
Model-Based Software Development – Perspectives and Challenges

Prof. Dr. Holger Schlingloff
Fraunhofer FIRST & Humboldt Universität

hs@informatik.hu-berlin.de



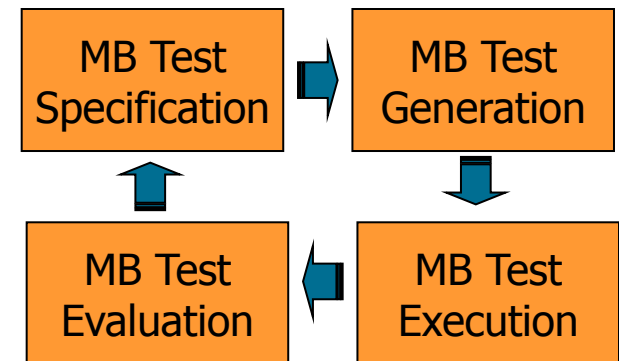
Structure of this Talk

I. Model Based Software Development

- models, roles, meanings, uses

II. Issues in MBT/MBD

- model-based test specification
- model-based test generation
- model-based test execution
- model-based test evaluation



III. Perspectives and Challenges

- pragmatic, practical and theory questions

Model-Based Design

Modeling is as old as engineering itself

Modulus = measure; unit or gauge
according to which scale the pillars of a
temple are made

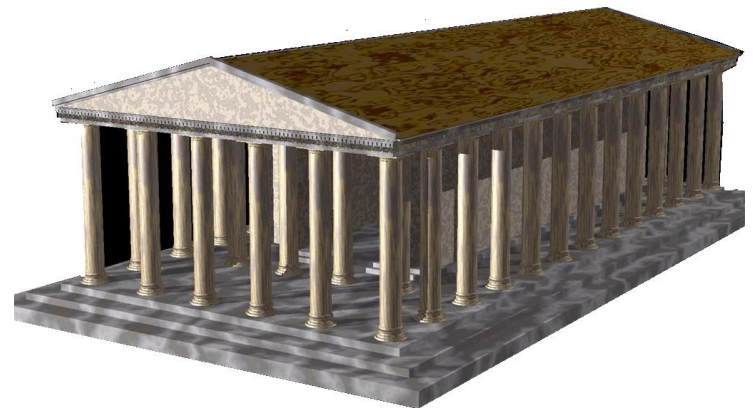
Model serves two purposes:

- reduced image of some existing thing
- antetype of something to be built

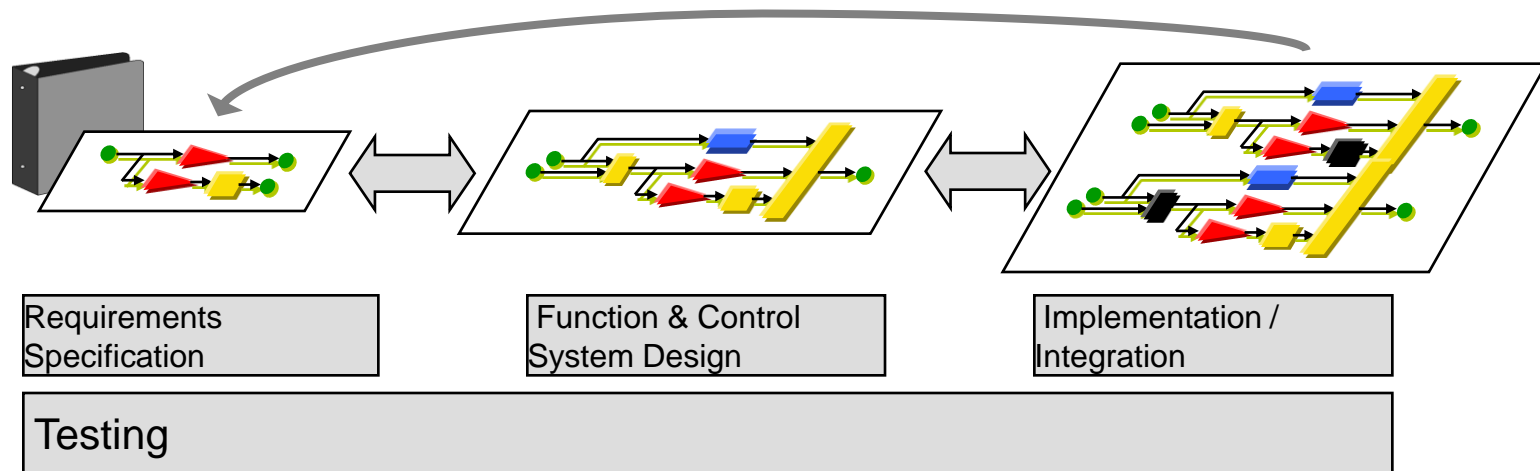
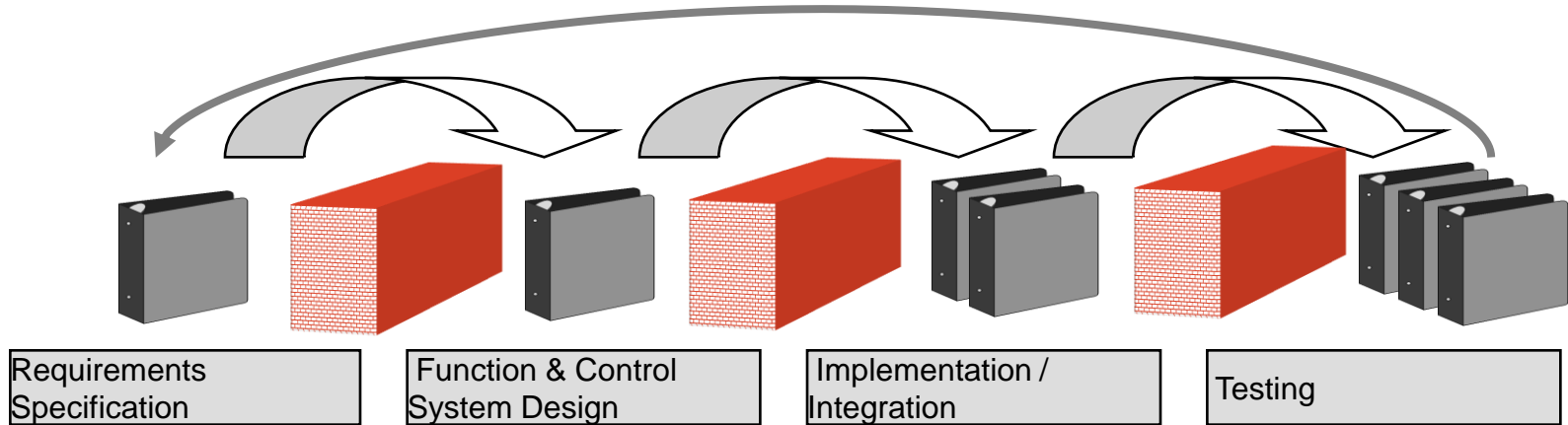
essential: reduction, purposefulness

models of software?

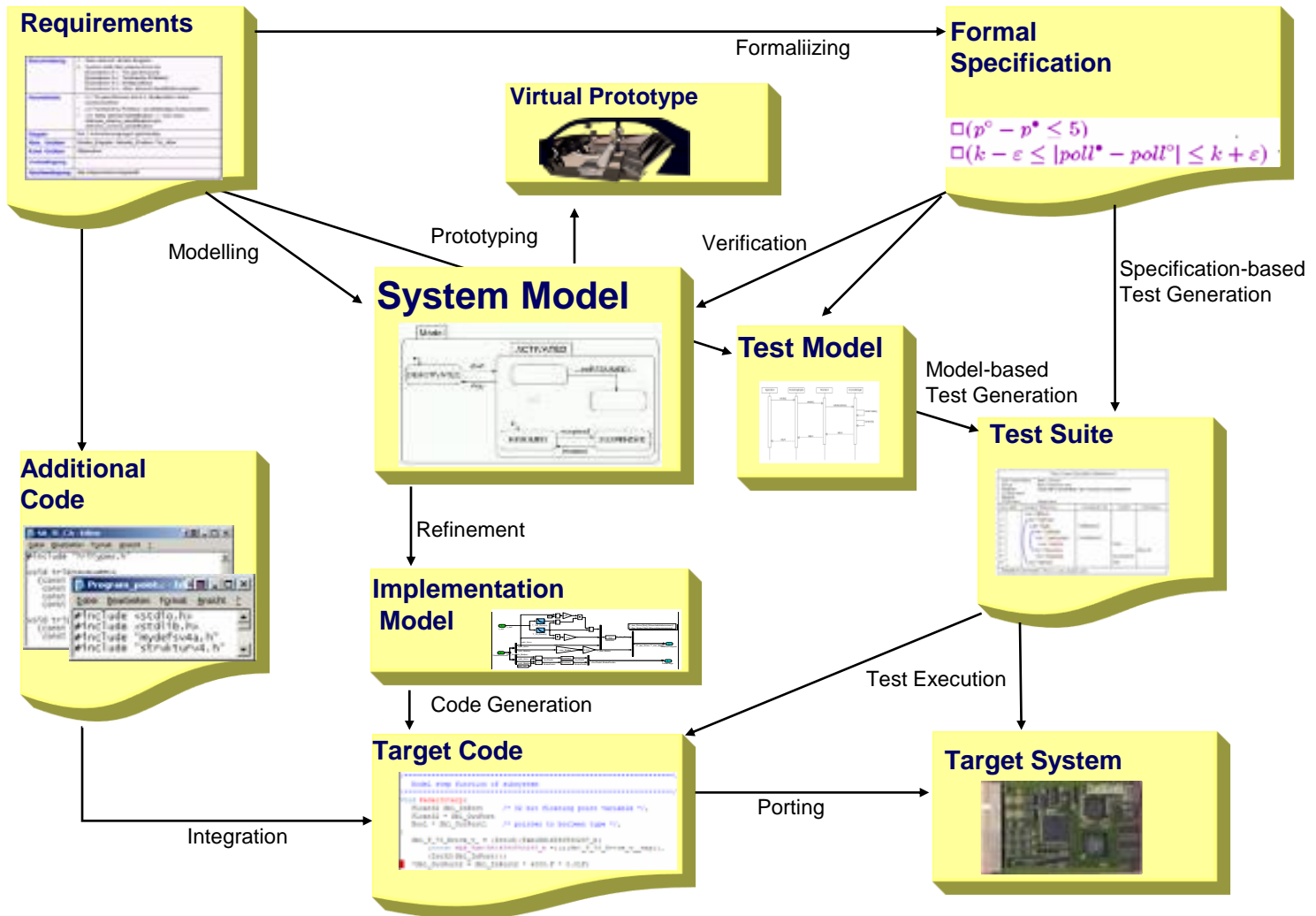
r



Model-Based Software Development



Artifacts and Activities



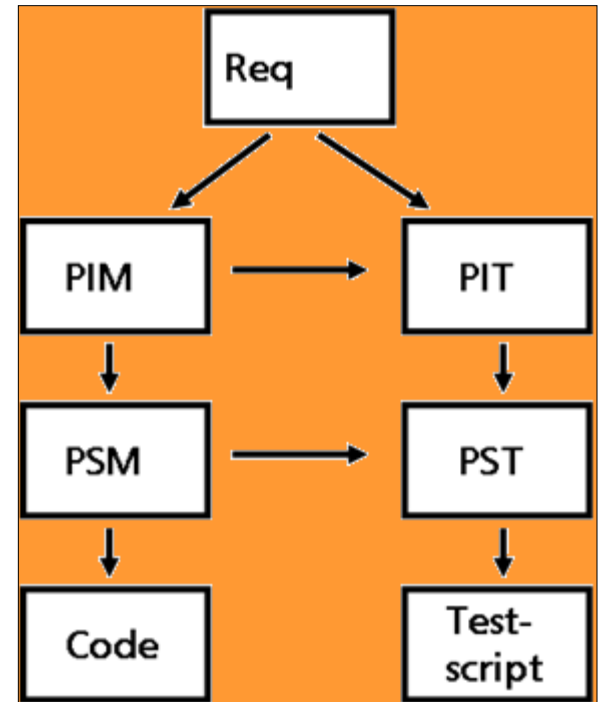
MBD and MBT

Model-based design (MBD)

- A system engineer ...
 - ... reads the specification
 - ... builds a system model
 - ...refines this to more detail
 - ... and generates an executable system

Model-based testing (MBT)

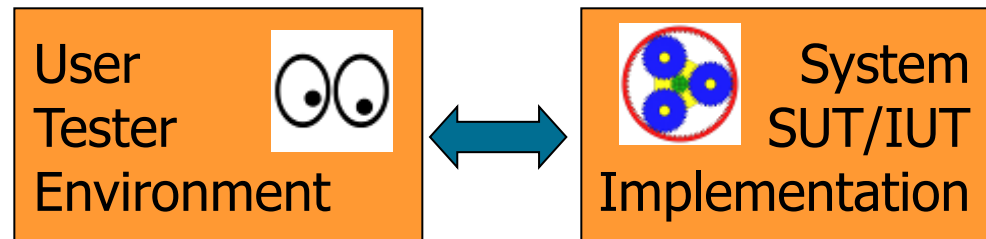
- A system tester ...
 - ... reads the specification
 - ... builds a test model
 - ... matches this to the SUT interfaces
 - ... and generates an executable test suite



Model-Based Software Testing

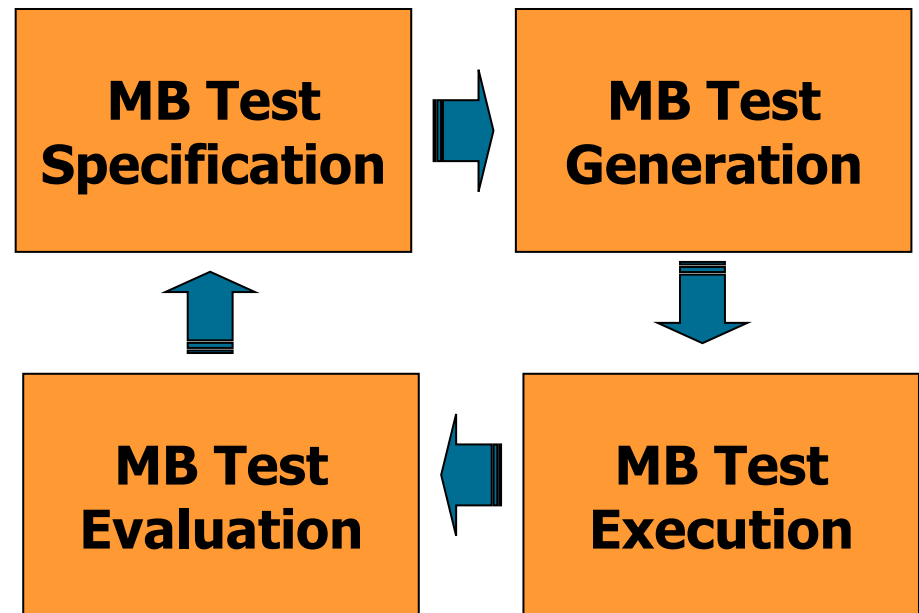
Implementation models vs. testing models

- system model
- user / environment model

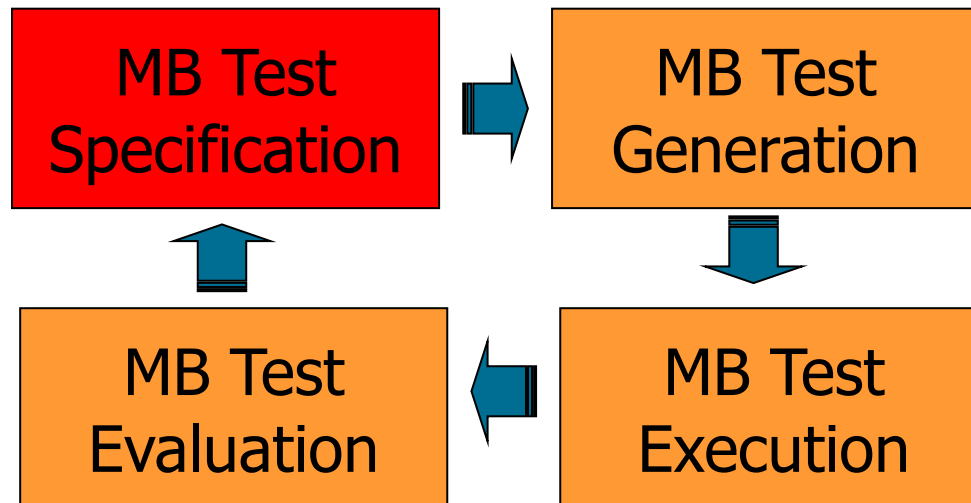


Use of Models in Design and Test

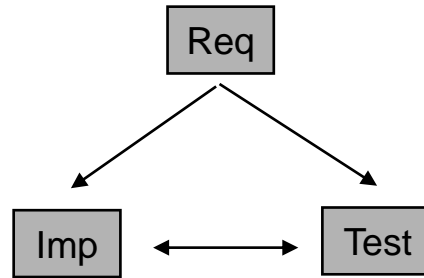
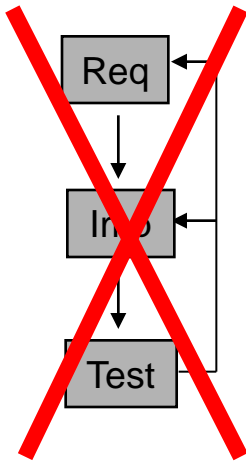
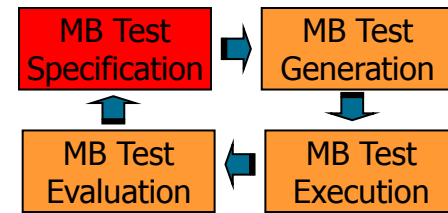
- Model as a means of **specification**, i.e., **formalizing** system requirements in models
- Model as a **source**, i.e., **generating** code and test cases from models
- Model as **object matter**, i.e., **executing** a test case on a model
- Model as **oracle**, i.e., **evaluating** tests and assigning a test verdict via a model



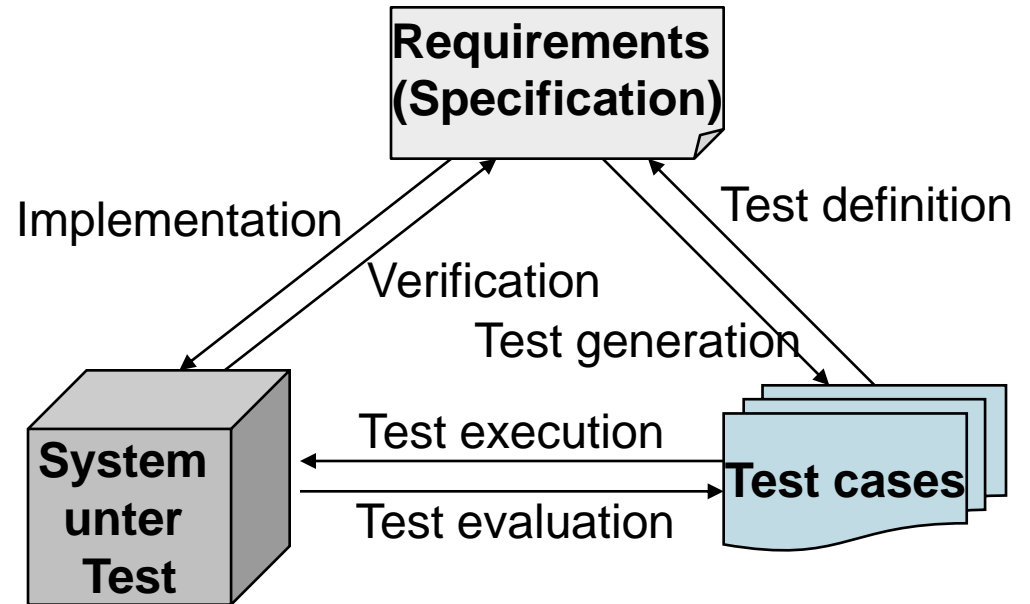
Model-Based Test Specification



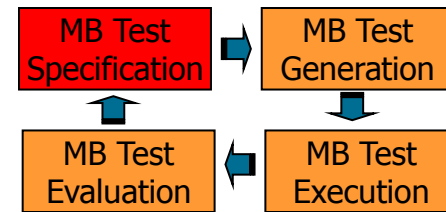
Test Specification



The validation triangle

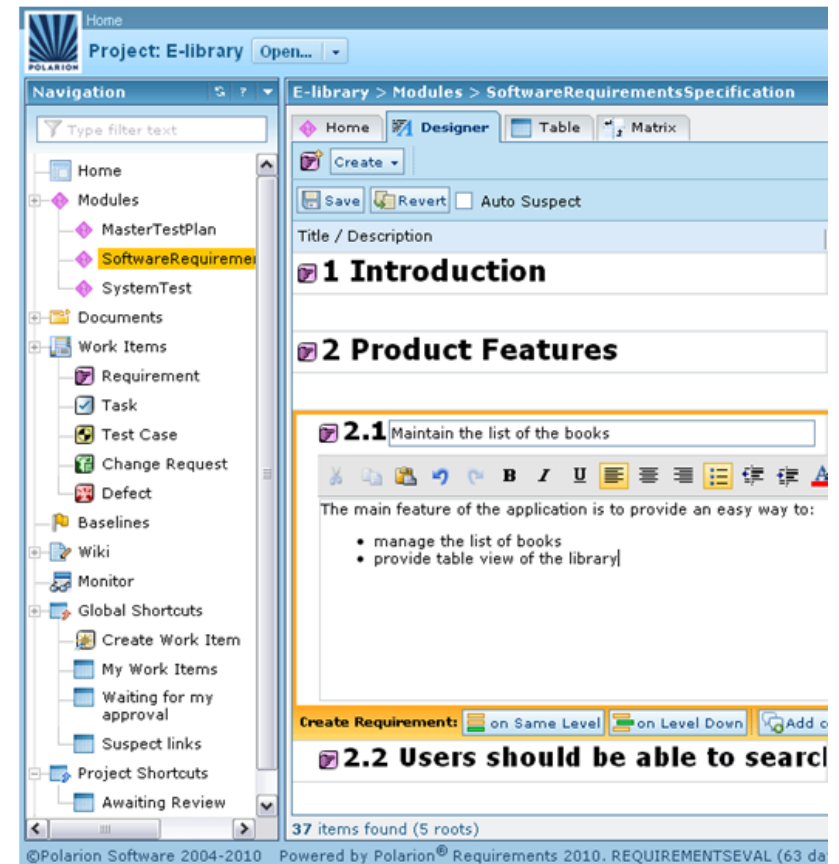


Model Development

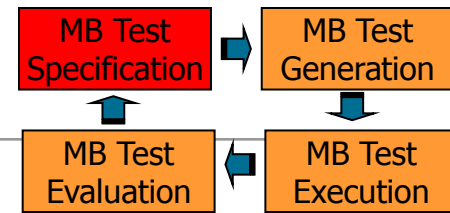


Requirements can be given as

- textual contract specification
- use case descriptions
- algebraic or logical formula
- class descriptions with pre- and postconditions
- UML state diagrams
- timed or hybrid automata
- Matlab/Simulink files,
- Code / Pseudo-Code,
- ...

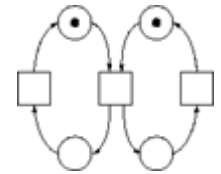


Requirements Modelling



Various formalisms are used for modelling

- classical formalisms
 - automata, Petri nets, StateCharts, MSCs, ...
- UML, OCL
- tool-oriented / domain specific modelling languages
 - Simulink, Scade, Ascet, Labview-G, (Modelica)
- logical and algebraic languages,
 - e.g. TLA, ASM/ASML, CSP, ...
- programming languages used as modelling languages
 - C#, VPL,

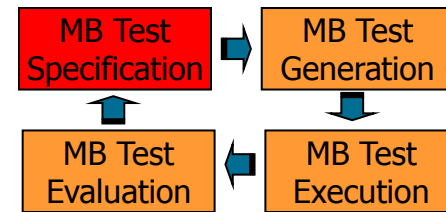


Use Case Descriptions

Requirements often are “use case descriptions” (standardized natural language)

Natural language is informal, imprecise, allows over/under-specification, ...

→ Methodology of transforming textual use case descriptions into models



2. Typical and Critical Scenarios

2.1 Brake failure while brake is closed

This scenario is part of the realization of DriveWrp.FMon.Brake operation failure [1].

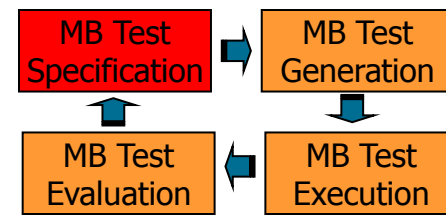
2.1.1 Preconditions

Brake is in state closing or closed, no Alarm E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or _FC_ELEVATOR_MECHANICAL_BRAKE_KB1 during previous trip occurred. (**Note:** The case of an Alarm E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or _FC_ELEVATOR_MECHANICAL_BRAKE_KB1 during a trip is handled in ch. 2.2)

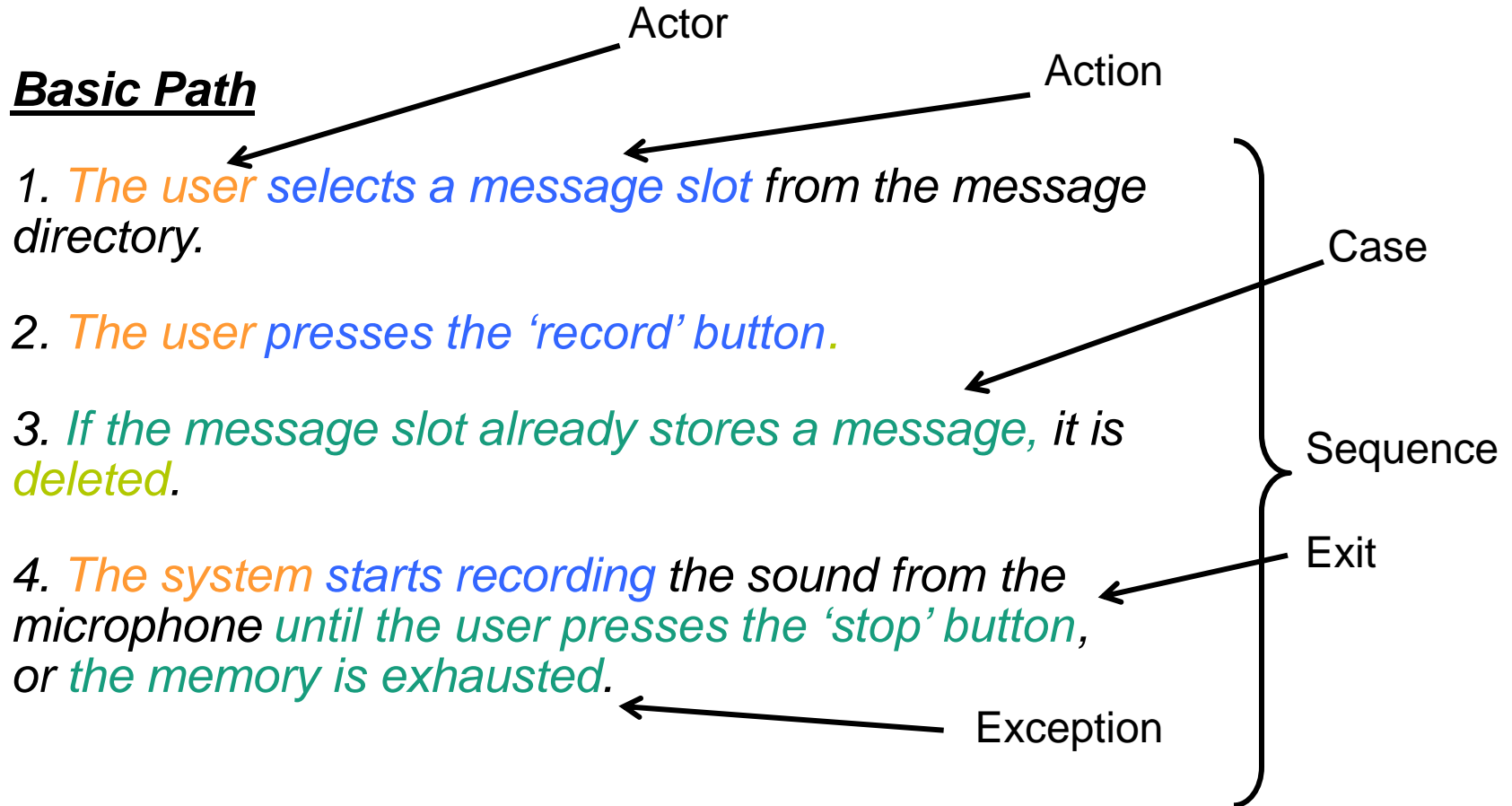
2.1.2 Basic Flow

Seq-Nr	Description	Junction
1.	FC sends error E_FC_ELEVATOR_MECHANICAL_BRAKE_KB or E_FC_ELEVATOR_MECHANICAL_BRAKE_KB1 to the Drive component	
2.	The FC blocks itself	
3.	The Drive component logs the Error "Error Brake Partial Failure"	
4.	The Drive components sets the state of the monitoring function "MOB" in the MonitoringSupervisor to "Error"	
5.	The MonitoringSupervisor logs the Status "Breakdown"	
6.	The MonitoringSupervisor activates the service ESLN in the service handler	
7.	The ServiceHandler blocks the Drive interface and opens the door.	
8.	The Drive component sets the state of the monitoring function "MOB" in the MonitoringSupervisor to "ReadyForReset"	
9.	END SysUC	

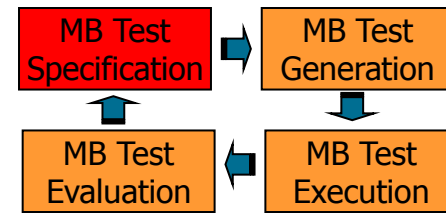
Formalization of Use Case Descriptions



Example Use Case “Start Record”



Translation into Intermediate Format



1. Identification of system interfaces
2. Identification of system functions and reactions
3. Formalization of control flow
4. Formalization of individual steps
5. Assignment of actions to steps

Step	Assigned Interaction
<input type="checkbox"/> Basic Path	◆ Basic Path
1. The use case begins when the custom...	◆ Costumer.selectPlaceOrder()
2. The system displays the Place Order s...	◆ System.displayPlaceOrderScreen()
3. The customer enters his or her name.	◆ Costumer.enterName(CName)
<input type="checkbox"/> Enter ProductCodes	◆ while not finished
4.a) The customer enters product cod...	◆ Costumer.enterPCode(PCode)
4.b) The system adds the price of the ...	◆ System.addPriceToTotal(PCode,Total)
5. The system displays the total price.	◆ System.displayPrice(Total)

User.selectMessageSlot(Slot)

User.startRecording()

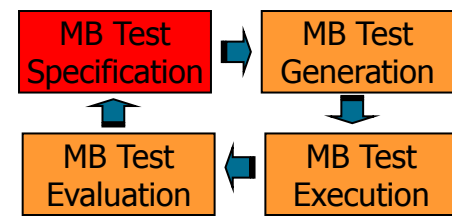
if (Slot.full())

 System.deleteMessageSlot(Slot)

Exception (System.exhaustMemory() || User.stopRecording())

 System.recordMessage(Slot)

Use Case Validator



Use Case Validator Perspective - SoundRecorder.ucf - Eclipse Platform

File Edit Navigate Search Project ATL Editor Uvalidator Editor Run Window Help

generate import

Use Case Vali...

Navigator

- SoundRecorder2
 - .project
 - model.xml
 - out.uml2
 - SoundRecorder.di
 - SoundRecorder.uc
 - SoundRecorder.ucf
 - SoundRecorder.uml2
- SoundRecorder3

SoundRecorder.ucf

Step	Assigned Interaction
Basic Path	Basic Path
1. The user selects a ...	User.selectMessageSlot(Slot)
2. The user presses t...	User.startRecording()
unnamed Switch	Slot.full()
If	If
3. If the messa...	System.deleteteMessageSlot(Slot)
4. The system starts r...	System.recordMessage(Slot)

Action View

- System
 - recordMessage(SlotNo: Integer)
 - deleteteMessageSlot(SlotNo: Integer)
 - exhaustMemory()
- User
 - startRecording()
 - selectMessageSlot(SlotNo: Integer)
 - stopRecording()

Binding View

Parameter	Type
no parameters	no parameters

Outline

- Record a Message
 - Basic Path
 - 1. The user selects a m
 - 2. The user presses the
 - unnamed Switch
 - 4. The system starts re

Console

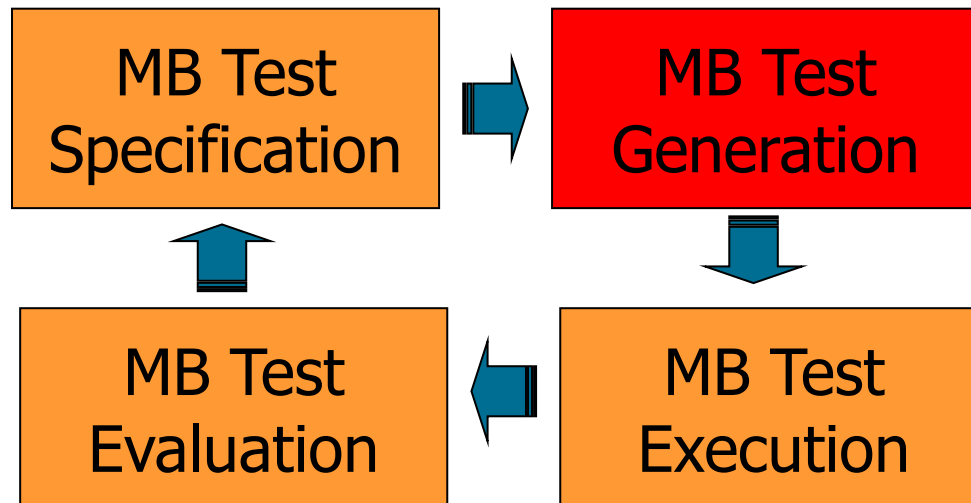
```

org.atl.eclipse.adt.editor.console
Starte Generierung des .xml ...
Temporäres UCV-Modell erstellt: platform:/resource/SoundRec
Generierung fertig
platform:/resource/SoundRecorder/SoundRecorder.uml2
Generierung fertig, Ausgabemodell: /resource/SoundRecorder/
  
```

Variable View

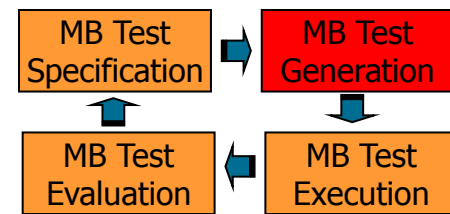
```
Slot: Integer
```

Model-Based Test Generation



Automated Test Generation...

- ... is of considerable industrial interest ...
- ... has been investigated for some time now ...
- ... is often used synonymously with “model-based testing”

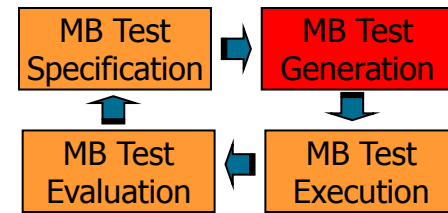


Main dimensions

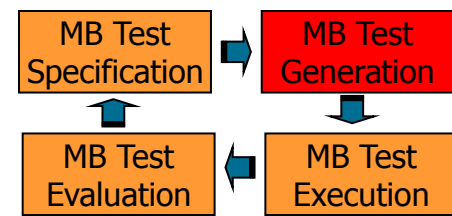
- supported modelling language
 - FSM/LTS, B, Z, CSP, UML, ML/SL/SF, C#, ...
- supported coverage criteria
 - All-States, All-Transitions, Modified Condition / Decision Coverage (MCDC)
 - data-flow criteria, mutation analysis
 - uniformity hypothesis / complete test suites
- test construction method
 - guided depth-first search, transition tour algorithm, model checking, SAT, constraint solving, theorem proving, search-based, ...

Automated Test Generation Tools

Mostly from UML State Machines
 More than a dozen commercial and experimental research tools available
 Comparison e.g. by mutation analysis



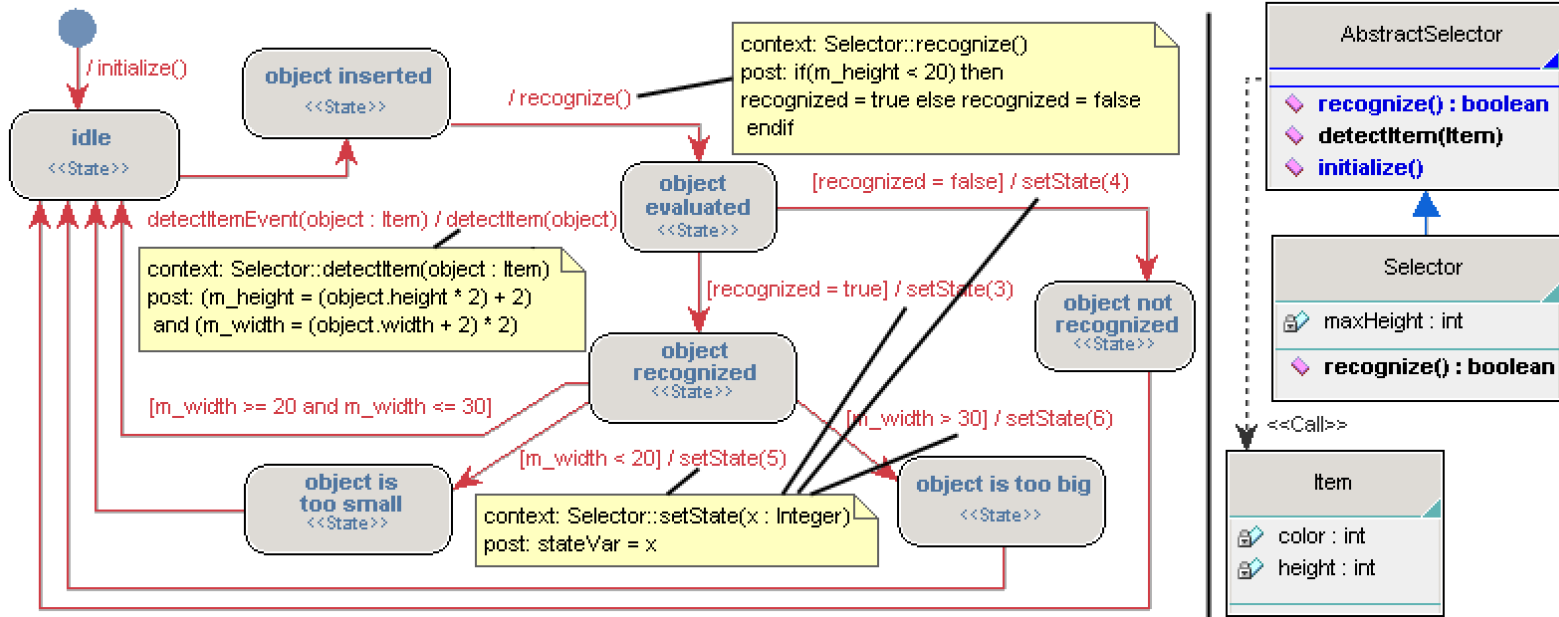
Coverage Criterion	Killed Mutants
state coverage	124 / 268
decision coverage	239 / 268
masked MC/DC	256 / 268
MDMCC	268 / 268

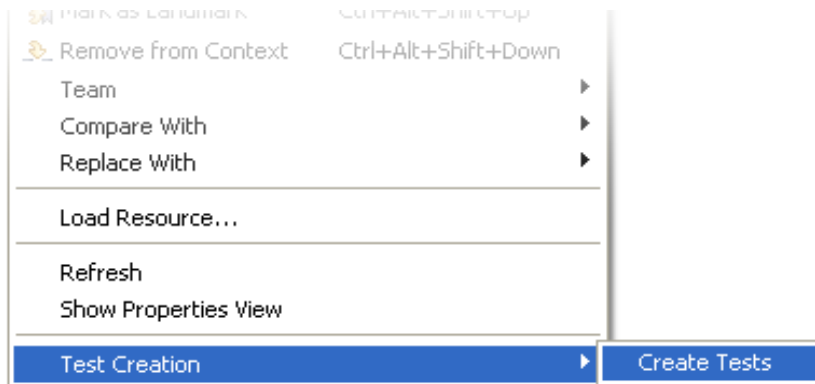
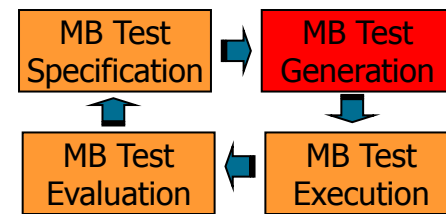


Since we didn't like the technologies (and the prizes), we built our own...

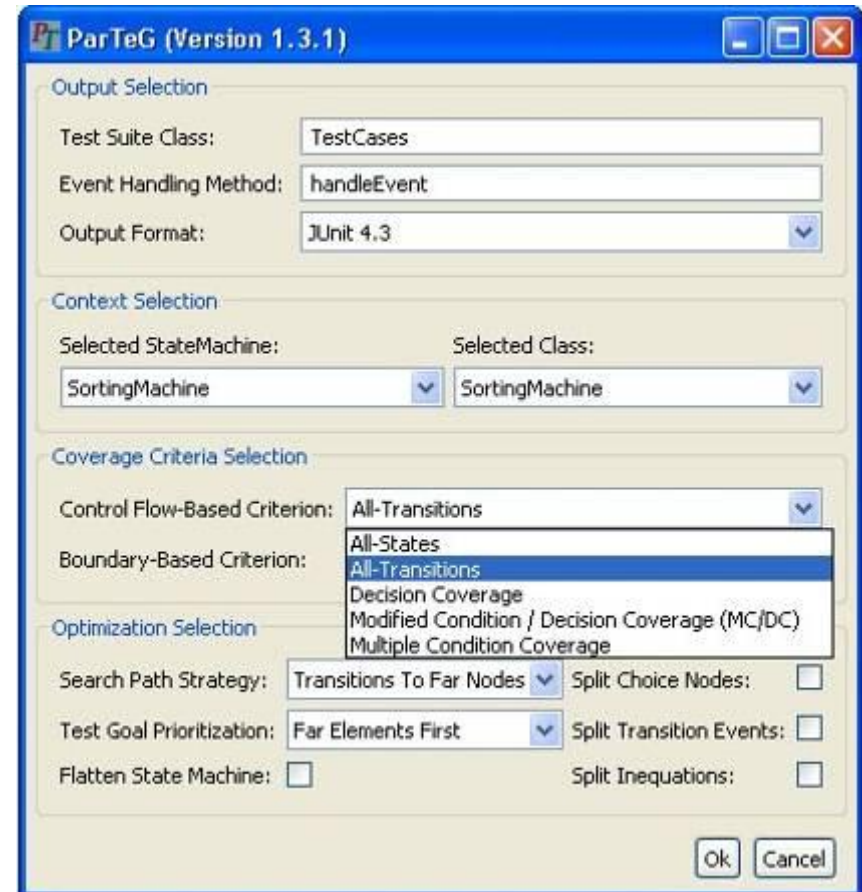
– <http://parteg.sourceforge.net/>

UML class & state transition diagrams, connected by OCL Plugin for Eclipse, supports XML import / export

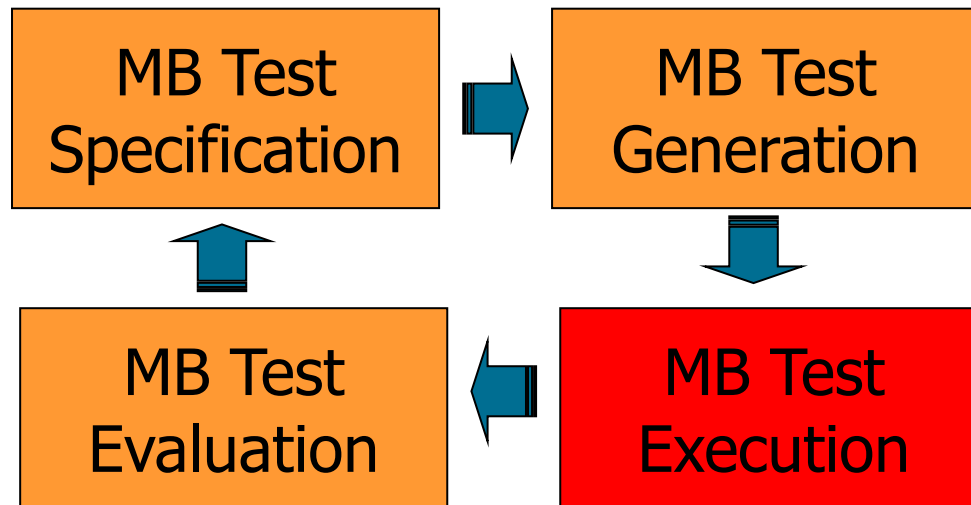




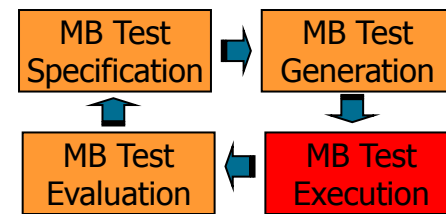
- use in industrial projects (ETCS railway switch)
- backward search algorithm with OCL constraint solving
- test suite quality evaluation experiments
- testability transformations on models



Model-Based Test Execution



Quality of Test Suites



Test execution can be expensive

- large test suites
- resetting of SUT
- actual road testing

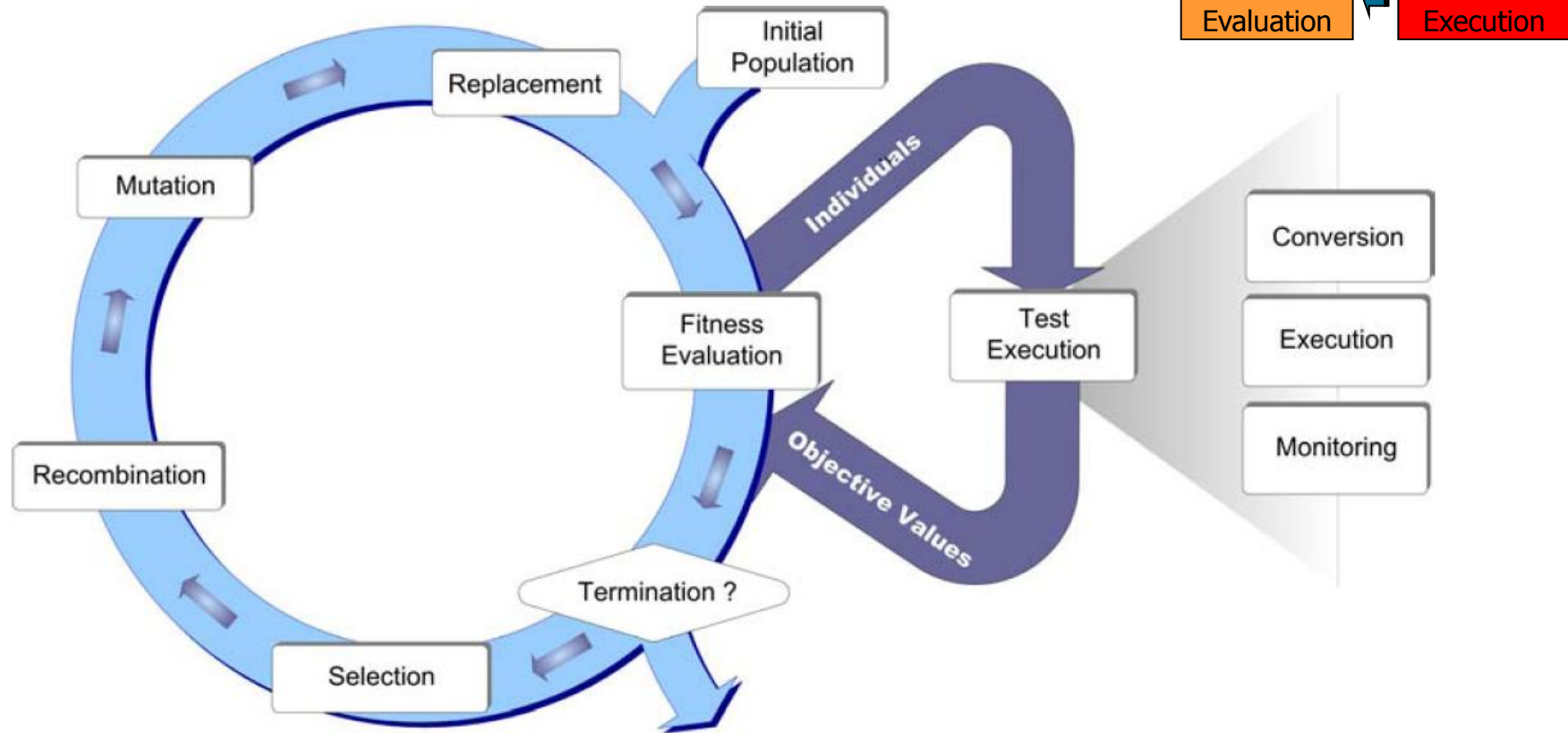


Solution: testing systems models

- how to find an “optimal” test suite minimizing the testing efforts?
- ➔ use genetic algorithm for heuristic search



Evolutionary Test Generation

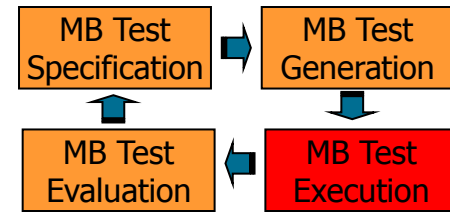


Test case: vector of continuous signals

Test execution via Simulink

Fitness user-defined C-function

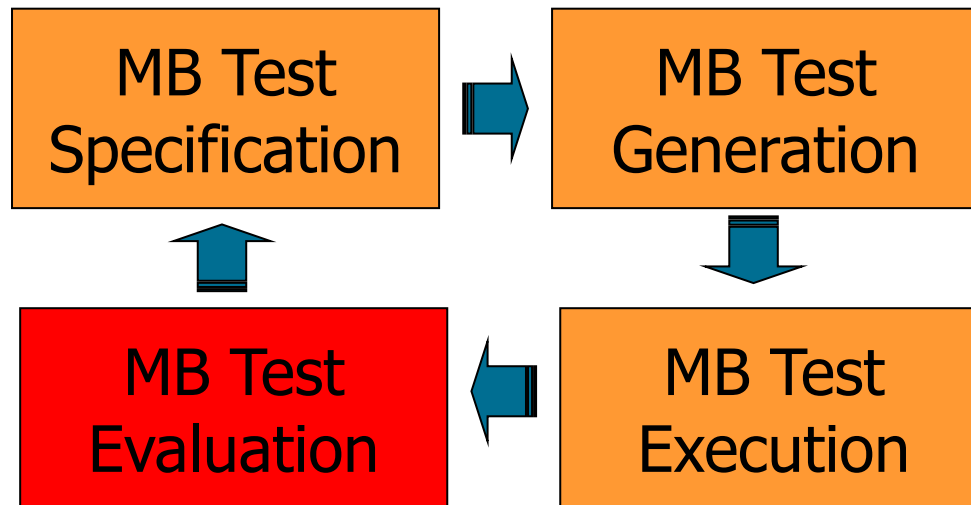
EvoTest Framework



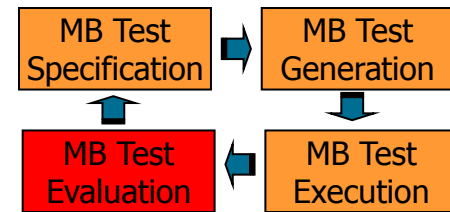
The screenshot shows the Eclipse IDE with the following components:

- Project Explorer:** Shows a project named 'TriangleExample' with a 'src' folder containing 'test22.c' and 'tri_type(short, short, short)'. There are also several test case files.
- Code Editor:** Displays the source code for 'int tri_type(short a, short b, short c)'. The code includes conditional logic for determining the type of a triangle based on side lengths. Some lines are highlighted in green and yellow.
- Outline:** Shows the function signature 'tri_type(short, short, short) : int'.
- Fitness Evaluation Chart:** Located at the bottom, it plots 'Fitness value' (y-axis, 0 to 2.2) against 'Generations' (x-axis, 1 to 14). Three lines represent 'Average Fitness' (green), 'Maximum Fitness' (blue), and 'Minimum Fitness' (red). The red line drops sharply from approximately 2.1 to 0.0 by generation 4. The green line decreases from 2.2 to 0.0 by generation 11. The blue line remains relatively stable around 2.2.

Model-Based Test Evaluation



The CSP-CASL Modelling Language



CSP: communicating sequential processes

- established formalism to describe concurrent systems
- ongoing research foundations; applications in industry

CASL: common algebraic specification language

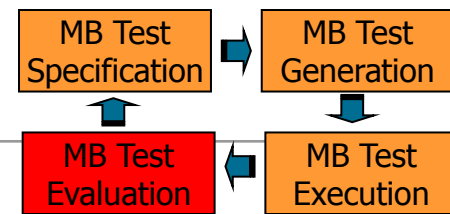
- de-facto standard in algebraic specification community
- well-suited for modelling data

CSP-CASL: combining process algebraic and algebraic specifications

- comparable: Lotos, ACP, ...

ccspec S_p = data D channel Ch process P end

Refinement



One of the most widely investigated formal notions
(„opposite“ of abstraction)

OMG: „A specialized state machine is an extension of the general state machine, in that regions, vertices, and transitions may be added or redefined (replaced by another state machine with at least the same entry/exit points)“

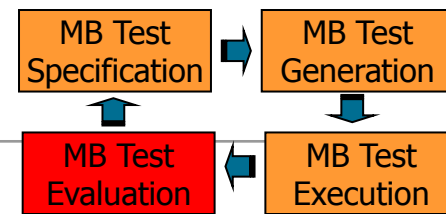
CSP-CASL: Refinement = Reduction of nondeterminism

- Replacement of abstract data types by concrete ones
- Adding algebraic laws to restrict executions



Theorem: Refinement preserves test cases

Enhancement



Very few systems are developed „from scratch“
Enhancement as a first step to evolution

Sp' is an *enhancement* of Sp ($Sp \gg Sp'$) iff

- the signature of Sp can be embedded into Sp'
- data models of Sp' are consistent with Sp
- traces of Sp are preserved in Sp' , and
- failures of Sp in its signature are also failures of Sp'

E.g., extending a specification by additional data or external choice leads to an enhancement



Theorem: Enhancement preserves test results (red / yellow/ green)

Perspectives and Challenges

- pragmatic
- practical
- theory

Challenges for Model-Based Software Development (1)

Pragmatic questions

(making the technology “smooth”):

- Requirements, formalisms
- Embedded systems
 - setting up testing interfaces
 - setting up environment models
- Eliminating “semantic variation points”
- DSLs, model transformations

“Modelling maturity levels”:

- **Level 0: No Specification**
- **Level 1: Textual Specification**
- **Level 2: Text with Models**
- **Level 3: Models with Text**
- **Level 4: Precise Models**
- **Level 5: Models only**

* Anneke Kleppe, Jos Warmer: MDA Explained



Challenges for Model-Based Software Development (2)

Practical questions:

- Parallelism
 - multicore / distribution
 - dealing with limited information
- Diagnosis / Debugging
 - monitoring / passive testing
- Quality of Testing
 - when to stop
 - fault detection capabilities



Challenges for Model-Based Software Development (3)

Theory questions:

- Modelling evolution
 - product lines, concerns
 - change, adaptability
- Modelling intent
 - agents, self-organizing systems
 - security modelling



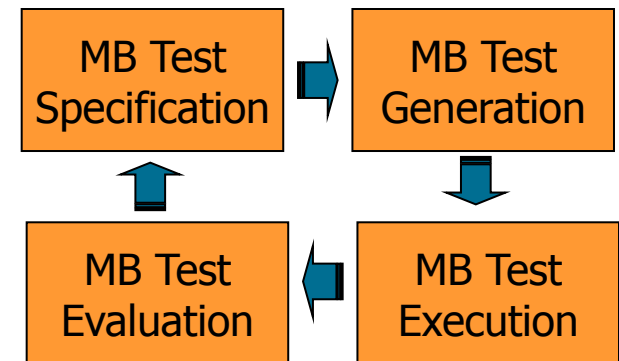
THANK YOU FOR YOUR ATTENTION!

I. Model Based Software Development

- models, roles, meanings, uses

II. Issues in MBT/MBD

- model-based test specification
- model-based test generation
- model-based test execution
- model-based test evaluation



III. Perspectives and Challenges

- pragmatic, practical and theory questions