Radboud Universiteit

ESI

Model-Based Testing

Jan Tretmans

ESI – Embedded Systems Innovation by TNO
Radboud University Nijmegen
Högskolan i Halmstad
jan.tretmans@tno.nl

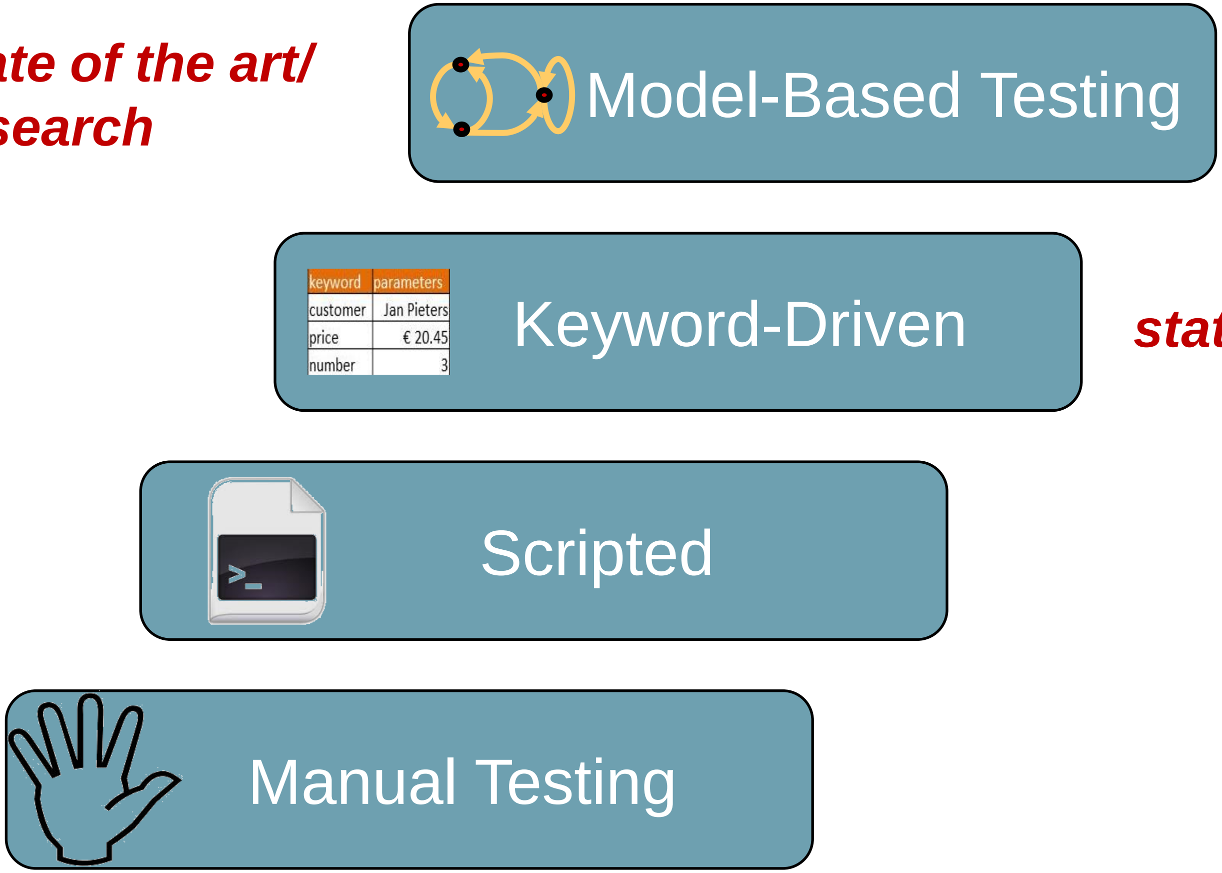
TNO innovation for life

Model-Based Testing

Basics

MBT : The Next Step in Test Automation

state of the art/ research



Model-Based Testing

Keyword-Driven

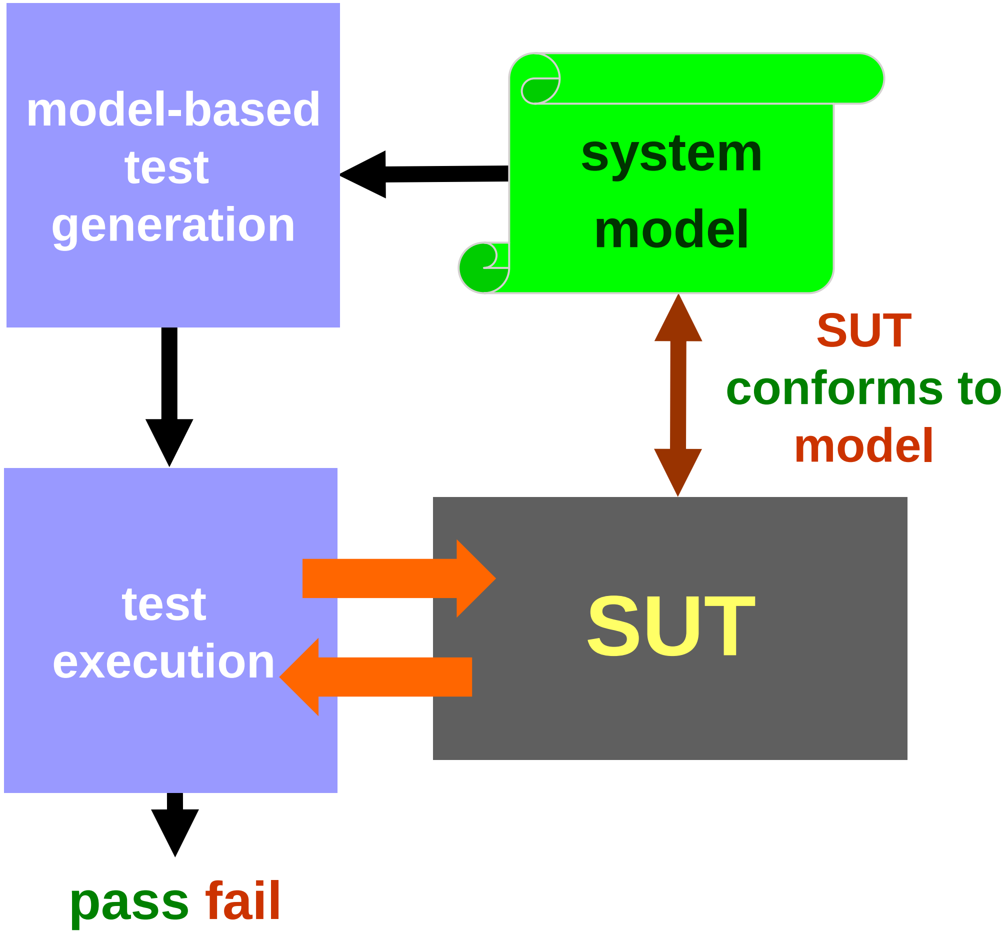
keyword	parameters
customer	Jan Pieters
price	€ 20.45
number	3

Scripted

Manual Testing

state of practice

Model Based Testing



model-based test generation

system model

SUT conforms to model

test execution

SUT

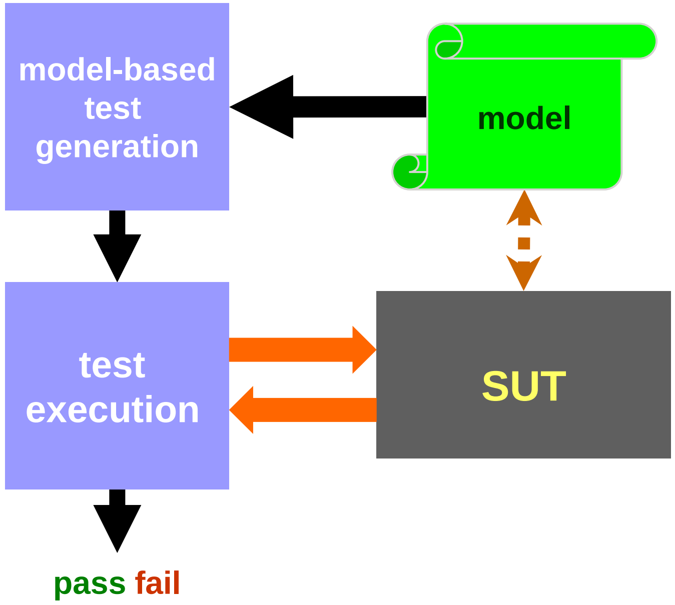
pass fail

SUT conforms to model

sound ⇓ ⇑ *exhaustive*

SUT passes tests

MBT : Benefits



detecting more bugs

faster and cheaper

MBT: next step in test automation

- **Automatic test generation**

  + test execution + result analysis

- **More, longer, and diversified test cases**

  more variation in test flow and in test data

- **Model is precise and consistent test basis**

  unambiguous analysis of test results

- **Test maintenance by maintaining models**

  improved regression testing

- **Expressing test coverage**

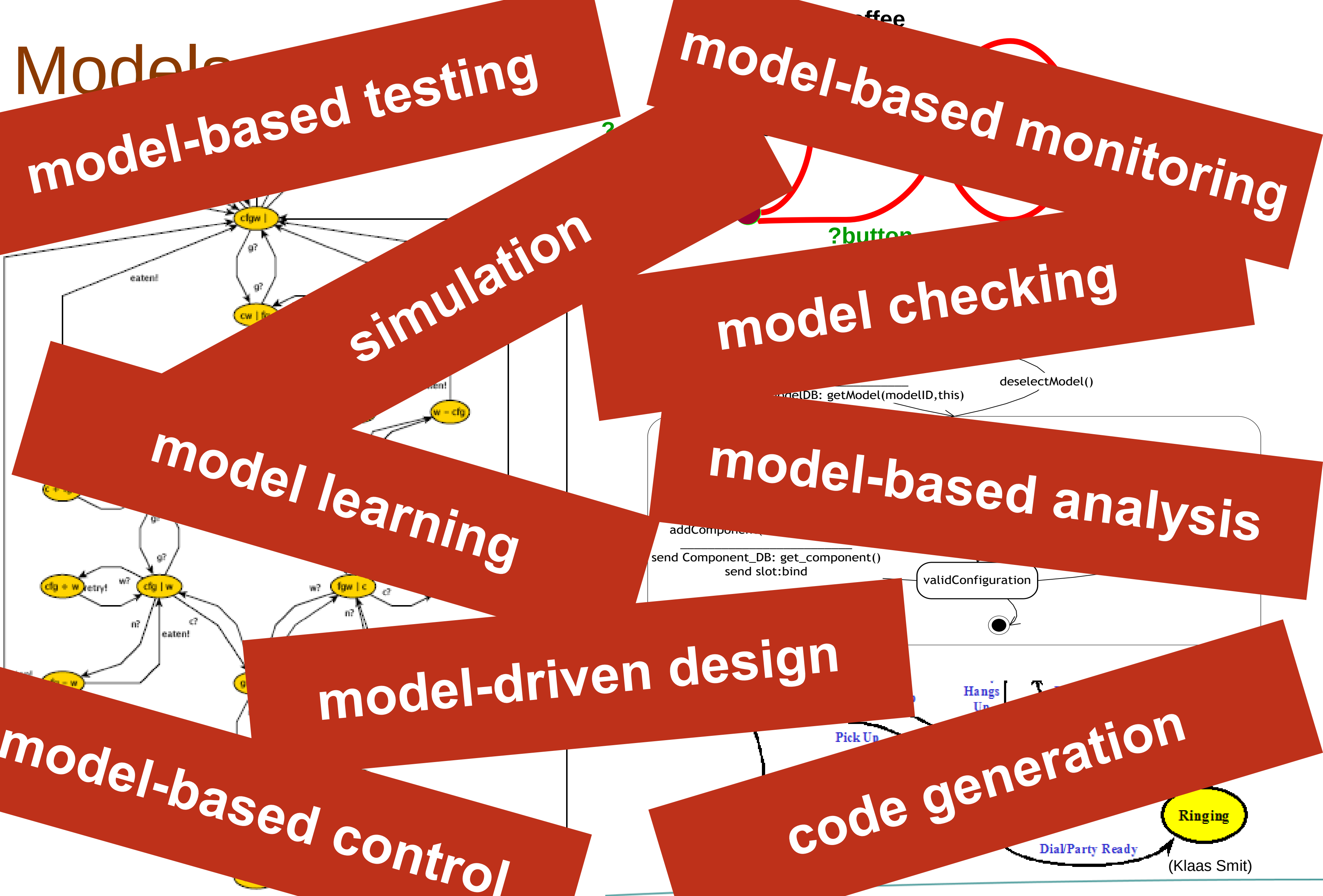
  model coverage

  customer profile coverage

Model-Based

Verification, Validation, Testing, ...

Models



(Klaas Smit)

model-based testing

model-based monitoring

simulation

model checking

model learning

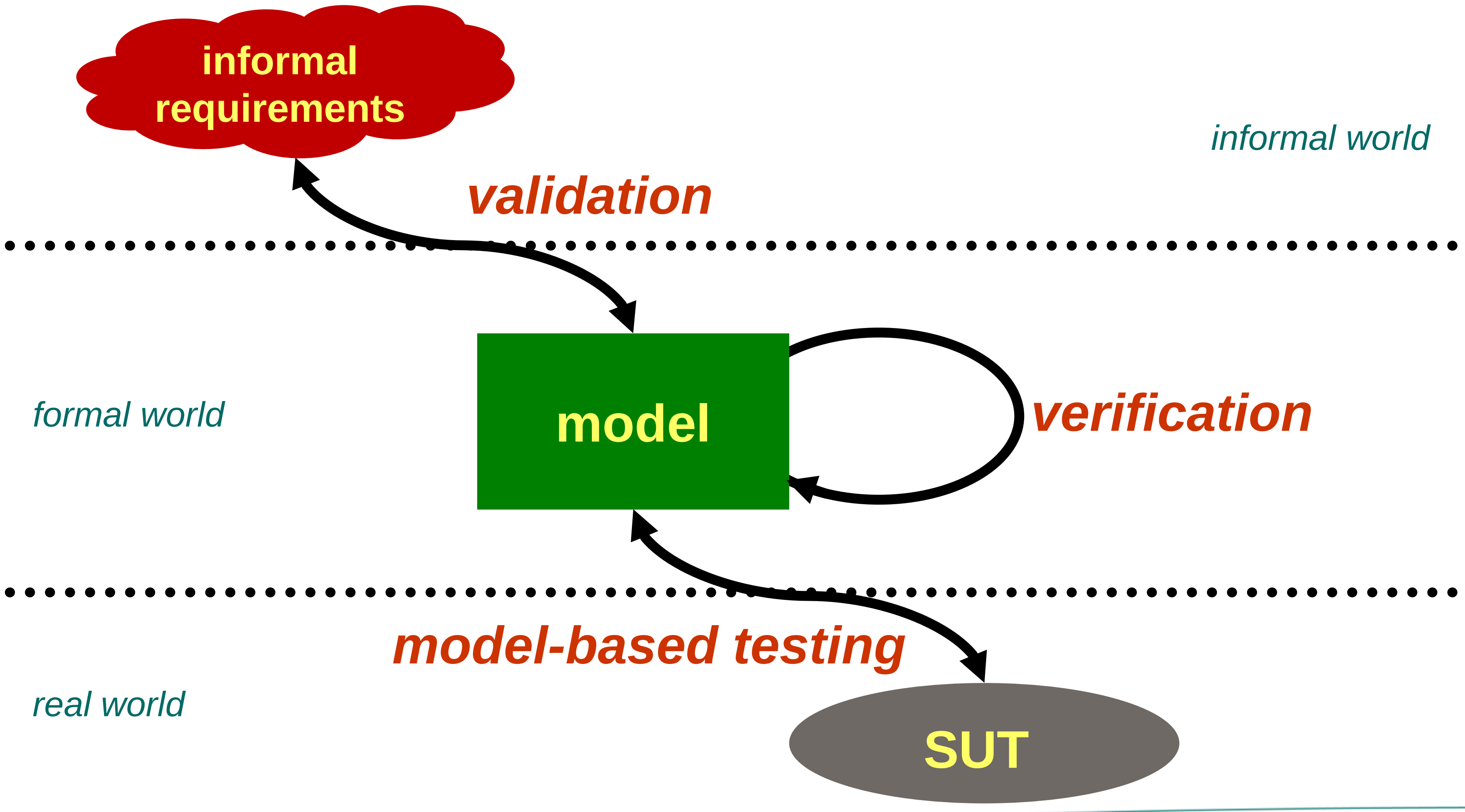
model-based analysis

model-driven design

model-based control

code generation

Validation, Verification, Testing



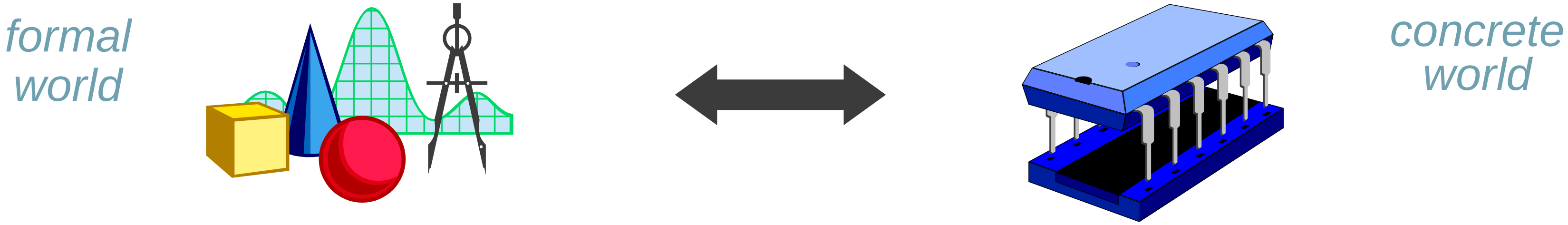
Verification and Testing

Model-based verification :

- formal manipulation
- prove properties
- performed on model

Model-based testing :

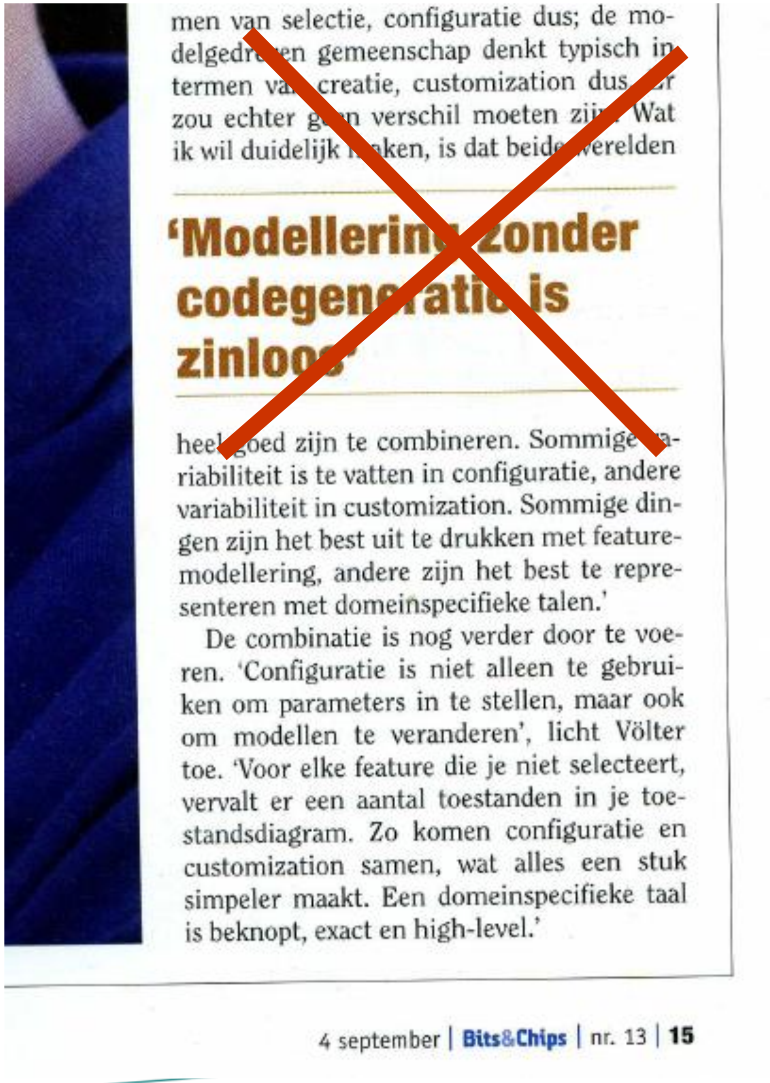
- experimentation
- show error
- concrete system



Verification is only as good as the validity of the model on which it is based

Testing can only show the presence of errors, not their absence

Code Generation from a Model



men van selectie, configuratie dus; de modelgedreven gemeenschap denkt typisch in termen van creatie, customization dus. Er zou echter geen verschil moeten zijn. Wat ik wil duidelijk maken, is dat beide werelden

'Modellering zonder codegeneratie is zinloos'

heel goed zijn te combineren. Sommige variabiliteit is te vatten in configuratie, andere variabiliteit in customization. Sommige dingen zijn het best uit te drukken met featuremodellering, andere zijn het best te representeren met domeinspecifieke talen.'

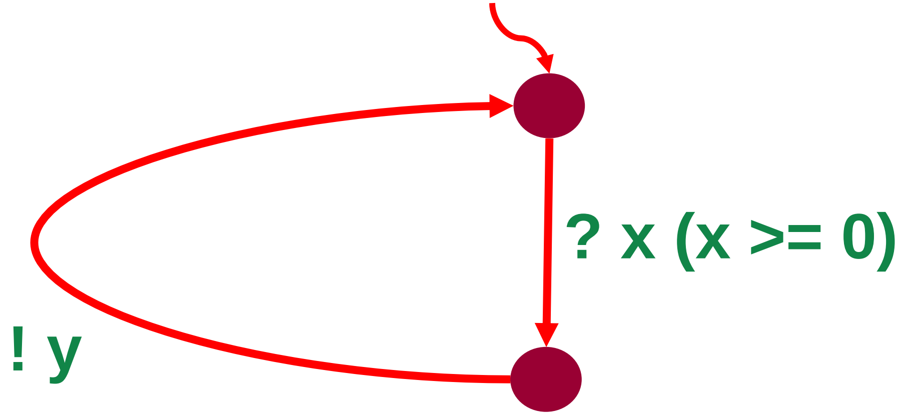
De combinatie is nog verder door te voeren. 'Configuratie is niet alleen te gebruiken om parameters in te stellen, maar ook om modellen te veranderen', licht Völter toe. 'Voor elke feature die je niet selecteert, vervalt er een aantal toestanden in je toestandsdiagram. Zo komen configuratie en customization samen, wat alles een stuk simpeler maakt. Een domeinspecifieke taal is beknopt, exact en high-level.'

4 september | Bits&Chips | nr. 13 | 15

A model is more (*less*) than code generation:

- views
- abstraction
- testing of aspects
- verification and validation of aspects

Code Generation from a Model Not Always Possible



$y \times y = x$

model of $\sqrt{x}$

- *specification of* ***properties*** *rather than construction*
- ***under-specification***
- ***non-determinism***

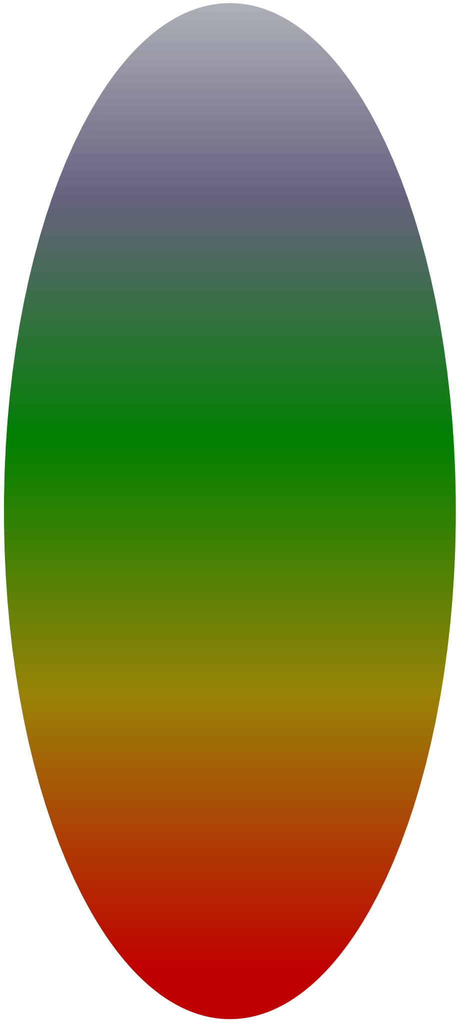
Spectrum of Models

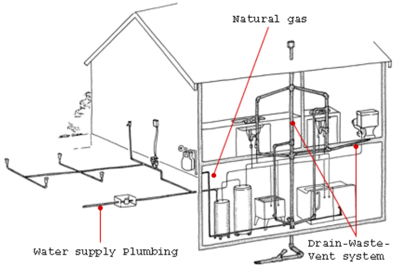
abstract (test) models

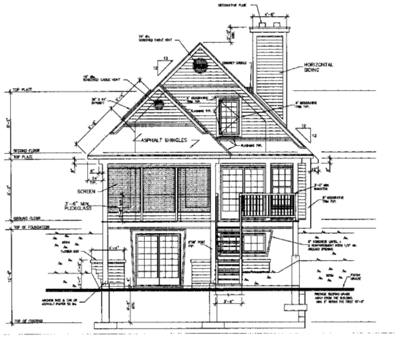
design models

virtualization

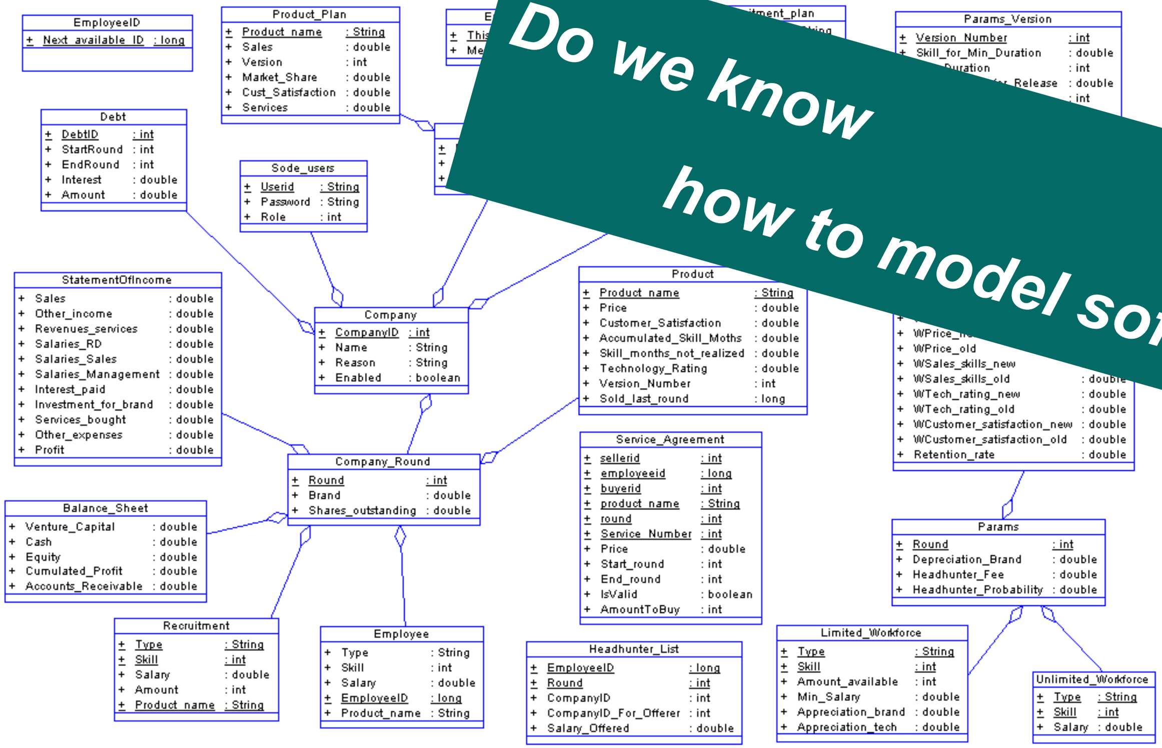
realization







A Software Model



EmployeeID
+ Next_available_ID : long

Product_Plan
+ Product_name : String
+ Sales : double
+ Version : int
+ Market_Share : double
+ Cust_Satisfaction : double
+ Services : double

Params_Version
+ Version_Number : int
+ Skill_for_Min_Duration : double
Duration : int
Release : double
: int

Debt
+ DebtID : int
+ StartRound : int
+ EndRound : int
+ Interest : double
+ Amount : double

Sode_users
+ Userid : String
+ Password : String
+ Role : int

StatementOfIncome
+ Sales : double
+ Other_income : double
+ Revenues_services : double
+ Salaries_RD : double
+ Salaries_Sales : double
+ Salaries_Management : double
+ Interest_paid : double
+ Investment_for_brand : double
+ Services_bought : double
+ Other_expenses : double
+ Profit : double

Company
+ CompanyID : int
+ Name : String
+ Reason : String
+ Enabled : boolean

Product
+ Product_name : String
+ Price : double
+ Customer_Satisfaction : double
+ Accumulated_Skill_Moths : double
+ Skill_months_not_realized : double
+ Technology_Rating : double
+ Version_Number : int
+ Sold_last_round : long

+ WPrice_old
+ WSales_skills_new
+ WSales_skills_old : double
+ WTech_rating_new : double
+ WTech_rating_old : double
+ WCustomer_satisfaction_new : double
+ WCustomer_satisfaction_old : double
+ Retention_rate : double

Company_Round
+ Round : int
+ Brand : double
+ Shares_outstanding : double

Service_Agreement
+ sellerid : int
+ employeeid : long
+ buyerid : int
+ product_name : String
+ round : int
+ Service_Number : int
+ Price : double
+ Start_round : int
+ End_round : int
+ IsValid : boolean
+ AmountToBuy : int

Balance_Sheet
+ Venture_Capital : double
+ Cash : double
+ Equity : double
+ Cumulated_Profit : double
+ Accounts_Receivable : double

Params
+ Round : int
+ Depreciation_Brand : double
+ Headhunter_Fee : double
+ Headhunter_Probability : double

Recruitment
+ Type : String
+ Skill : int
+ Salary : double
+ Amount : int
+ Product_name : String

Employee
+ Type : String
+ Skill : int
+ Salary : double
+ EmployeeID : long
+ Product_name : String

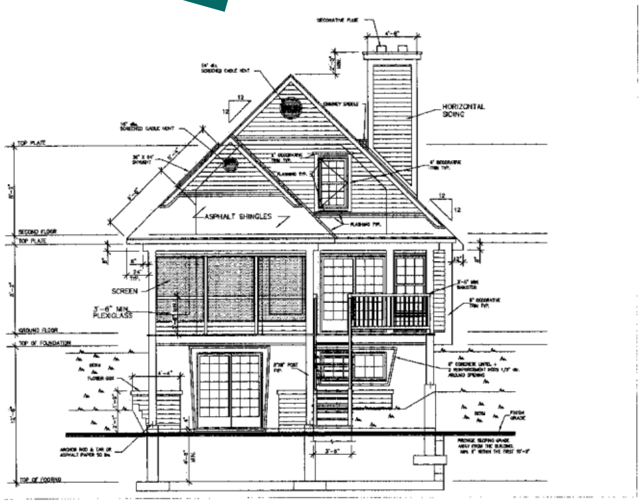
Headhunter_List
+ EmployeeID : long
+ Round : int
+ CompanyID : int
+ CompanyID_For_Offerer : int
+ Salary_Offered : double

Limited_Workforce
+ Type : String
+ Skill : int
+ Amount_available : int
+ Min_Salary : double
+ Appreciation_brand : double
+ Appreciation_tech : double

Unlimited_Workforce
+ Type : String
+ Skill : int
+ Salary : double

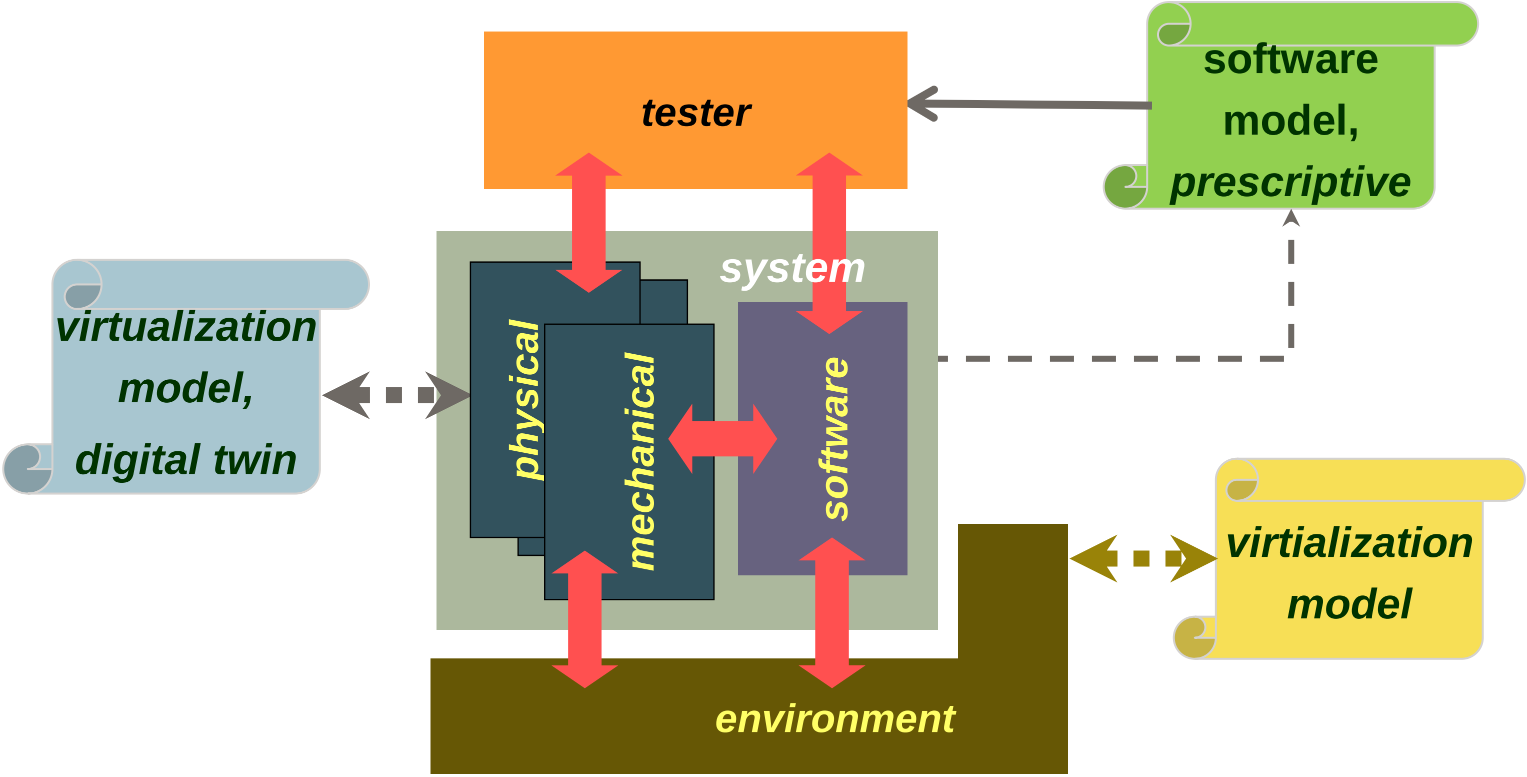
Do we know

how to model software ?



Testing with Models

Models for Testing



Model-Based Testing

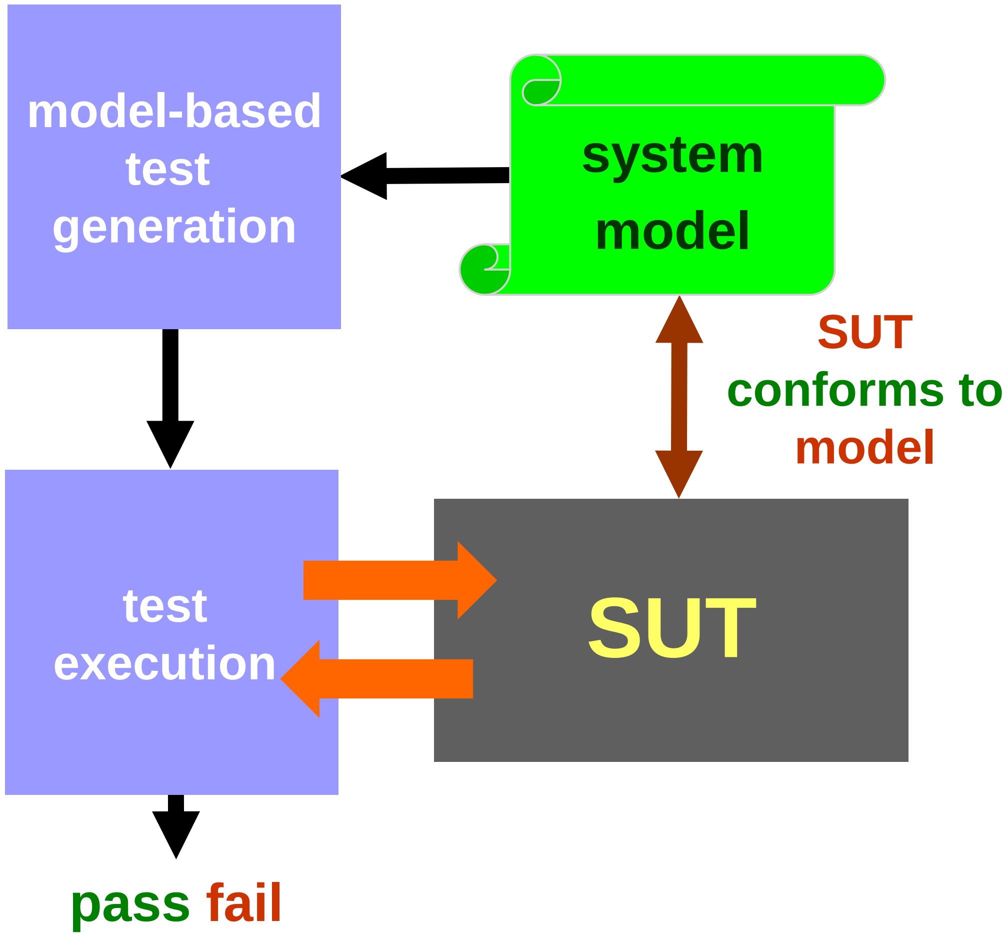
Theory

Model Based Testing

SUT conforms to model

sound ⇓ ⇑ *exhaustive*

SUT passes tests



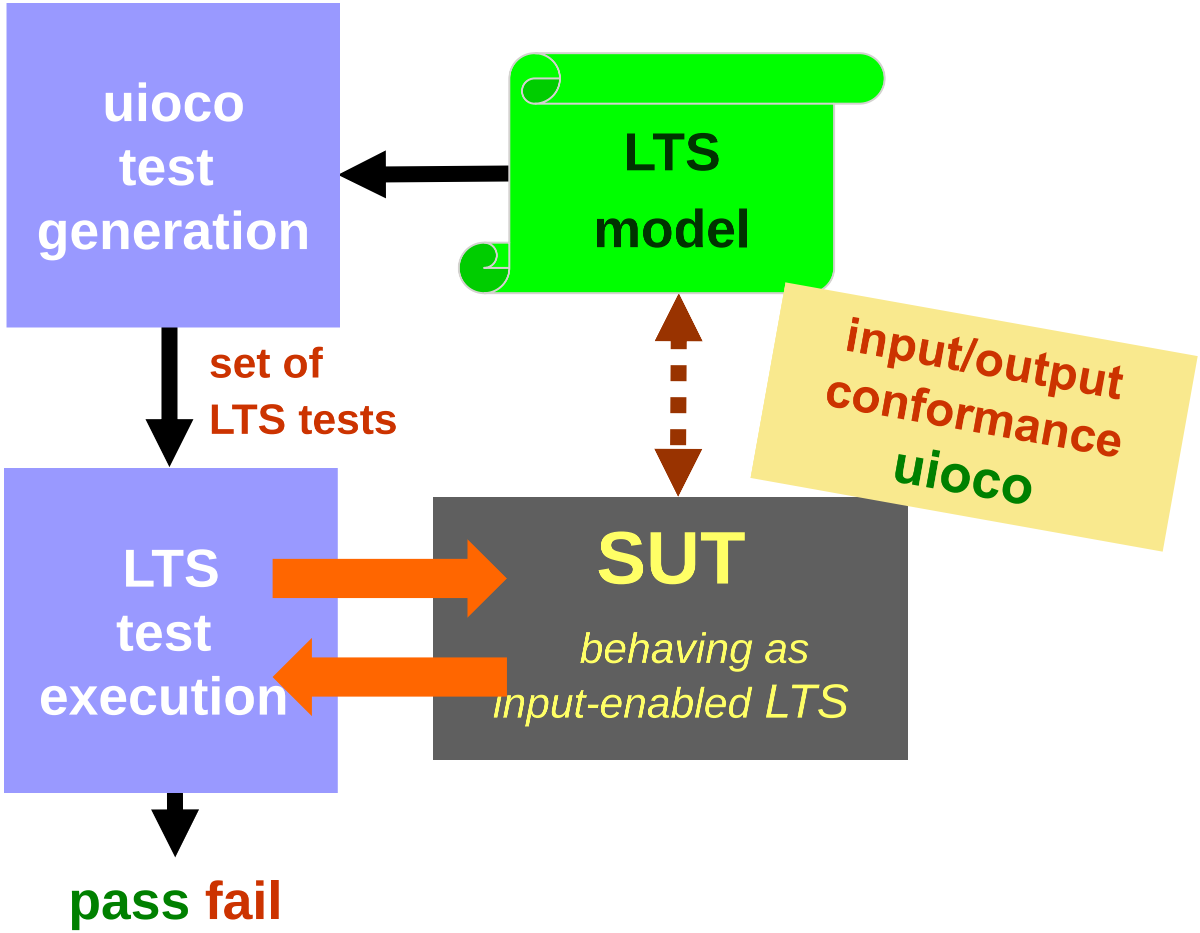
Model-Based Testing Theory with Labelled Transition Systems and **uioco**

MBT : LTS & uioco

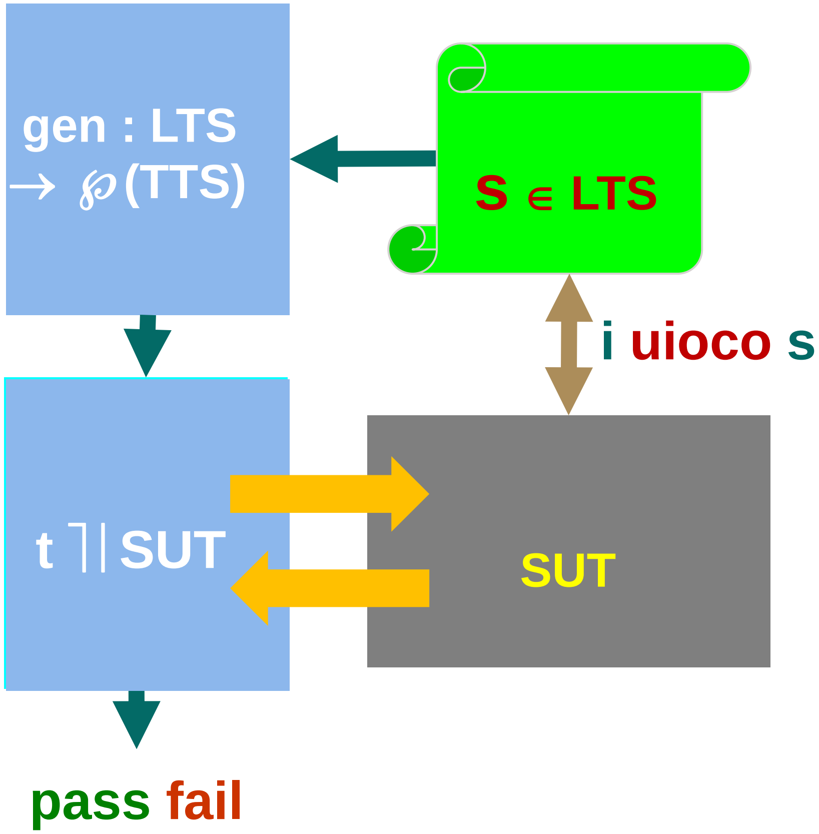
SUT uioco model

sound ⇓ ⇑ *exhaustive*

SUT passes tests



MBT with *uioco* is Sound and Exhaustive



Test assumption :

$\forall \mathbf{SUT} \in \mathrm{IMP}\ .\ \exists \mathbf{m}_{\mathbf{SUT}} \in \mathrm{IOTS}\ .$

$\forall t \in \mathrm{TESTS}\ .$

$\mathbf{SUT}$ **passes** $t \Leftrightarrow \mathbf{m}_{\mathbf{SUT}}$ **passes** t

Prove soundness and exhaustiveness:

$\forall \mathbf{m} \in \mathrm{IOTS}\ .$

$(\ \forall t \in \mathrm{gen}(\mathbf{s})\ .\ \mathbf{m}$ **passes** $t\ )$

$\Leftrightarrow$ **m uioco s**

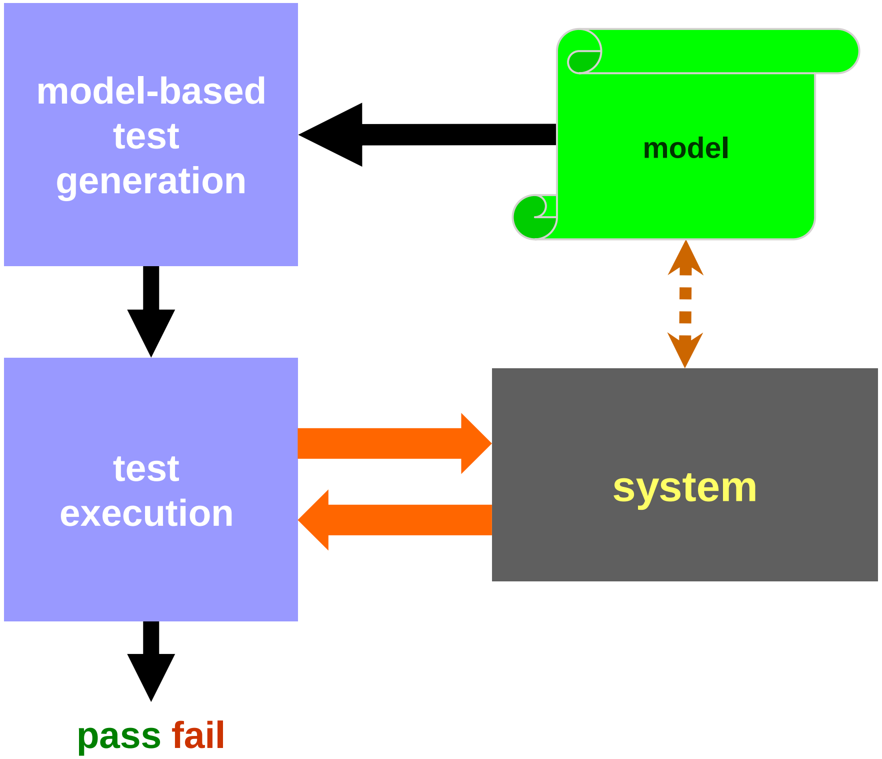
SUT comforms to s

exhaustive ⇑ ⇓ *sound*

SUT passes gen(s)

MBT

?


model-based test generation
model
test execution
system
pass fail