PROTOS

Kombinatorische State-Transition Tests für Embedded Systeme

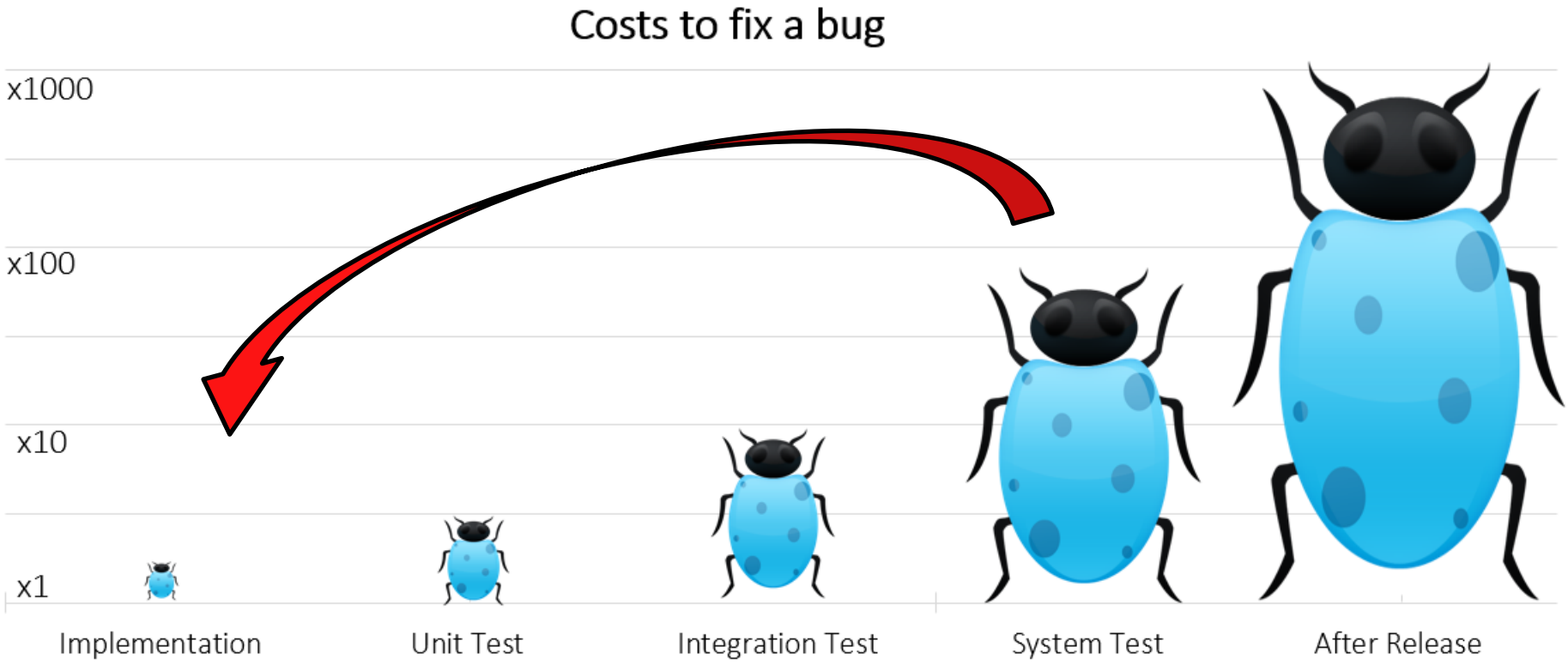
Thomas Schütz

Embedded Testing - MBT

München - 19.06.2018

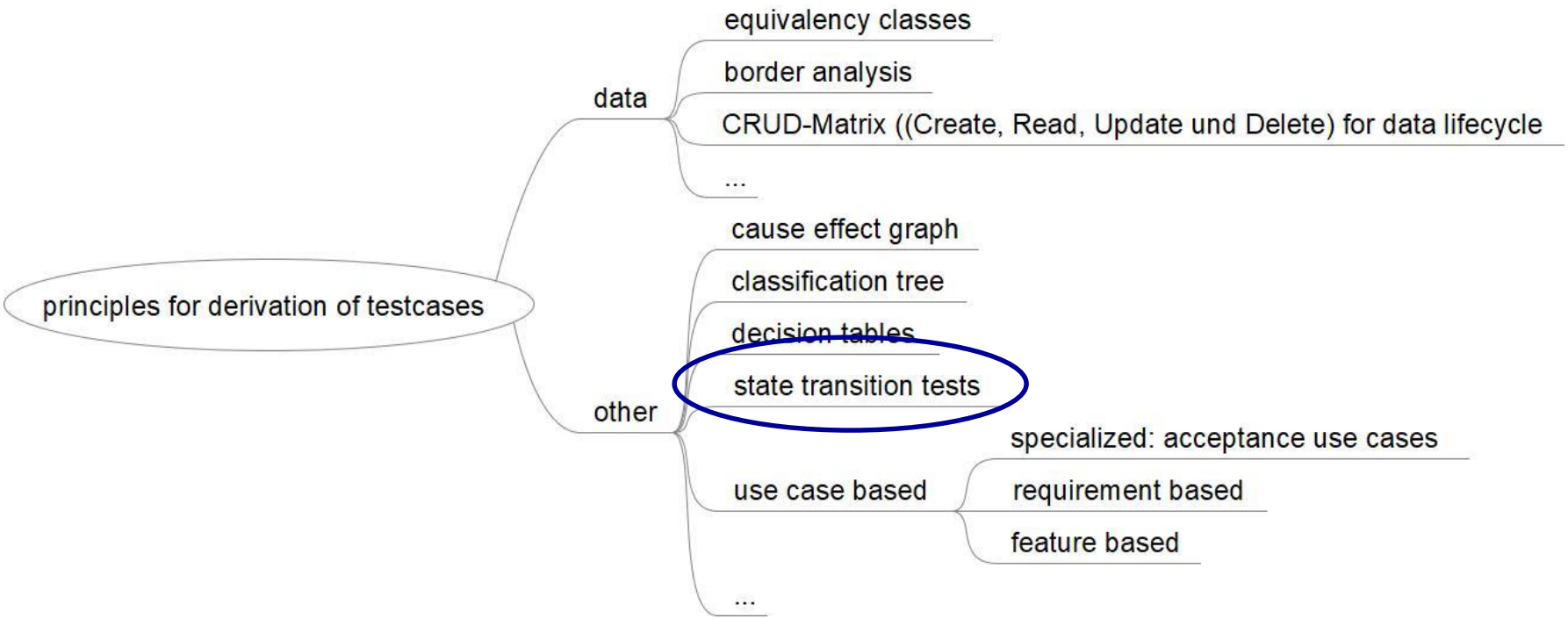
Why?

What is the Problem and why is it so big?

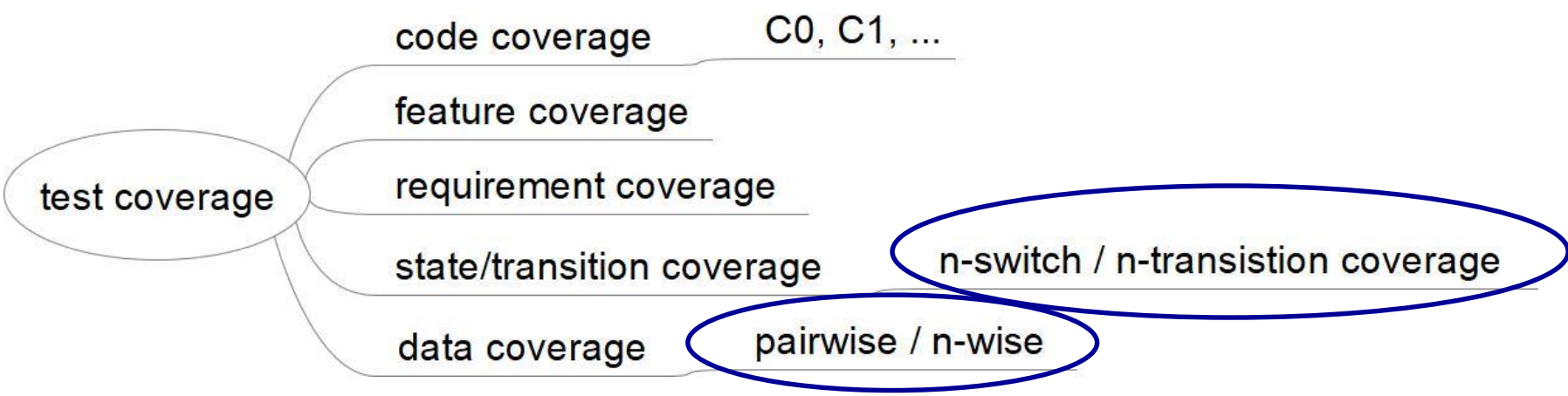


We need to learn and test more, earlier, faster!

How do we derive Testcases?

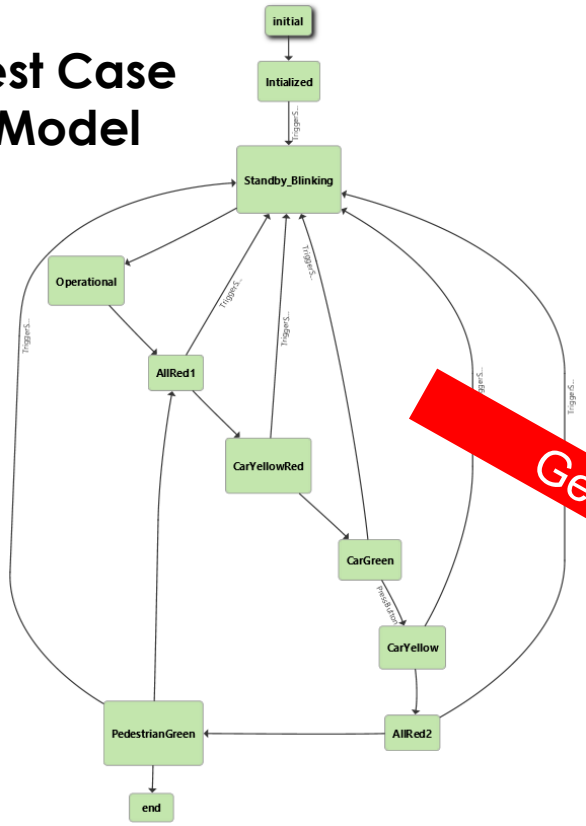


How do we define Coverage?



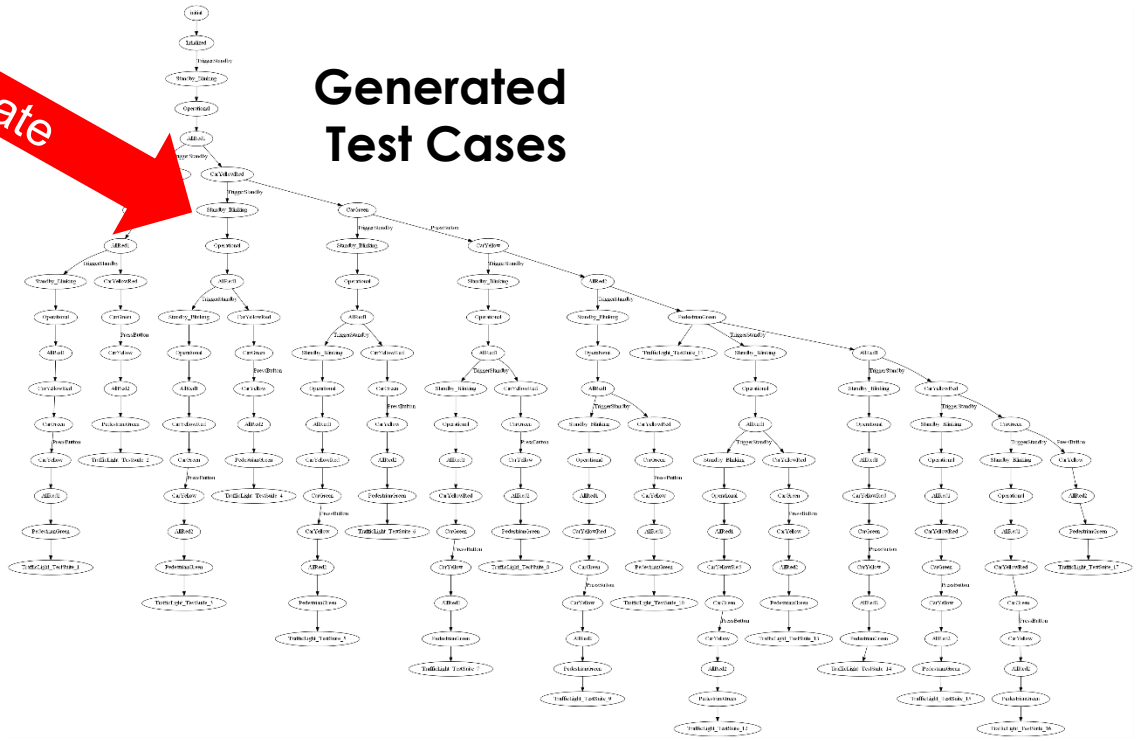
State Transition Testing

Test Case Model



Generate

Generated Test Cases



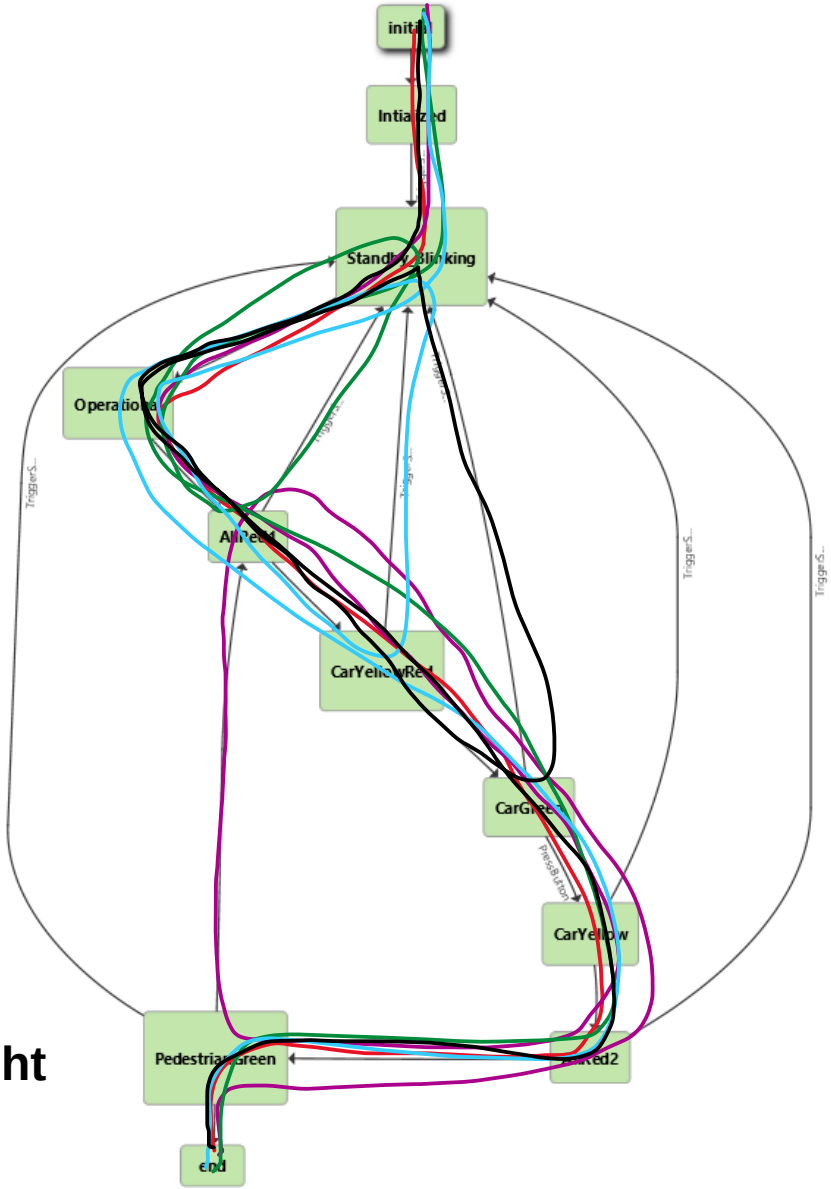
- Coverage for Data Combinations (n-wise)
- Coverage for Path Coverage (n-transition)
- Automatic Generation for all Test Cases

State Transition Testing

- every path from initial to end is a test case
- goal: at least complete state and transition coverage (0-switch)
- better: (complete) path coverage (n-switch)
- automatic generation for higher n-switch coverage is very useful (many test cases needed for coverage)

Example:

state/transition test for traffic light



n-switch Coverage

Goals:

- state / transition coverage
- (complete) path coverage

n-switch (n-transition) coverage:

- 0-switch: every transition at least once (includes complete state coverage)
- 1-switch: all combinations of two transitions at least once
- n-switch: all combinations of n+1 transitions at least once

n-wise Coverage

Goals:

- Coverage for data combinations

n-wise coverage:

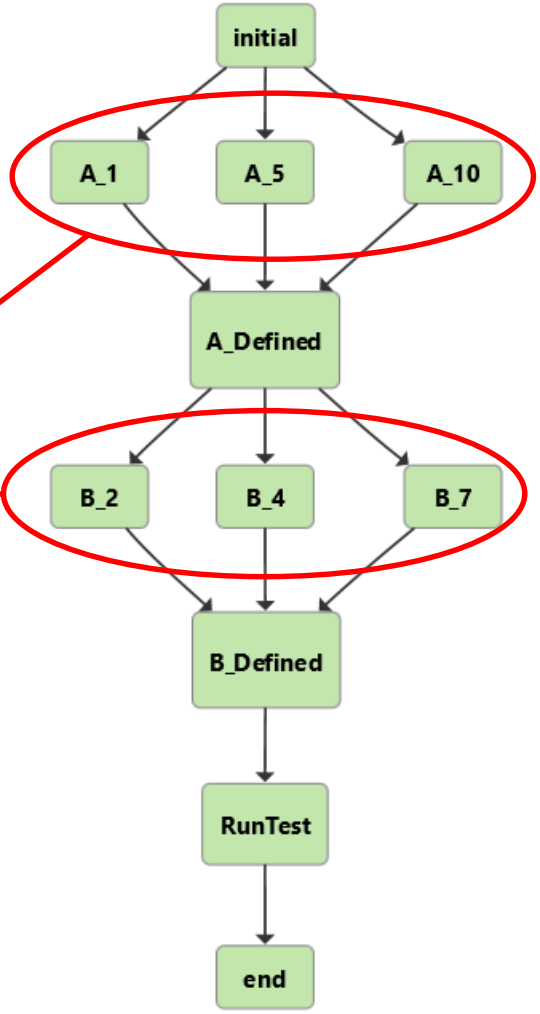
- start with equivalency classes and border analysis for each parameter (test vectors)
- pairwise (2-wise): all possible test vector combinations for 2 parameters
- n-wise: all possible test vector combinations for n parameters

Example:

Parameter combinations for f(a,b)		**Parameter a**		
		1	5	10
Parameter b	2	(1,2)	(5,2)	(10,2)
	4	(1,4)	(5,4)	(10,4)
	7	(1,7)	(5,7)	(10,7)

n-wise Coverage with state/transition tests

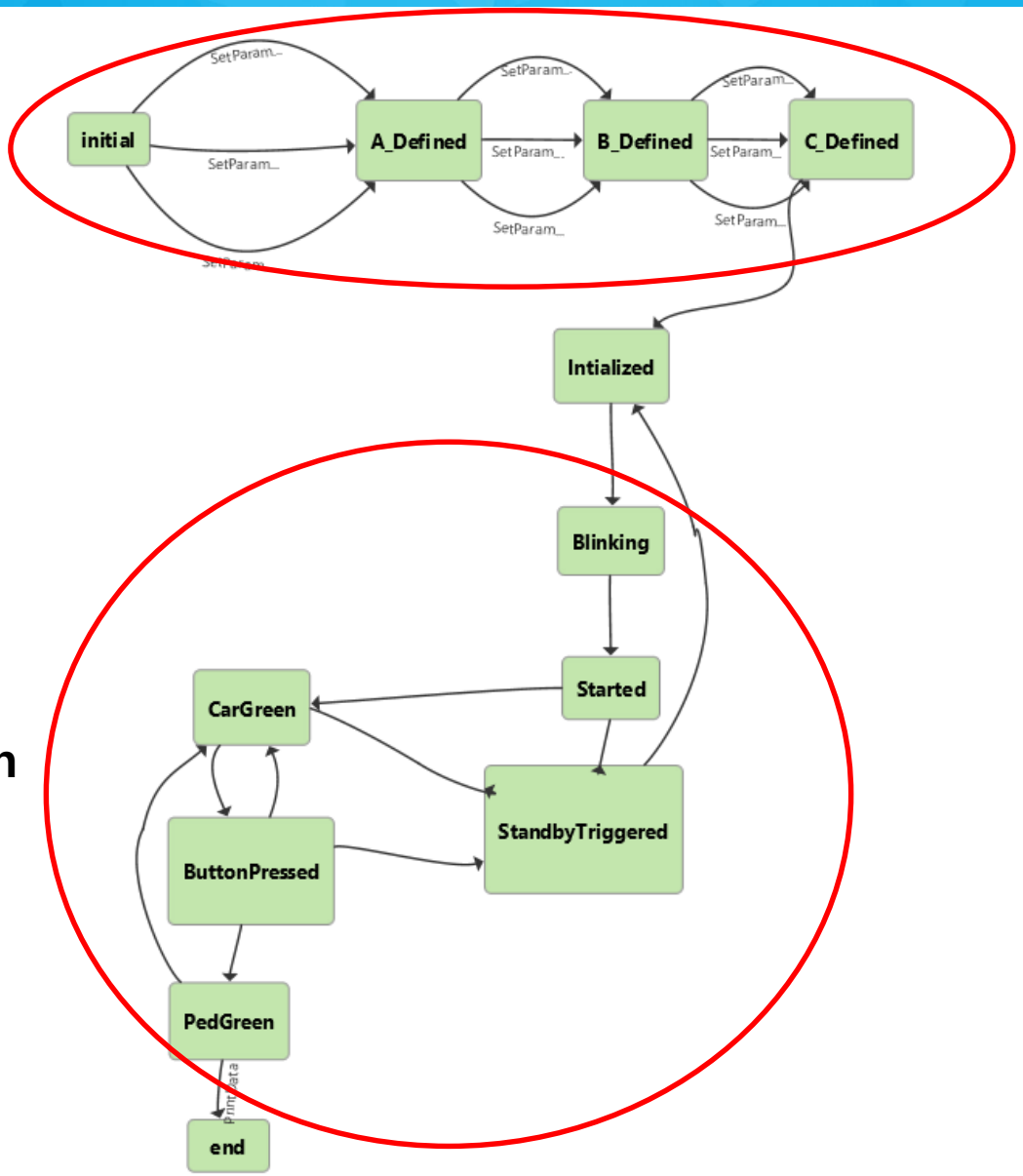
Parameter combinations for f(a,b)		Parameter a		
		1	5	10
Parameter b	2	(1,2)	(5,2)	(10,2)
	4	(1,4)	(5,4)	(10,4)
	7	(1,7)	(5,7)	(10,7)



Comination of n-wise and n-switch

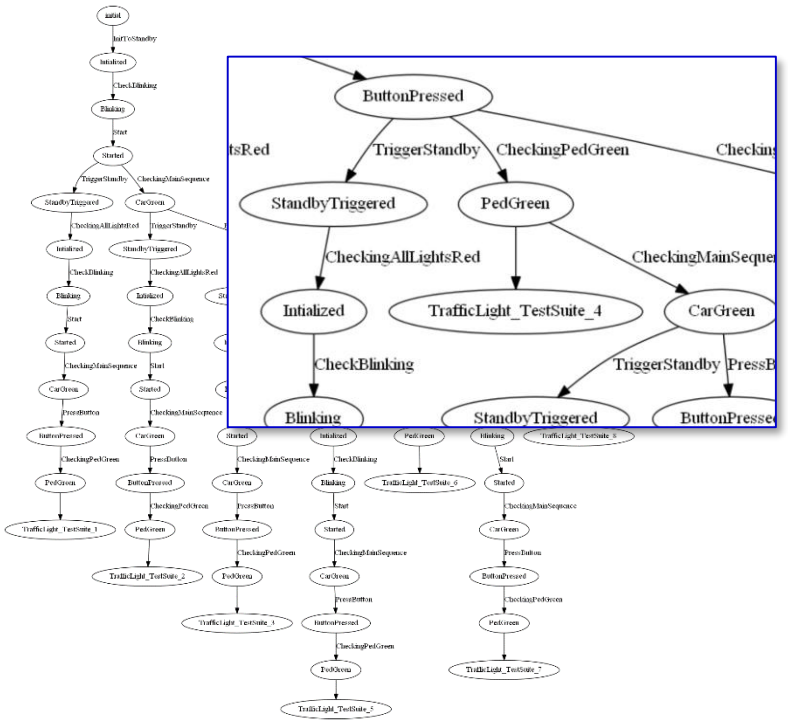
n-wise for data coverage

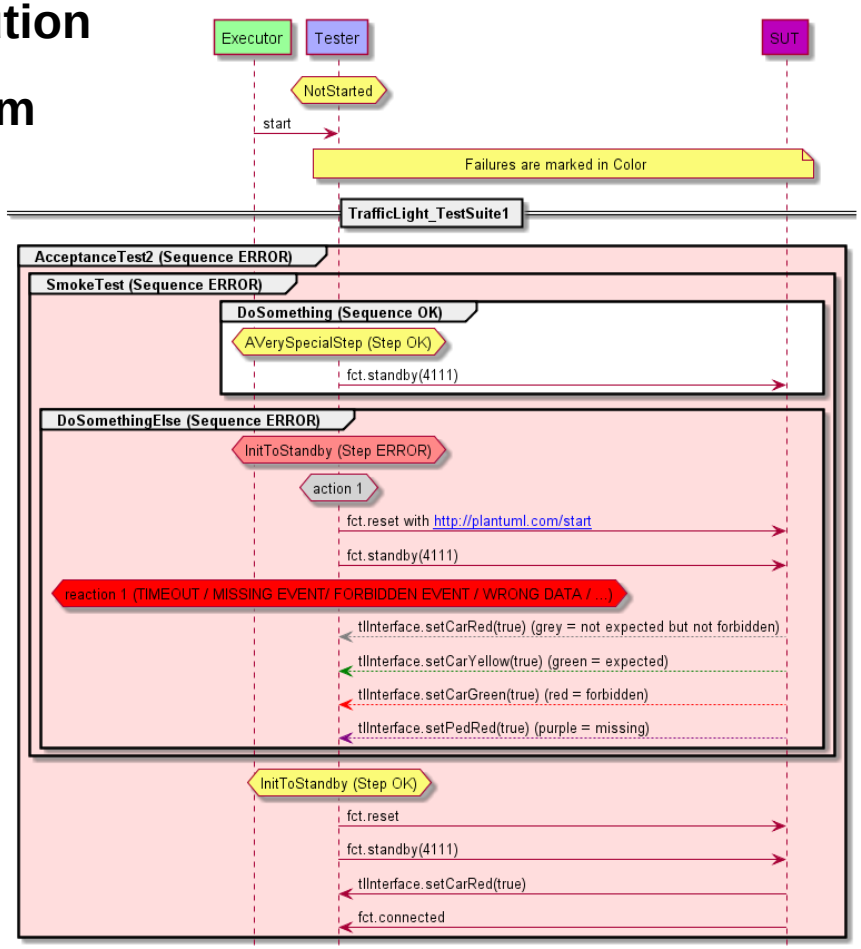
n-switch for path coverage



Documentation and Analysis

graphical views of test cases and test execution are very important to understand the problem

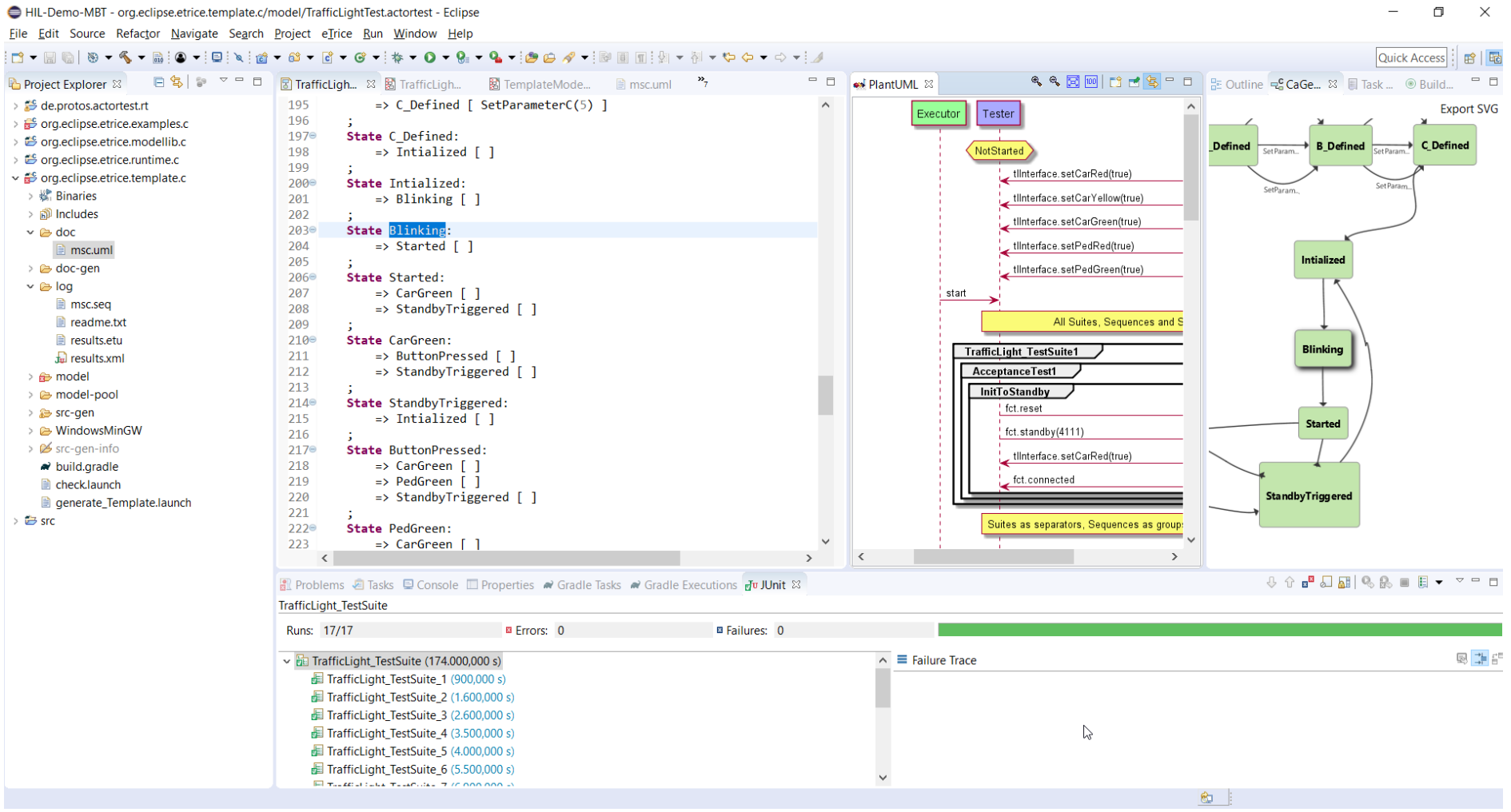




directed graphs help to understand the different execution paths of all test cases

sequence diagrams help to understand the dynamic behavior of a specific testcase

Demo


HIL-Demo-MBT - org.eclipse.etrice.template.c/model/TrafficLightTest.actortest - Eclipse
File Edit Source Refactor Navigate Search Project eTrice Run Window Help
Project Explorer
de.protos.actortest.rt
org.eclipse.etrice.examples.c
org.eclipse.etrice.modellib.c
org.eclipse.etrice.runtime.c
org.eclipse.etrice.template.c
Binaries
Includes
doc
msc.uml
doc-gen
log
msc.seq
readme.txt
results.etu
results.xml
model
model-pool
src-gen
WindowsMinGW
src-gen-info
build.gradle
check.launch
generate_Template.launch
src
=> C_Defined [SetParameterC(5)]
;
State C_Defined:
=> Intialized [ ]
;
State Intialized:
=> Blinking [ ]
;
State Blinking:
=> Started [ ]
;
State Started:
=> CarGreen [ ]
=> StandbyTriggered [ ]
;
State CarGreen:
=> ButtonPressed [ ]
=> StandbyTriggered [ ]
;
State StandbyTriggered:
=> Intialized [ ]
;
State ButtonPressed:
=> CarGreen [ ]
=> PedGreen [ ]
=> StandbyTriggered [ ]
;
State PedGreen:
=> CarGreen [ ]
PlantUML
Executor
Tester
NotStarted
tlInterface.setCarRed(true)
tlInterface.setCarYellow(true)
tlInterface.setCarGreen(true)
tlInterface.setPedRed(true)
tlInterface.setPedGreen(true)
start
TrafficLight_TestSuite1
AcceptanceTest1
InitToStandby
fct.reset
fct.standby(4111)
tlInterface.setCarRed(true)
fct.connected
Suites as separators, Sequences as group
Export SVG
B_Defined
C_Defined
Intialized
Blinking
Started
StandbyTriggered
Problems Tasks Console Properties Gradle Tasks Gradle Executions JUnit
TrafficLight_TestSuite
Runs: 17/17
Errors: 0
Failures: 0
TrafficLight_TestSuite (174.000,000 s)
TrafficLight_TestSuite_1 (900,000 s)
TrafficLight_TestSuite_2 (1.600,000 s)
TrafficLight_TestSuite_3 (2.600,000 s)
TrafficLight_TestSuite_4 (3.500,000 s)
TrafficLight_TestSuite_5 (4.000,000 s)
TrafficLight_TestSuite_6 (5.500,000 s)
Failure Trace

Conclusion

- state transition tests are a good method to define testcases for path- and data-coverage
- state transition tests can be used without tools
- for combinatorial testcases modeling tools and code generators are necessary

... any questions?

Thomas Schütz (PROTOS GmbH)

http://www.eclipse.org/etrice

http://www.protos.de