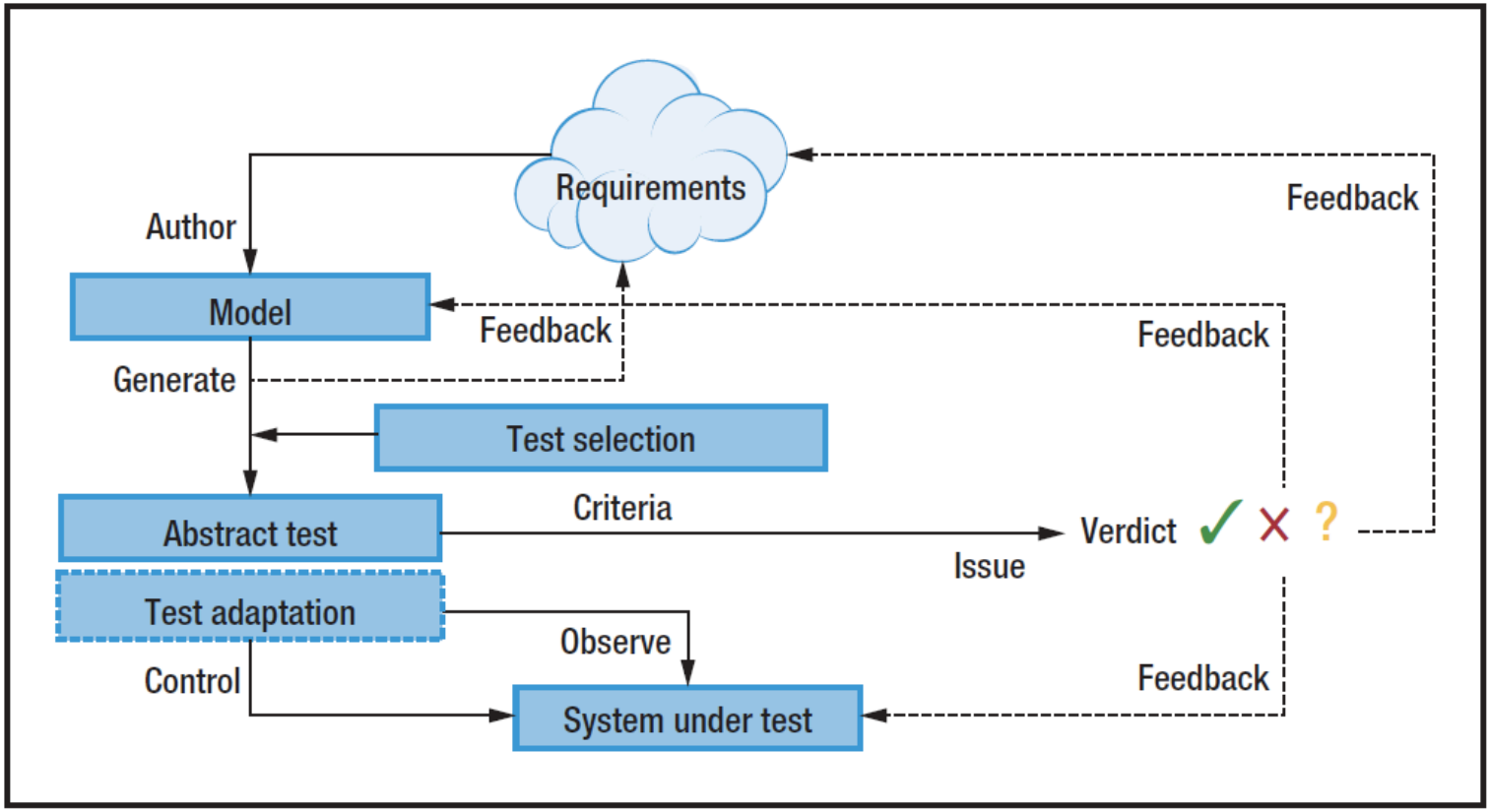
(5) Test execution

(5-1) Concretization of test cases

(5-2) Establishment of the test verdict

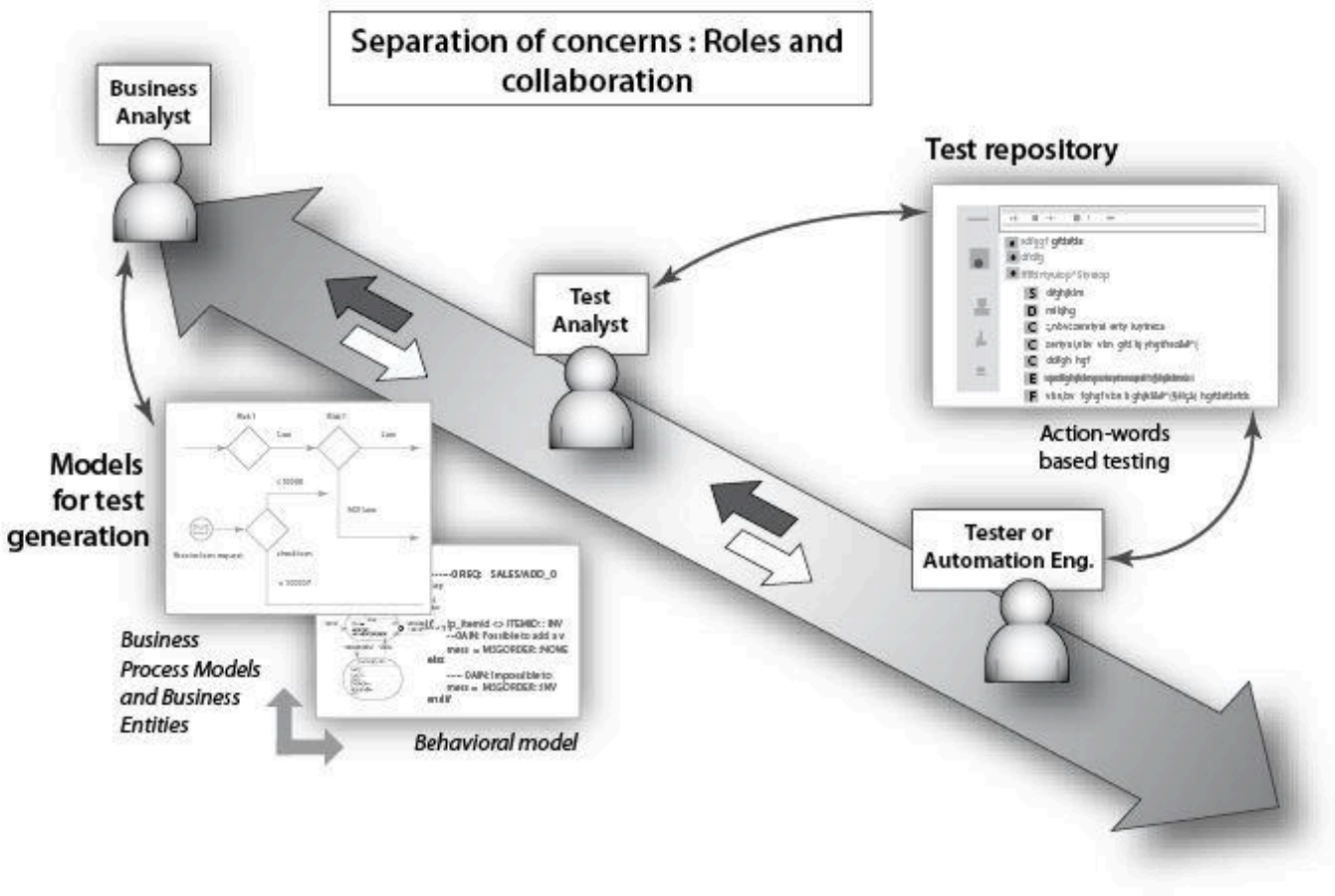
Source: M. Utting, A. Pretschner, B. Legeard
Taxonomy of MBT approaches. STVR, 22-5, 2012

1. What is Model-Based Testing?



Source: Methods for Testing & Specification (MTS); Model-Based Testing (MBT); Requirements for Modelling Notations, ES 202 951 v.1.1.1 European Telecommunications Standards Institute, 2011

1. What is Model-Based Testing?



1. What is Model-Based Testing?

The fine art of MBT modelling…

Models are strongly related to the test objectives!

MBT **test models** should be:

- abstract:
  - to cover what is intended to be tested
- detailed and precise enough:
  - to compute the expected result (oracle) → test verdict assignment
- validated and verified[1]:
  - if the test « fails » what is wrong? the model or the SUT?
    (keep in mind that MBT is « back-to-back testing » - MBT model vs. SUT)

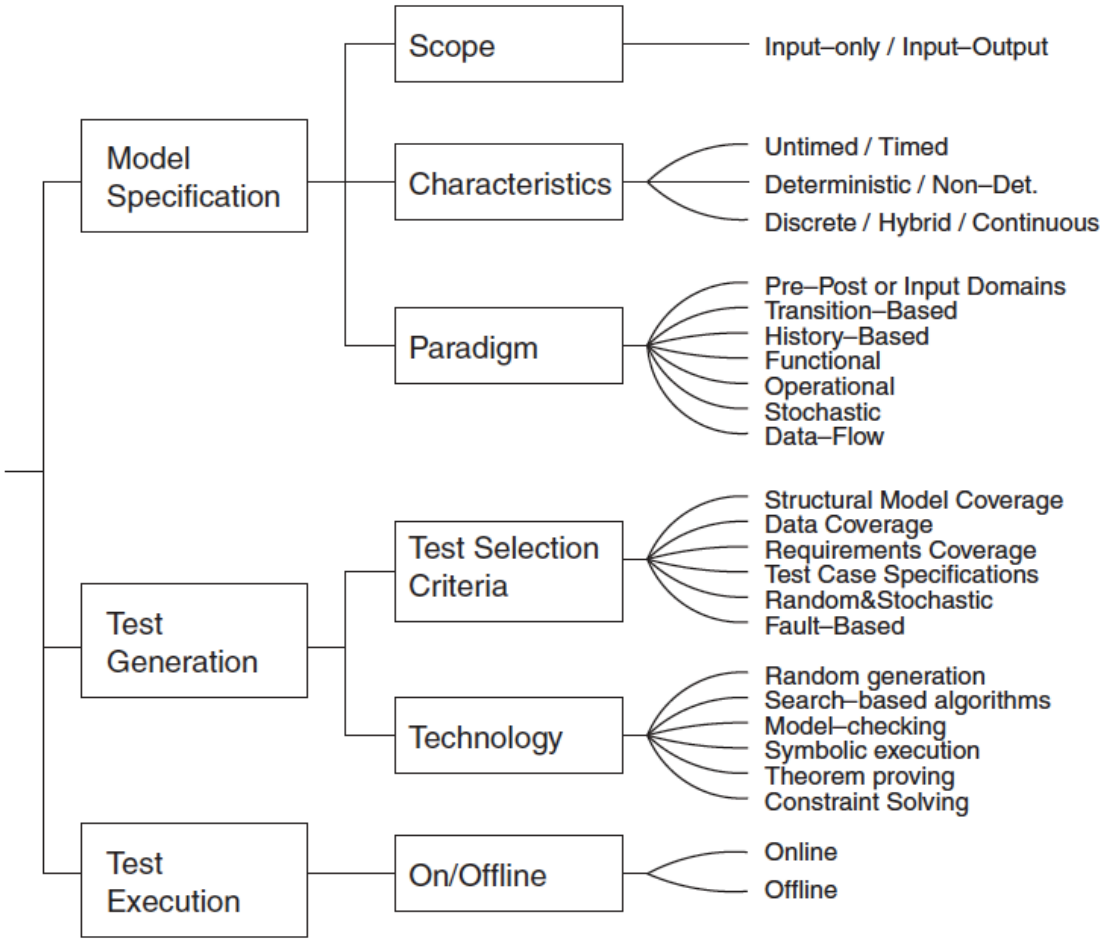
…a difficult trade-off!

[1]a must-read: M.-C. Gaudel. *Checking models, proving programs, testing systems*. Tests & Proofs'2011. LNCS.

Agenda

1. What is Model-Based Testing?
2. **Taxonomy of Model-Based Testing approaches**
3. Test execution & conformance relationships
4. Practical session – MBT of a web application with ModelJUnit
5. Demonstration – MBT with Smartesting CertifyIt
6. Summary: benefits/drawbacks of the MBT approaches

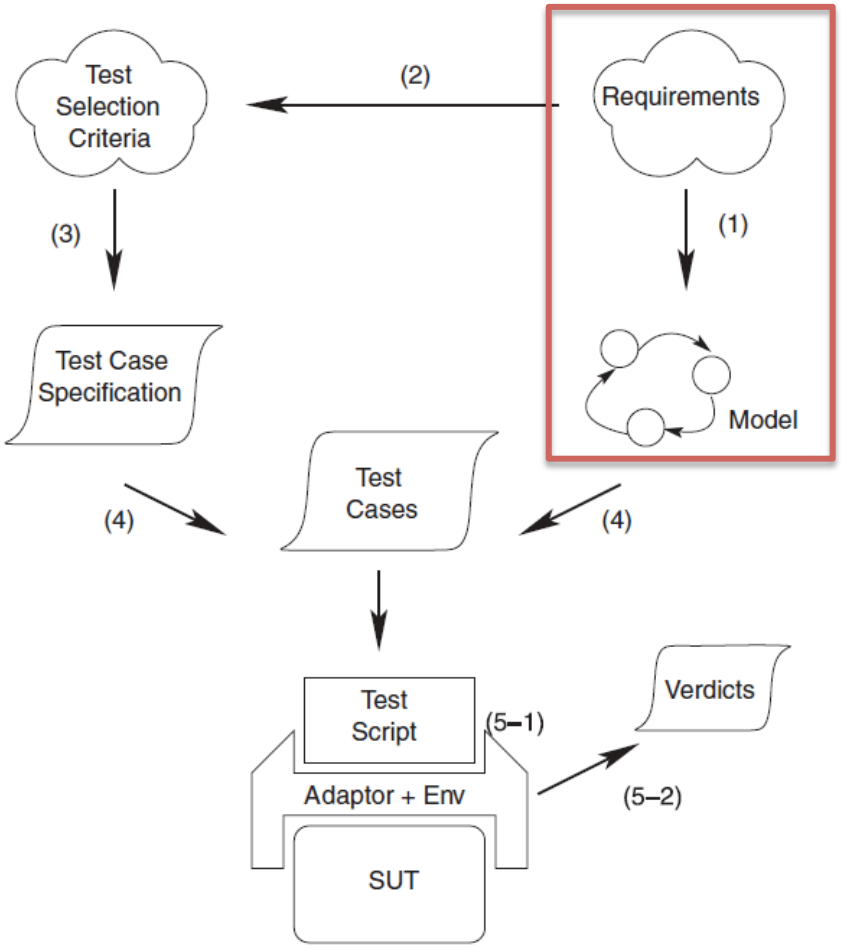
2. Taxonomy of MBT approaches



Source:
M. Utting, A. Pretschner, B. Legeard
Taxonomy of MBT approaches.
Soft. Testing Verif. & Reliability, 22-5, 2012

+ evolutions w.r.t. ISTQB syllabus

2. Taxonomy of MBT approaches – Model Specification



(1) Model design

(2) Choice of the appropriate test selection criteria

(3) Transformation of TS criteria into an « operational form » (algorithms)

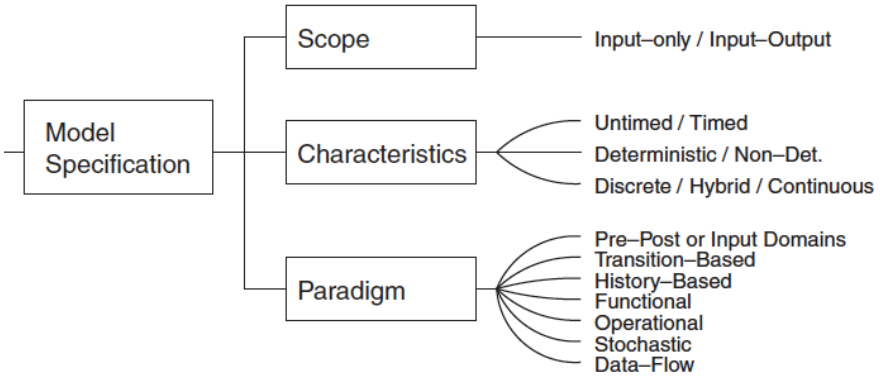
(4) Test cases generation

(5) Test execution

(5-1) Concretization of test cases

(5-2) Establishment of the test verdict

2. Taxonomy of MBT approaches



Scope of the model

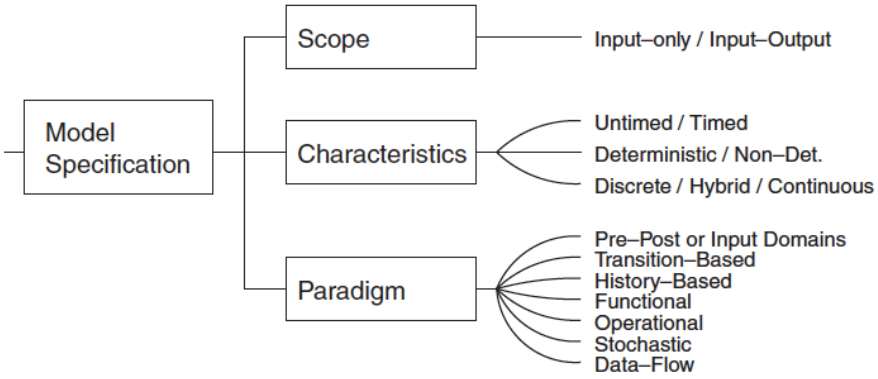
- Input-only: the model only characterises the inputs of the system, not the outputs

  → restricts its usage to test data generation (weak oracle – robustness « no crash »)

- Input-Output: the model defines both input and outputs

  → provides the oracle (excepted result) of the test, useful for functional testing

2. Taxonomy of MBT approaches



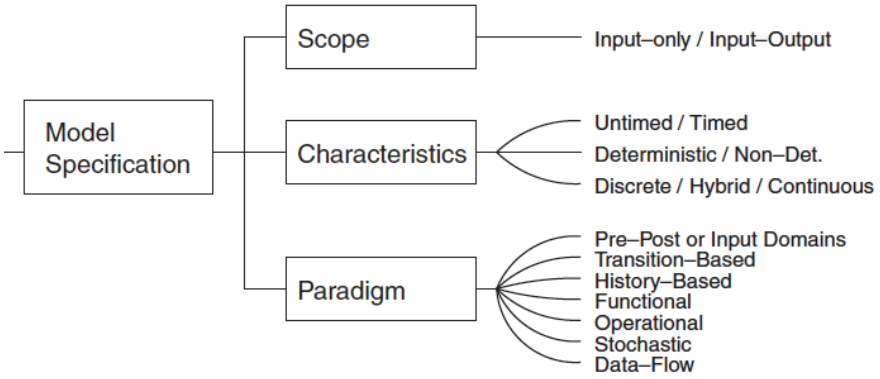
Characteristics of the model

- Untimed: no time constraints to consider
- Timed: used to model real-time systems (with time constraints)

→ Delays/timers/time constraints can be abstracted in the model and later re-introduced in the concretization layer

- Berkenkötter K, Kirner R. Real-time and hybrid systems testing. Model-based Testing of Reactive Systems (LNCS, vol. 3472), Broy M, Jonsson B, Katoen J-P, Leucker M, Pretschner A (eds.). Springer: Berlin, 2005; 355–387.

2. Taxonomy of MBT approaches

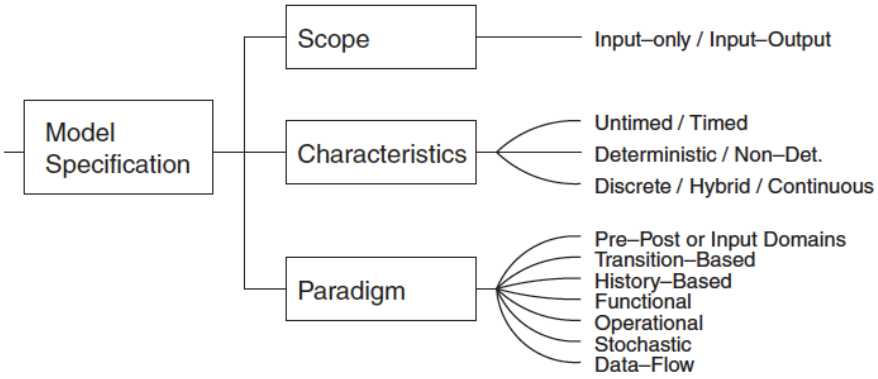


Characteristics of the model

- Deterministic: the same inputs always provide the same outputs
  → favorable situation: accuracy of the test verdict

- Non-Deterministic:
  - frequent in concurrent systems
  - difficulties to establish the test verdict (leading to inconclusive verdicts)
    → if observable non-determinism, define test cases as trees or graphs

2. Taxonomy of MBT approaches

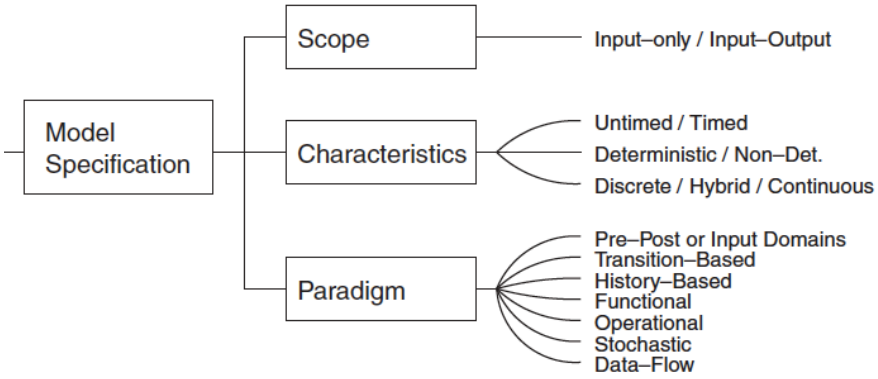


Characteristics of the model

- Event-discrete systems mostly targeted by MBT approaches
- Continuous/hybrid models often used for embedded systems

- Berkenkötter K, Kirner R. Real-time and hybrid systems testing. Model-based Testing of Reactive Systems (LNCS, vol. 3472), Broy M, Jonsson B, Katoen J-P, Leucker M, Pretschner A (eds.). Springer: Berlin, 2005; 355–387.
- Model-Based Testing for Embedded Systems. J. Zander, I. Schieferdecker, P.J. Mosterman. CRC Press, 2012

2. Taxonomy of MBT approaches




```
1if (p_Code=CODE::FAUX)then
2    ---@REQ:COUNTER_DECREASED
3    carte.nbEssaisCode = carte.nbEssaisCode-1 and
4    afficherMessage(MESSAGE::CODE_ERRONE) and
5    if (carte.nbEssaisCode <= 0) then
6        ---@REQ:CARD_BLOCKED
7        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
8        restituerCarte()
9    else
0        ---@REQ:CARD_NOT_BLOCKED
1        afficherMessage(MESSAGE::ENTRER_CODE)
2    endif
3else
4    ---@REQ:OK
5    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
```

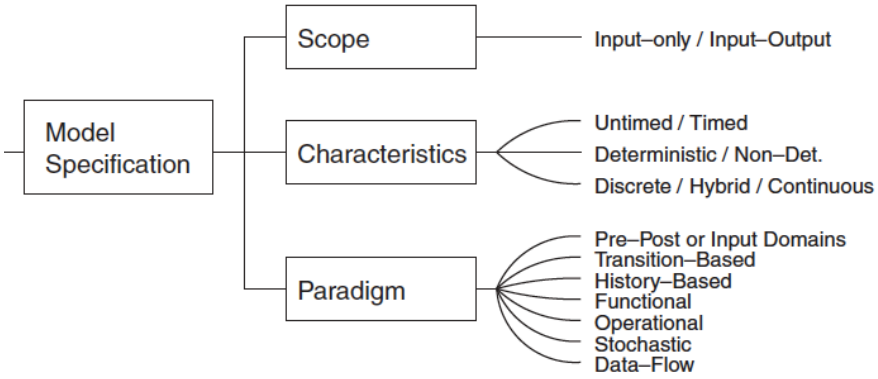
Model paradigm

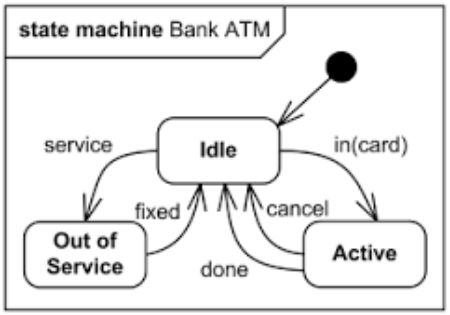
- Pre-Post or Input Domains (a.k.a. State-based) notations

System model = collection of variables + operations written using pre/postconditions

Examples: B (generalized substitutions), Z, VDM, JML (contracts), OCL (preconditions/ postconditions), SpecExplorer (C#-plus-preconditions)

2. Taxonomy of MBT approaches





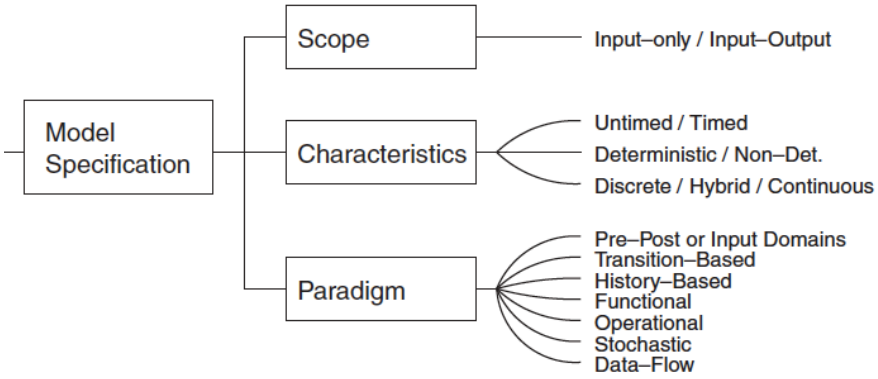
Model paradigm

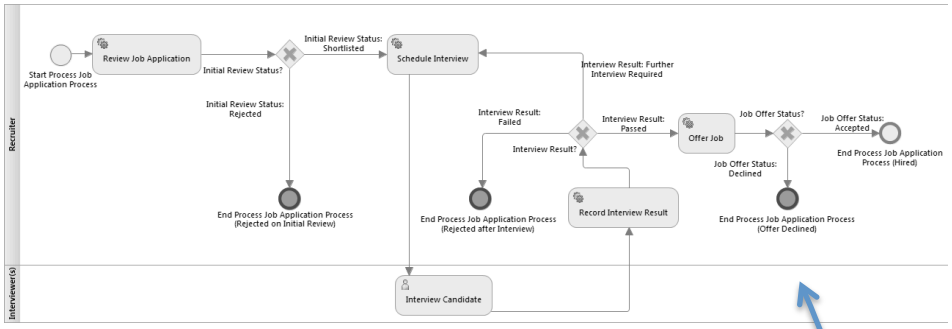
- Transition-based notations

Usually a graphical node-and-arc notations, possibly including hierarchical states, parallelism, etc.

Examples: (Extended-)Finite State Machines, Mealy Machines, statecharts, (Input-Output) Labelled Transitions Systems

2. Taxonomy of MBT approaches





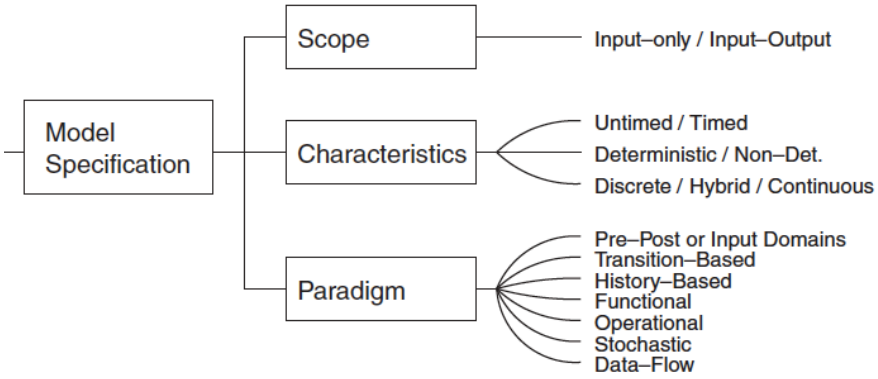
Model paradigm

- History-Based notation / flow charts

Describes the system using allowable traces of its behavior

Examples: Message Sequence Charts, sequence diagrams, activity diagrams, Business Process Model Notation (BPMN)

2. Taxonomy of MBT approaches



ARRAY (Elem: [Undefined → Elem])

sort Array
imports INTEGER

Arrays are collections of elements of generic type Elem. They have a lower and upper bound (discovered by the operations First and Last) Individual elements are accessed via their numeric index.
Create takes the array bounds as parameters and creates the array, initialising its values to Undefined. Assign creates a new array which is the same as its input with the specified element assigned the given value. Eval reveals the value of a specified element. If an attempt is made to access a value outside the bounds of the array, the value is undefined.

Create (Integer, Integer) → Array
Assign (Array, Integer, Elem) → Array
First (Array) → Integer
Last (Array) → Integer
Eval (Array, Integer) → Elem

First (Create (x, y)) = x
First (Assign (a, n, v)) = First (a)
Last (Create (x, y)) = y
Last (Assign (a, n, v)) = Last (a)
Eval (Create (x, y), n) = Undefined
Eval (Assign (a, n, v), m) =
if m < First (a) **or** m > Last (a) **then** Undefined **else**
if m = n **then** v **else** Eval (a, m)

Model paradigm

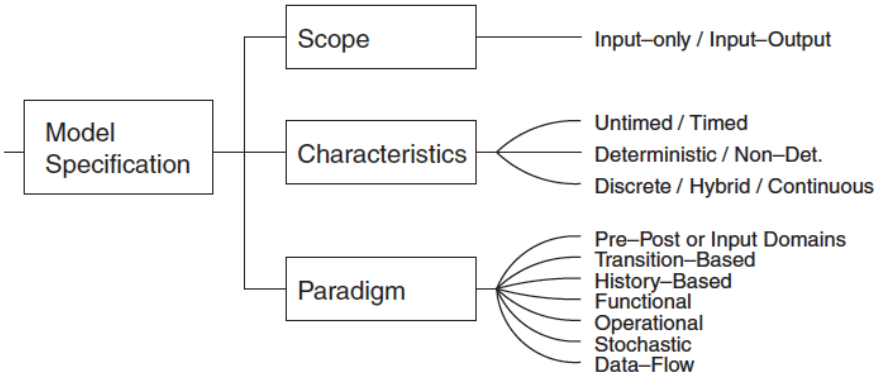
- Functional notation

Describes the system using mathematical functions

Examples: algebraic specifications (first-order logics), HOL (higher-order logics)

- M.-C. Gaudel, P. Le Gall. Testing data type implementations from algebraic specifications. Proceedings of Formal Methods and Testing (LNCS vol. 4949). Springer 2008.

2. Taxonomy of MBT approaches




```
role alice(
  A,B     : agent,
  K       : symmetric_key,
  Hash    : hash_func,
  SND,RCV : channel(dy))
played_by A def=

  local
    State   : nat,
    Na,Nb   : text,
    K1      : message

  init
    State := 0

  transition

  1. State  = 0 /\ RCV(start) =|>
     State':= 2 /\ Na' := new()
                /\ SND({Na'}_K)

  2. State  = 2 /\ RCV({Nb'}_K) =|>
     State':= 4 /\ K1' := Hash(Na.Nb')
                /\ SND({Nb'}_K1')
                /\ witness(A,B,bob_alice_nb,Nb')

end role
```

```
role bob (A,B: agent,
          Ka: public_key,
          Kb: public_key,
          Snd,Rcv: channel(dy))
played_by B
def=

  local State   : nat,
        Na,Nb : text
  const sec_k2 : protocol_id
  init  State := 0

  transition
   1. State = 0 /\ Rcv({Na'.A}_Kb)
      =|>
      State' := 1
      /\ Nb' := new()
      /\ Snd({Na'.Nb'}_Ka)
      /\ witness(B,A,nb,Nb')
      /\ secret(Nb',sec_k2,{A,B})
   2. State = 1 /\ Rcv({Nb}_Kb)
      =|>
      State' := 2
      /\ request(A,B,na,Na)
end role
```

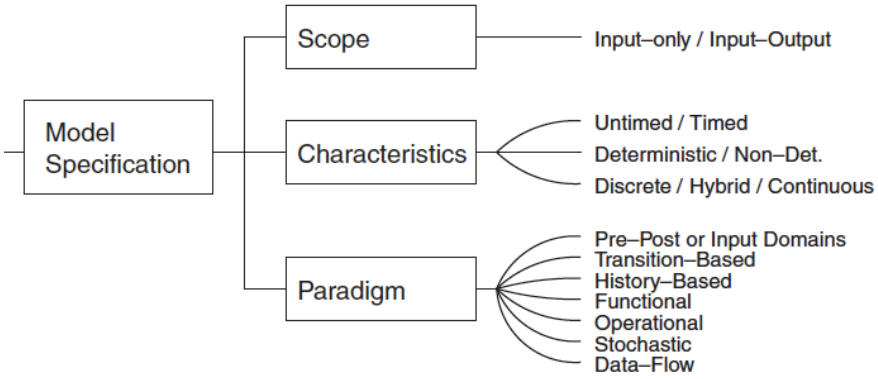
Model paradigm

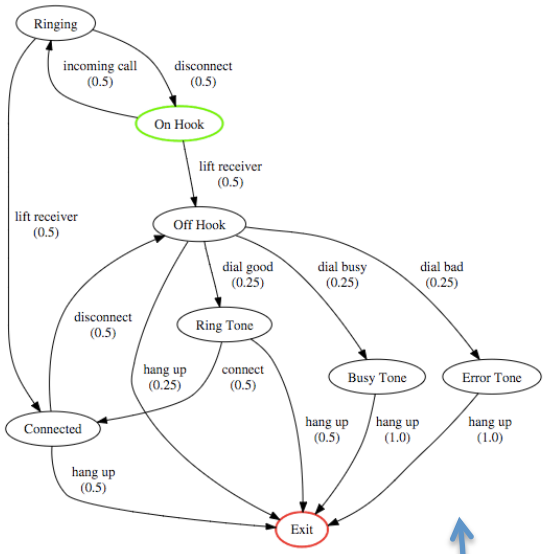
- Operational notation

Describes the system as a collection of executable processes, executing in parallel.

Examples: process algebra such as Communicating Sequential Processes (CSP), VHDL, Petri Nets, High-Level Protocol Specification Language (HLPSL)

2. Taxonomy of MBT approaches





Model paradigm

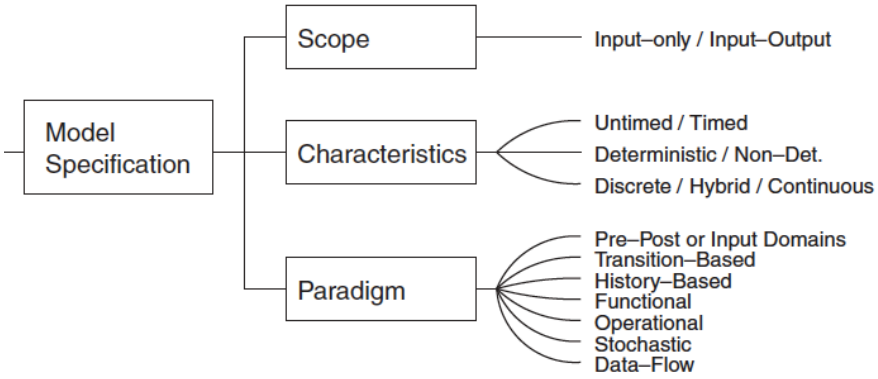
- Stochastic notation

Describes the system by a probabilistic model of the events and input values (mainly used for modeling environments rather than SUT – usage model)

Examples: Markov chains

- J. A. Whittaker and M. G. Thomason, *A Markov chain model for statistical software testing*, in IEEE Trans. on Software Engineering, vol. 20, no. 10, pp. 812-824, Oct 1994.

2. Taxonomy of MBT approaches




```
node TIME_STABLE(set, second:  bool; delay:  int)
returns (level:  bool);
var ck:  bool;
let
  level = current(STABLE((set, delay) when ck));
  ck = true -> set or second;
tel;

node STABLE(set:  bool; delay:  int)
returns (level:  bool);
var count:  int;
let
  level = (count>0);
  count = if set then delay
        else if false->pre(level) then pre(count)-1
        else 0;

tel;
```

Model paradigm

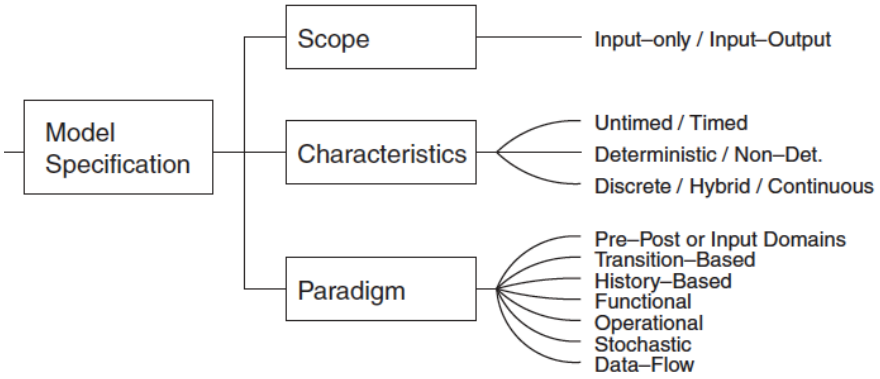
- Data-flow notation

Describes the system by focusing on data rather than control flow

Examples: Lustre, block diagrams of Matlab Simulink (continuous systems)

- B. Marre and A. Arnould, *Test sequences generation from LUSTRE descriptions: GATEL. Proceedings 15th IEEE Int. Conf. on Automated Soft. Engineering, 2000.*

2. Taxonomy of MBT approaches



+ Subject & Focus (ISTQB novelty)

A MBT model generally combines these various subject/focus aspects.

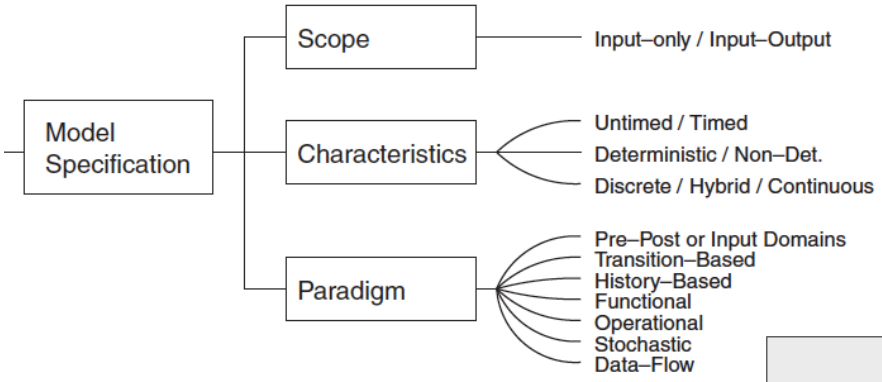
Subject

- System (most approaches)
- Environment
  e.g. Matlab Simulink, usage models
- Tests: behaviour of the tester
  (e.g. activate X and check value of Y)

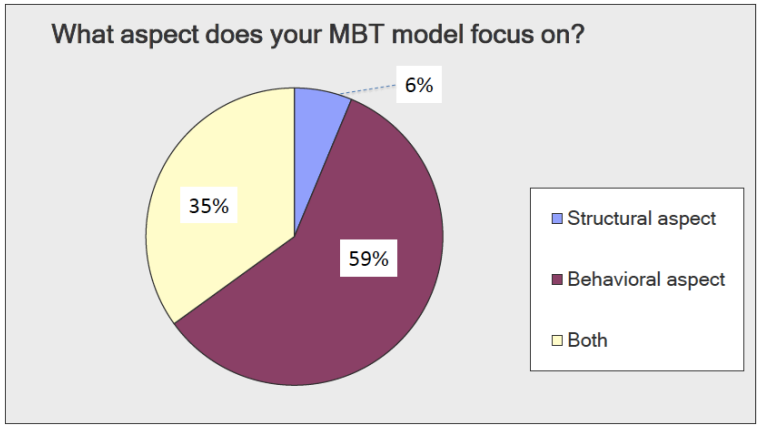
Focus

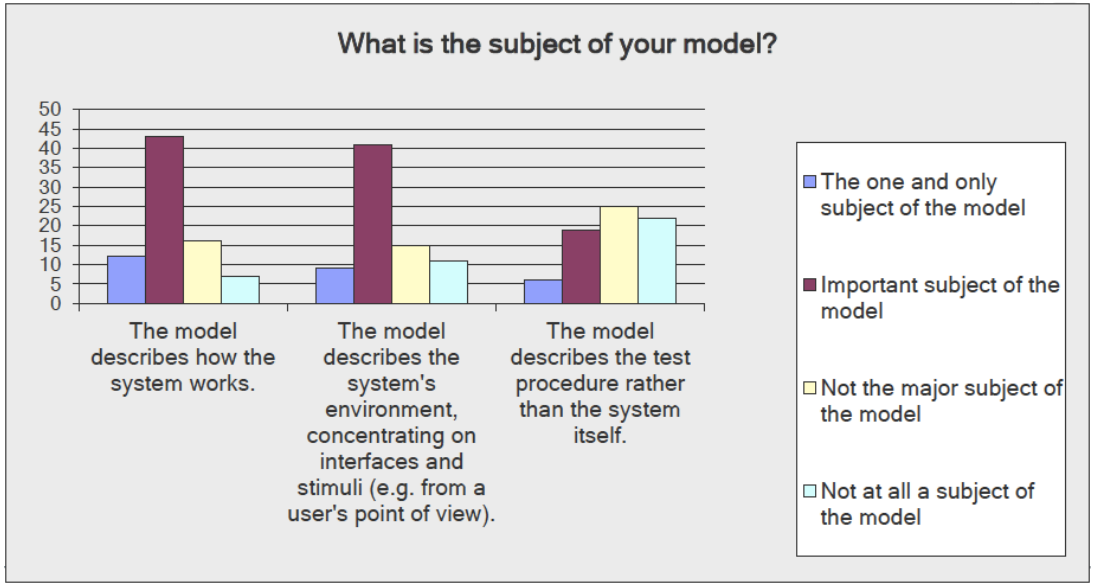
- Structure
- Behaviour

2. Taxonomy of MBT approaches

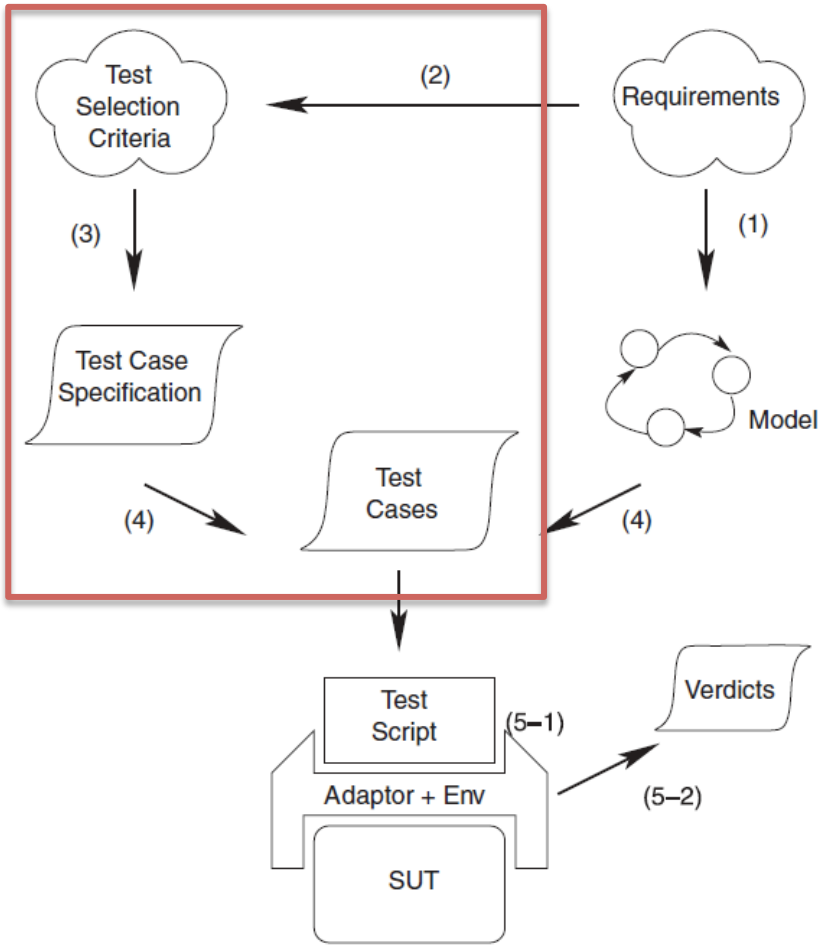


+ Subject & Focus





2. Taxonomy of MBT approaches – Test Generation



(1) Model design

(2) Choice of the appropriate test selection criteria

(3) Transformation of TS criteria into an « operational form » (algorithms)

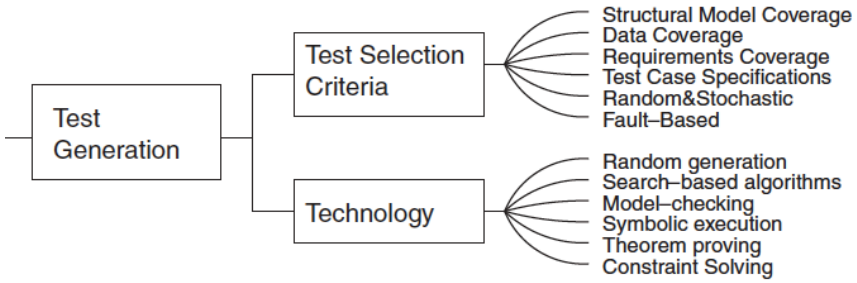
(4) Test cases generation

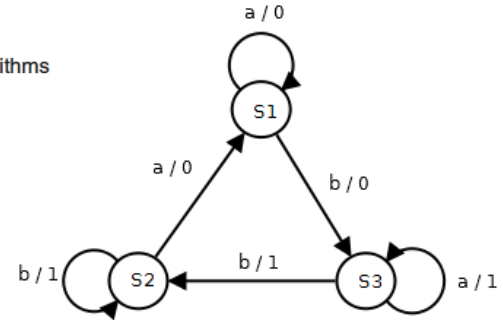
(5) Test execution

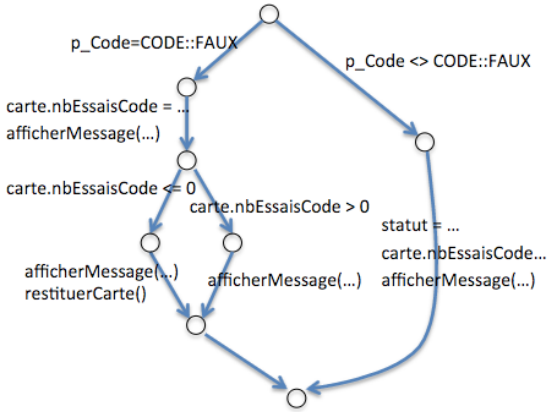
(5-1) Concretization of test cases

(5-2) Establishment of the test verdict

2. Taxonomy of MBT approaches







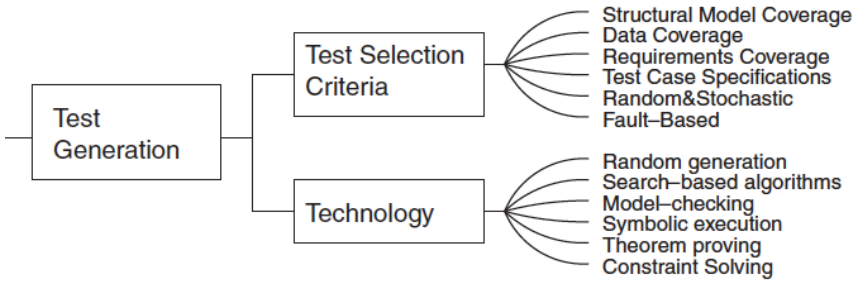
Test selection criteria

- Structural model coverage

The generated test cases aim to cover the structure of the model (control flow graph coverage for textual notations, node/arc-coverage for FSM, etc.)

- Jeremy Dick and Alain Faivre. 1993. Automating the Generation and Sequencing of Test Cases from Model-Based Specifications. In Proceedings of the 1st Int. Symposium of Formal Methods Europe on Industrial-Strength Formal Methods (FME'93), Springer.
- Aho A, Dahbura A, Lee D, Uyar MU. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. IEEE Transactions on Communications 1991; 39(11):1604–1615.
- Lee D, Yannakakis M. Principles and methods of testing finite state machines—A survey. Proceedings of the IEEE 1996; 84(2):1090–1126.

2. Taxonomy of MBT approaches



# Cas	OS	Réseau	Imprimante	Application
Cas 1	XP	ATM	Canon-EX	Pwpoint
Cas 2	XP	Bluetooth	Canon900	Word
Cas 3	XP	Wifi	HP35	Excel
Cas 4	Linux	ATM	HP35	Word
Cas 5	Linux	Bluetooth	Canon-EX	Excel
Cas 6	Linux	Wifi	Canon900	Pwpoint
Cas 7	Mac OS X	ATM	Canon900	Excel
Cas 8	Mac OS X	Bluetooth	HP35	Pwpoint
Cas 9	Mac OS X	Wifi	Canon-EX	Word

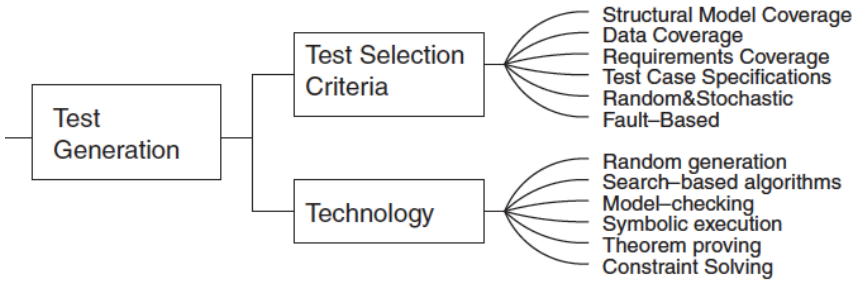
Test selection criteria

- Data coverage

The generated test cases aim to cover particular data values that can be identified using equivalence classes, N-Wise coverage, boundary analysis, etc.

- D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, *The Combinatorial Design Approach to Automatic Test Generation*, IEEE Software, vol. 13, no. 5, Sept. 1996
- A.W. Williams and R.L. Probert, *A Practical Strategy for Testing Pair-Wise Coverage of Network interfaces*, Proc. IEEE Int'l Symp. Software Reliability Eng., 1996.
- B. Legeard, F. Peureux, and M. Utting. *Automated Boundary Testing from Z and B*. In Proceedings of the Int. Sym. of Formal Methods Europe on Formal Methods, 2002.

2. Taxonomy of MBT approaches




```
1if (p_Code=CODE::FAUX)then
2    ---@REQ:COUNTER_DECREASED
3    carte.nbEssaisCode = carte.nbEssaisCode-1 and
4    afficherMessage(MESSAGE::CODE_ERRONE) and
5    if (carte.nbEssaisCode <= 0) then
6        ---@REQ:CARD_BLOCKED
7        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
8        restituerCarte()
9    else
0        ---@REQ:CARD_NOT_BLOCKED
1        afficherMessage(MESSAGE::ENTRER_CODE)
2    endif
3else
4    ---@REQ:OK
5    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
```

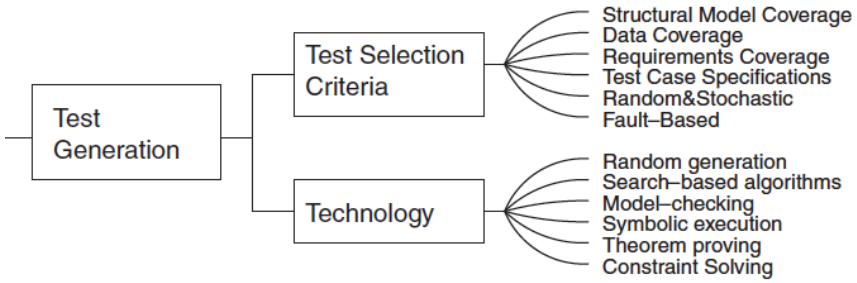
Test selection criteria

- Requirements coverage

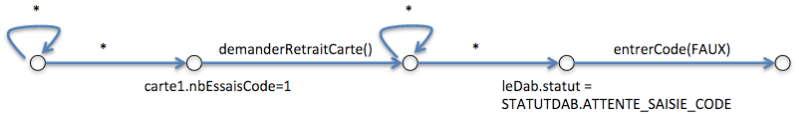
The generated test cases aim to cover the parts of the model that are explicitely related to initial requirements (e.g. preconditions, effect, transition)

- Mogyorodi, G., (2003), *What Is Requirements-Based Testing?* - http://www.stsc.hill.af.mil/crosstalk/2003/03/Mogyorodi.html
- C. Nebut, S. Pickin, Y. Le Traon and J. M. Jezequel, *Automated requirements-based generation of test cases for product families. 18th IEEE Int. Conf. on Automated Software Engineering*, 2003.
- F. Bouquet, E.Jaffuel, B. Legeard, F. Peureux, and M. Utting. *Requirement Traceability in Automated Test Generation - Application to Smart Card Software Validation*. In Procs. of the ICSE Int. Workshop on Advances in Model-Based Software Testing (A-MOST'05), 2005.

2. Taxonomy of MBT approaches




```
use any_operation any_number_of_times
        to_reach "nbEssaisCode = 1" on_instance carte1
then use leDab.demanderRetraitCarte()
then use any_operation any_number_of_times
        to_reach "statut=STATUTDAB::ATTENTE_SAISIE_CODE" on_instance leDab
then use leDab.entrerCode(FAUX)                    // should block and eject the card
```



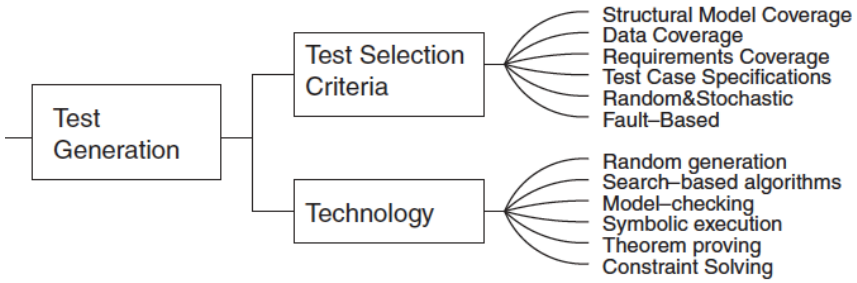
Test selection criteria

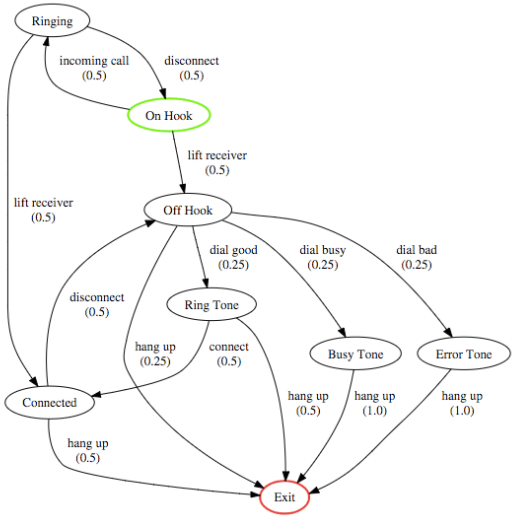
- Test Case Specifications

The generated tests aim to follow a given scenario (TCS) that can be written using a formal notation, restricting the paths of the model that will be exercised.

- J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. *Using On-The-Fly Verification Techniques for the Generation of test Suites*. In Proceedings of the 8th International Conference on Computer Aided Verification (CAV '96), 1996. (Outil TGV)
- Y. Ledru, L. du Bousquet, P. Bontron, O. Maury, C. Oriat and M. L. Potet, *Test purposes: adapting the notion of specification to testing*, Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on, 2001, pp. 127-134.
- Tsai WT, Saimi A, Yu L, Paul R. *Scenario-based object-oriented testing framework*. QSIC 03, Dallas, U.S.A., 2003; 410.

2. Taxonomy of MBT approaches





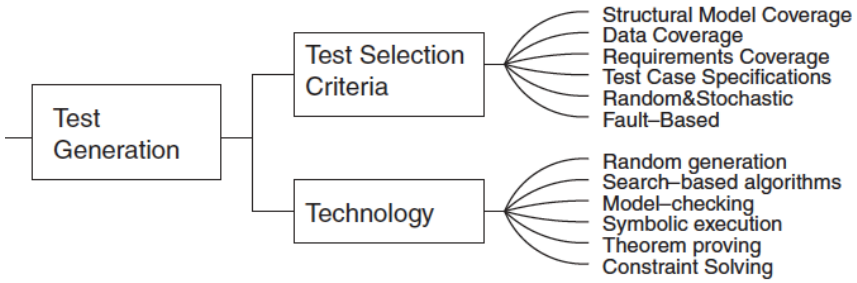
Test selection criteria

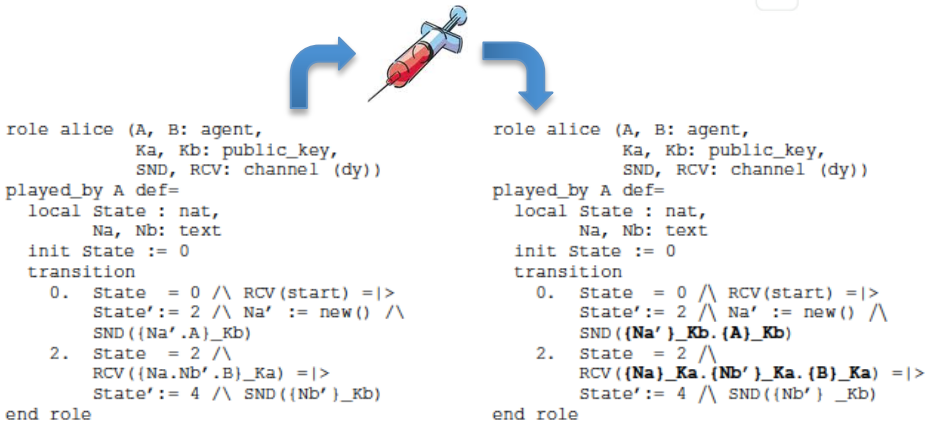
- Random & Stochastic

The generated tests aim to follow an expected usage profile (mainly used with environment models).

- Flajolet, P., Zimmermann, P., Cutsem, B.V.: *A calculus for the random generation of labelled combinatorial structures*. Theoretical Computer Science 132, 1{3}, 1994.
- A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of large models and application to testing. STTT, International Journal on Software Tools for Technology Transfer, 14(1):73-93, 2012
- P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet. *Software statistical testing*. Predictably dependable computing systems, ISBN 3-540-59334-9. Springer, 1995.

2. Taxonomy of MBT approaches





```
role alice (A, B: agent,
            Ka, Kb: public_key,
            SND, RCV: channel (dy))
played_by A def=
  local State : nat,
        Na, Nb: text
  init State := 0
  transition
    0.  State  = 0 /\ RCV(start) =|>
        State':= 2 /\ Na' := new() /\
        SND({Na'.A}_Kb)
    2.  State  = 2 /\
        RCV({Na.Nb'.B}_Ka) =|>
        State':= 4 /\ SND({Nb'}_Kb)
end role
```

```
role alice (A, B: agent,
            Ka, Kb: public_key,
            SND, RCV: channel (dy))
played_by A def=
  local State : nat,
        Na, Nb: text
  init State := 0
  transition
    0.  State  = 0 /\ RCV(start) =|>
        State':= 2 /\ Na' := new() /\
        SND({Na'}_Kb.{A}_Kb)
    2.  State  = 2 /\
        RCV({Na}_Ka.{Nb'}_Ka.{B}_Ka) =|>
        State':= 4 /\ SND({Nb'}_Kb)
end role
```

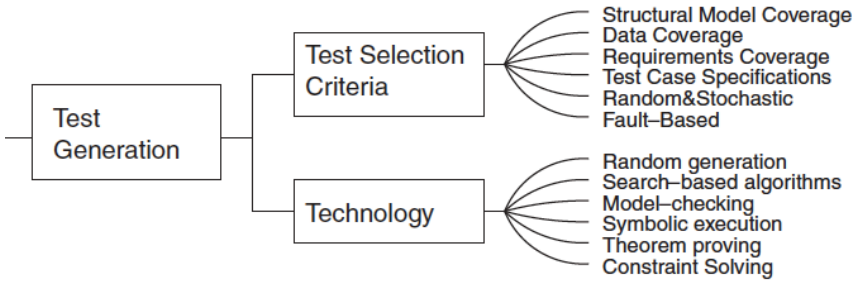
Test selection criteria

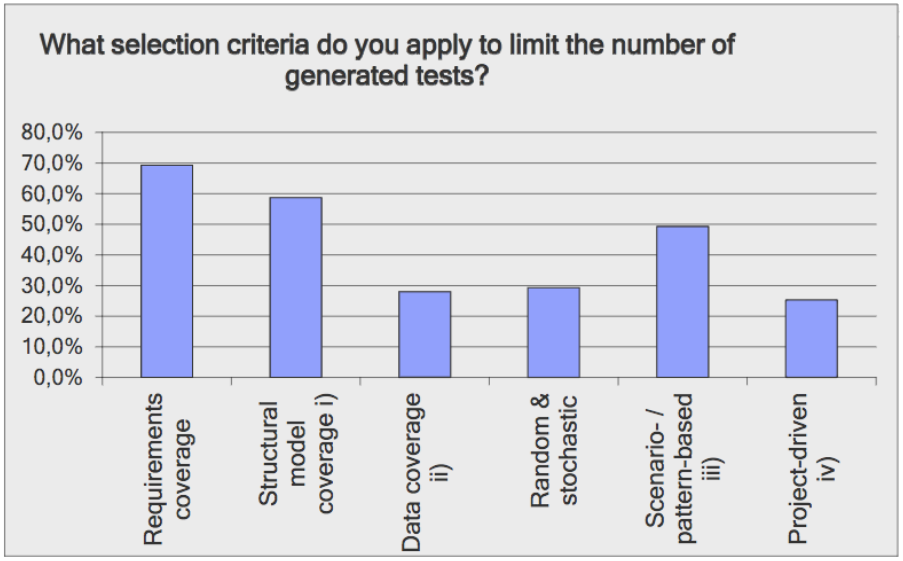
- Fault-based

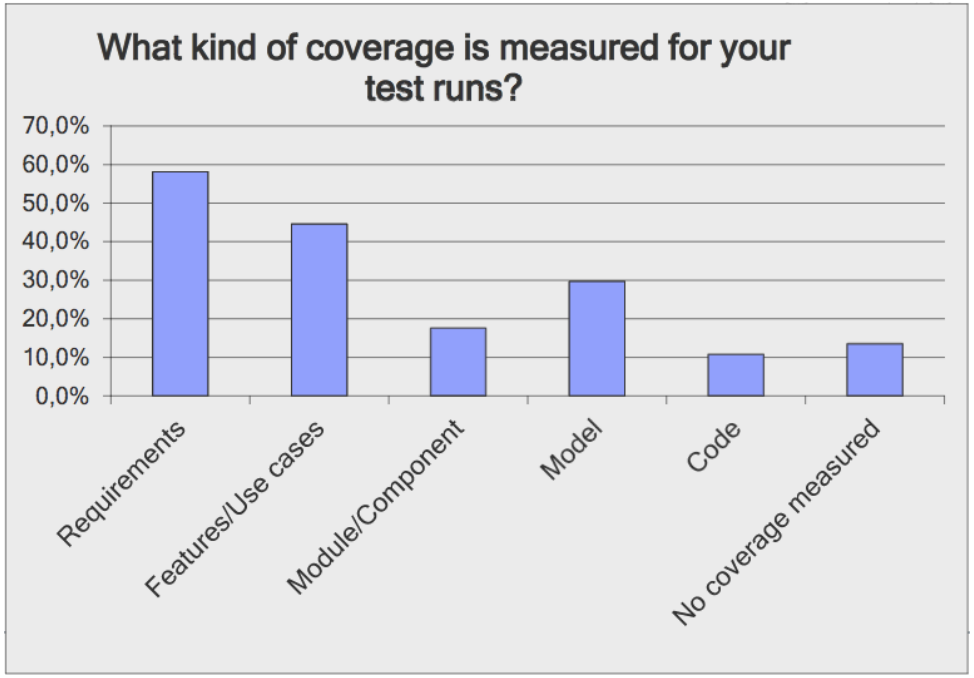
The generated tests aim to exhibit a given set of faults seeded in the model (w.r.t. the initial model). Model faults are assumed to represent a « real » implementation fault.

- T. A. Budd and A. S. Gopal, *Program Testing by Specification Mutation*, Computer Languages, vol. 10, no. 1, pp. 63–73, 1985.
- Y. Jia and M. Harman, *An Analysis and Survey of the Development of Mutation Testing*, in IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 649-678, Sept.-Oct. 2011.
- F. Dadeau, P.-C. Héam, R. Kheddam, G. Maatoug, and M. Rusinowitch. Model-Based Mutation Testing from Security Protocols in HLPSL. Software Testing, Verification and Reliability, 25(5-7):684--711, 2015.

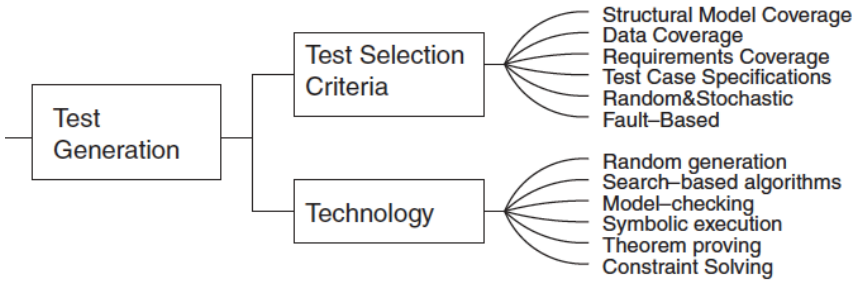
2. Taxonomy of MBT approaches







2. Taxonomy of MBT approaches



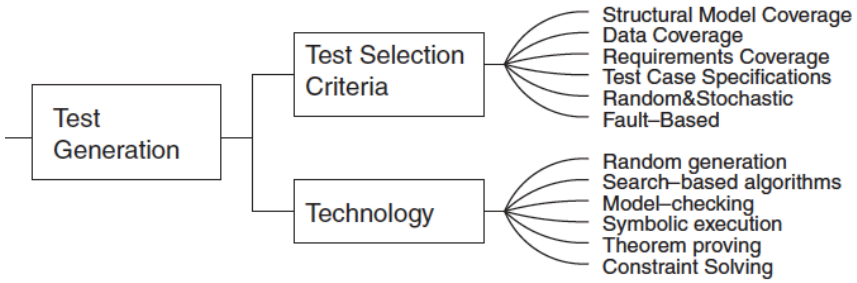
Technology

- Random generation

Sampling of the input space of a system. Can be used for data generation or test cases (operation sequences), possibly biaised to satisfy usage profile.

- A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lasseigne, and S. Peyronnet. *Uniform random sampling of traces in very large models*. In 1st International ACM Workshop on Random Testing, pages 10 -19, July 2006.
- S. J. Prowell. *JUMBL: a tool for model-based statistical testing*, Proceedings of the 36th Int. Conference on System Sciences, 2003.
- C. Oriat. *Jartege: a tool for random generation of unit tests for java classes*. In Proceedings of the 1st int. conf. on Quality of Software Architectures and Software Quality, 2005.

2. Taxonomy of MBT approaches



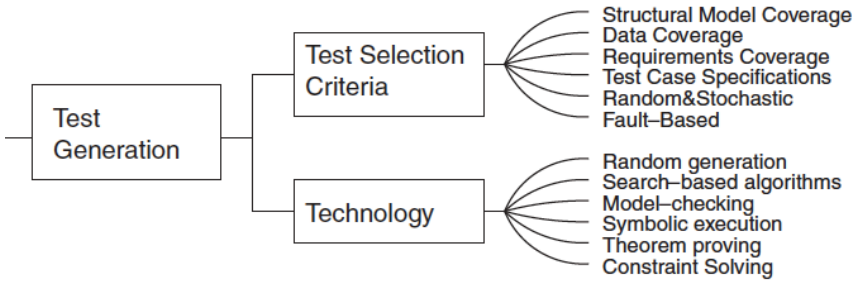
Technology

- Search-based algorithms

Application of optimization and metaheuristic search techniques such as evolutionary algorithms (genetic programming), simulated annealing, etc. Especially used for test data selection.

- P. McMinn. *Search-Based Software Test Data Generation: A Survey*. STVR vol. 14, no. 2, pp. 105–156, 2004
- A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2003).
- P. McMinn. *Search-Based Software Testing: Past, Present and Future*. In Proceedings of the 2011 IEEE 4th Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW '11). IEEE Computer Society, Washington, DC, USA, 153-163.

2. Taxonomy of MBT approaches



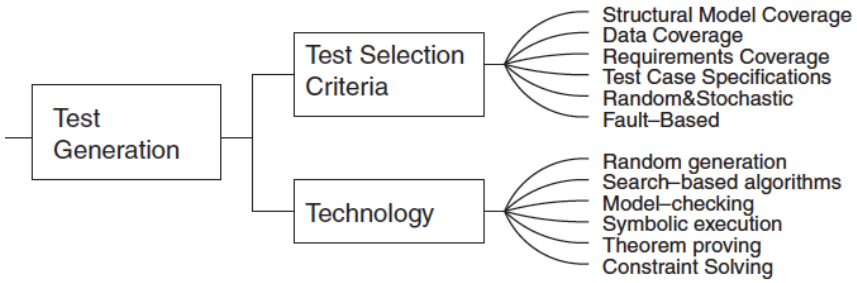
Technology

- Model-checking

Exhaustive exploration of state space. Often used in conjunction with (the negation of) a temporal property.

- J. Callahan, F. Schneider, and S. Easterbrook. *Automated Software Testing Using Model-Checking*. In Proceedings 1996 SPIN Workshop
- G. Fraser, F. Wotawa, and P.E. Ammann, *Testing with model checkers: a survey*, Software Testing, Verification and Reliability, vol. 19, 2009.
- A. Gargantini , C. Heitmeyer. *Using model checking to generate tests from requirements specifications*. Proceedings of the 7th European software engineering conference. 1999.
- L. Tan, O. Sokolsky, I. Lee. *Specification-based testing with linear temporal logic*, In: IRI'2004, 493--498.

2. Taxonomy of MBT approaches



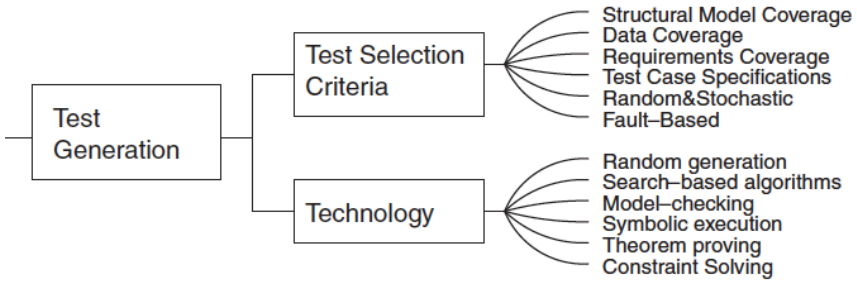
Technology

- Symbolic execution

Run the (executable) model with a set of input values (symbolic values), to obtain symbolic traces, representing a set of executions, that can be conceretized.

- Pretschner A. *Classical search strategies for test case generation with constraint logic programming*. Proceedings of the Formal Approaches to Testing of Software, Aalborg, Denmark, 2001; 47–60.
- C. S. Pasareanu, W. Visser, D. H. Bushnell, J. Geldenhuys, P. C. Mehlitz, N. Rungta. *Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis*. Autom. Softw. Eng. 20(3): 391-425 (2013)
- Colin S, Legeard B, Peureux F. Preamble computation in automated test case generation using constraint logic programming. Journal of Software Testing, Verification and Reliability 2004; 14(3):213–235.

2. Taxonomy of MBT approaches



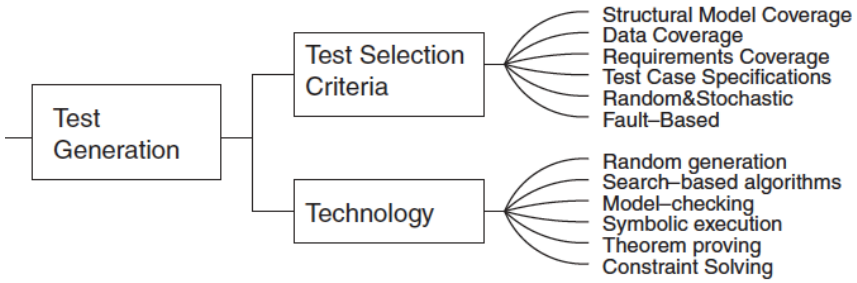
Technology

- Theorem proving

Use of deductive theorem provers, supporting the generation of witness traces or counterexamples, that check the satisfiability of a formula (guard on transitions, path condition, etc.)

- Achim D. Brucker and Burkhart Wolff. On Theorem Prover-based Testing. In Formal Aspects of Computing, 25 (5), pages 683-721, 2013.
- C. Jard and T. Jéron. *TGV: theory, principles and algorithms*. Software Tools for Technology Transfer, 7(4):297–315, 2005.
- Bentakouk, L., Poizat, P., Zaïdi, F.: Checking the behavioral conformance of web services with symbolic testing and an smt solver. In Gogolla, M., Wolff, B., eds.: TAP. Volume 6706 of LNCS., Springer (2011) 33-50.

2. Taxonomy of MBT approaches



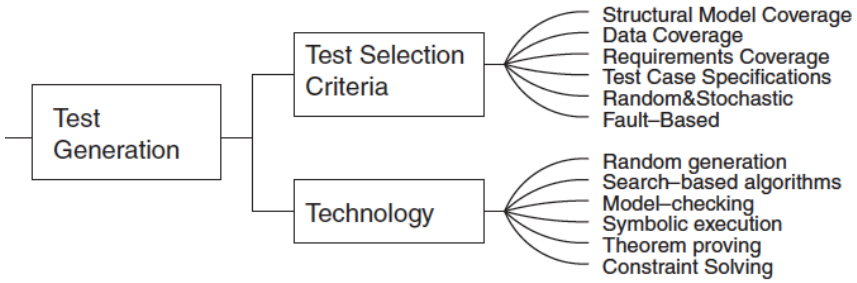
Technology

- Constraint solving

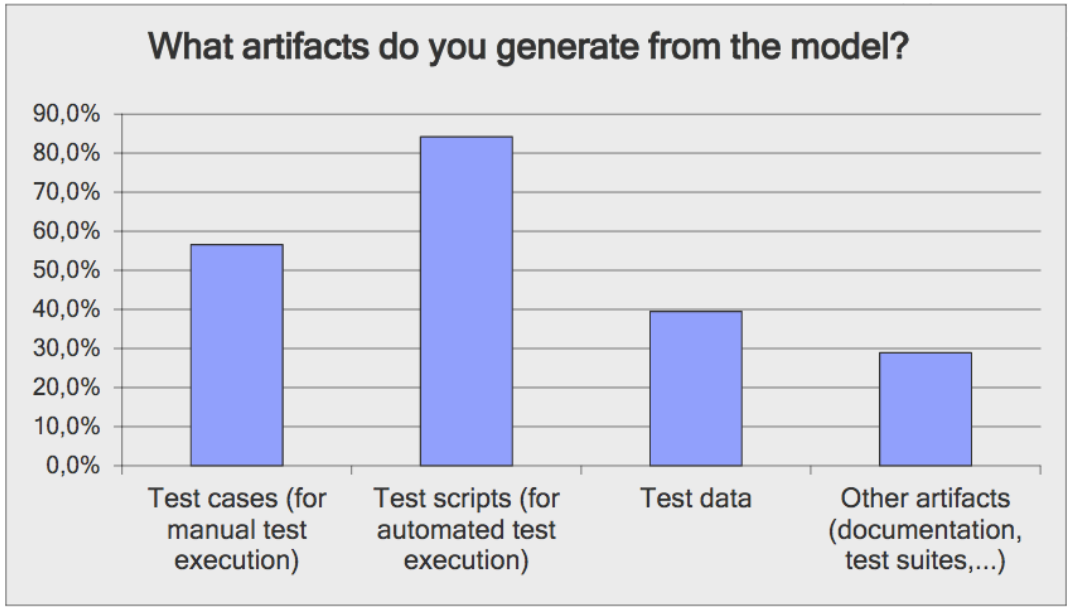
Represent the test case specification as a Constraint Satisfaction Problem (variables with finite domains, and associated constraints) that is instantiated to get a solution (or not).

- Clarke D, Jéron T, Rusu V, Zinovieva E. *STG: A symbolic test generation tool*. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02) (Lecture Notes Computer Science, vol. 2280). Springer: Berlin, 2002; 470–475.
- Colin S, Legeard B, Peureux F. *Preamble computation in automated test case generation using constraint logic programming*. Journal of Software Testing, Verification and Reliability 2004; 14(3):213–235.
- B. Marre and B. Blanc. *Test selection strategies for lustre descriptions in Gatel*. Electronic Notes in Theoretical Computer Science, 111:93 – 111, 2005.

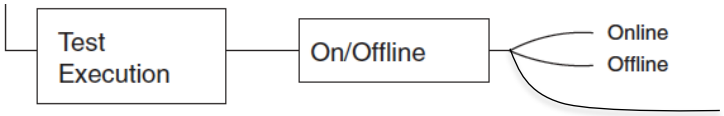
2. Taxonomy of MBT approaches



+ Generated artifacts (ISTQB novelty)



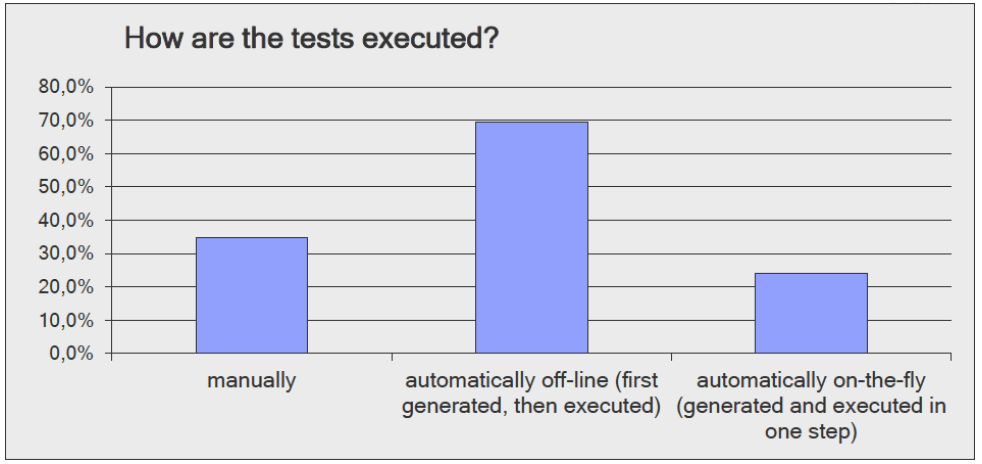
2. Taxonomy of MBT approaches



+ Manual (ISTQB novelty)

Execution

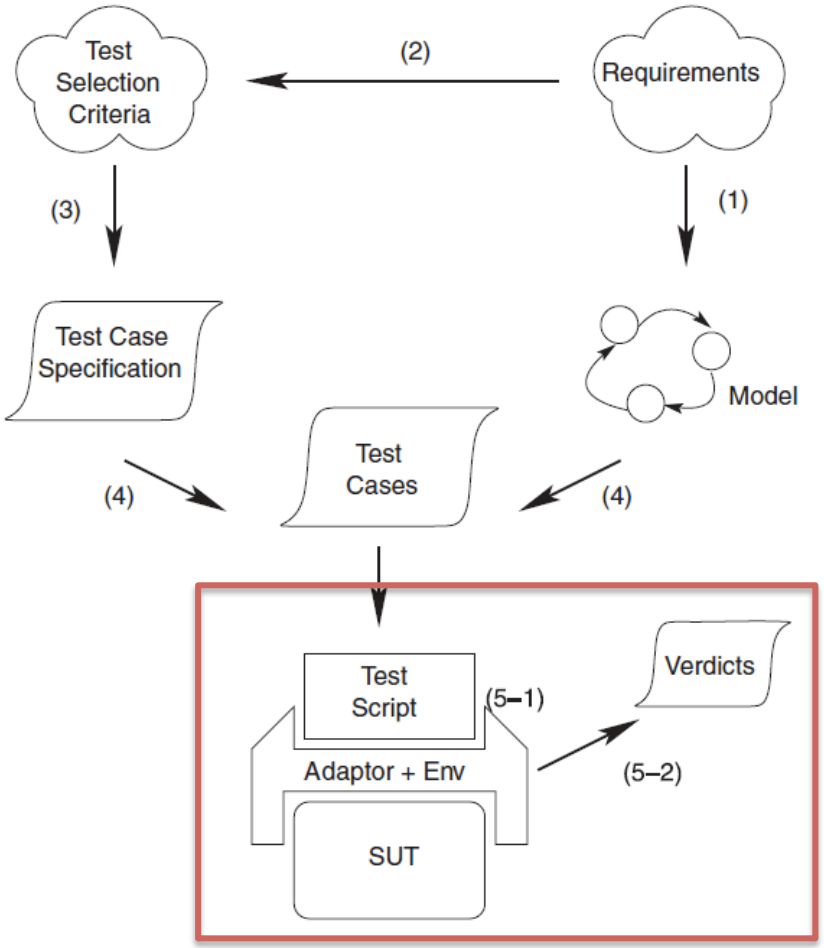
- Manual: test cases are executed manually
- Offline: tests are generated into a test repository for future execution
- Online: each generated test is generated and executed on the SUT simultaneously



Agenda

1. What is Model-Based Testing?
2. Taxonomy of Model-Based Testing approaches
3. **Test execution & conformance relationships**
4. Practical session – MBT of a web application with ModelJUnit
5. Demonstration – MBT with Smartesting CertifyIt
6. Summary: benefits/drawbacks of the MBT approaches

3. Concretization and conformance



(1) Model design

(2) Choice of the appropriate test selection criteria

(3) Transformation of TS criteria into an « operational form » (algorithms)

(4) Test cases generation

(5) Test execution

(5-1) Concretization of test cases

(5-2) Establishment of the test verdict

3. Concretization and conformance

Two issues to consider:

- Bridge the gap between the abstract and concrete level
  - Control: abstract operations + parameters
  - Observations: return values, specific operations

- Implement the conformance relationship and establish the test verdict:
  - « Pass »
  - « Fail »
  - possibly, « Inconclusive »

3. Concretization

- abstract tests → concrete tests

- Control points:
  - Map abstract operations with concrete « actions »
  - Also map parameters list (if necessary adapt it)
  - Translate abstract values into concrete ones (especially enumerations)

- Observations:
  - Return values (to be translated) of the operations
  - Dedicated operations in the model
  - Internal state variables values (if accessible)

  → of the utmost importance: determines the accuracy of the test
  → hopefully, the model provides the test oracle (the expected result)

3. Conformance relationship

Many different conformance relationships: isomorphism, bisimulation, trace equivalence, etc.

Reasonable compromise: **ioco** defined on IOLTS

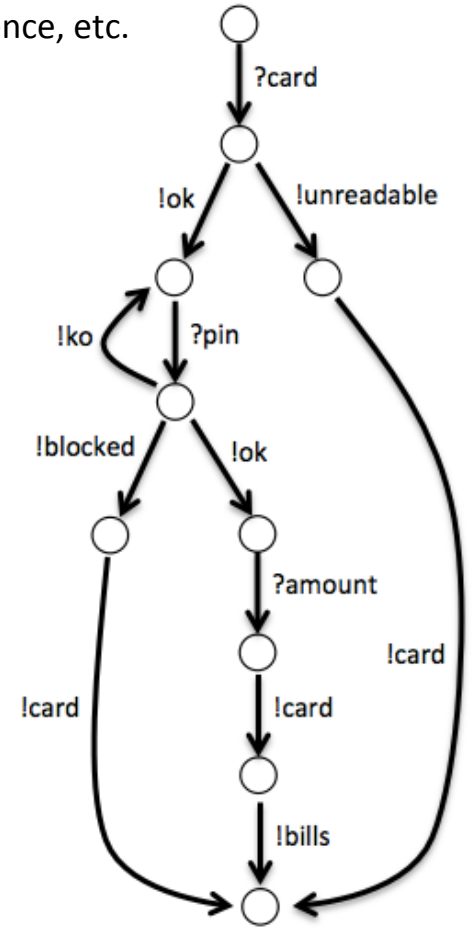
IOLTS = < Q, A, →, q0 >

- Q = set of states
- $A = A_i \cup A_o \cup \{\tau\}$ with
  - A_i = input actions (prefixed by ?)
  - A_o = output actions (prefixed by !)
  - τ = internal action
- $\rightarrow \subseteq Q \times A \times Q$
- q0 = initial state

δ : quiescence (observation of no output) : deadlock/livelock

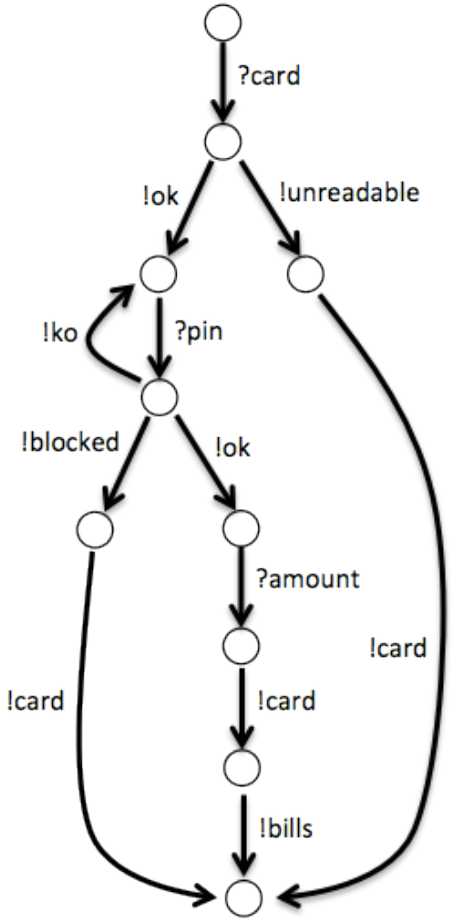
IUT ioco S: $\forall \sigma$: Straces(S) : out(IUT after σ) $\subseteq$ out(S after σ)

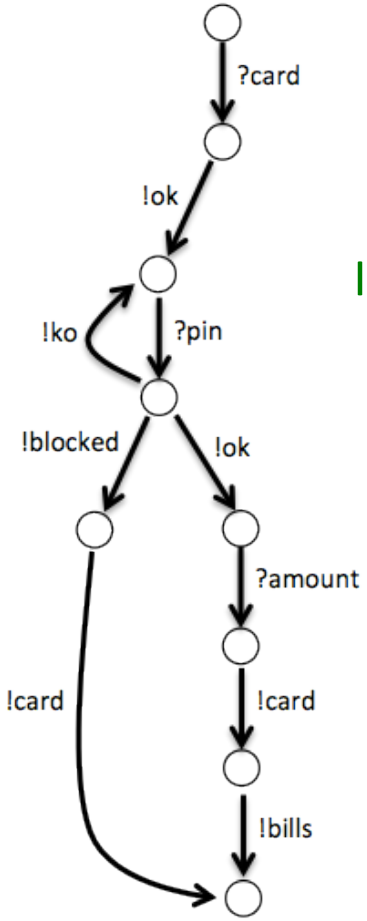
After each suspended trace (ie. an execution up to a quiescence), IUT exhibits only outputs and quiescences present in S.



- J Tretmans. *Test generation with inputs, outputs and repetitive quiescence.* Software---Concepts and Tools, 1996

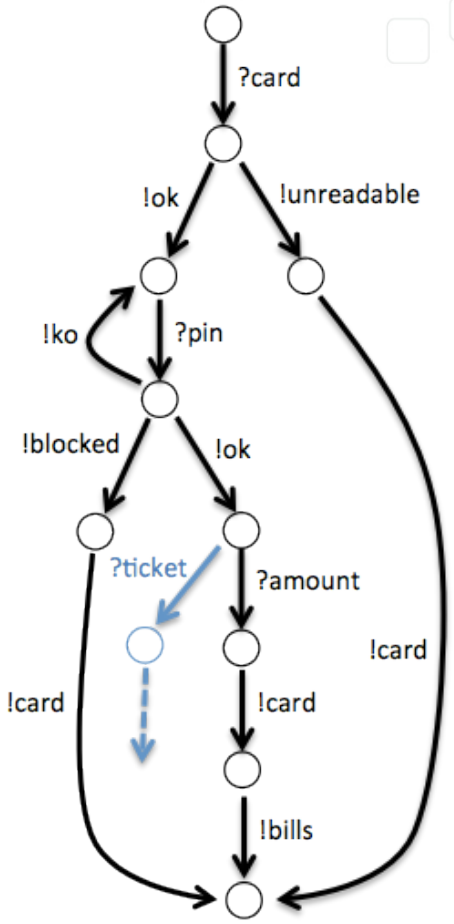
3. Conformance relationship





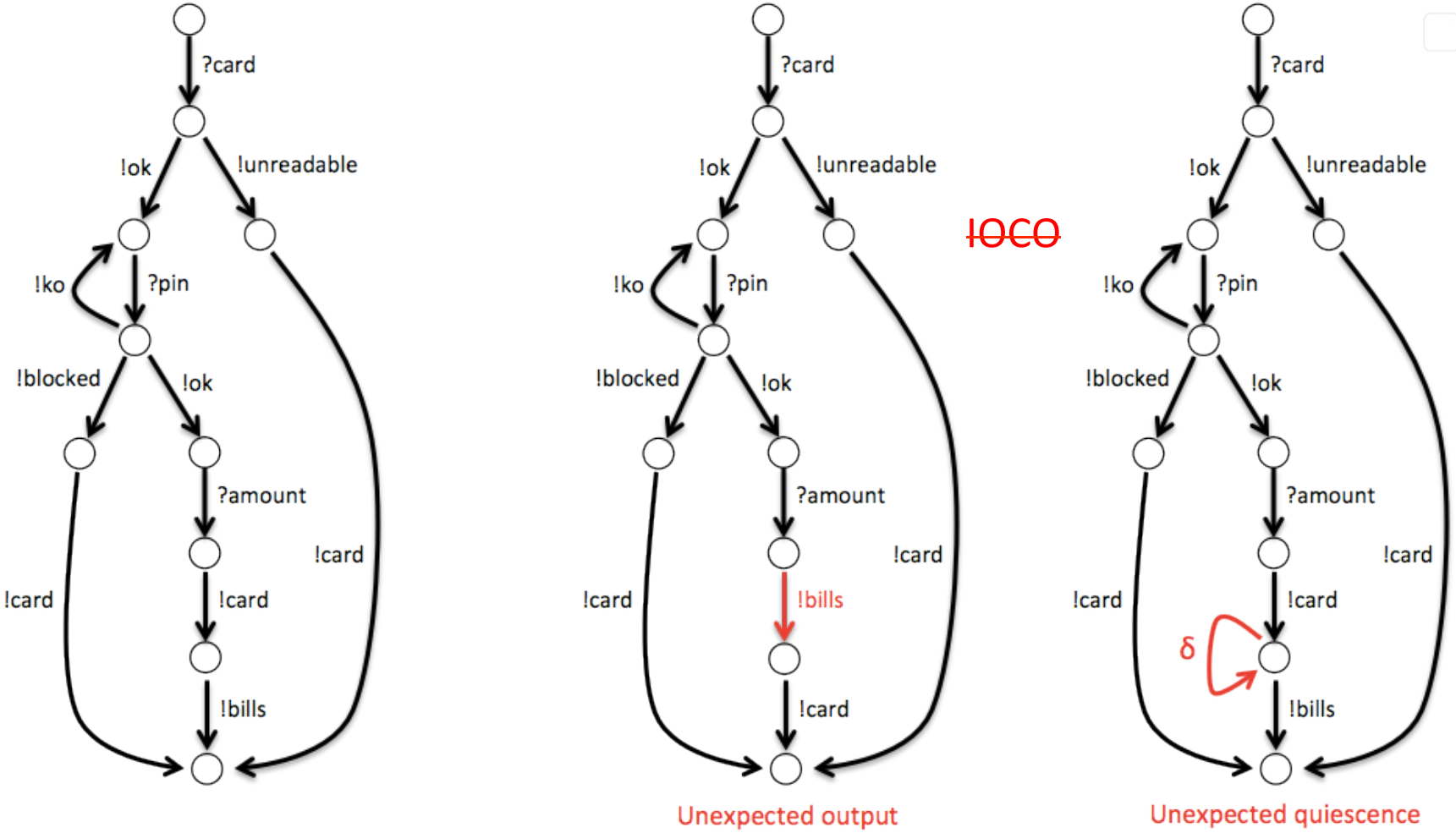
Implementation choice

IOCO



Implementation of partial specification

3. Conformance relationship



Agenda

1. What is Model-Based Testing?
2. Taxonomy of Model-Based Testing approaches
3. Test execution & conformance relationships
4. **Practical session – MBT of a web application with ModelJUnit**
5. Demonstration – MBT with Smartesting CertifyIt
6. Summary: benefits/drawbacks of the MBT approaches

4. MBT with ModelJUnit – ATM example

Automated Teller Machine (ATM) - withdraw cash with a credit card

- System Under Test (SUT) = cash machine
- Test data = credit cards with associated bank accounts
- Control points = reader, pad (0-9 + cancel, delete, validate)
  → abstracted into « actions » (insert card, type PIN, etc.)
- Observation points = messages on the screen, card/bills ejected

- Behaviours = « usual » behaviour of an ATM (functional testing)

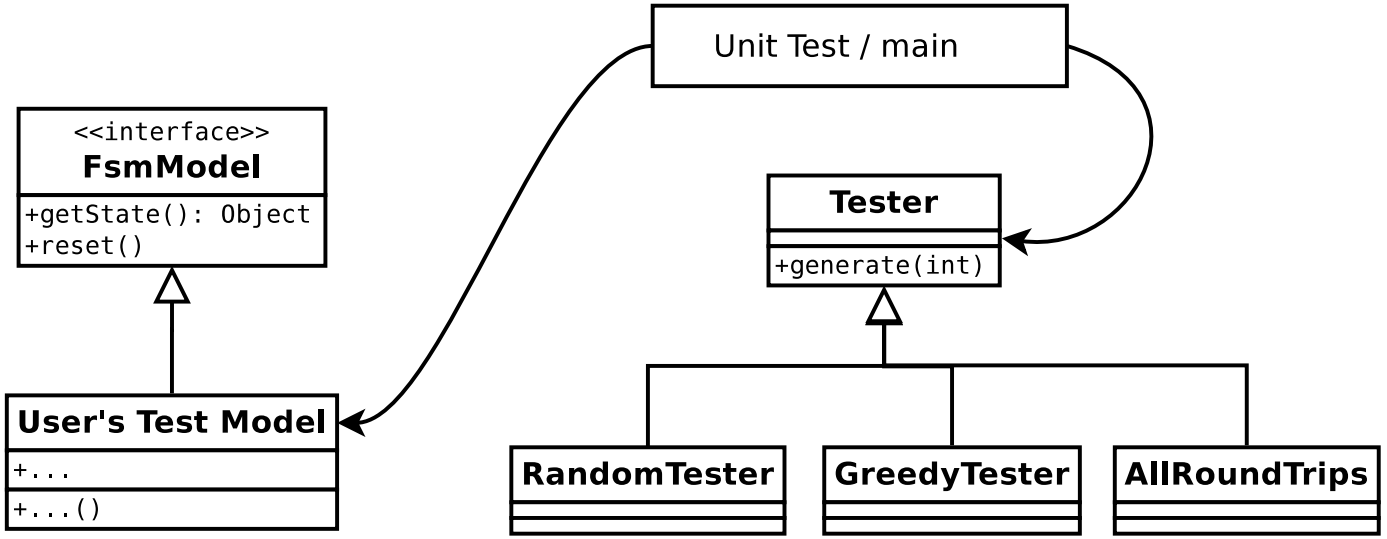
No physical device → simulation on a web application (HTML5+JS)

Visit http://bit.ly/294MsPR for the specification & SUT

4. MBT with ModelJUnit – ATM example

ModelJUnit - https://sourceforge.net/projects/modeljunit/

Library to perform model based tests, by defining a Java class of a user model.



4. MBT with ModelJUnit – Summary

Step 1 – simple FSM, abstraction of money withdrawal
- structural model coverage (states, transitions, actions)
- various algorithms: model-checking, (biaised) random

Step 2 – connection to SUT
- online testing (each step is executed on the SUT)
- abstracted delays reintroduced in the concretization layer

Step 3 – adding observations (screen messages)
- possible errors in observations

Step 4 – extended FSM with variables for wallet & balance

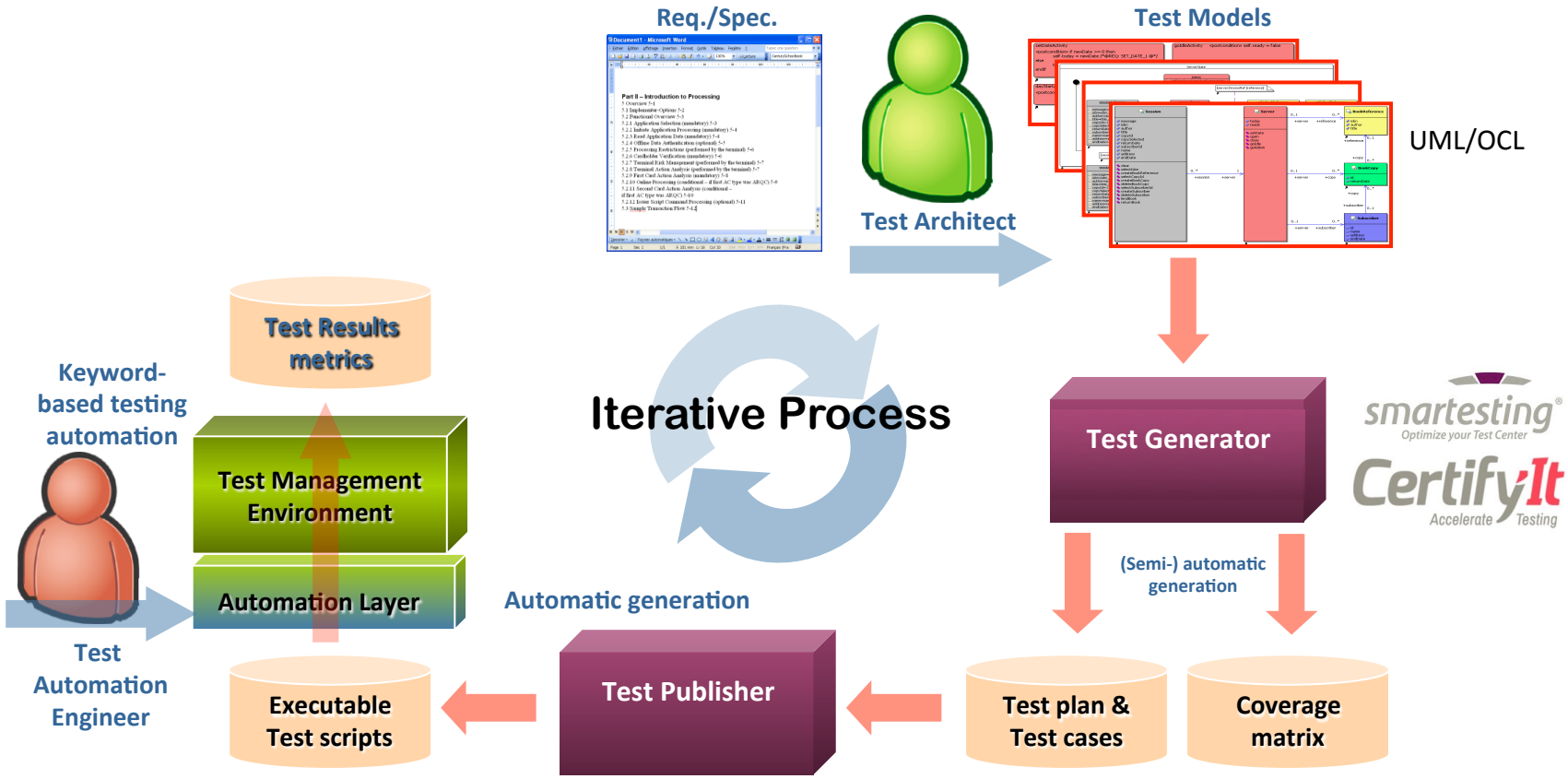
Step 5 – fine-grained observations
- non-conformance w.r.t. the specification

Variant – non-deterministic model and adaptative testing

Agenda

1. What is Model-Based Testing?
2. Taxonomy of Model-Based Testing approaches
3. Test execution & conformance relationships
4. Practical session – MBT of a web application with ModelJUnit
5. **Demonstration – MBT with Smartesting CertifyIt**
6. Summary: benefits/drawbacks of the MBT approaches

5. Model-Based Testing with CertifyIt



5.1. Models: UML/OCL

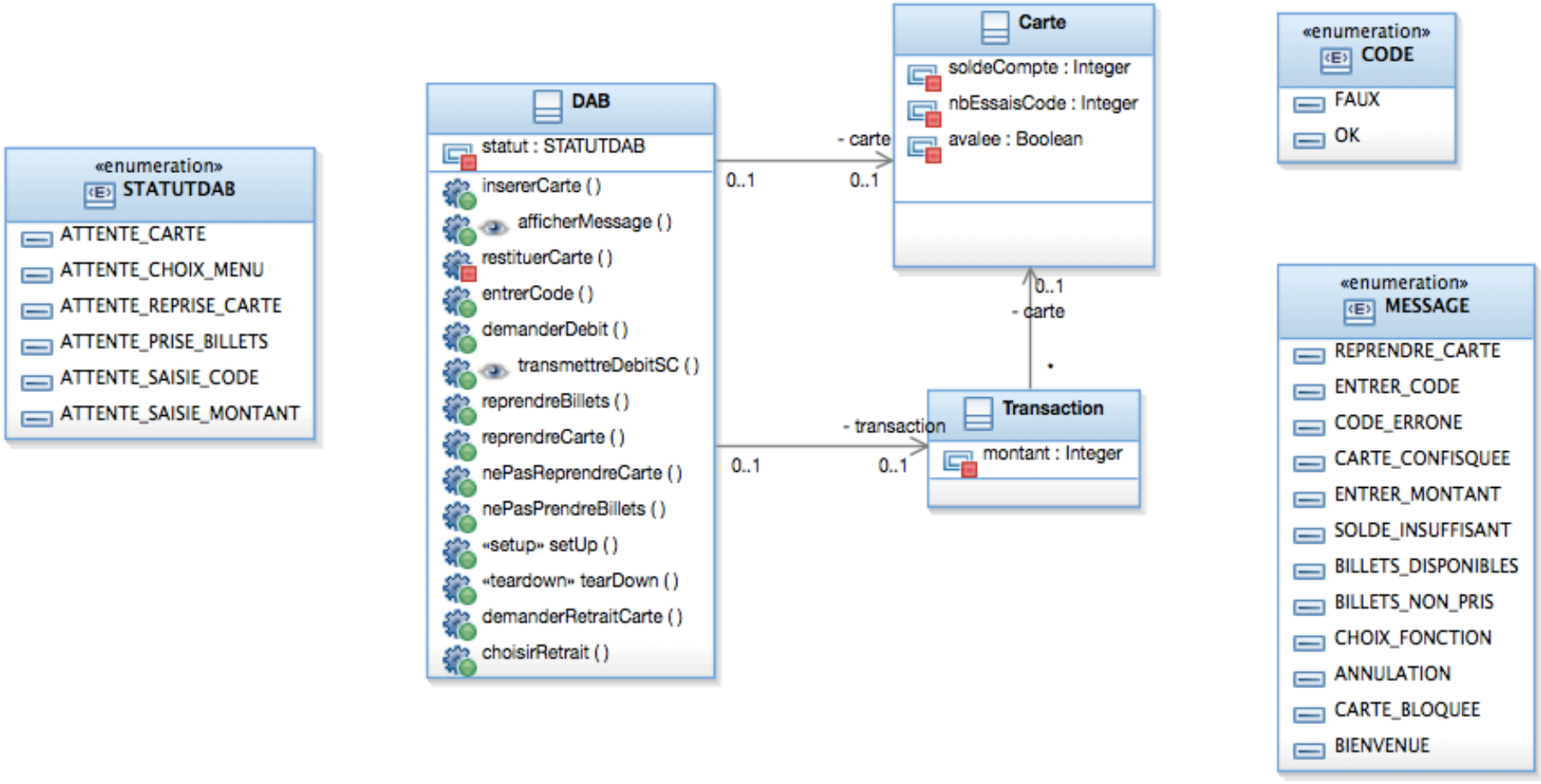
- Unified Modeling Language + Object Constraint Language
  - use of a subset of UML, called UML4ST : no inheritance, binary associations, no dynamic creation of instances
  - adaptation of the usual semantics of OCL as an action language (to make OCL executable)

  → test model ≠ design model

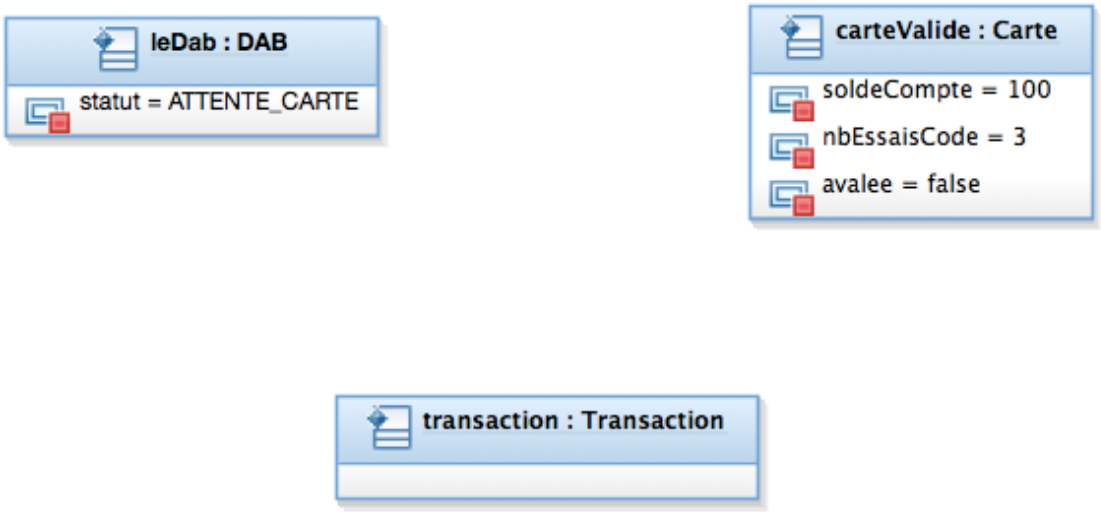
- Three kinds of UML diagrams are considered:
  - class diagram → data model,
  - object diagram → initial state, and
  - statecharts → dynamics – not considered here, for simplicity

- Instead, OCL code is used to describe the behaviour of the operations

Running example: ATM – Class diagram



Running example: ATM – Object diagram



Running example: ATM – OCL

- Precondition of the entrerCode(PIN) operation

```
not carte.oclIsUndefined() and statut=STATUTDAB::ATTENTE_SAISIE_CODE
```

- Postcondition of the entrerCode(PIN) operation

```
if (p_Code=CODE::FAUX)then
    ---@REQ:COUNTER_DECREASED
    carte.nbEssaisCode = carte.nbEssaisCode-1 and
    afficherMessage(MESSAGE::CODE_ERRONE) and
    if (carte.nbEssaisCode <= 0) then
        ---@REQ:CARD_BLOCKED
        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
        restituerCarte()
    else
        ---@REQ:CARD_NOT_BLOCKED
        afficherMessage(MESSAGE::ENTRER_CODE)
    endif
else
    ---@REQ:OK
    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
    carte.nbEssaisCode = 3 and
    afficherMessage(MESSAGE::ENTRER_MONTANT)

endif
```

5. Test selection criteria & test generation

- Static test selection criteria
  - Structural coverage of the OCL code
  - Requirement coverage

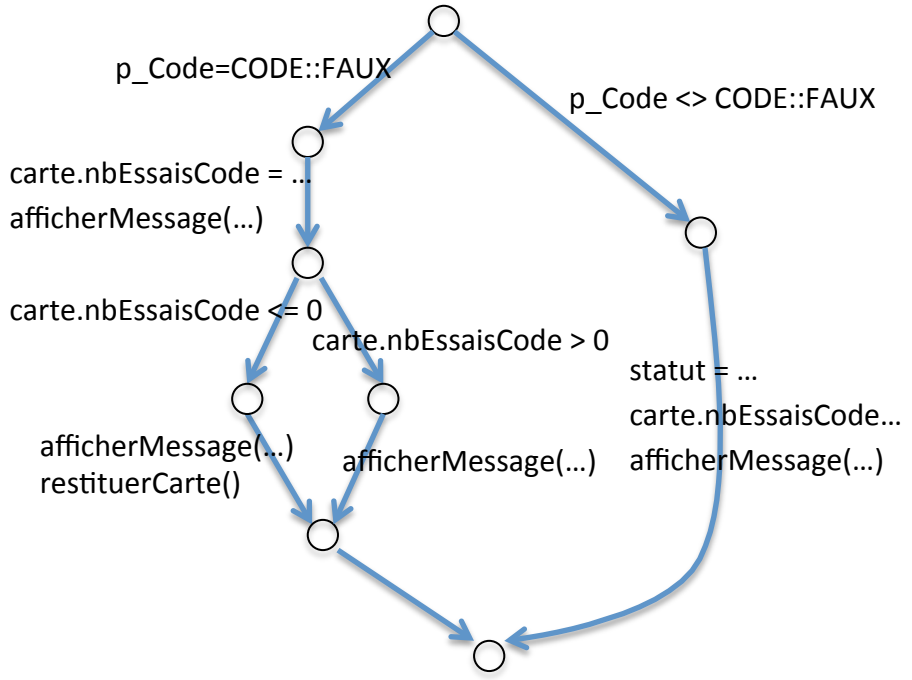
- Dynamic test selection criteria
  - Test purposes (abstract test scenarios)
  - Temporal properties

Static test selection criteria

- Goal: activate each behaviour of each operation of the SUT
  - Behaviour = branch in the CFG of the operation
  - Test Target = state that makes the execution of the behavior possible

```
if (p_Code=CODE::FAUX)then
    ---@REQ:COUNTER_DECREASED
    carte.nbEssaisCode = carte.nbEssaisCode-1 and
    afficherMessage(MESSAGE::CODE_ERRONE) and
    if (carte.nbEssaisCode <= 0) then
        ---@REQ:CARD_BLOCKED
        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
        restituerCarte()
    else
        ---@REQ:CARD_NOT_BLOCKED
        afficherMessage(MESSAGE::ENTRER_CODE)
    endif
else
    ---@REQ:OK
    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
    carte.nbEssaisCode = 3 and
    afficherMessage(MESSAGE::ENTRER_MONTANT)

endif
```



Static test selection criteria

- For each test target, automatically explore the model states and compute a sequence of operations that reaches the target

- Shape of a test case:
  - <preamble> = sequence of operations, from the initial state that reaches the target
  - <body> = invocation of the operation to active the targeted behavior
  - <observation> = possible additional operations that can be executed to check that the targeted operation was correctly executed

- Test cases can be merged to minimize the size of the test suite

Static test selection criteria

Examples of functional tests:

- leDab.insererCarte(carteValide) // @REQ: OK
  leDab.choisirRetrait() // @REQ: OK
  leDab.entrerCode(OK) // @REQ: OK
  leDab.demanderDebit(50) // @REQ: OK
  leDab.reprendreCarte() // @REQ: OK, @REQ: TRANSACTION_DONE
  leDab.reprendreBillets() // @REQ: OK

- leDab.insererCarte(carteValide) //@REQ: OK
  leDab.choisirRetrait() // @REQ: OK
  leDab.entrerCode(OK) // @REQ: OK
  leDab.demanderDebit(150) // @REQ: INSUFFICIENT_BALANCE

Static test selection criteria - limitations

- Limitations of automated testing based on static criteria (structural/ requirement coverage)
  - test cases with limited size (steps)
  - difficulty to take into account the dynamics of the system (must be hard-coded into the model)
  - possible issues with the test target's reachability

- Two complementary ways to drive the test generation:
  - test scenarios
  - temporal test properties

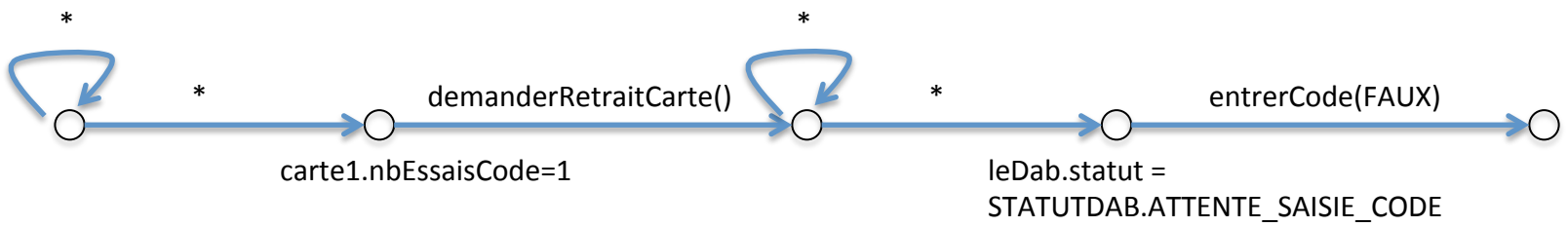
Dynamic criteria: test purposes

- Test scenarios that help the user describing test sequences that cannot be computed by the tool
- Based on regular expressions involving operations and state predicates
- However, textual description, close to natural language (to help the test designer)
- Unfolded on the model to be instantiated as a test case

Test purposes

Example on the ATM: a test scenario that checks that pin retry counter is correctly implemented

use any_operation any_number_of_times
to_reach "*nbEssaisCode = 1*" on_instance *carteValide*
then use leDab.demanderRetraitCarte()
then use any_operation any_number_of_times
to_reach "statut=STATUTDAB::ATTENTE_SAISIE_CODE" on_instance leDab
then use leDab.entrerCode(FAUX) // should block and eject the card



Test purposes

Example on the ATM: a test scenario that checks that pin retry counter is correctly implemented

Once unfolded on the model:

leDab.insererCarte(carteValide)

leDab.choisirRetrait()

leDab.entrerCode(FAUX)

leDab.entrerCode(FAUX)

leDab.demanderRetraitCarte()

leDab.reprendreCarte()

leDab.insererCarte(carteValide)

leDab.choisirRetrait()

leDab.entrerCode(FAUX)

Dynamic criteria: test properties

- Some test scenarios address specific test intentions, that could be formalized by high-level properties

- TOCL = Temporal OCL
  - overlay of OCL to express temporal properties
  - based on Dwyer *et al.* property patterns [DAC99]
  - does not require the use of a complex formalism (e.g. LTL, CTL)

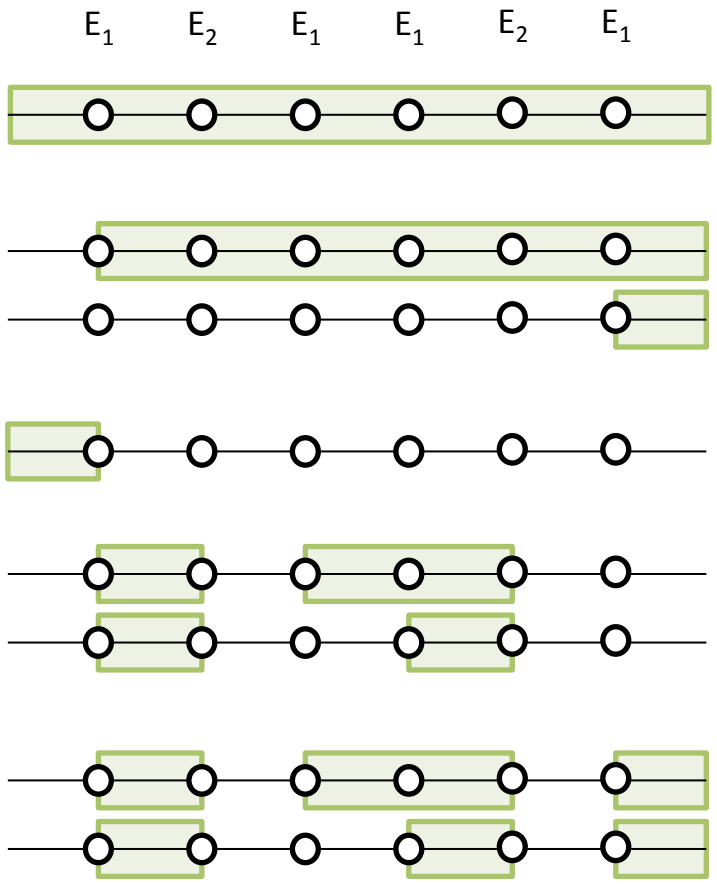
- Property = Pattern + Scope
  - Pattern: describes occurrences or orderings of events
  - Scope: describes the observation window on which the pattern is supposed to hold

[DAC99] M. Dwyer, G. Avrunin, and J. Corbett. *Patterns in property specifications for finite-state verification*. ICSE'99.

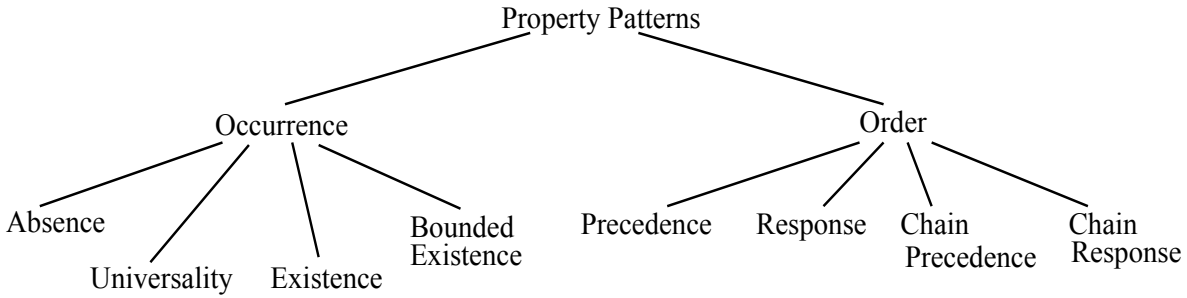
Temporal Properties in TOCL

Scopes

- globally
- after E_1
  - after last E_1
- before E_1
- between E_1 and E_2
  - between last E_1 and E_2
- after E_1 until E_2
  - after last E_1 until E_2



Temporal Properties in TOCL

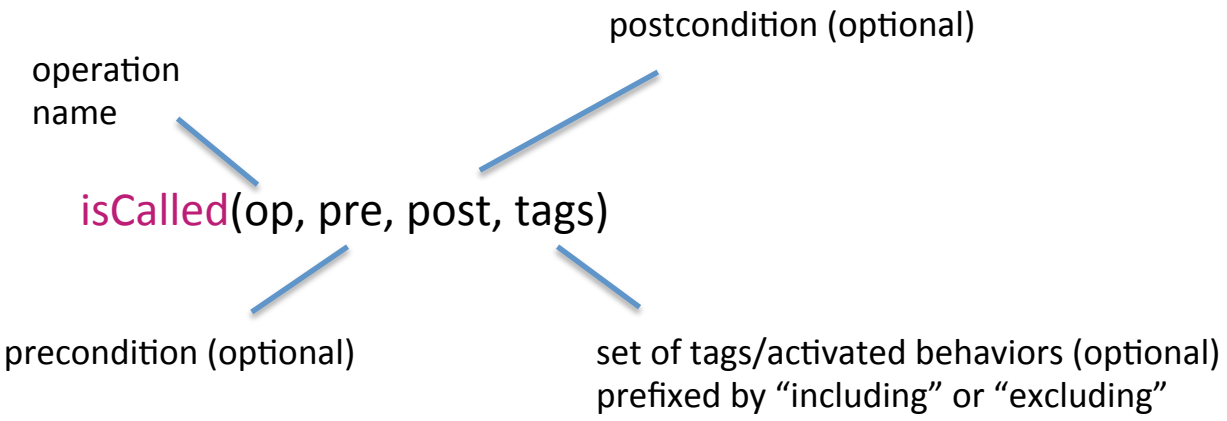


Patterns

- always P
- never E
- eventually E at least/at most/exactly k times
- E_1 [directly] precedes E_2
- E_1 [directly] follows E_2

Temporal Properties in TOCL

Events: operation calls



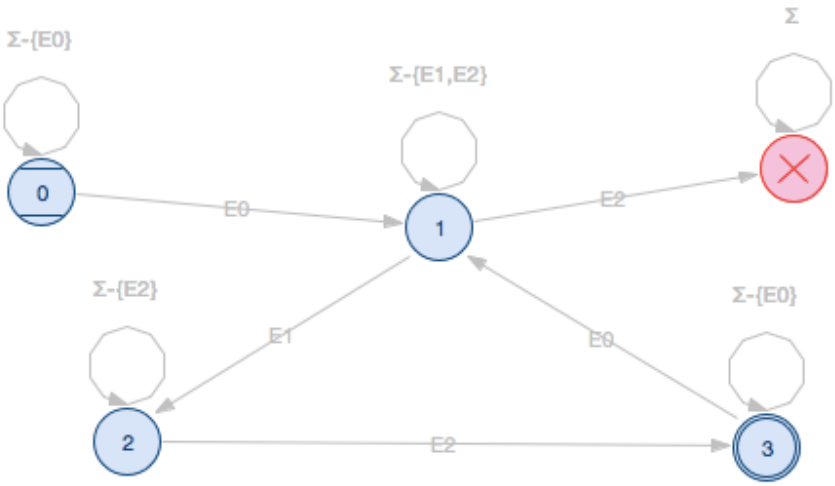
Temporal Properties in TOCL

"Once a card is inserted, it is necessary to authenticate to get bills."

between isCalled(leDab.insererCarte,including:{@REQ:OK})
and isCalled(leDab.reprendreBillets,including:{@REQ:OK})

eventually isCalled(leDab.entrerCode,including:{@REQ:OK})
    at least 1 times

E0 = insererCarte
E1 = entrerCode
E2 = reprendreBillets

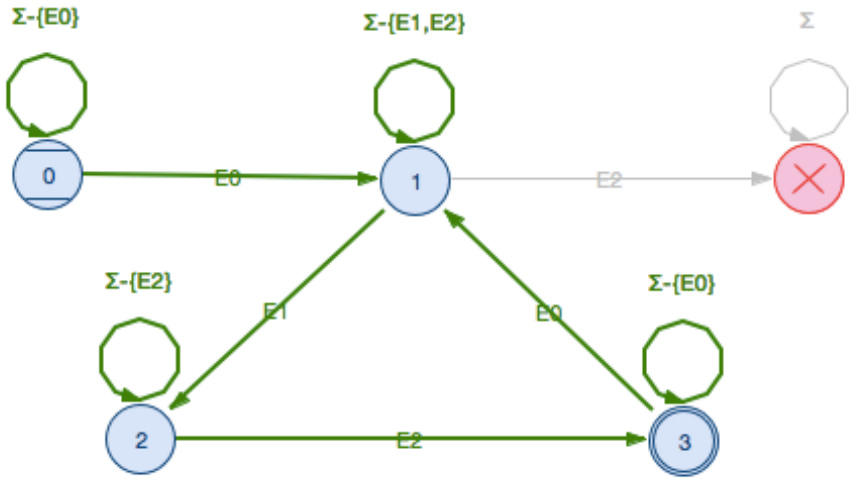


Using the properties for testing

- Two possible uses for these properties

1. Measure the quality of a test suite

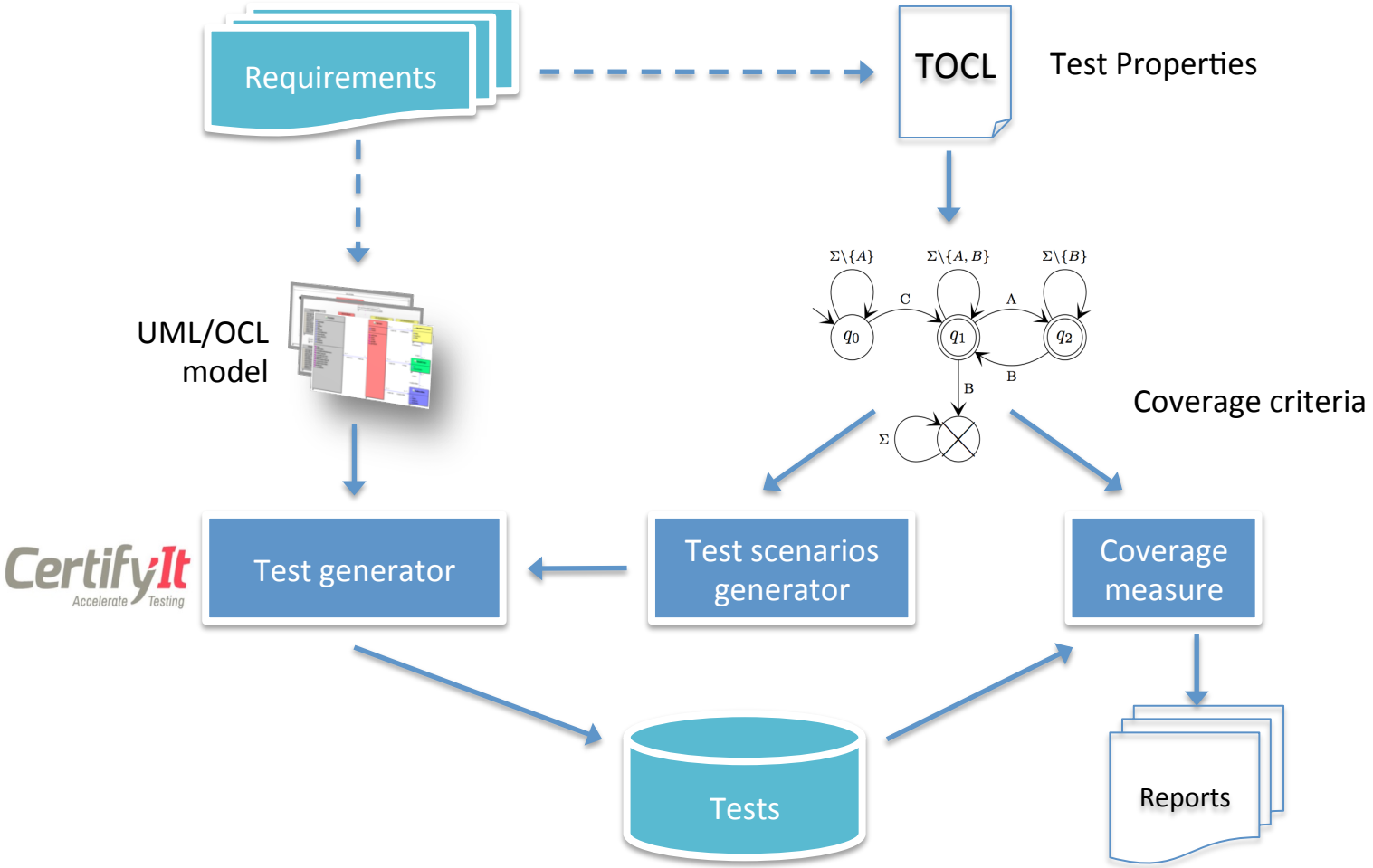
leDab.insererCarte(carteValide)	0 → 1
leDab.choisirRetrait()	1 → 1
leDab.entrerCode(OK)	1 → 2
leDab.demanderDebit(50)	2 → 2
leDab.reprendreCarte()	2 → 2
leDab.reprendreBillets()	2 → 3



2. Generate new tests

use any_operation any_number_of_times then
use leDab.insererCarteValide to_activate {@REQ:OK} then
use any_operation any_number_of_times then
use entrerCode() to_activate {@REQ:OK} then
use any_operation any_number_of_times then
use reprendreBillets() to_activate {@REQ:OK} … x2

Use of temporal test properties



Interest of test properties

- Language is easy to learn and use to design test properties
- Usefulness of the coverage reports
  - shows which part of the properties are not covered by the tests
- Relevance of the coverage criteria
  - Property automata are rarely 100% covered by the functional test suite
  - “Shows test configurations that one may not easily think of”
- Unintended use of the properties: model validation
  - Use of the test cases coverage measure to detect violations of the property by the model

Agenda

1. What is Model-Based Testing?
2. Taxonomy of Model-Based Testing approaches
3. Test execution & conformance relationships
4. Practical session – MBT of a web application with ModelJUnit
5. Demonstration – MBT with Smartesting CertifyIt
6. **Summary: benefits/drawbacks of the MBT approaches**

6. Summary: benefits and drawbacks

Benefits:

- back-to-back validation of a system: a comparison of two point of views
- functional testing: does not aim at runtime errors (null pointers, divisions by 0, etc.) but focus on specification mistakes (40% of the errors in a program)
- look for automation

Drawbacks:

- Model design step:
  - learning curve to take into account (language)
  - keep in mind you design a test model, not a design model
- Test generator: need to know how it works to produce the right tests
- Test verdict:
  - implement the conformance relationship you want (ioco might not be sufficient!)
  - in case of non-conformance: where is the error?

Our advise: perform MBT iteratively and incrementally

6. MBT: where to go?

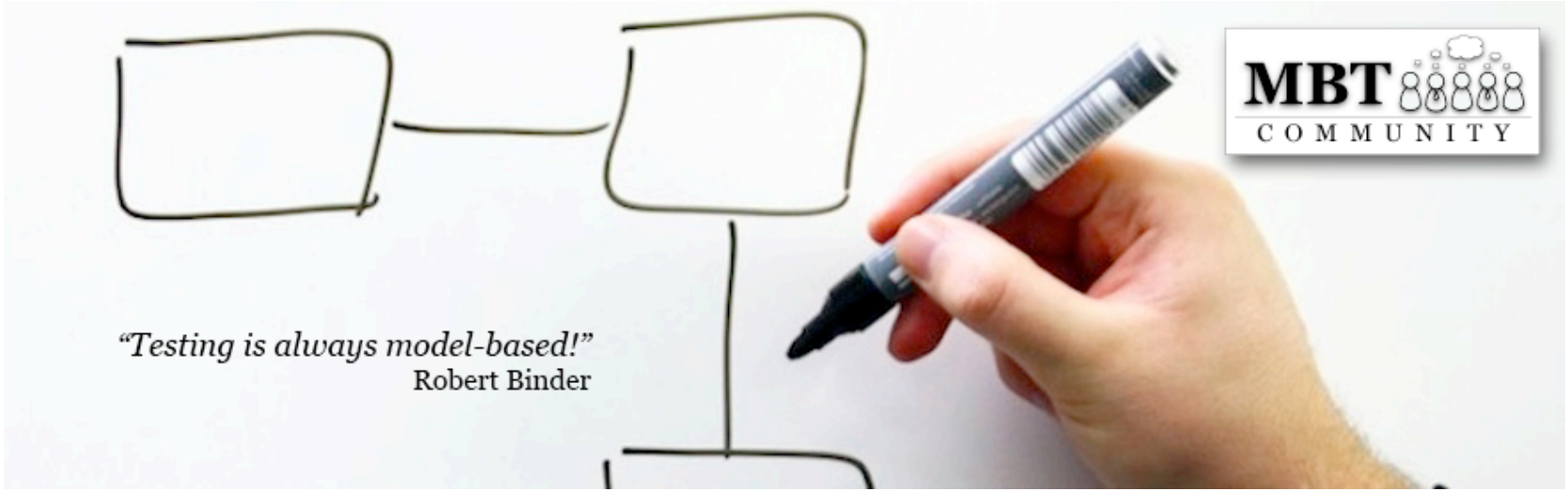
Journals
- TSE: Transactions on Software Engineering
- STVR: Software Testing Verification and Reliability
- STTT: Software Tools for Technology Transfer
- JSS: Journal of Software and Systems
- SoSyM: Software and Systems Modeling
- …

Conferences
- ICST: Int. Conf. Software Testing Verification and Validation
- ICSE: Int. Conf. on Software Engineering
- ASE: Automated Software Engineering
- ISSRE: Int. Symposium on Software Reliability and Engineering
- ICFEM: Int. Conf. on Formal Engineering Methods
- TAP: Tests and Proofs
- …

Workshops
- AMOST: Advances in Model Based Testing (co-located with ICST)
- MBT: Model-Based Testing (co-located with ETAPS)

Meet the French community:
- Conférence AFADL
- Journées du GDR GPL
- Groupe de travail MTV2 (méthodes de test pour la vérification et la validation)

Questions?



www.model-based-testing.info

Some (general) references on MBT

- Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing. STVR 22:5, 2012. [previous version appeared as a tech report of The University of Waikato (April 2006)] Useful surveys in the field of model-based testing: M. Shafique, Y. Labiche. A systematic review of state-based test tools. STTT, 2013. [previous version as tech report of Carleton University, SCE-10-04, May 2010.]

- G. Fraser, F. Wotawa, and P.E. Ammann, "Testing with model checkers: a survey," Software Testing, Verification and Reliability, vol. 19, 2009, pp. 215-261. [Draft version availiable]

- Dias Neto, A. C., Subramanyan, R., Vieira, M., and Travassos, G. H. "A survey on model-based testing approaches: a systematic review." In Proc. of WEASELTech '07. ACM, New York, NY, 31-36. [detailed technical report]

- MOGENTES Consortium: State of the Art Survey - Part a: Model-based Test Case Generation, Deliverable D1.2. (2008)

- Broy, M., Jonsson, B., Katoen, J., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science). Springer-Verlag New York, Inc. (2005)

- Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann (2007) ISBN 978-0-12-372501-1.

Some of our references DISC/Smartesting

- Julien Botella, Jürgen Grossmann, Bruno Legeard, Fabien Peureux, Martin Schneider, and Fredrik Seehusen. Model-Based Security Testing with Test Patterns. In UCAAT 2014, 2nd User Conference on Advanced Automated Testing, Munich, Germany, September 2014. ETSI.

- Frédéric Dadeau, Kalou Cabrera Castillos, and Jacques Julliand. Coverage Criteria for Model-Based Testing using Property Patterns. In A.K. Petrenko and H. Schlingloff, editors, MBT 2014, 9th Workshop on Model-Based Testing, Satellite workshop of ETAPS 2014, volume 141 of EPTCS, Grenoble, France, pages 29--43, April 2014. Open Publishing Association.

- Julien Botella, Fabrice Bouquet, Jean-François Capuron, Franck Lebeau, Bruno Legeard, and Florence Schadle. Model-Based Testing of Cryptographic Components -- Lessons Learned from Experience. In ICST'13, 6th IEEE Int. Conf. on Software Testing, Verification and Validation, pages 192--201, March 2013.

- Kalou Cabrera Castillos, Frédéric Dadeau, and Jacques Julliand. Scenario-Based Testing from UML/OCL Behavioral Models -- Application to POSIX Compliance. STTT, International Journal on Software Tools for Technology Transfer, 13(5):431--448, 2011. Note: Special Issue on Verified Software: Tools, Theory and Experiments (VSTTE'09)

- Fabrice Bouquet, Christophe Grandpierre, Bruno Legeard, and Fabien Peureux. A test generation solution to automate software testing. In AST'08, 3rd Int. workshop on Automation of Software Test, Leipzig, Germany, pages 45--48, May 2008. ACM Press.