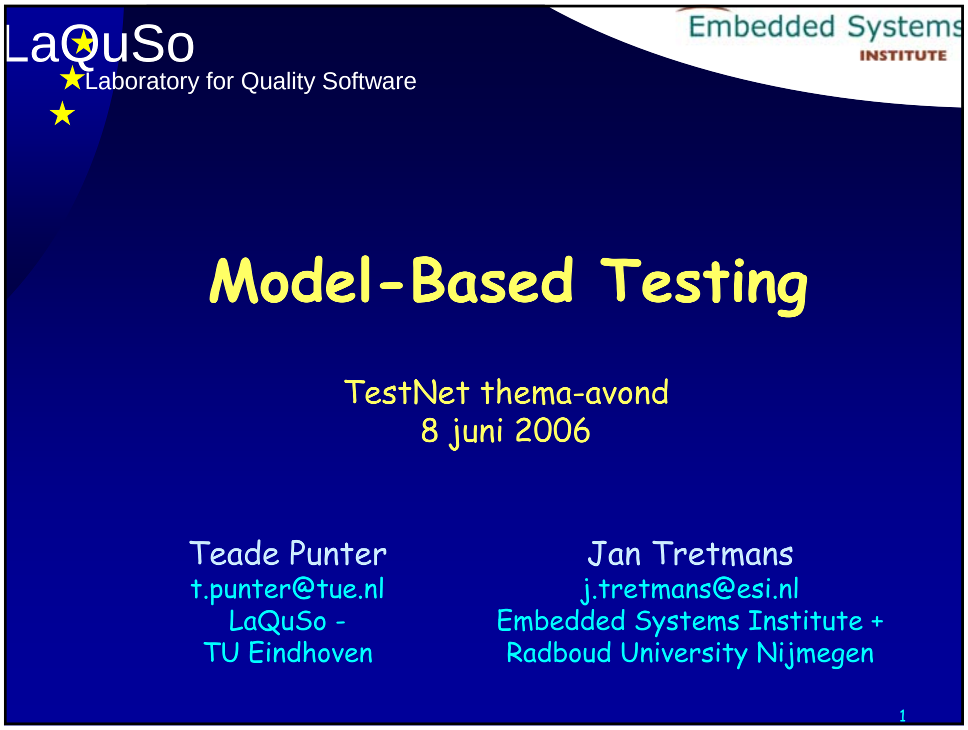


LaQuSo
Laboratory for Quality Software

Embedded Systems
INSTITUTE

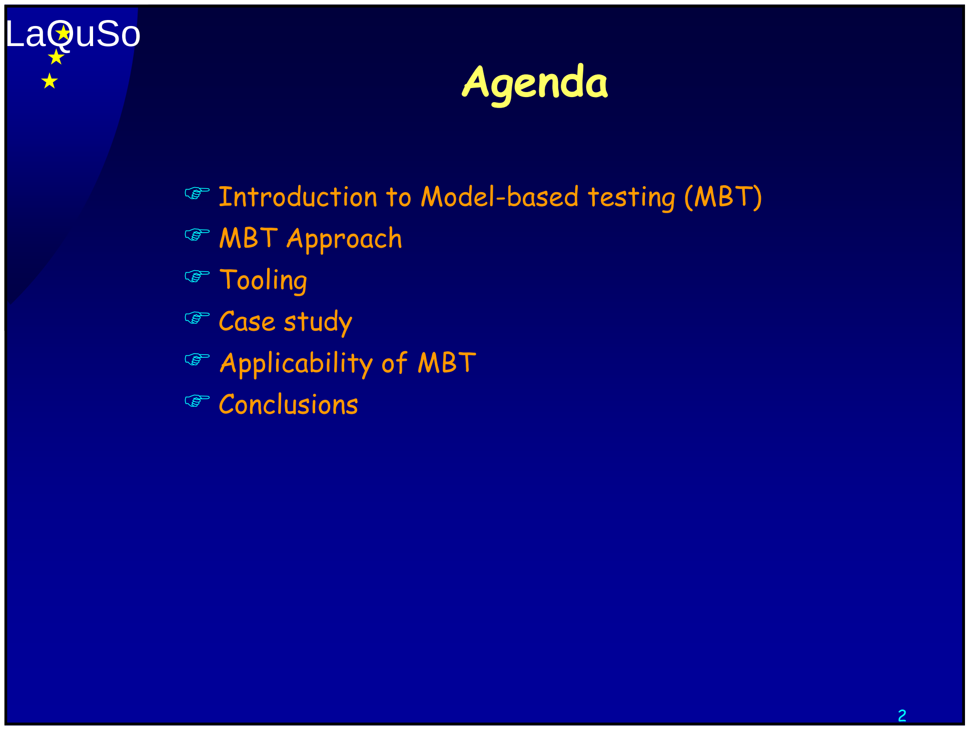
Model-Based Testing

TestNet thema-avond
8 juni 2006

Teade Punter
t.punter@tue.nl
LaQuSo -
TU Eindhoven

Jan Tretmans
j.tretmans@esi.nl
Embedded Systems Institute +
Radboud University Nijmegen

1

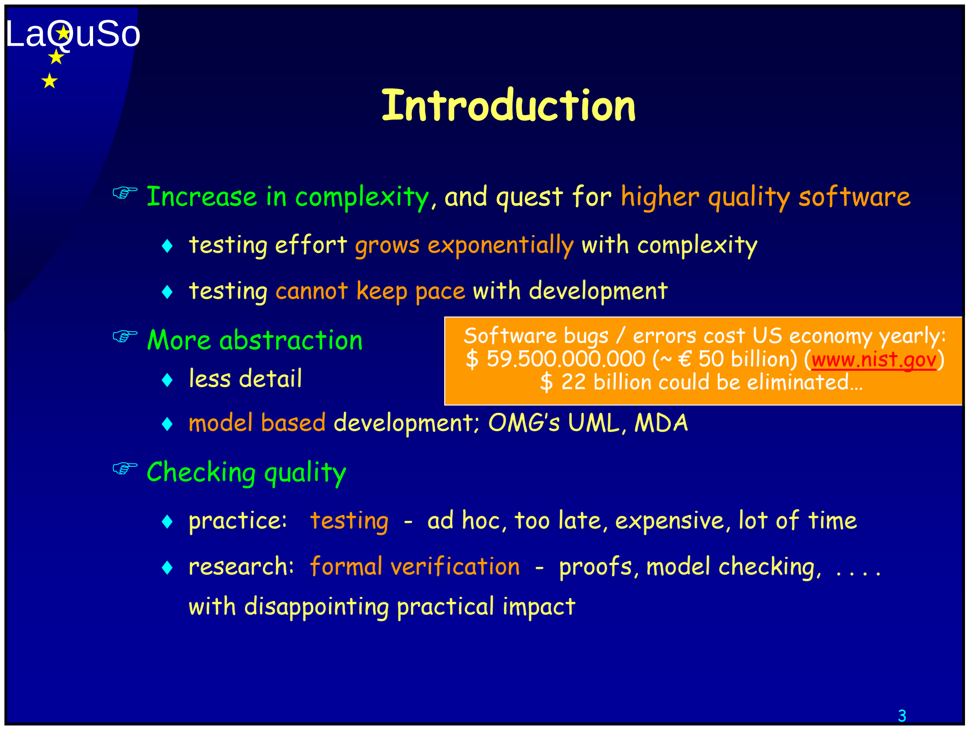


LaQuSo

Agenda

- Introduction to Model-based testing (MBT)
- MBT Approach
- Tooling
- Case study
- Applicability of MBT
- Conclusions

2



LaQuSo

Introduction

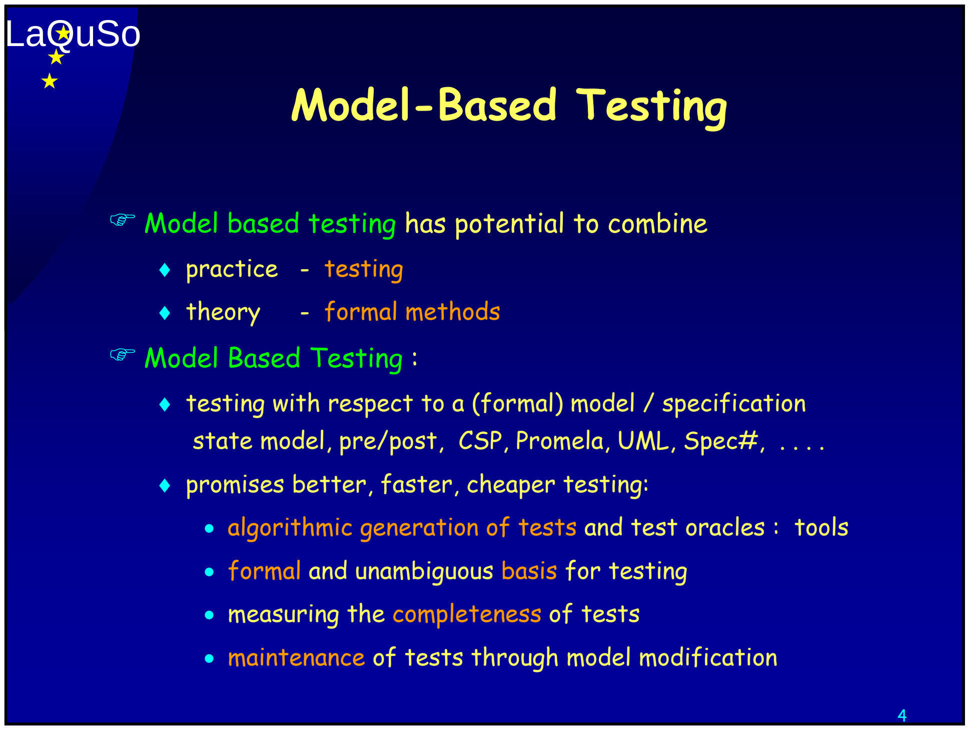
☞ Increase in complexity, and quest for higher quality software
- testing effort grows exponentially with complexity
- testing cannot keep pace with development

☞ More abstraction
- less detail
- model based development; OMG's UML, MDA

Software bugs / errors cost US economy yearly:
$ 59.500.000.000 (~ € 50 billion) (www.nist.gov)
$ 22 billion could be eliminated…

☞ Checking quality
- practice: testing - ad hoc, too late, expensive, lot of time
- research: formal verification - proofs, model checking, ...
  with disappointing practical impact

3



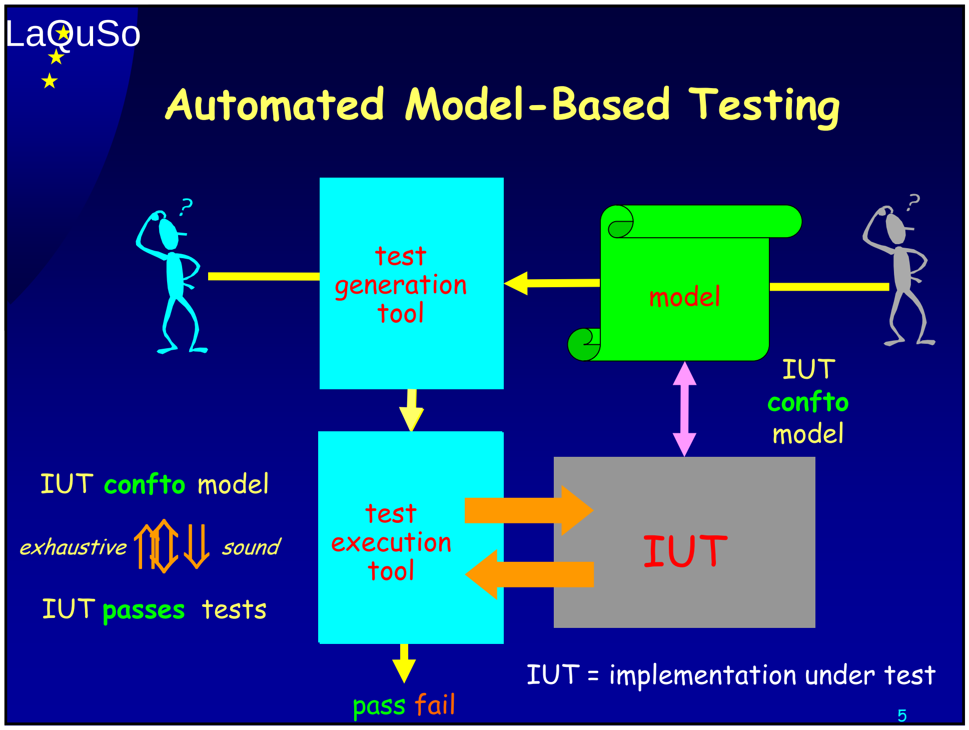
LaQuSo

Model-Based Testing

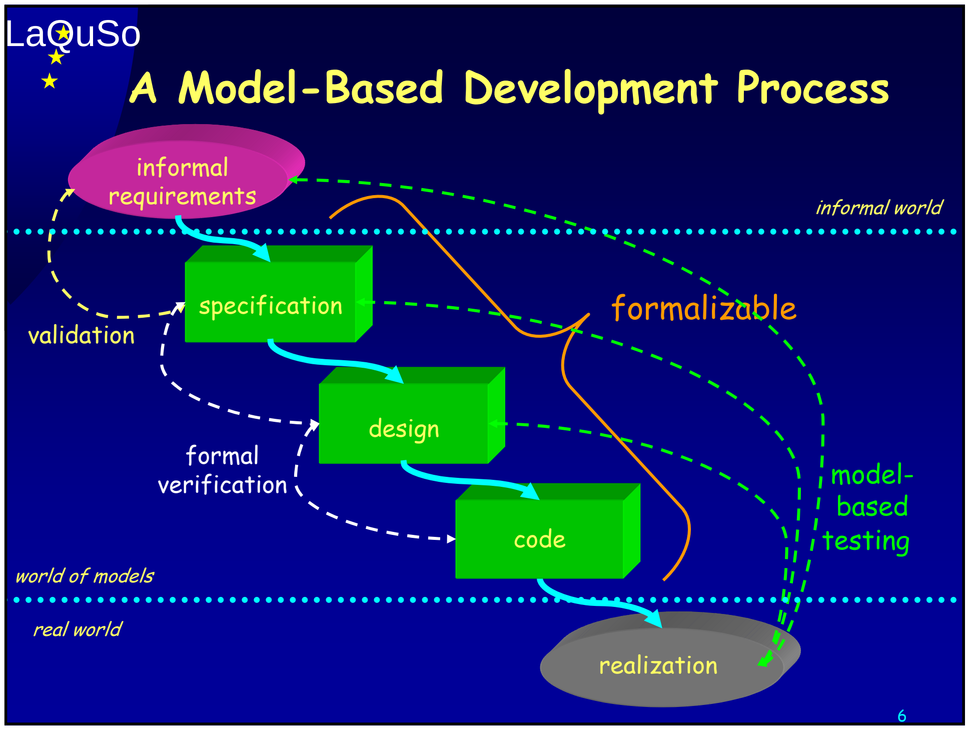
☞ Model based testing has potential to combine
- practice - testing
- theory - formal methods

☞ Model Based Testing :
- testing with respect to a (formal) model / specification
  state model, pre/post, CSP, Promela, UML, Spec#, ...
- promises better, faster, cheaper testing:
  - algorithmic generation of tests and test oracles : tools
  - formal and unambiguous basis for testing
  - measuring the completeness of tests
  - maintenance of tests through model modification

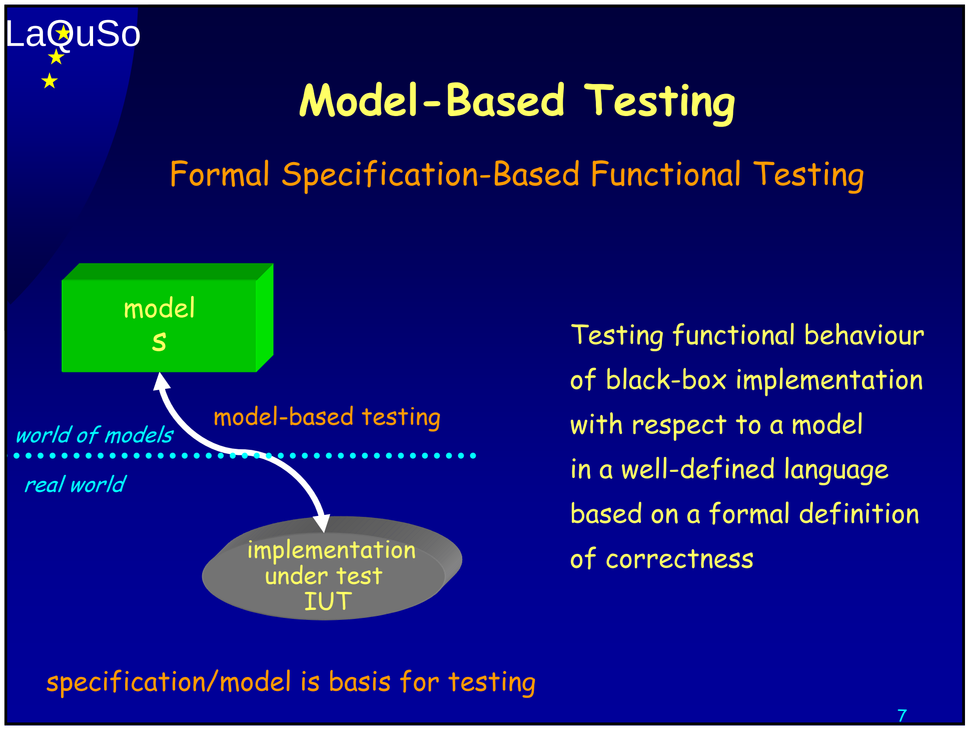
4



Automated Model-Based Testing



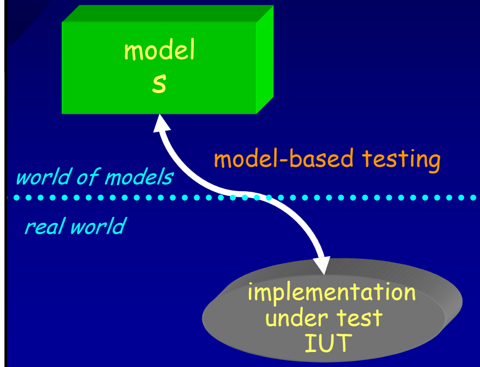
A Model-Based Development Process



LaQuSo

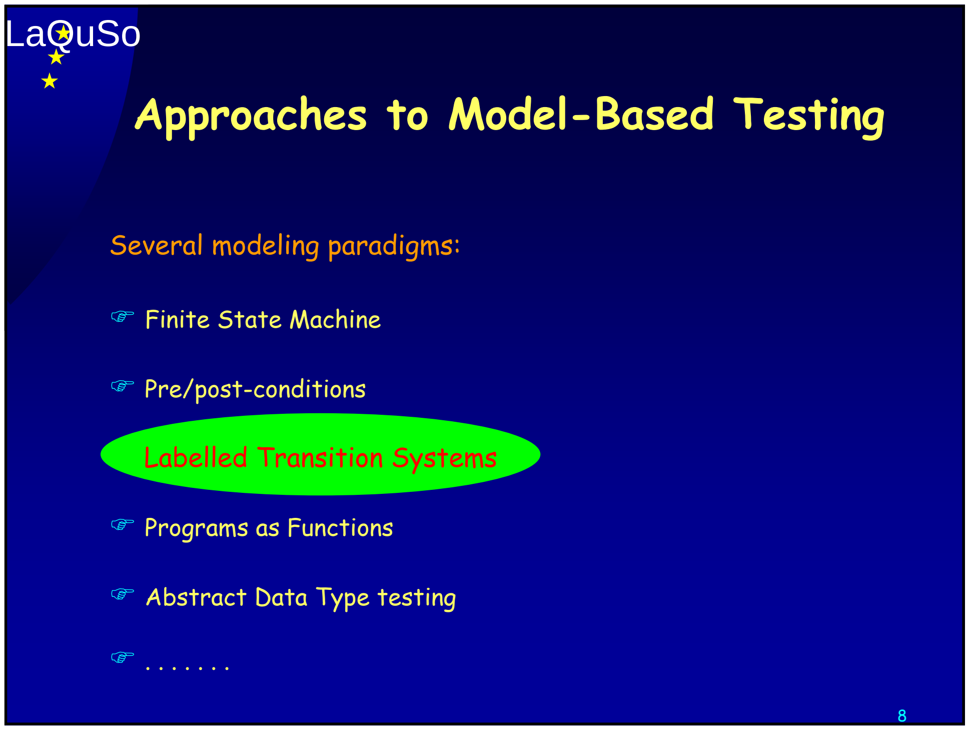
Model-Based Testing

Formal Specification-Based Functional Testing



Testing functional behaviour of black-box implementation with respect to a model in a well-defined language based on a formal definition of correctness

specification/model is basis for testing

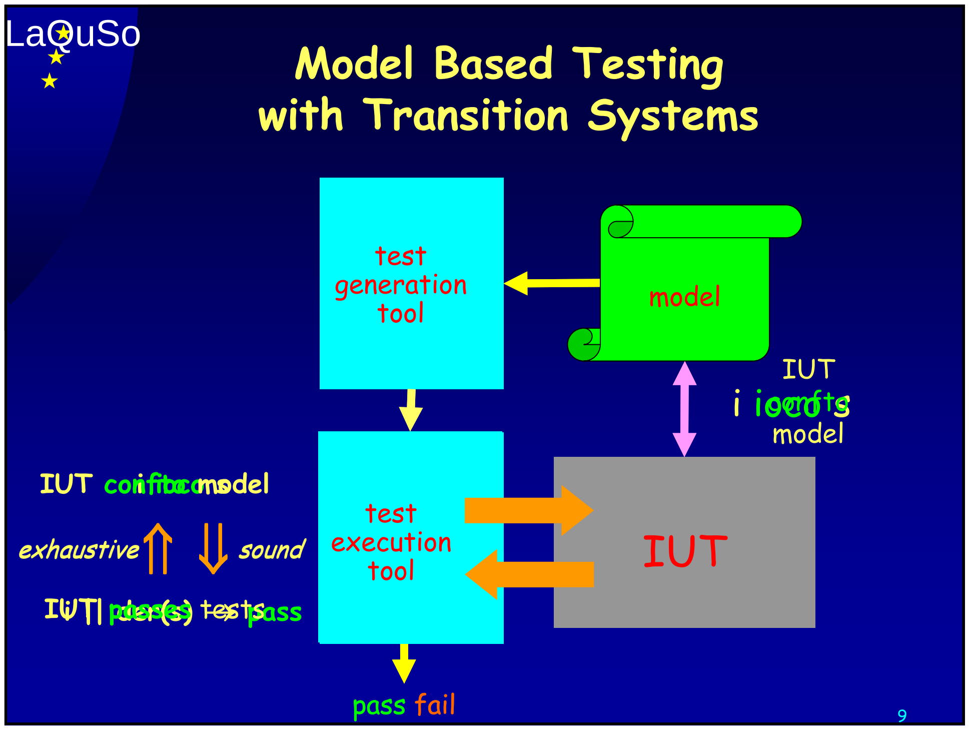


LaQuSo

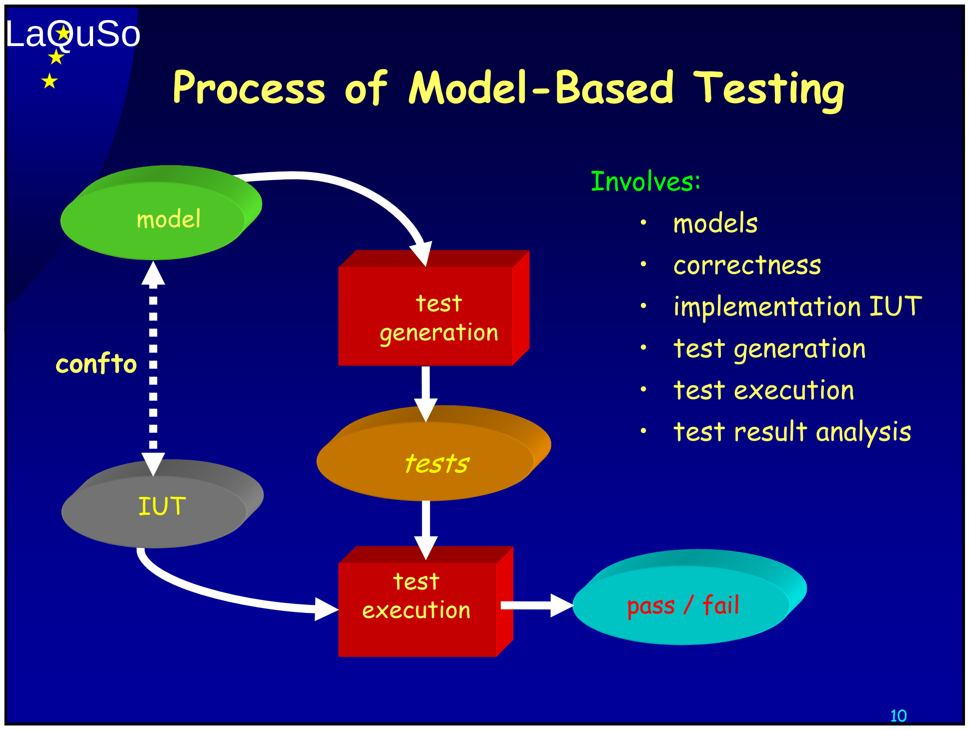
Approaches to Model-Based Testing

Several modeling paradigms:

- ☞ Finite State Machine
- ☞ Pre/post-conditions
- Labelled Transition Systems
- ☞ Programs as Functions
- ☞ Abstract Data Type testing
- ☞ . . . .



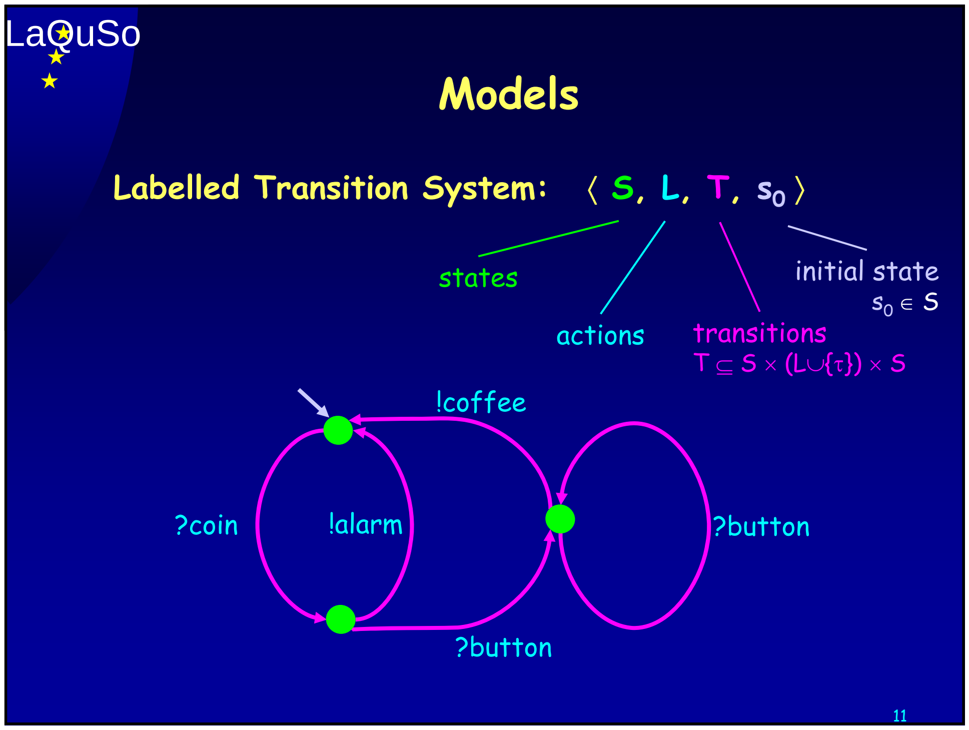
Model Based Testing with Transition Systems



Process of Model-Based Testing

Involves:

- models
- correctness
- implementation IUT
- test generation
- test execution
- test result analysis

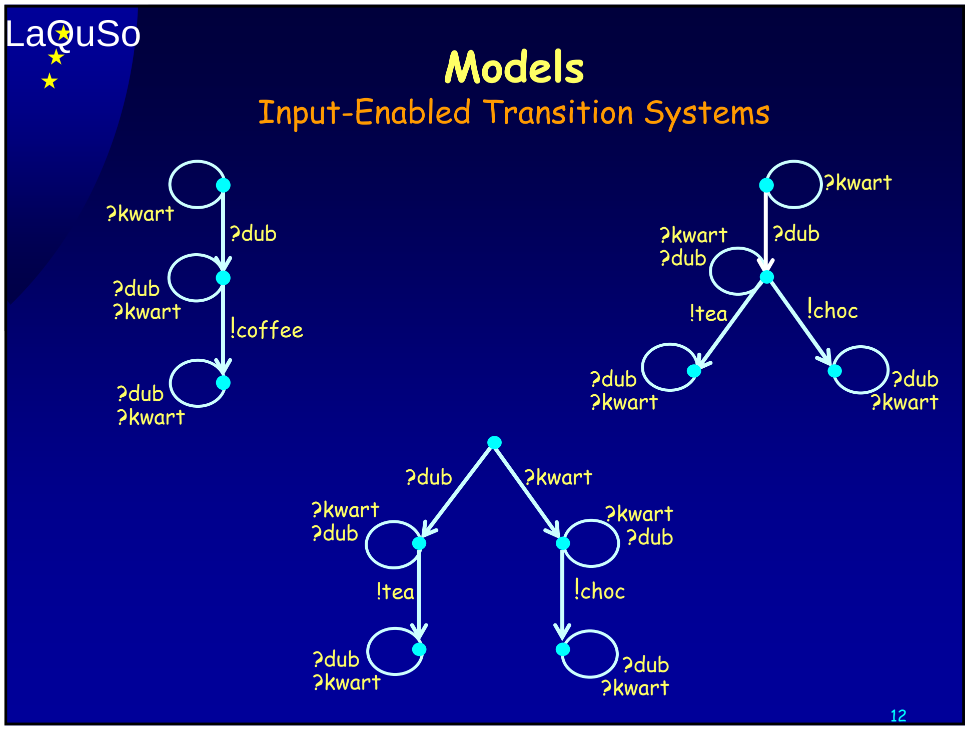


LaQuSo

Models

Labelled Transition System: $\langle$ S, L, T, s_0 $\rangle$

11

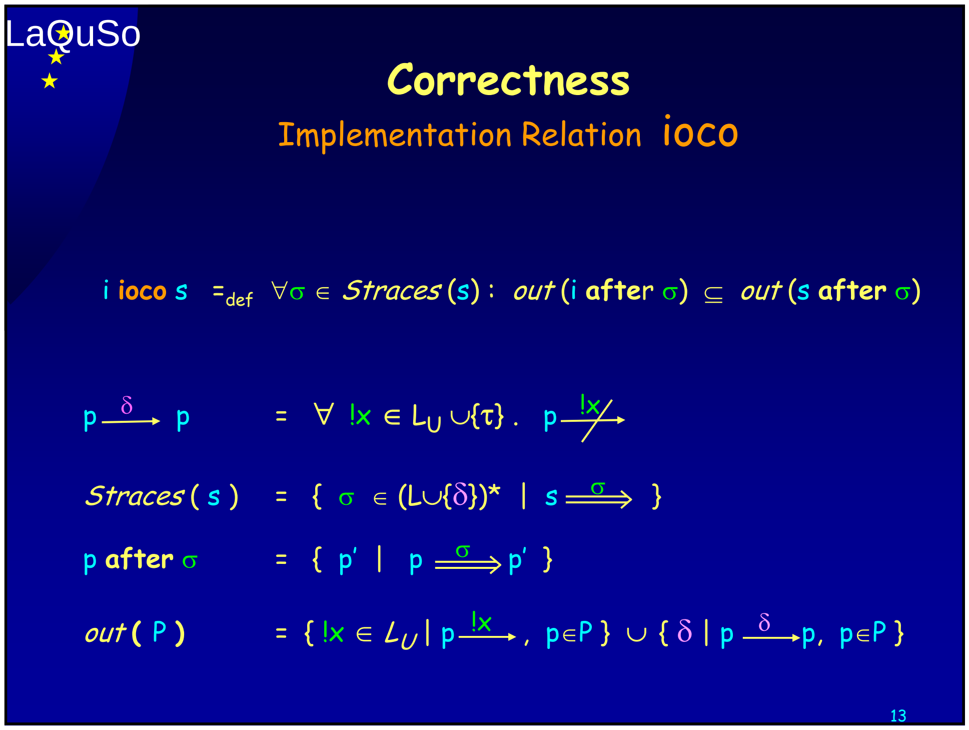


LaQuSo

Models

Input-Enabled Transition Systems

12


LaQuSo

Correctness

Implementation Relation ioco

$$i \;\textbf{ioco}\; s \;=_{def}\; \forall \sigma \in \mathit{Straces}(s) : \mathit{out}(i \;\textbf{after}\; \sigma) \subseteq \mathit{out}(s \;\textbf{after}\; \sigma)$$

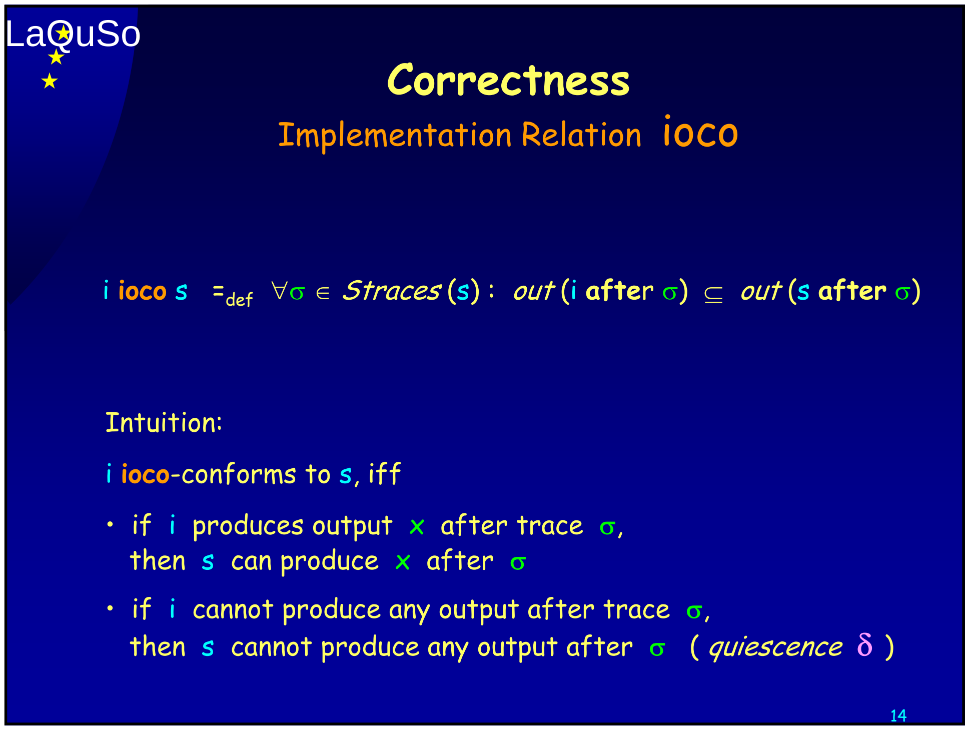
$$p \xrightarrow{\delta} p \quad = \quad \forall\, !x \in L_U \cup \{\tau\} .\; p \not\xrightarrow{!x}$$

$$\mathit{Straces}(s) \quad = \quad \{\, \sigma \in (L \cup \{\delta\})^* \mid s \xRightarrow{\sigma} \,\}$$

$$p \;\textbf{after}\; \sigma \quad = \quad \{\, p' \mid p \xRightarrow{\sigma} p' \,\}$$

$$\mathit{out}(\, P \,) \quad = \quad \{\, !x \in L_U \mid p \xrightarrow{!x} ,\; p \in P \,\} \cup \{\, \delta \mid p \xrightarrow{\delta} p,\; p \in P \,\}$$

13


LaQuSo

Correctness

Implementation Relation ioco

$$i \;\textbf{ioco}\; s \;=_{def}\; \forall \sigma \in \mathit{Straces}(s) : \mathit{out}(i \;\textbf{after}\; \sigma) \subseteq \mathit{out}(s \;\textbf{after}\; \sigma)$$

Intuition:

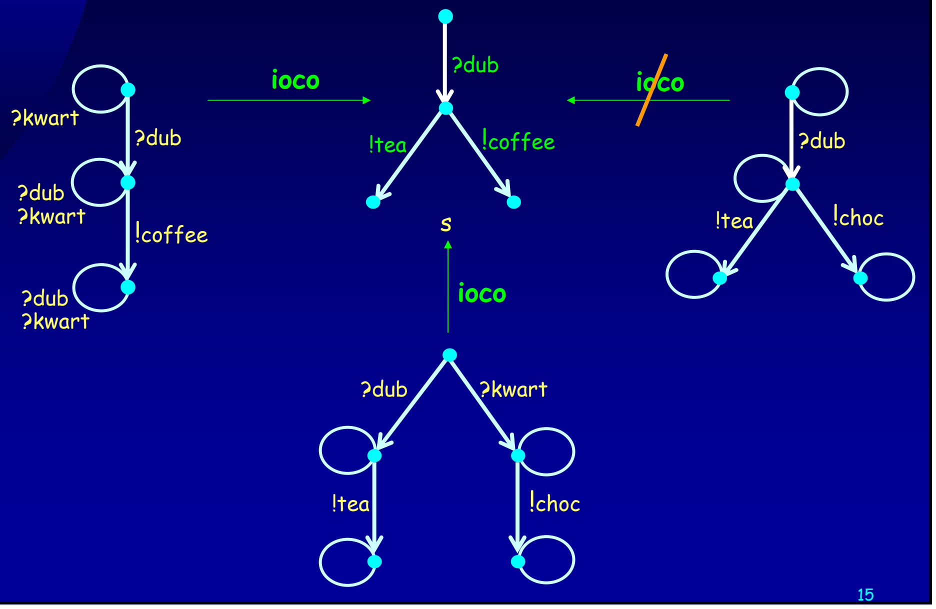
i **ioco**-conforms to s, iff

- if i produces output x after trace σ,
  then s can produce x after σ
- if i cannot produce any output after trace σ,
  then s cannot produce any output after σ (*quiescence* δ)

14

LaQuSo

Implementation Relation ioco

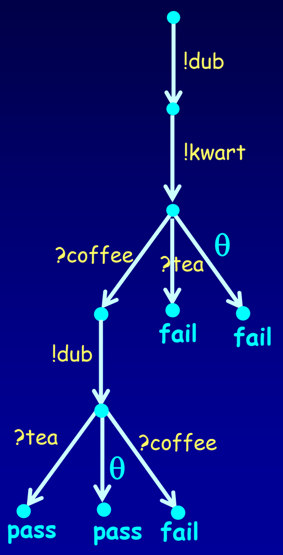


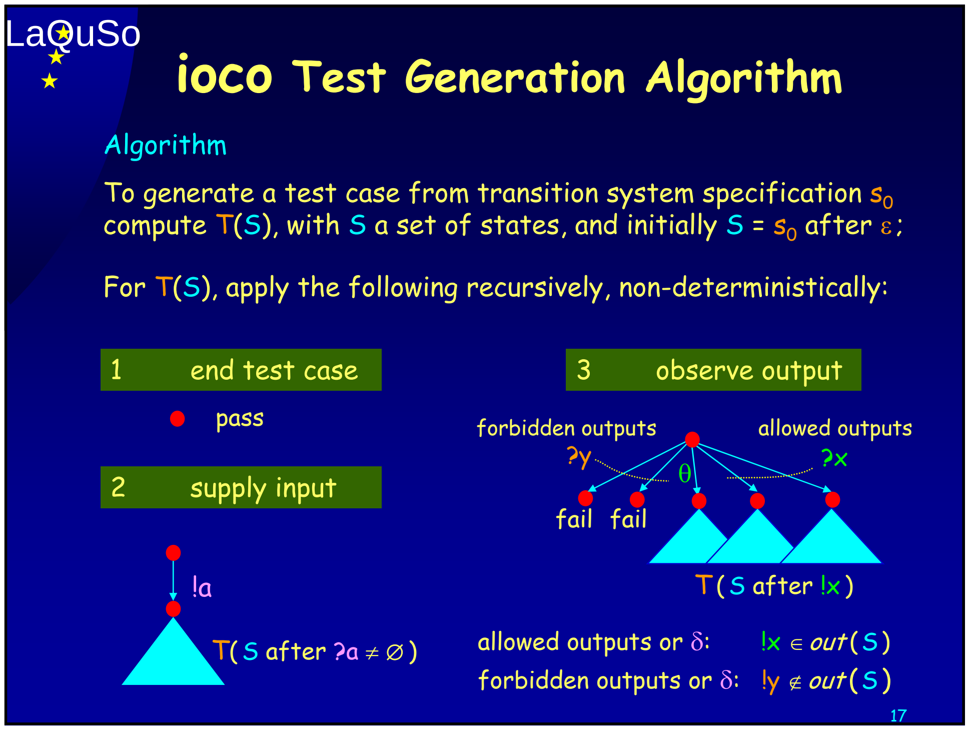
LaQuSo

Test Cases

Model of test case
= transition system :

- 'quiescence' label θ
- tree-structured
- finite, deterministic
- final states **pass** and **fail**
- from each state ≠ **pass**, **fail** :
  - either one input !a
  - or all outputs ?x and θ





LaQuSo

ioco Test Generation Algorithm

Algorithm

To generate a test case from transition system specification s_0 compute T(S), with S a set of states, and initially S = s_0 after ε;

For T(S), apply the following recursively, non-deterministically:

1 end test case

● pass

2 supply input

!a

T(S after ?a $\neq \varnothing$)

3 observe output

forbidden outputs

allowed outputs

?y

θ

?x

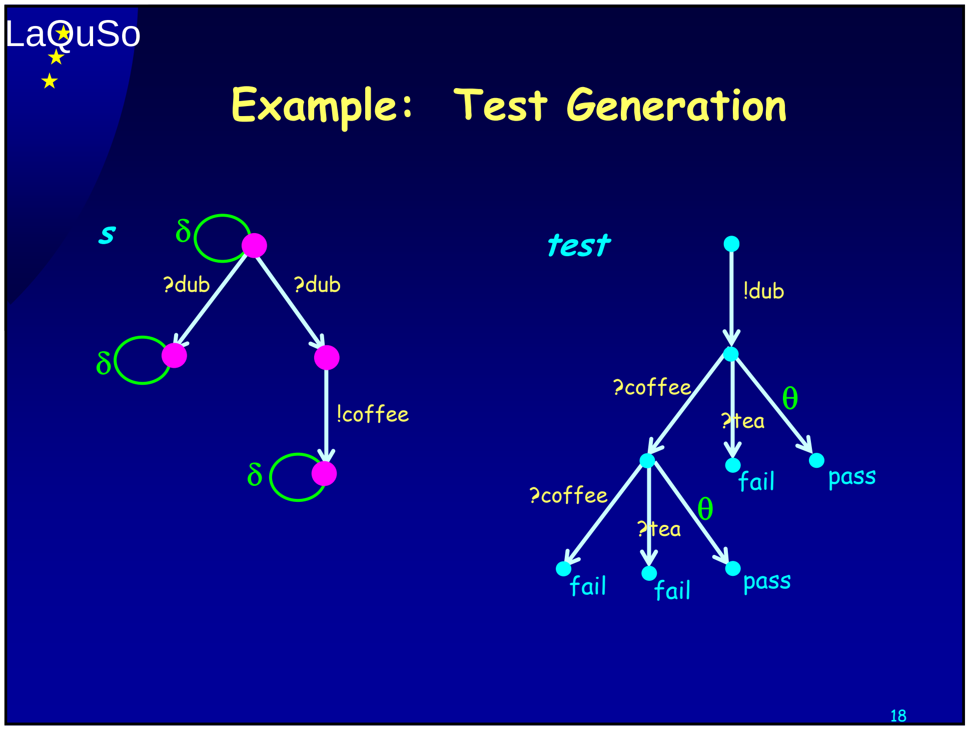
fail fail

T(S after !x)

allowed outputs or δ: !x $\in$ *out*(S)

forbidden outputs or δ: !y $\notin$ *out*(S)

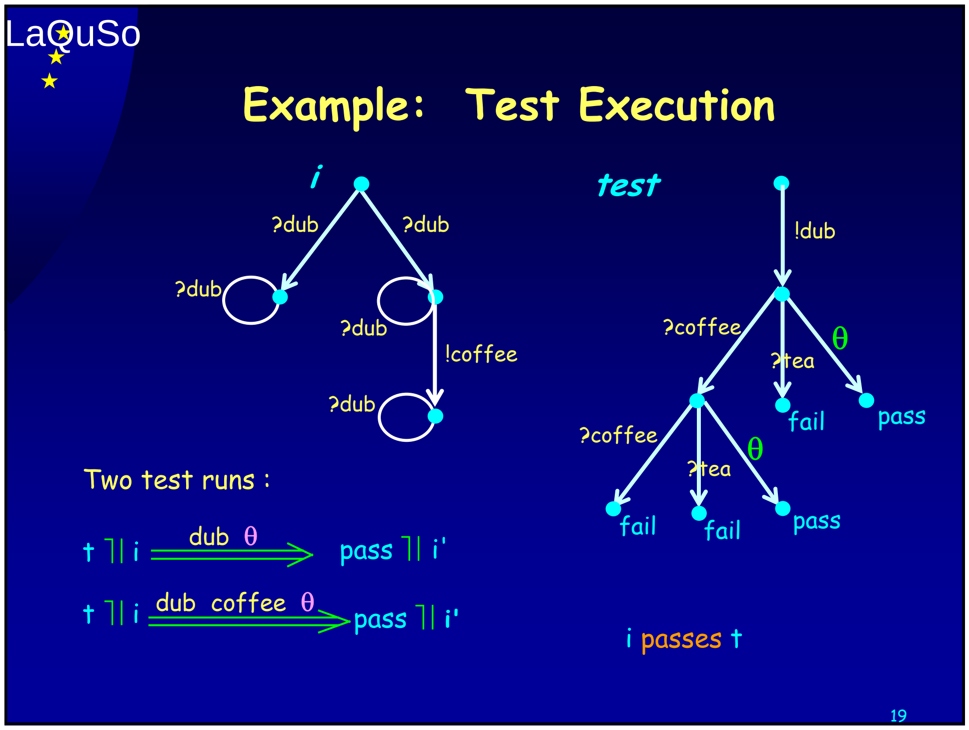
17



LaQuSo

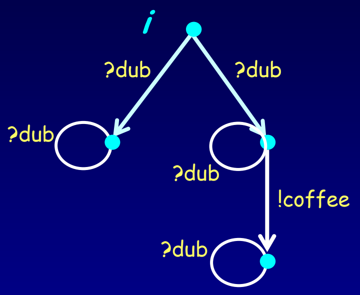
Example: Test Generation

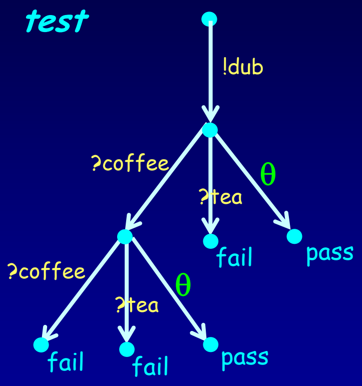
18



LaQuSo

Example: Test Execution





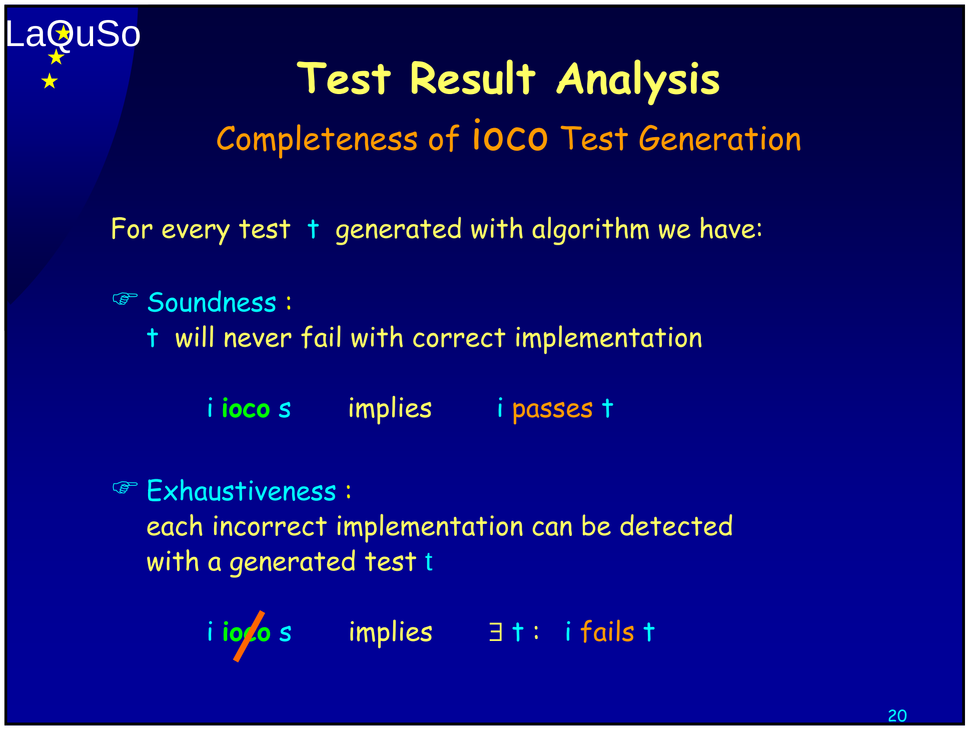
Two test runs :

$t \rceil\!| \, i \xRightarrow{\text{dub } \theta} \text{pass} \rceil\!| \, i'$

$t \rceil\!| \, i \xRightarrow{\text{dub coffee } \theta} \text{pass} \rceil\!| \, i'$

i passes t

19



LaQuSo

Test Result Analysis

Completeness of ioco Test Generation

For every test t generated with algorithm we have:

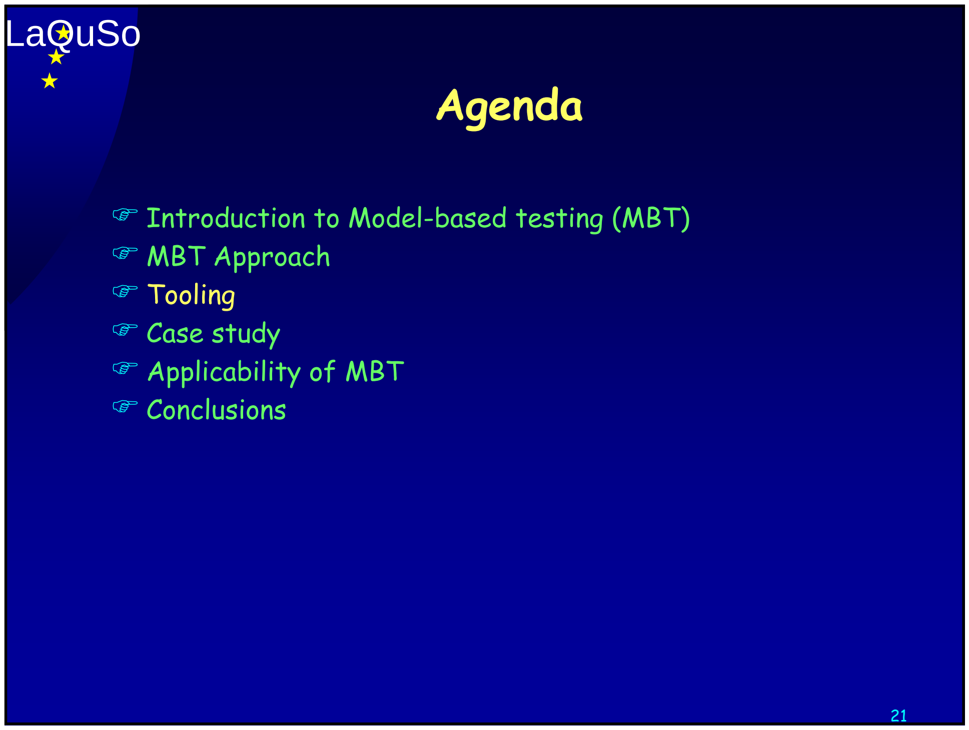
☞ Soundness :
t will never fail with correct implementation

i **ioco** s implies i passes t

☞ Exhaustiveness :
each incorrect implementation can be detected with a generated test t

i $\cancel{\textbf{ioco}}$ s implies ∃ t : i fails t

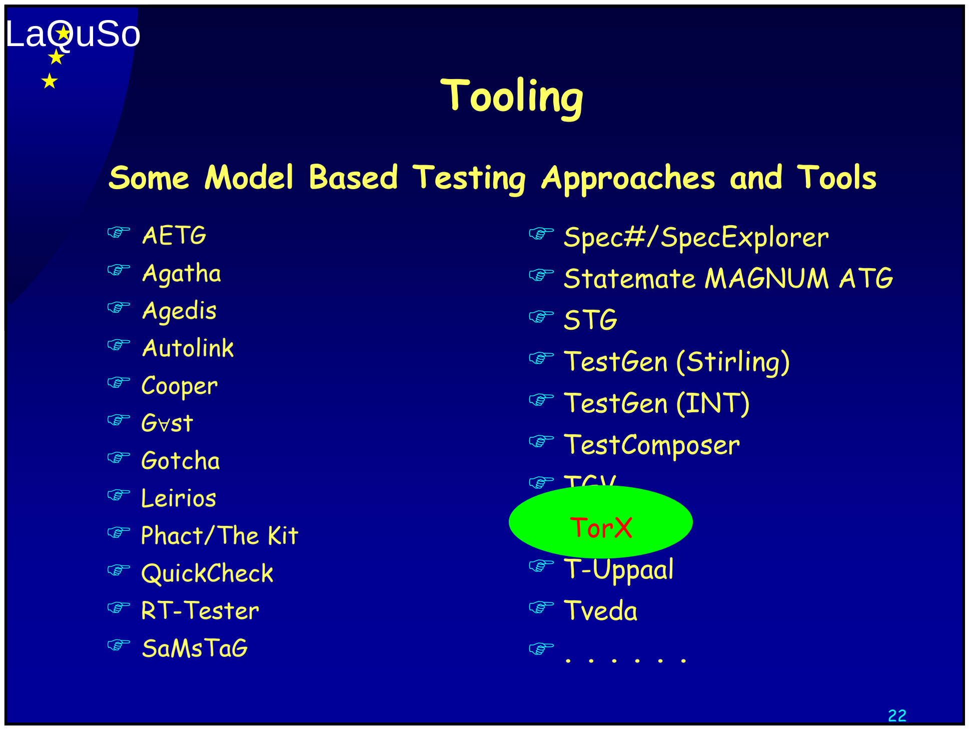
20


LaQuSo

Agenda

- Introduction to Model-based testing (MBT)
- MBT Approach
- Tooling
- Case study
- Applicability of MBT
- Conclusions

21


LaQuSo

Tooling

Some Model Based Testing Approaches and Tools

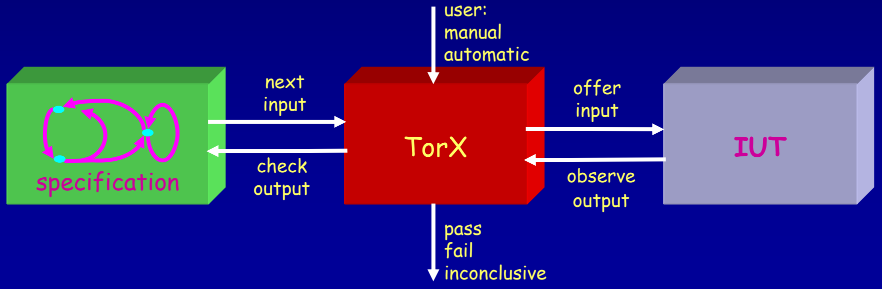
- AETG
- Agatha
- Agedis
- Autolink
- Cooper
- G∀st
- Gotcha
- Leirios
- Phact/The Kit
- QuickCheck
- RT-Tester
- SaMsTaG
- Spec#/SpecExplorer
- Statemate MAGNUM ATG
- STG
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TGV
- TorX
- T-Uppaal
- Tveda
- . . . . . .

22

LaQuSo

A Tool for Transition Systems Testing: TorX

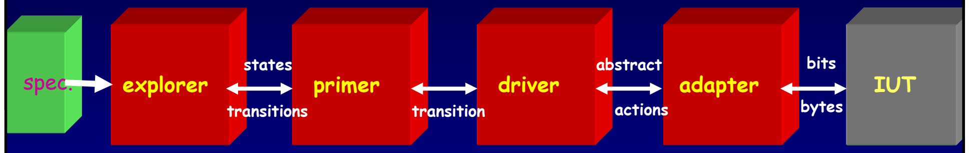
- On-the-fly test generation and test execution
- Implementation relation: **ioco**
- Mainly applicable to reactive systems / state based systems;
  - specification languages: LOTOS, Promela, FSP, Automata

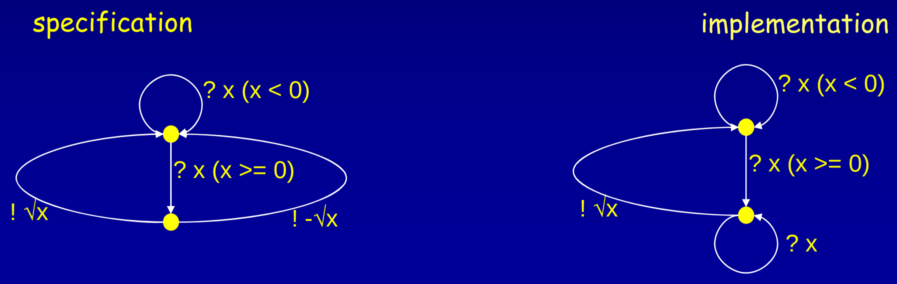


23

LaQuSo

On-the-Fly Testing

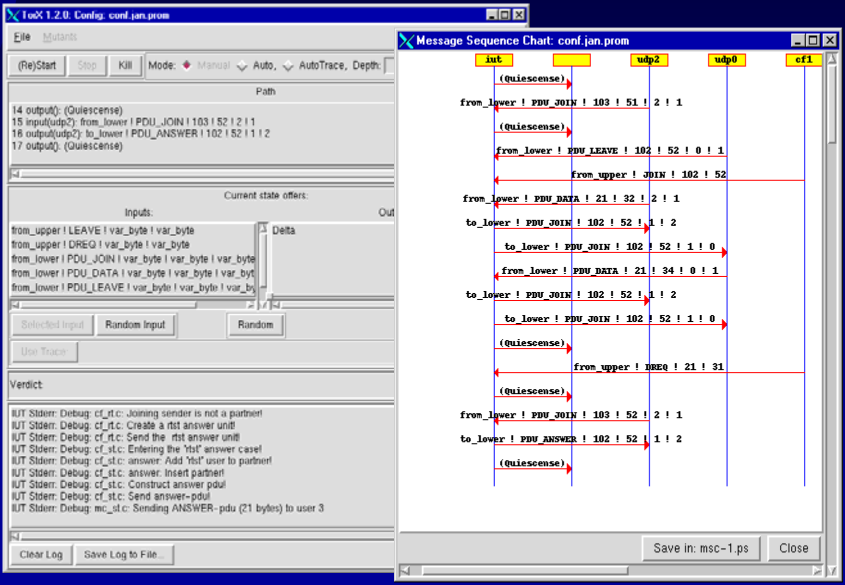




24

LaQuSo

TorX



```
14 output(): (Quiescense)
15 input(udp2): from_lower ! PDU_JOIN ! 103 ! 52 ! 2 ! 1
16 output(udp2): to_lower ! PDU_ANSWER ! 102 ! 52 ! 1 ! 2
17 output(): (Quiescense)
```

```
from_upper ! LEAVE ! var_byte ! var_byte
from_upper ! DREQ ! var_byte ! var_byte
from_lower ! PDU_JOIN ! var_byte ! var_byte ! var_byte
from_lower ! PDU_DATA ! var_byte ! var_byte ! var_byt
from_lower ! PDU_LEAVE ! var_byte ! var_byte ! var_by
```

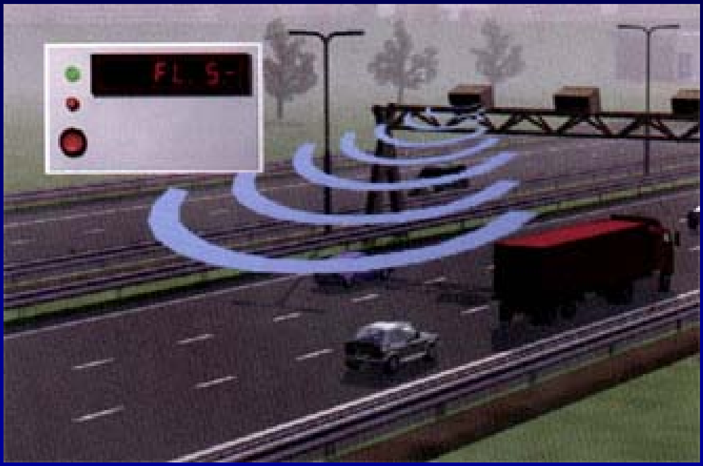
```
Delta
```

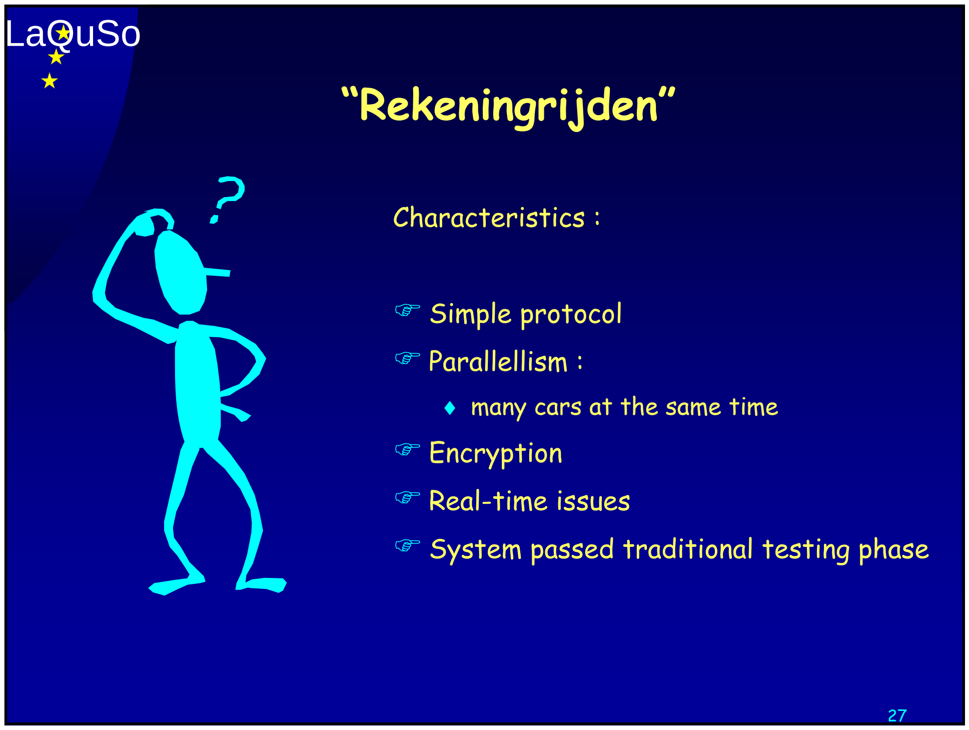
```
IUT Stderr: Debug: cf_rt.c: Joining sender is not a partner!
IUT Stderr: Debug: cf_rt.c: Create a rtst answer unit!
IUT Stderr: Debug: cf_rt.c: Send the  rtst answer unit!
IUT Stderr: Debug: cf_st.c: Entering the "rtst" answer case!
IUT Stderr: Debug: cf_st.c: answer: Add "rtst" user to partner!
IUT Stderr: Debug: cf_st.c: answer: Insert partner!
IUT Stderr: Debug: cf_st.c: Construct answer pdu!
IUT Stderr: Debug: cf_st.c: Send answer-pdu!
IUT Stderr: Debug: mc_st.c: Sending ANSWER-pdu (21 bytes) to user 3
```

```
(Quiescense)
from_lower ! PDU_JOIN ! 103 ! 51 ! 2 ! 1
(Quiescense)
from_lower ! PDU_LEAVE ! 102 ! 52 ! 0 ! 1
from_upper ! JOIN ! 102 ! 52
from_lower ! PDU_DATA ! 21 ! 32 ! 2 ! 1
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 2
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 0
from_lower ! PDU_DATA ! 21 ! 34 ! 0 ! 1
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 2
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 0
(Quiescense)
from_upper ! DREQ ! 21 ! 31
(Quiescense)
from_lower ! PDU_JOIN ! 103 ! 52 ! 2 ! 1
to_lower ! PDU_ANSWER ! 102 ! 52 ! 1 ! 2
(Quiescense)
```

LaQuSo

Interpay "Rekeningrijden" Payment Box Protocol





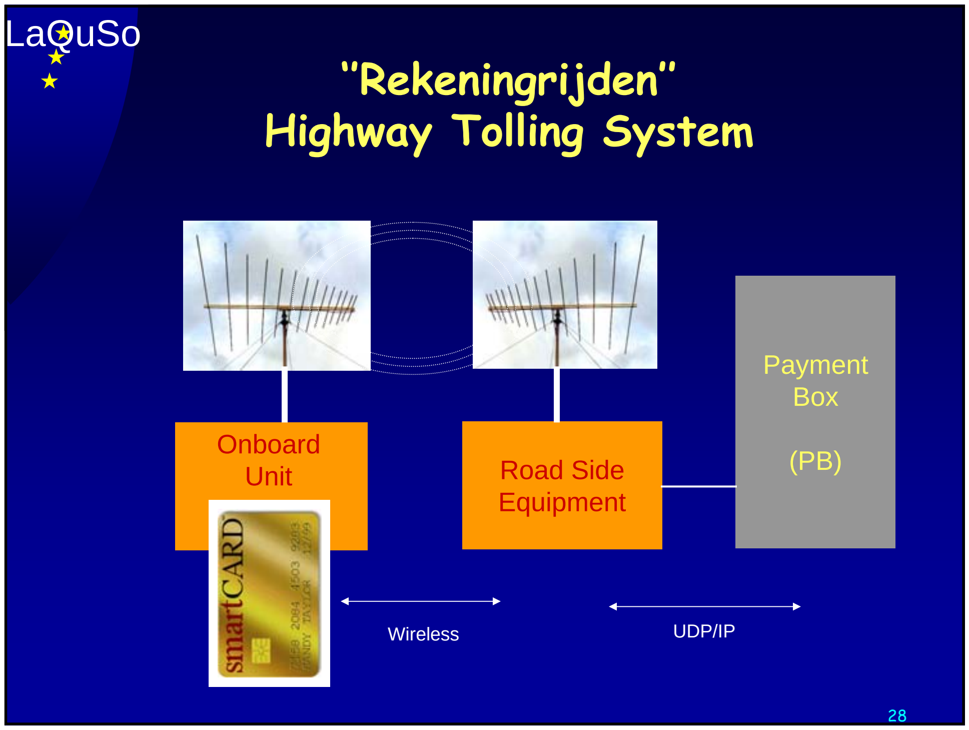
LaQuSo

"Rekeningrijden"

Characteristics :

- Simple protocol
- Parallellism :
  - many cars at the same time
- Encryption
- Real-time issues
- System passed traditional testing phase

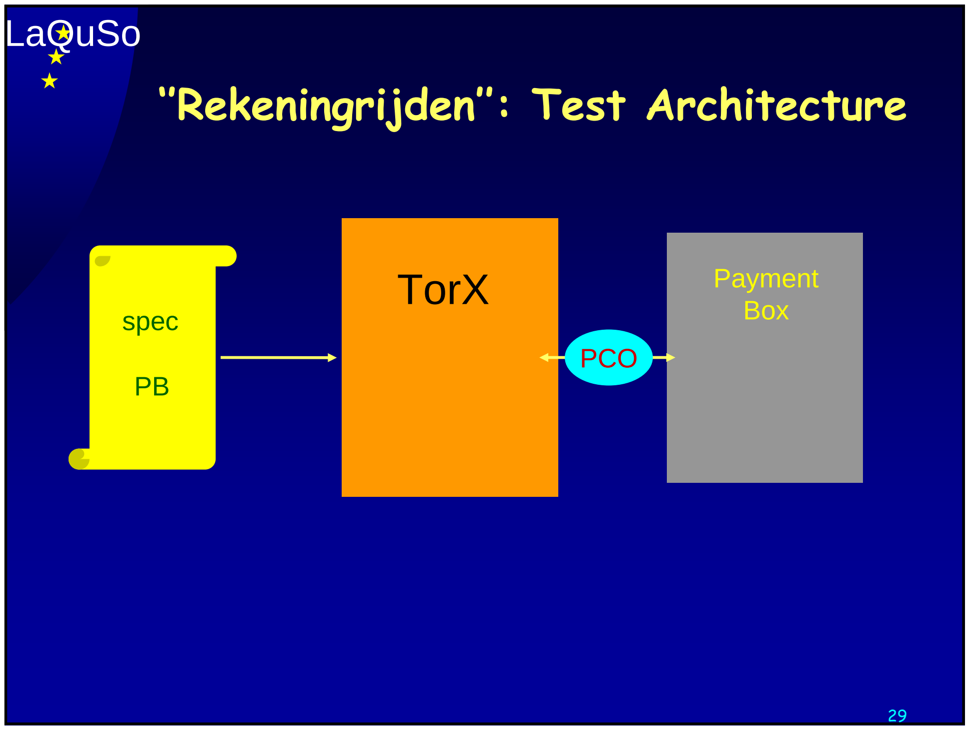
27



LaQuSo

"Rekeningrijden" Highway Tolling System

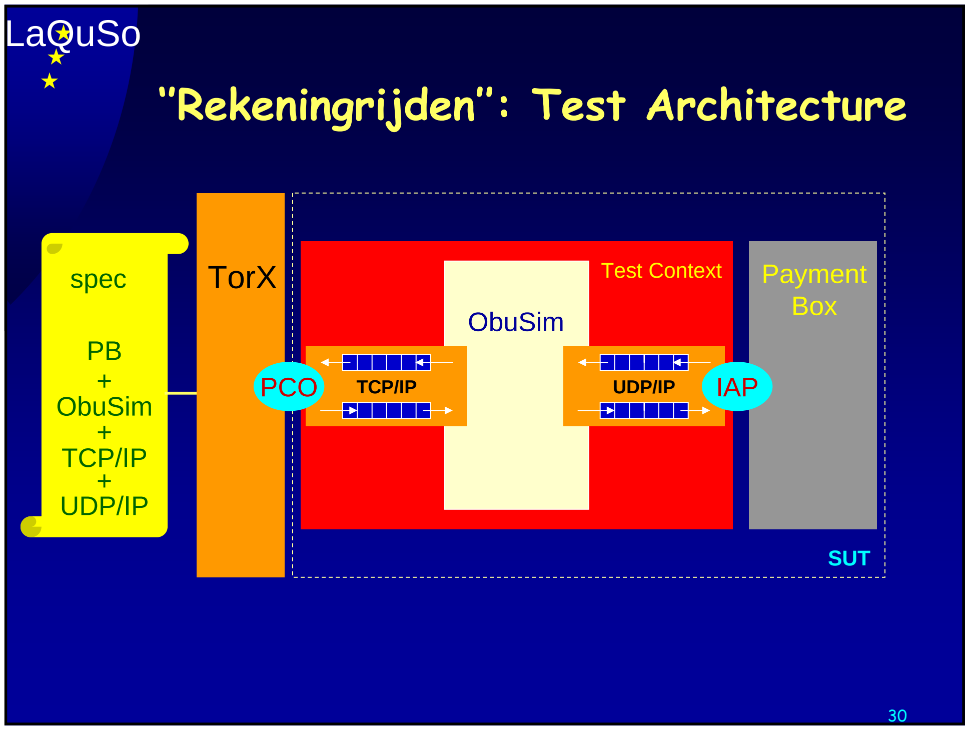
28



LaQuSo

"Rekeningrijden": Test Architecture

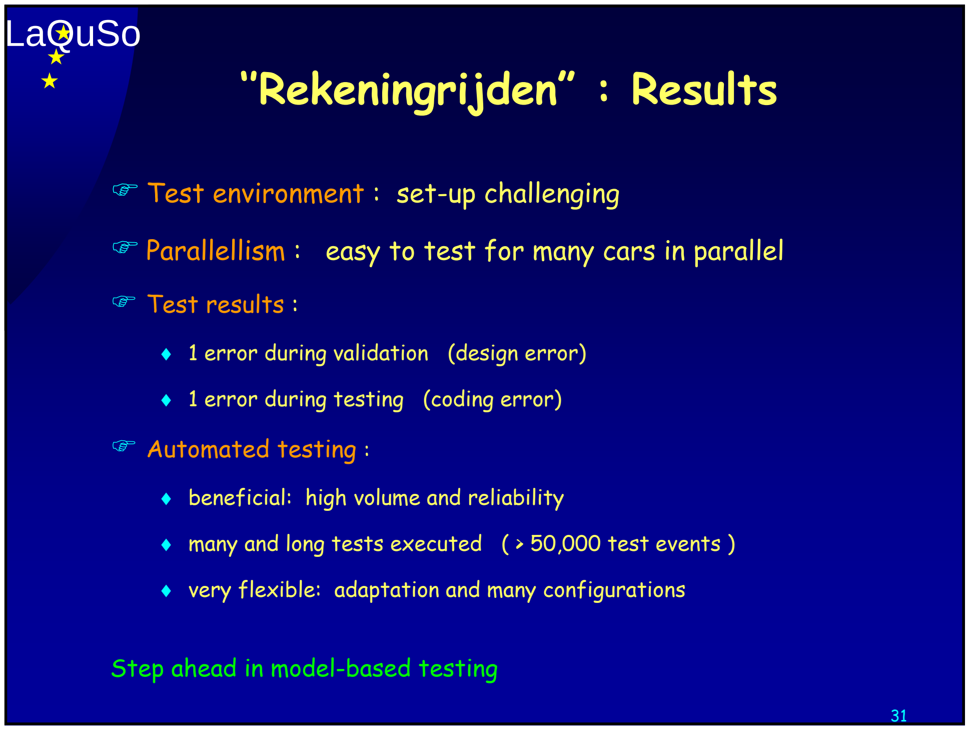
29



LaQuSo

"Rekeningrijden": Test Architecture

30

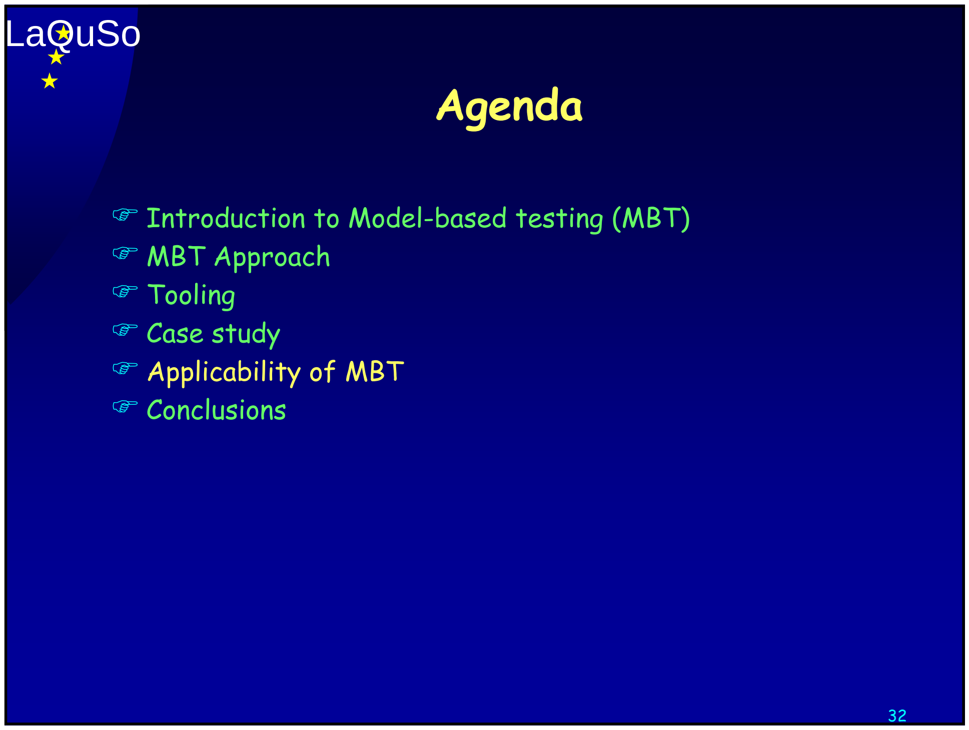

LaQuSo

"Rekeningrijden" : Results

- Test environment : set-up challenging
- Parallellism : easy to test for many cars in parallel
- Test results :
  - 1 error during validation (design error)
  - 1 error during testing (coding error)
- Automated testing :
  - beneficial: high volume and reliability
  - many and long tests executed (> 50,000 test events)
  - very flexible: adaptation and many configurations

Step ahead in model-based testing

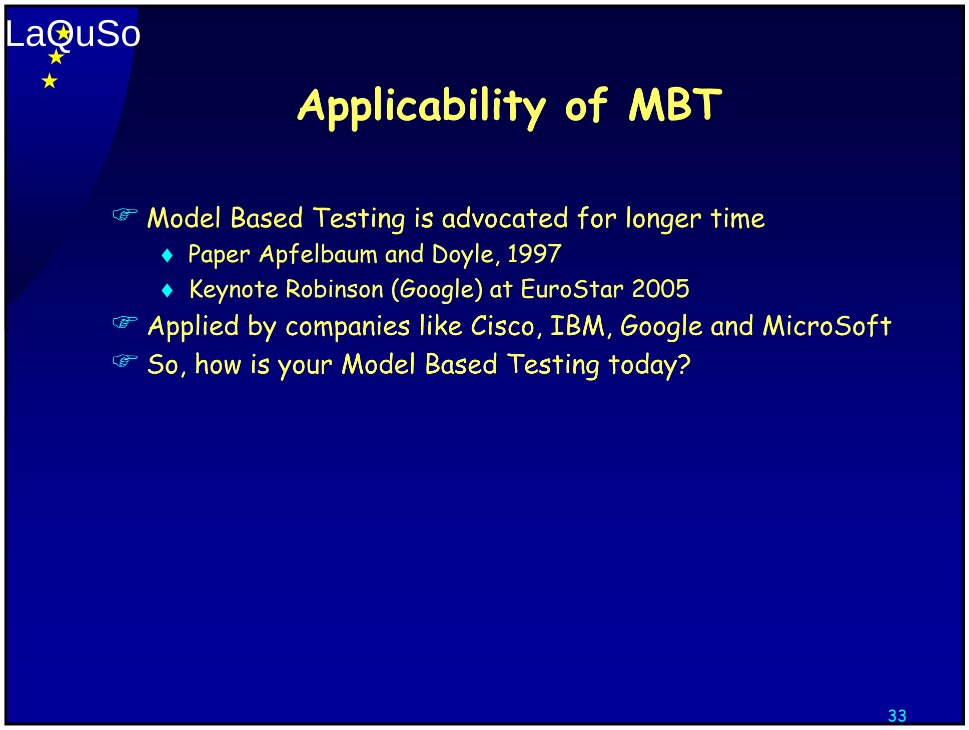
31


LaQuSo

Agenda

- Introduction to Model-based testing (MBT)
- MBT Approach
- Tooling
- Case study
- Applicability of MBT
- Conclusions

32

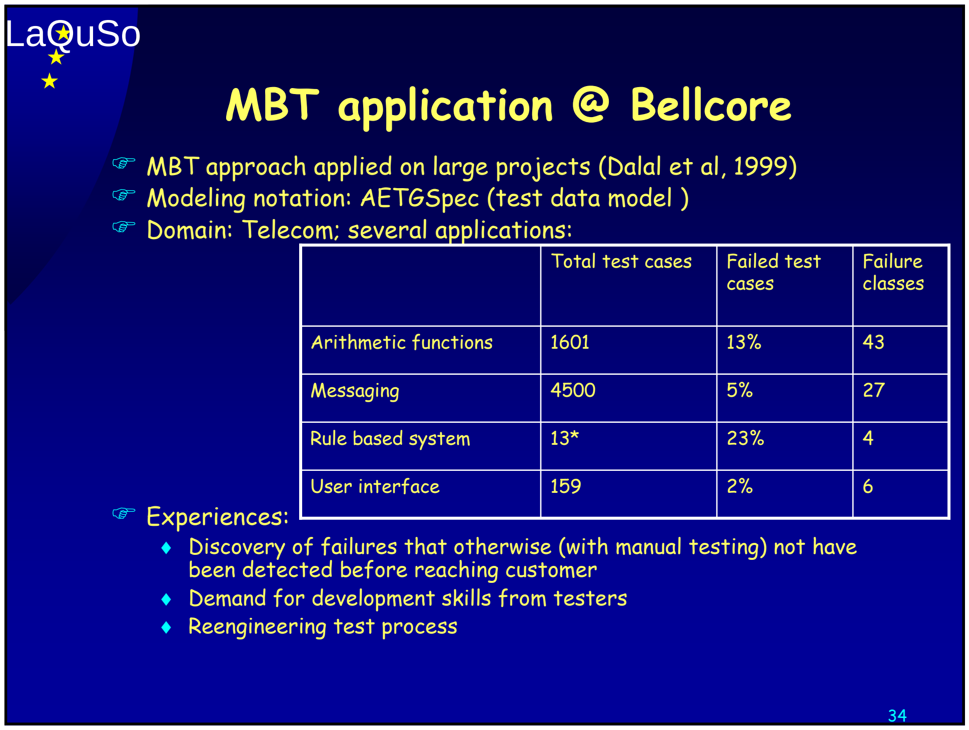


LaQuSo

Applicability of MBT

- Model Based Testing is advocated for longer time
  - Paper Apfelbaum and Doyle, 1997
  - Keynote Robinson (Google) at EuroStar 2005
- Applied by companies like Cisco, IBM, Google and MicroSoft
- So, how is your Model Based Testing today?

33



LaQuSo

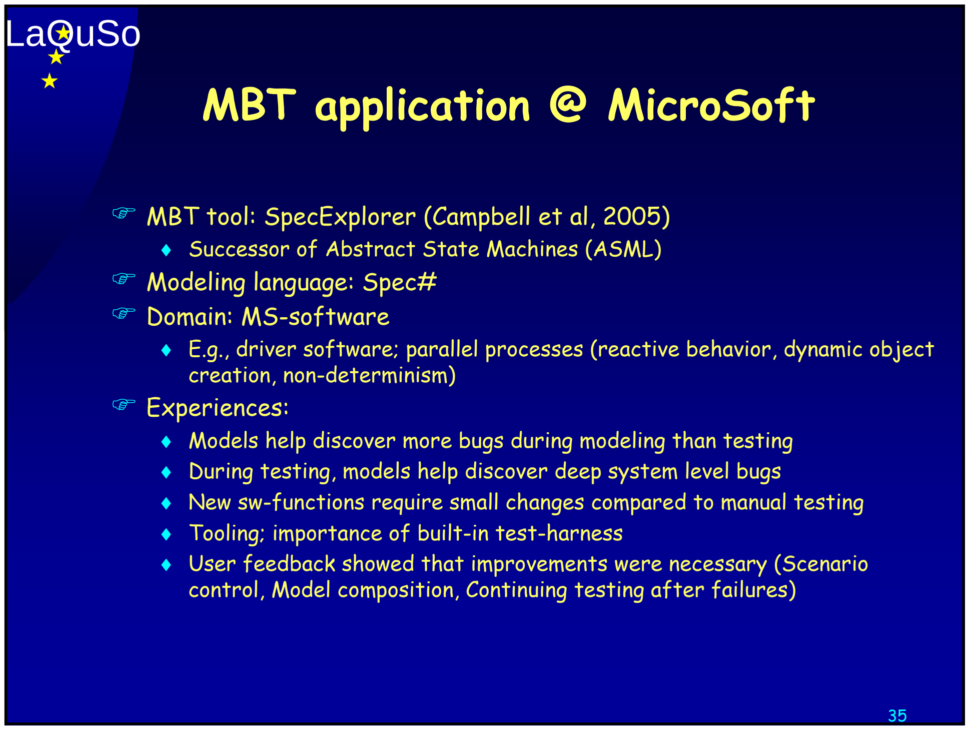
MBT application @ Bellcore

- MBT approach applied on large projects (Dalal et al, 1999)
- Modeling notation: AETGSpec (test data model)
- Domain: Telecom; several applications:

	Total test cases	Failed test cases	Failure classes
Arithmetic functions	1601	13%	43
Messaging	4500	5%	27
Rule based system	13*	23%	4
User interface	159	2%	6

- Experiences:
  - Discovery of failures that otherwise (with manual testing) not have been detected before reaching customer
  - Demand for development skills from testers
  - Reengineering test process

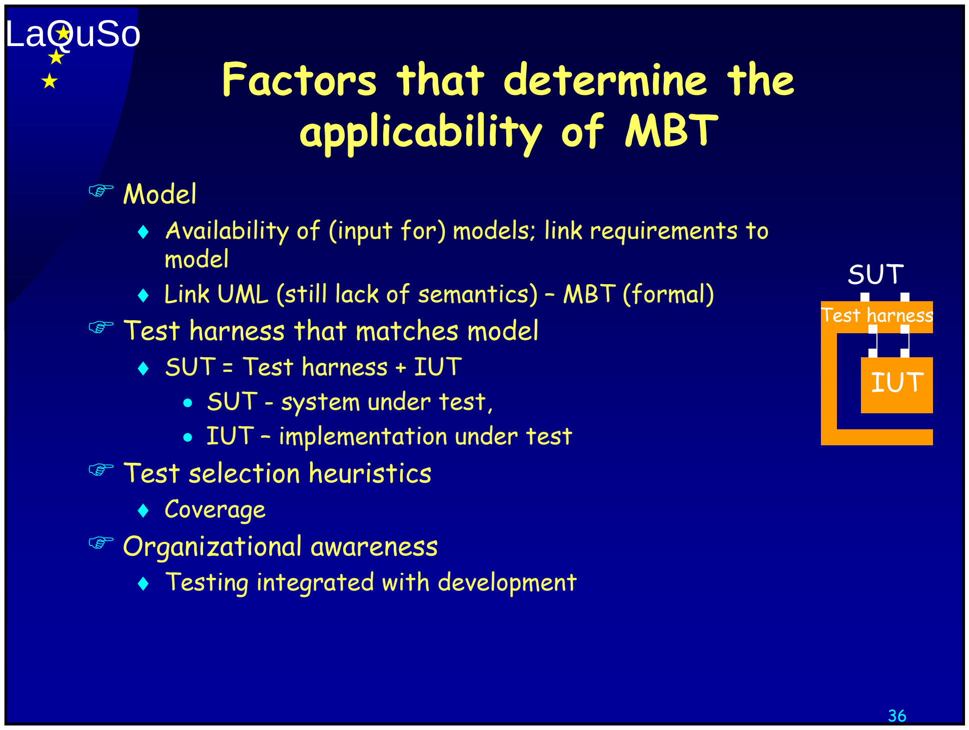
34


LaQuSo

MBT application @ MicroSoft

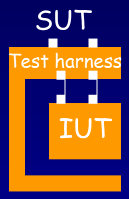
- MBT tool: SpecExplorer (Campbell et al, 2005)
  - Successor of Abstract State Machines (ASML)
- Modeling language: Spec#
- Domain: MS-software
  - E.g., driver software; parallel processes (reactive behavior, dynamic object creation, non-determinism)
- Experiences:
  - Models help discover more bugs during modeling than testing
  - During testing, models help discover deep system level bugs
  - New sw-functions require small changes compared to manual testing
  - Tooling; importance of built-in test-harness
  - User feedback showed that improvements were necessary (Scenario control, Model composition, Continuing testing after failures)

35

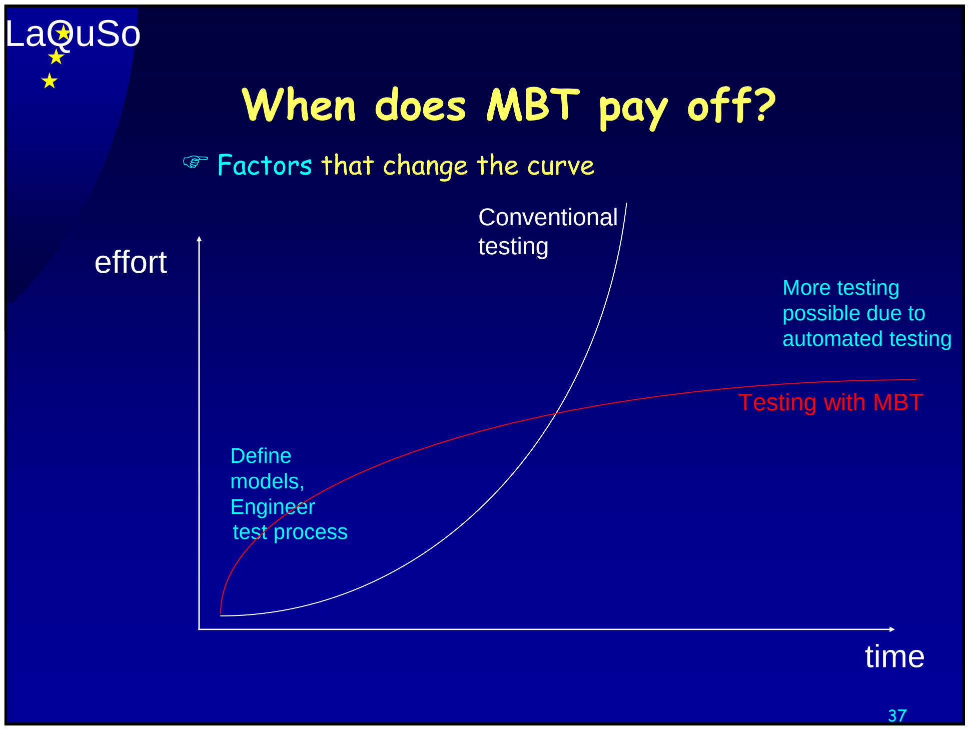

LaQuSo

Factors that determine the applicability of MBT

- Model
  - Availability of (input for) models; link requirements to model
  - Link UML (still lack of semantics) – MBT (formal)
- Test harness that matches model
  - SUT = Test harness + IUT
    - SUT - system under test,
    - IUT – implementation under test
- Test selection heuristics
  - Coverage
- Organizational awareness
  - Testing integrated with development



36

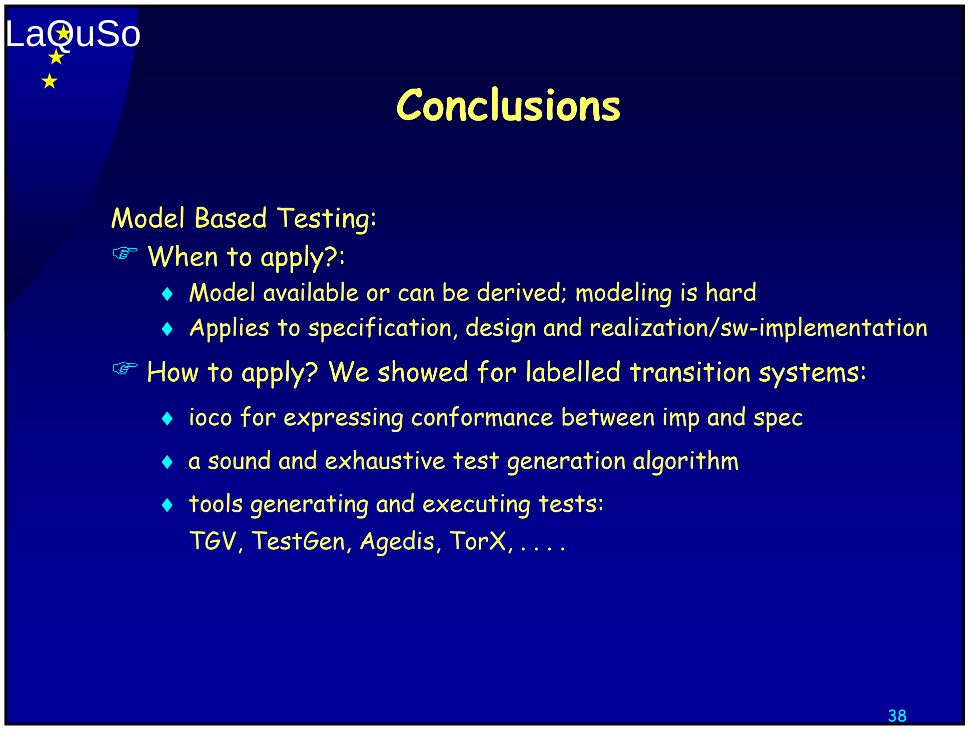


LaQuSo

When does MBT pay off?

☞ Factors that change the curve

37



LaQuSo

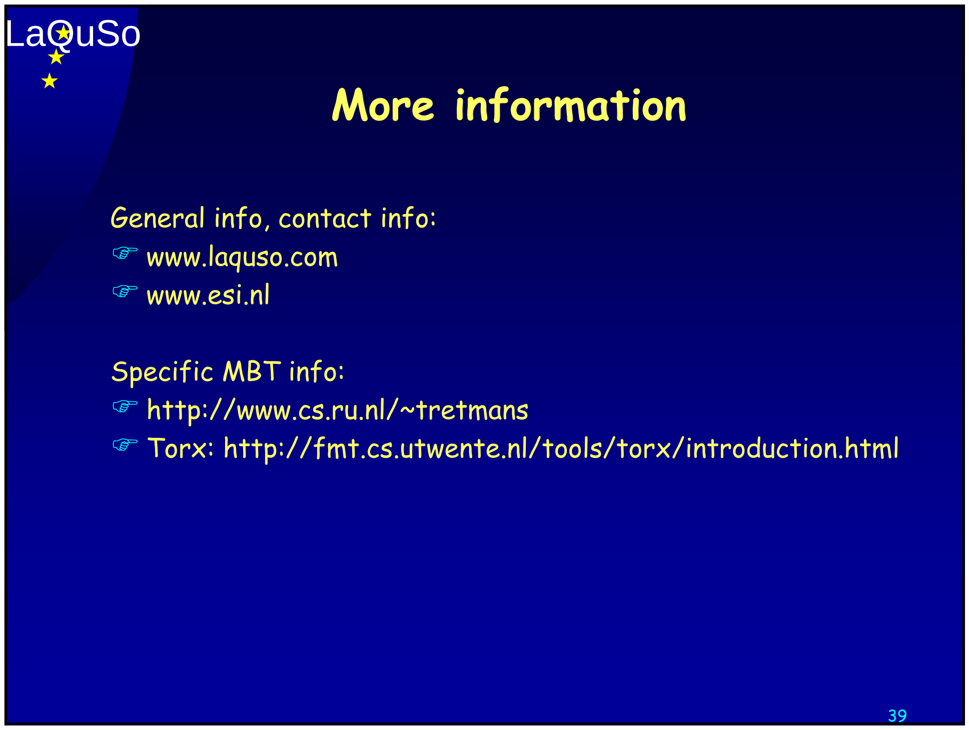
Conclusions

Model Based Testing:

☞ When to apply?:
- ♦ Model available or can be derived; modeling is hard
- ♦ Applies to specification, design and realization/sw-implementation

☞ How to apply? We showed for labelled transition systems:
- ♦ ioco for expressing conformance between imp and spec
- ♦ a sound and exhaustive test generation algorithm
- ♦ tools generating and executing tests:
  TGV, TestGen, Agedis, TorX, ...

38



LaQuSo

More information

General info, contact info:

☞ www.laquso.com

☞ www.esi.nl

Specific MBT info:

☞ http://www.cs.ru.nl/~tretmans

☞ Torx: http://fmt.cs.utwente.nl/tools/torx/introduction.html