# Testing Against Requirements Using UML Environment Models

Denis Hatebur, Maritta Heisel

{denis.hatebur, maritta.heisel}@uni-due.de

Thomas Santen and Dirk Seifert

{santen,seifert}@cs.tu-berlin.de,

dirk.seifert@loria.fr

Universität Duisburg-Essen
Fakultät Ingenieurwissenschaften, Fachgebiet Software Engineering

Technische Universität Berlin
Fakultät Elektrotechnik und Informatik, Fachgebiet Softwaretechnik

LORIA – Université Nancy 2, France

ITESYS Institut für technische Systeme GmbH, Dortmund

June 5, 2008

## Our Test Approach I

### Problem

- model-based software development: set up models of software systems to be constructed
- models should not be used for code generation as well as for testing

### Our Approach

- test against requirements, not against specification
- explicitly model the environment in which the software will operate
- use UML state machines with tool support
- consider reactive/embedded systems

## Our Test Approach II

### On-the-fly testing

- generating and executing test cases are intertwined

+ no state explosion

– can start if implementation is available

### Batch testing

- test cases are generated and stored for later execution

+ early test preparation possible

– all possible behavior variants must be computed

## Terminology (Jackson) I

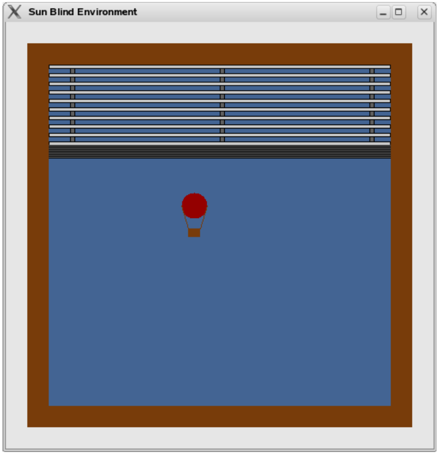
| | |
|---|---|
| Machine | thing we are going to build; may consist of software and hardware |
| Environment | part of the world where the machine will be integrated |
| System | consists of machine and its environment; consists of domains |
| Requirements | *optative* statements; describe how the *environment* should behave when the machine is in action |
| Specification | *implementable* requirements; describe the machine at its external interfaces; are basis for its construction |

## Terminology (Jackson) II

| | |
|---|---|
| Domain knowledge | *indicative* statements; consist of facts and assumptions: |
| Facts | describe what holds in the environment, no matter how we build the machine |
| Assumptions | describe things that cannot always be guaranteed, but which are needed to fulfill the requirements, e.g., rules for user behavior |

Domain knowledge is needed to derive specifications from requirements: $S \wedge D \Rightarrow R$, $D \equiv F \wedge A$

# Running Example: SunBlindControl



The sunblind can manually be lowered or pulled up.

The sunblind is automatically lowered on sunshine for more than one minute.

The sunblind should not be destroyed by heavy wind.

The environment consists of user, sunblind, sun and wind.

# Transforming a Requirement into a Specification I

*R*1 The sunblind is not destroyed by wind.

*F*1 Heavy wind for more than 30 sec is destructive.

*A*1 Heavy wind for less than 30 sec is not destructive.

*F*2 If the sunblind is up, it cannot be destroyed by wind.

*R*1′ The sunblind is up if there is heavy wind for more than 30 sec.

$$F1 \wedge A1 \wedge F2 \wedge R1' \Rightarrow R1$$

## Transforming a Requirement into a Specification II

- $F3$ It takes less than 30 sec to pull up the sunblind.
- $R1''$ If there is heavy wind and the sunblind is not up, it is pulled up.

$$F3 \wedge R1'' \Rightarrow R1'$$

- $F4$ There is heavy wind if and only if the wind sensor generates more than 75 pulses per sec.
- $F5$ Turning the motor left pulls up the sunblind.
- $S1$ If the wind sensor generates more than 75 pulses per sec and the last signals to the motor have not been *turn left*, followed by *motor left blocked* and *stop motor*, then the *turn left* signal is sent to the motor.

$$F4 \wedge F5 \wedge S1 \Rightarrow R1''$$

$$F1 \wedge F2 \wedge F3 \wedge F4 \wedge F5 \wedge A1 \wedge S1 \Rightarrow R1$$

## Testing Against Requirements vs. Specifications

**Testing against the specification**

- Does the machine generate the *turn left* signal?
- If the specification was not correctly derived from the requirements, the SUT passes the test nevertheless.

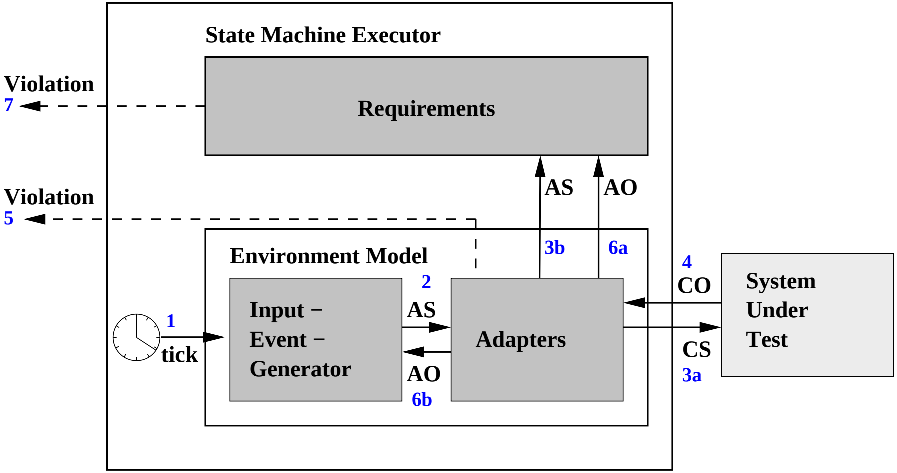
**Testing against the requirements**

- Does the sunblind enter a state where it would be destroyed?
- Detects errors made in transforming requirements into specifications.
- Checks if customer needs are satisfied (acceptance test).

# Testing with Environment Models

- the environment is modeled using UML state machines
- this model explicitly contains the facts and assumptions about the environment
- the environment model consists of the input event generator and adapters:
    - the input event generator produces abstract events
    - adapters transform abstract events such as *pull up sun blind* into concrete ones, such as *turn motor left*.
- the requirements are expressed as state machines that check whether a requirement is violated

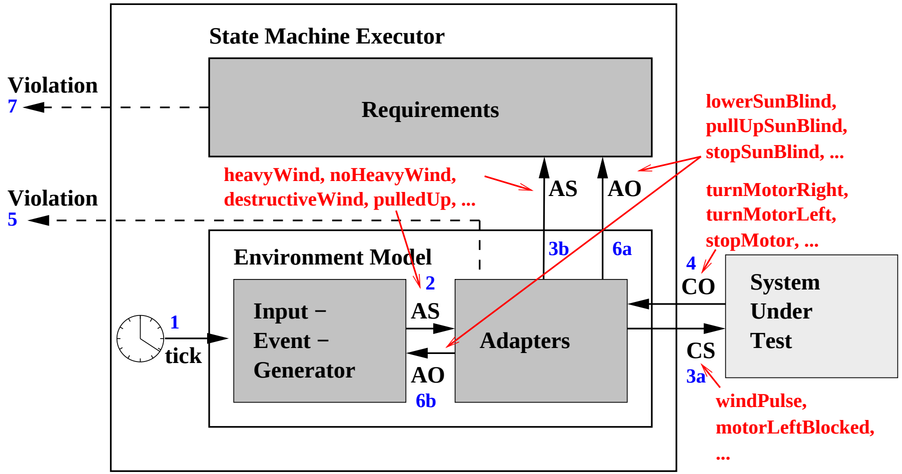
# “On the Fly” - Test System Architecture



CO: Concrete Observation
AO: Abstract Observation
CS: Concrete Stimulus

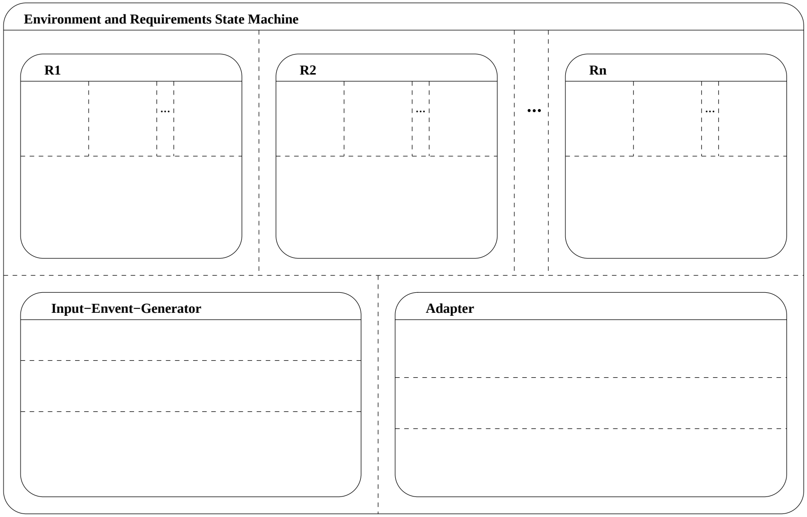
AS: Abstract Stimulus
tick: Request for new Abstract Stimulus
Violation: Test Result

# “On the Fly” - Test System Architecture: Sun Blind

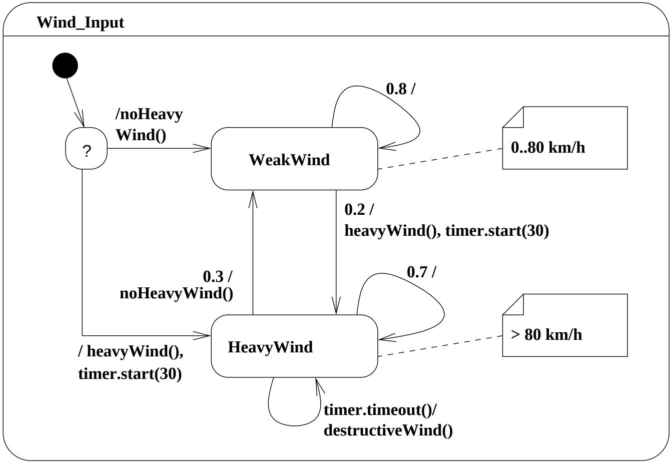


| | |
|---|---|
| CO: Concrete Observation | AS: Abstract Stimulus |
| AO: Abstract Observation | tick: Request for new Abstract Stimulus |
| CS: Concrete Stimulus | Violation: Test Result |

# Pattern for Requirements and Environment Model



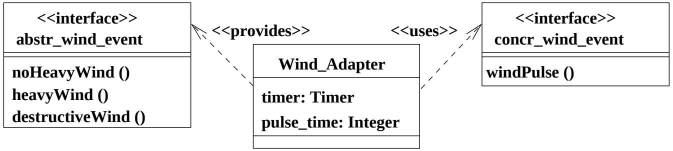
# Input Event Generator: Wind Example

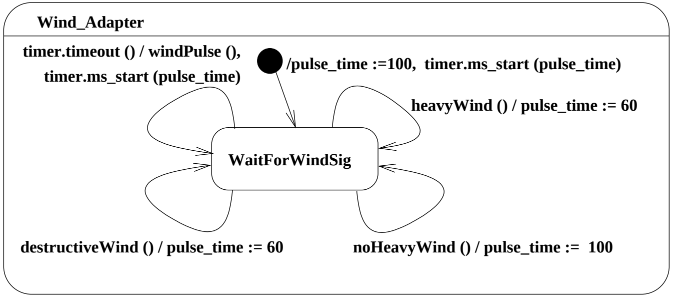


Wind_Input
?
/noHeavy Wind()
WeakWind
0.8 /
0..80 km/h
0.2 / heavyWind(), timer.start(30)
0.3 / noHeavyWind()
0.7 /
HeavyWind
> 80 km/h
/ heavyWind(), timer.start(30)
timer.timeout()/ destructiveWind()

Assumption *A*1 is modeled explicitly.
Probabilities are given for the different transitions.

## Adapter: Abstract to Concrete Stimuli

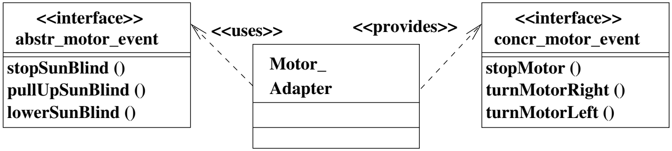


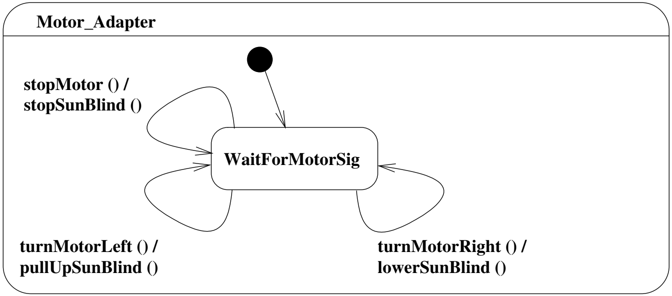


Wind_Adapter
timer.timeout () / windPulse (), timer.ms_start (pulse_time)
/pulse_time :=100, timer.ms_start (pulse_time)
heavyWind () / pulse_time := 60
WaitForWindSig
destructiveWind () / pulse_time := 60
noHeavyWind () / pulse_time := 100

The adapter transforms the abstract stimuli into concrete ones.

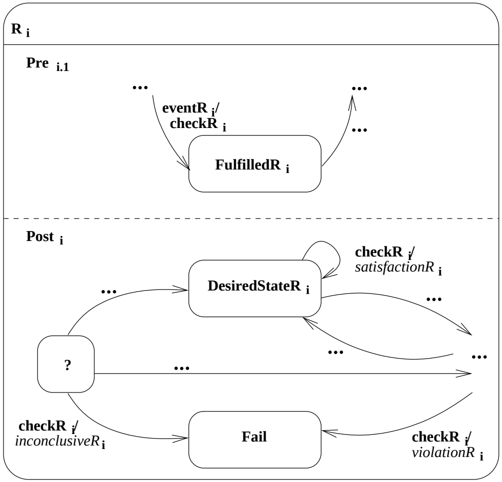
## Adapter: Concrete to Abstract Observations





The adapter transforms concrete observations into abstract ones.

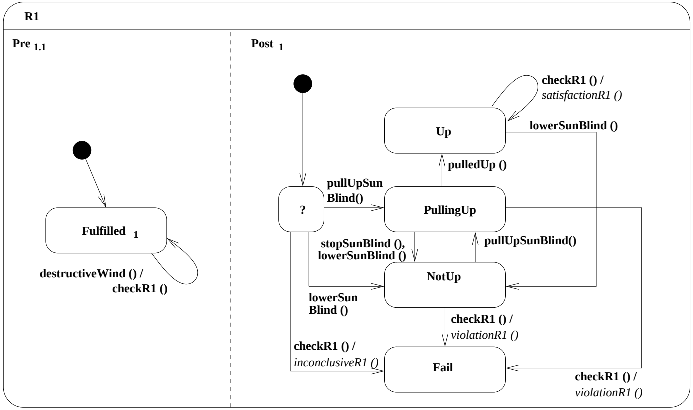
# Pattern for Modeling Requirements



The postcondition of the requirement is checked when all preconditions are fulfilled.

“When [*eventR*$_i$] happens, [controlled domain] should be in [*desiredStateR*$_i$]”.

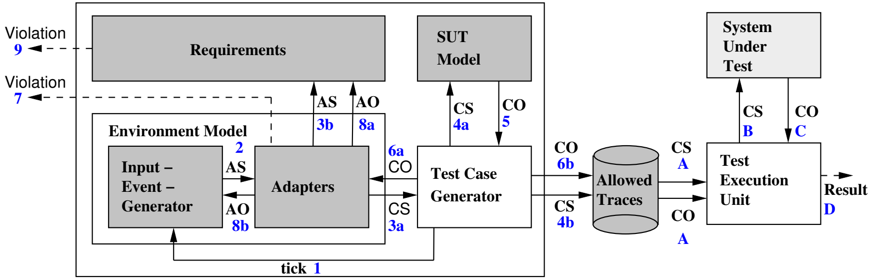
# Requirement: Example R1



R1
Pre 1.1
Fulfilled 1
destructiveWind () / checkR1 ()
Post 1
?
pullUpSun Blind()
Up
checkR1 () / *satisfactionR1 ()*
lowerSunBlind ()
pulledUp ()
PullingUp
stopSunBlind (), lowerSunBlind ()
pullUpSunBlind()
NotUp
lowerSun Blind ()
checkR1 () / *violationR1 ()*
checkR1 () / *inconclusiveR1 ()*
Fail
checkR1 () / *violationR1 ()*

If there is destructive wind, it must be checked if the sunblind is up. If yes, the requirement is satisfied. Otherwise, it is violated. The *Fail* state corresponds to a state where the sunblind would be destroyed.

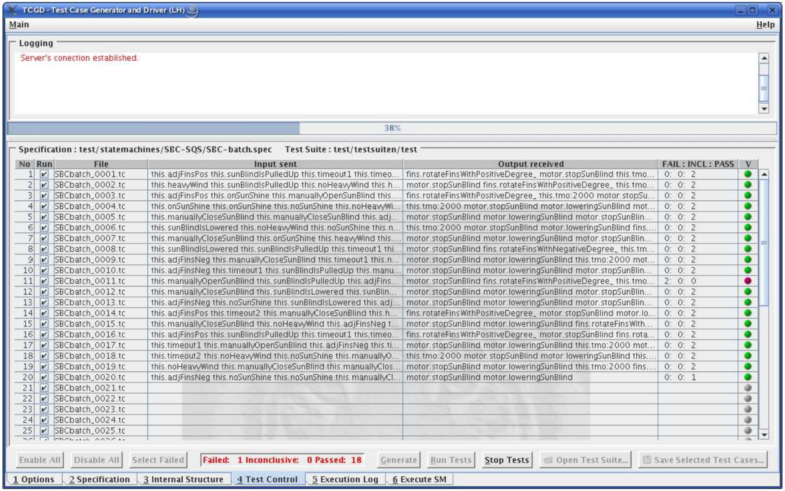
# Test Architecture for Generated Batch-Tests



The test case generator simulates the SUT in its environment and stores the resulting test cases.

The test execution unit executes the test cases on the SUT.

# Tool Support: TEAGER I



Server's conection established.

38%

Specification : test/statemachines/SBC-SQS/SBC-batch.spec    Test Suite : test/testsuiten/test

| No | Run | File | Input sent | Output received | FAIL : INCL : PASS | V |
|---|---|---|---|---|---|---|
| 1 | ✓ | SBCbatch_0001.tc | this.adjFinsPos this.sunBlindIsPulledUp this.timeout1 this.timeo... | fins.rotateFinsWithPositiveDegree_ motor.stopSunBlind this.tmo... | 0: 0: 2 | 🟢 |
| 2 | ✓ | SBCbatch_0002.tc | this.heavyWind this.sunBlindIsPulledUp this.noHeavyWind this.h... | motor.stopSunBlind fins.rotateFinsWithPositiveDegree_ this.tmo... | 0: 0: 2 | 🟢 |
| 3 | ✓ | SBCbatch_0003.tc | this.adjFinsPos this.onSunShine this.manuallyOpenSunBlind this.... | fins.rotateFinsWithPositiveDegree_ this.tmo:2000 motor.stopSu... | 0: 0: 2 | 🟢 |
| 4 | ✓ | SBCbatch_0004.tc | this.onSunShine this.onSunShine this.noSunShine this.noHeavyWi... | this.tmo:2000 motor.stopSunBlind motor.loweringSunBlind mot... | 0: 0: 2 | 🟢 |
| 5 | ✓ | SBCbatch_0005.tc | this.manuallyCloseSunBlind this.manuallyCloseSunBlind this.adj... | motor.stopSunBlind motor.loweringSunBlind motor.stopSunBlin... | 0: 0: 2 | 🟢 |
| 6 | ✓ | SBCbatch_0006.tc | this.sunBlindIsLowered this.noHeavyWind this.noSunShine this.n... | this.tmo:2000 motor.stopSunBlind motor.loweringSunBlind fins.... | 0: 0: 2 | 🟢 |
| 7 | ✓ | SBCbatch_0007.tc | this.manuallyCloseSunBlind this.onSunShine this.heavyWind this.... | motor.stopSunBlind motor.loweringSunBlind motor.stopSunBlin... | 0: 0: 2 | 🟢 |
| 8 | ✓ | SBCbatch_0008.tc | this.sunBlindIsLowered this.sunBlindIsPulledUp this.timeout1 thi... | motor.stopSunBlind fins.rotateFinsWithNegativeDegree_ this.tm... | 0: 0: 2 | 🟢 |
| 9 | ✓ | SBCbatch_0009.tc | this.adjFinsNeg this.manuallyCloseSunBlind this.timeout1 this.n... | motor.stopSunBlind motor.loweringSunBlind this.tmo:2000 mot... | 0: 0: 2 | 🟢 |
| 10 | ✓ | SBCbatch_0010.tc | this.adjFinsNeg this.timeout1 this.sunBlindIsPulledUp this.manu... | motor.stopSunBlind motor.loweringSunBlind motor.stopSunBlin... | 0: 0: 2 | 🟢 |
| 11 | ✓ | SBCbatch_0011.tc | this.manuallyOpenSunBlind this.sunBlindIsPulledUp this.adjFins... | motor.stopSunBlind fins.rotateFinsWithPositiveDegree_ this.tmo... | 2: 0: 0 | 🔴 |
| 12 | ✓ | SBCbatch_0012.tc | this.manuallyCloseSunBlind this.sunBlindIsLowered this.sunBlin... | motor.stopSunBlind motor.loweringSunBlind motor.stopSunBlin... | 0: 0: 2 | 🟢 |
| 13 | ✓ | SBCbatch_0013.tc | this.adjFinsNeg this.noSunShine this.sunBlindIsLowered this.adj... | motor.stopSunBlind motor.loweringSunBlind motor.stopSunBlin... | 0: 0: 2 | 🟢 |
| 14 | ✓ | SBCbatch_0014.tc | this.adjFinsPos this.timeout2 this.manuallyCloseSunBlind this.h... | fins.rotateFinsWithPositiveDegree_ motor.stopSunBlind motor.lo... | 0: 0: 2 | 🟢 |
| 15 | ✓ | SBCbatch_0015.tc | this.manuallyCloseSunBlind this.noHeavyWind this.adjFinsNeg t... | motor.stopSunBlind motor.loweringSunBlind fins.rotateFinsWith... | 0: 0: 2 | 🟢 |
| 16 | ✓ | SBCbatch_0016.tc | this.adjFinsPos this.sunBlindIsPulledUp this.timeout1 this.timeo... | fins.rotateFinsWithPositiveDegree_ motor.stopSunBlind fins.rota... | 0: 0: 2 | 🟢 |
| 17 | ✓ | SBCbatch_0017.tc | this.timeout1 this.manuallyOpenSunBlind this.adjFinsNeg this.ti... | motor.stopSunBlind motor.loweringSunBlind this.tmo:2000 mot... | 0: 0: 2 | 🟢 |
| 18 | ✓ | SBCbatch_0018.tc | this.timeout2 this.noHeavyWind this.noSunShine this.manuallyO... | this.tmo:2000 motor.stopSunBlind motor.loweringSunBlind this.... | 0: 0: 2 | 🟢 |
| 19 | ✓ | SBCbatch_0019.tc | this.noHeavyWind this.manuallyCloseSunBlind this.manuallyClos... | motor.stopSunBlind motor.loweringSunBlind this.tmo:2000 fins.... | 0: 0: 2 | 🟢 |
| 20 | ✓ | SBCbatch_0020.tc | this.adjFinsNeg this.noSunShine this.noSunShine this.manuallyCl... | motor.stopSunBlind motor.loweringSunBlind | 0: 0: 1 | 🟢 |
| 21 | ✓ | SBCbatch_0021.tc | | | | ⚪ |
| 22 | ✓ | SBCbatch_0022.tc | | | | ⚪ |
| 23 | ✓ | SBCbatch_0023.tc | | | | ⚪ |
| 24 | ✓ | SBCbatch_0024.tc | | | | ⚪ |
| 25 | ✓ | SBCbatch_0025.tc | | | | ⚪ |

Failed: 1 Inconclusive: 0 Passed: 18

swt.cs.tu-berlin.de/~seifert/teager.html

## Tool Support: TEAGER II

**TEAGER**

- research prototype
- clarifies UML state machine semantics
- executes imported UML state machines (for model validation and on-the-fly testing)
- generates test cases according to imported UML state machines (for batch testing)
  - probabilistic trigger selection
  - computes a complete behavioral model for a given search depth to determine the expected behavior of the SUT
- executes generated test cases
- LOCs: some 18k

## Summary

**Approach**

- based on Jackson's terminology
- uniform architecture for testing of reactive/embedded systems
- requirements are modeled explicitly with state machines using patterns
- facts and assumptions are modeled in the input-event generator or in the adapter state machines
- once these models have been set up manually (but systematically), the tests are performed automatically, using the tool TEAGER

**Benefits**

- environment models allow testing against requirements
- modeling patterns make approach systematic
- on-the-fly as well as batch testing is supported
- tool-support makes environment-based testing practical