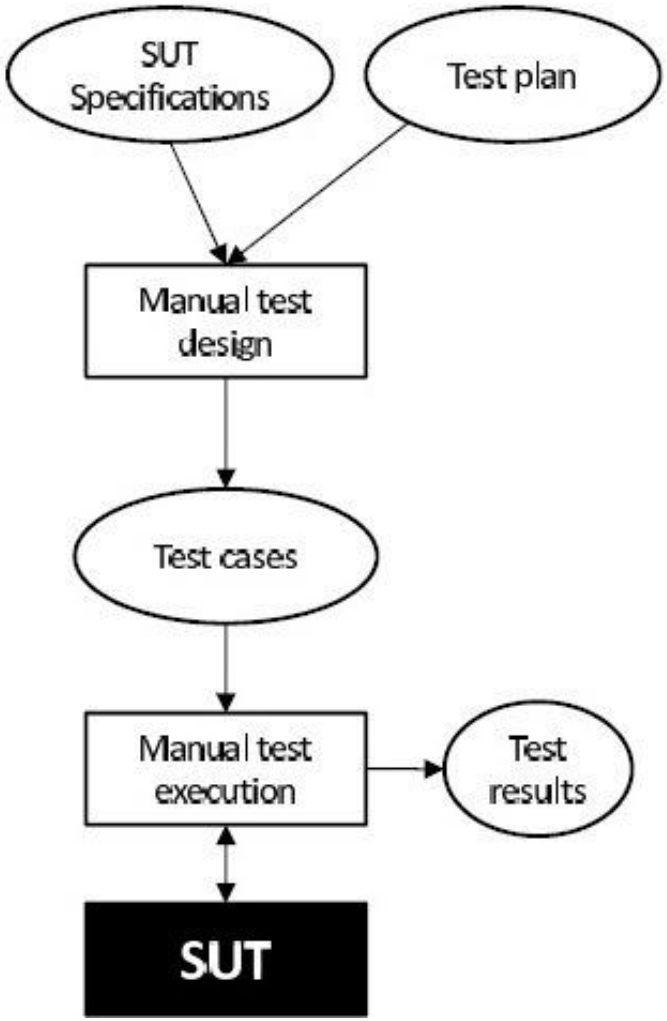
MBT – MODEL-BASED TESTING

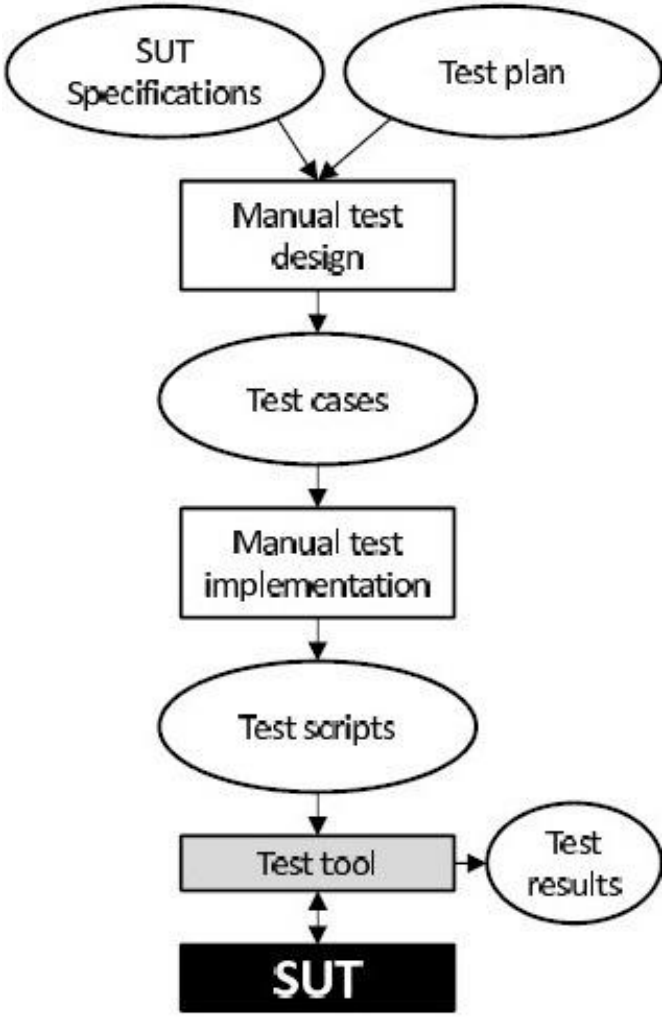
Gusztáv Adamis
adamis@tmit.bme.hu

MANUAL TESTING



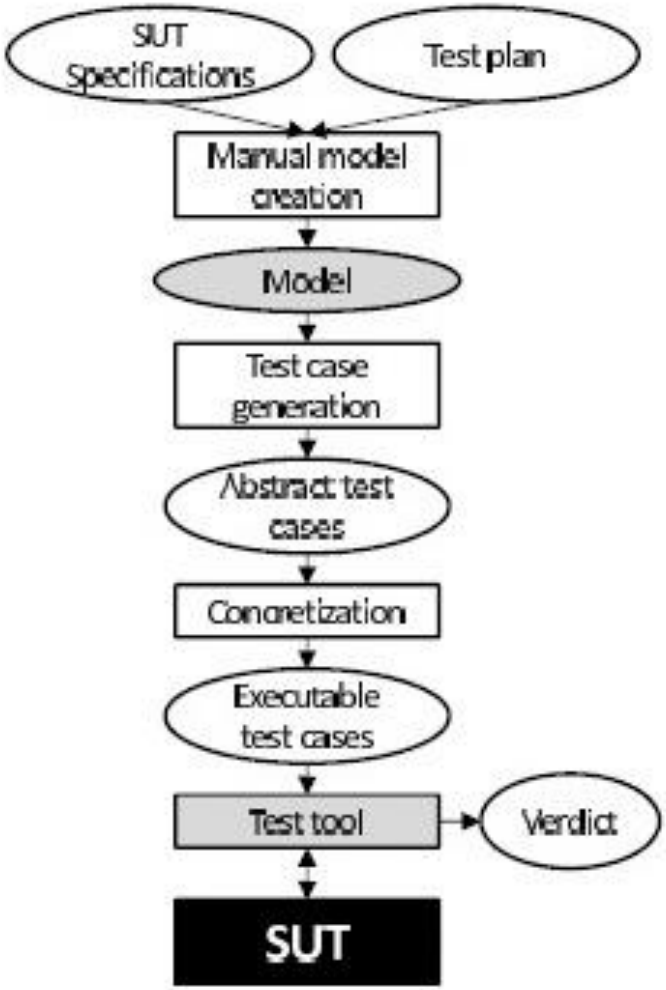
- Manual execution
  - Slow
  - Time consuming
  - Documentation
  - E.g. GUI test
    - Though several solutions of partly automating (Selenium, Jasmine)
- In most of the cases automation is necessary to achieve acceptable coverage in acceptable time frame

AUTOMATED TESTING (TEST SCRIPTS)



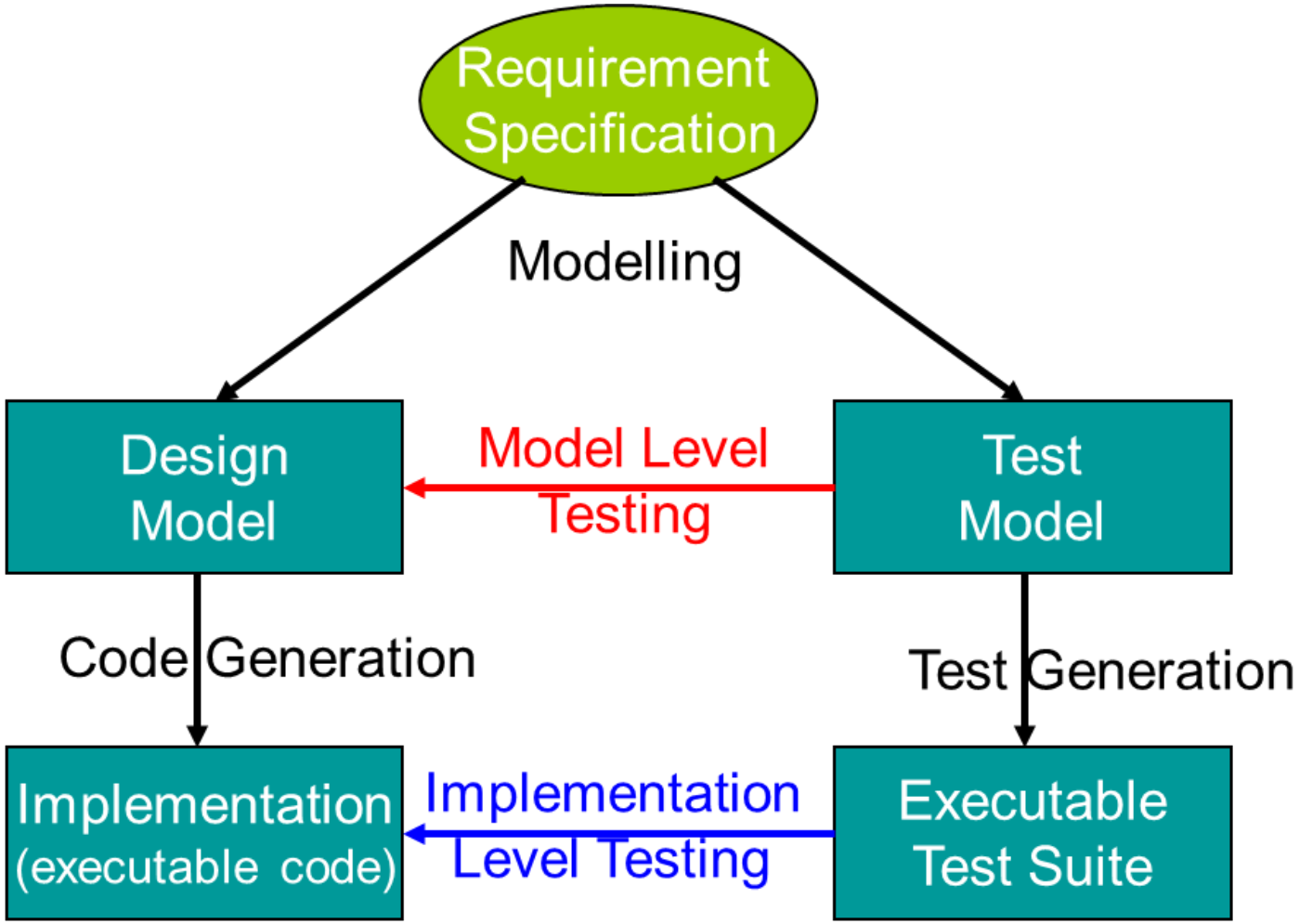
- Test programs (scripts) to write
  - Time
- Test execution tool to develop
  - Time
- But when they are ready, fast execution
- Problems:
  - Test development
  - Test validation
  - Test maintenance

MODEL-BASED TESTING



- Automatic (abstract) test case generation
  - Different strategies, goals
  - Model shall be verified, generated tests are correct
  - Any change in specification -> Re-generate the tests
    - No need to touch the test scripts
    - Easier maintenance
  - Problem:
    - Test Harness
    - Good only for ‘green field’ test development

MBT AT DIFFERENT LEVELS



TEST GENERATION ALGORITHMS

- Measuring the adequacy of a test suite
  - Coverage
- Deciding when to stop test generation

- Typically 100% coverage is not possible
  - Time consuming
  - Non-reachable states
    - Theoretically – if the model is wrong
    - Practically – complex guards, lot of variables

CLASSIFICATION OF TEST SELECTION CRITERIA

- Most of them can also be applied to code, but now for MODEL
  - Not the same
  - Both shall be tested

- Structural Model Coverage
- Data Coverage
- Fault-Model
- Requirements-Based
- Explicit Test Case Specification

CONTROL-FLOW-ORIENTED COVERAGE CRITERIA

- Mainly for code, but
  - For models in (E)FSM, UML, OCL, etc. : Decisions, Pre/Post conditions
- Statement Coverage (SC)
  - execute every reachable statement
- Decision (or Branch) Coverage (DC)
  - each reachable *decision* is made true by some tests and false by other tests
- Condition Coverage
  - each *condition* is tested with a true and with a false result
- Path Coverage (PC)
  - execute every satisfiable path through the control-flow graph
  - generally impossible to reach (loops!)

CONTROL-FLOW-ORIENTED COVERAGE CRITERIA

```
context SmartCard::verifyPin(p:PIN_CODES):MESSAGES
   post:
       if pinTry = 0 then
          result = MESSAGES::NO_MORE_TRIES
       else
          if (p=pin or statusPin=PINSTATUS::DISABLE) then
              result = MESSAGES::SUCCESS
          else
              result = MESSAGES::ERROR
              endif
       endif
```

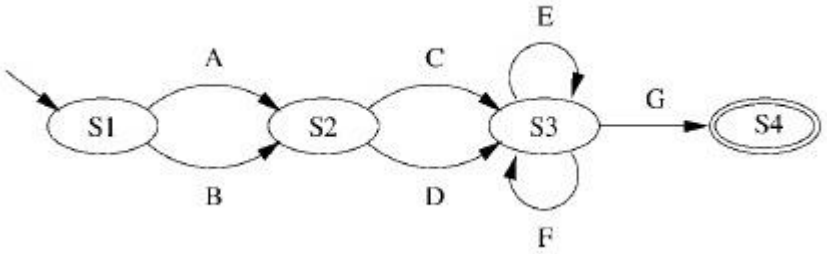
DATA-FLOW-ORIENTED COVERAGE CRITERIA

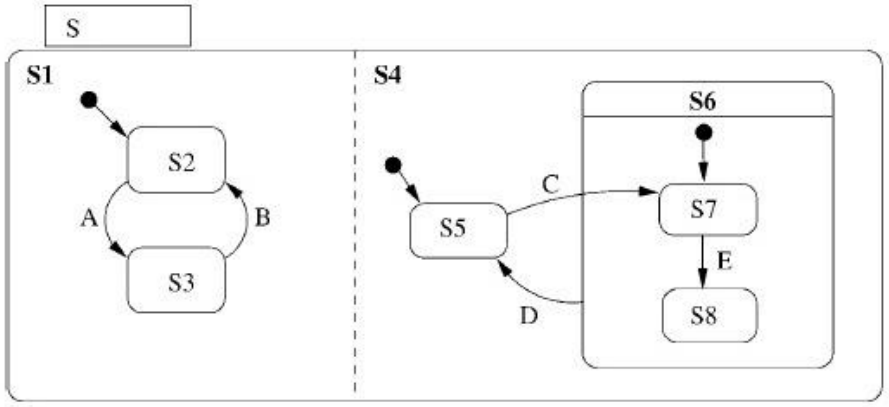
- For code
- Definition (assignment to) and use of variables
  - Data Flow Graphs
  - Definition-use paths

- All-Defs
  - test at least one def-use pair (d_v, u_v) for every definition d_v, that is, at least one path from each definition to one of its feasible uses
- All-Uses
  - test all def-use pairs (d_v, u_v). (testing all feasible uses of all defs)
- All-Def-Use-Paths
  - test all def-use pairs (d_v, u_v) and to test all paths from d_v, to u_v
  - Practically unrealistic

TRANSITION-BASED COVERAGE CRITERIA

› For (E)FSM, LTS, UML state charts, etc. models



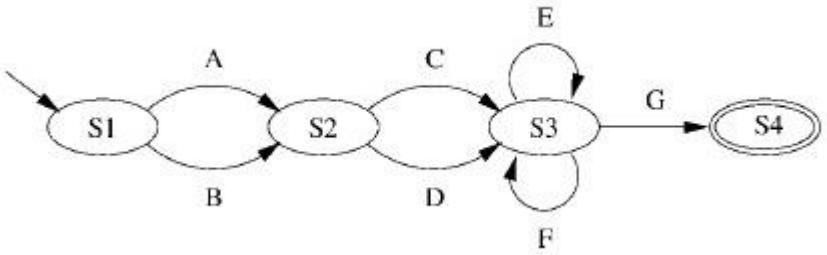


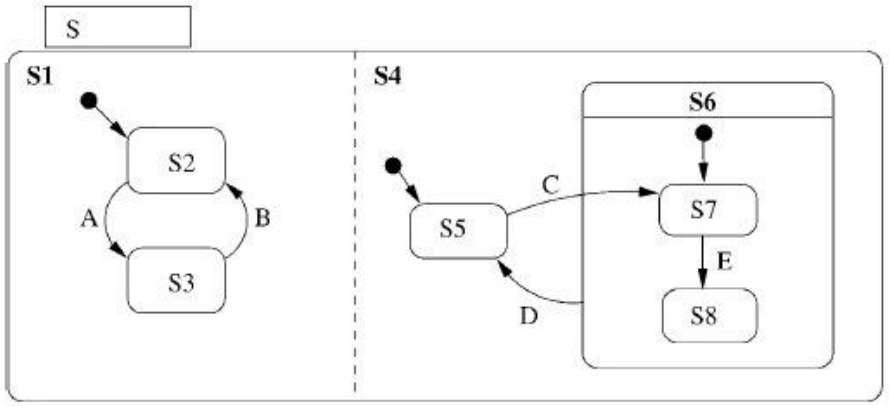
› All-states Coverage
  – Every state of the model is visited at least once

› e.g. ACG

e.g. ACE

TRANSITION-BASED COVERAGE CRITERIA



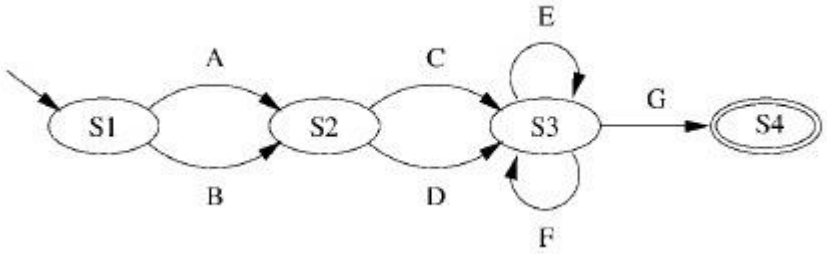


- All-transitions Coverage
  - Every transition of the model must be traversed at least once
  - e.g. ACEFG + BD
  - e.g. ACEFG + BDG (if shall end at final state)

ABCED or ABCEDCD

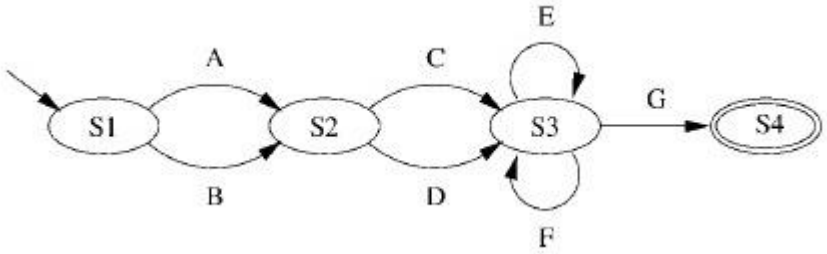
D is one or two transactions?

TRANSITION-BASED COVERAGE CRITERIA



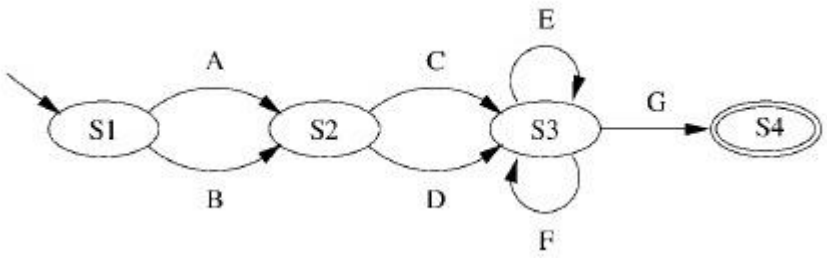
- All-transition-pairs Coverage
  - Every pair of adjacent transitions in the FSM or statechart model must be traversed at least once
- For S2:
  - AC+AD+BC+BD

TRANSITION-BASED COVERAGE CRITERIA



- All-loop-free-paths Coverage:
  - Every loop-free path must be traversed at least once
  - ACG + ADG + BCG + BDG
    - Does not cover all transitions or even all states!
- All-one-loop-paths Coverage
  - Every path containing at most two repetitions of one (and only one) state must be traversed at least once
  - All the loop-free paths and all the paths that loop once
    - 4*3 = 12 test cases

TRANSITION-BASED COVERAGE CRITERIA



- All-paths Coverage
  - Every path must be traversed at least once (exhaustive testing)
  - If loop: infinite number of paths
  - ACG+ADG+BCG+BDG+ACEG+ACEEG+….

TRANSITION-BASED COVERAGE CRITERIA

- All-loop-free-paths, All-one-loop-paths are inadequate on their own, since they do not guarantee to cover all transitions or even all states
  - Extreme example: in the first state we have to take a loop at least twice, nothing else than the first state is reachable…
- In practice, the All-transition Coverage is the minimum to reach

DATA COVERAGE CRITERIA

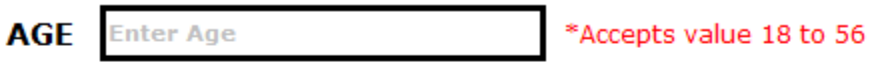
- For choosing a few good data values to use as test inputs when there is a huge number of possible input values
- Two extremes:
  - **One-value:** simply requires to test at least one value from the domain *D*. Often too simple.
  - **All-values:** requires to test every value in the domain *D*.
    - This is not practical if *D* is large (e.g., 0*. . .* 999999), but when *D* is small, such as an enumerated type, it can be useful to test all possibilities

EQUIVALENT PARTITIONING AND BOUNDARY VALUE TESTING

- A valid password contains 6..10 characters
  - Equivalent partitions:
    - 0..5 – shall not accept
    - 6..10 – shall accept
    - 11.. – shall not accept
    - Choose any value from the partitions
      - E.g. 3, 7, 186
  - Boundary value testing
    - Lots of faults in the SUT are located at the boundary between two functional behaviours
    - 5, 6, *(7), (9),* 10, 11

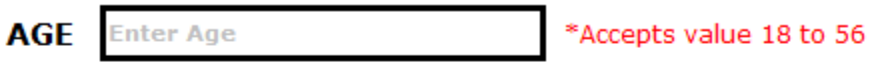
BOUNDARY VALUE TESTING

Assume, we have to test a field which accepts Age 18 – 56



BOUNDARY VALUE TESTING

Assume, we have to test a field which accepts Age 18 – 56



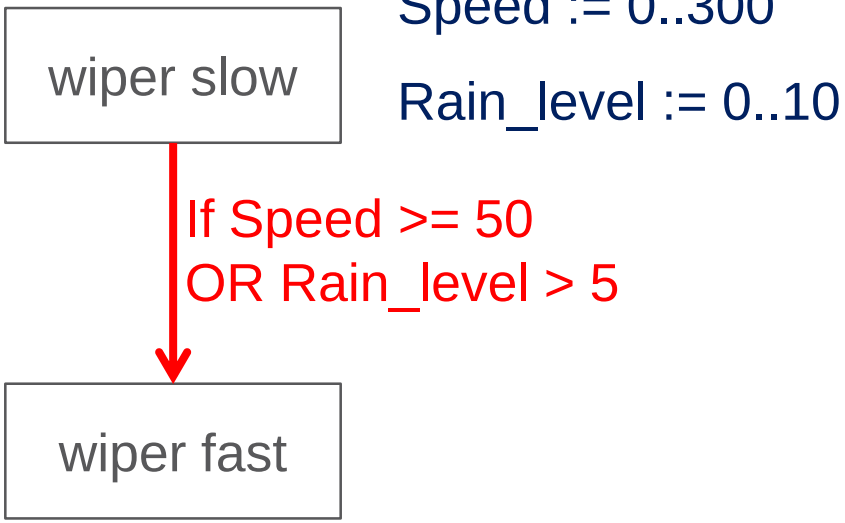
BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

Minimum boundary value is 18
Maximum boundary value is 56
Valid Inputs: 18,19,55,56
Invalid Inputs: 17 and 57

Test case 1: Enter the value 17 (18-1) = Invalid
Test case 2: Enter the value 18 = Valid
Test case 3: Enter the value 19 (18+1) = Valid
Test case 4: Enter the value 55 (56-1) = Valid
Test case 5: Enter the value 56 = Valid
Test case 6: Enter the value 57 (56+1) =Invalid

BOUNDARY VALUE TESTING

- Choose test input values at the boundaries of the input domains
  - Lots of faults in the SUT are located at the boundary between two functional behaviours



Speed := 0..300

Rain_level := 0..10

Speed = 0, (1), 49, 50, (51), 300

Rain_level = 0, (1), (4), 5, 6, 10

Speed= 0, Rain_level = 0

Speed= 0, Rain_level = 1, …

Speed= 1, Rain_level = 0, …

Speed= 300, Rain_level = 10

PAIRWISE TESTING

- Pairwise testing is based on the assumption that most defects are created as a result of no more than two test parameters (test values) being in a certain combination
- E.g.:
  - Destinations: Canada, Mexico, USA
  - Class: Tourist, Business, First
  - Seat: Aisle, Window
- Exhaustive testing:
  - 3*3*2 = 18 combinations

PAIRWISE TESTING

Test	Destination	Class	Seat Preference
1	Canada	Tourist	Aisle
2	Mexico	Tourist	Aisle
3 (defect!)	USA	Tourist	Aisle
4	Canada	Business Class	Aisle
5	Mexico	Business Class	Aisle
6	USA	Business Class	Aisle
7	Canada	First Class	Aisle
8	Mexico	First Class	Aisle
9	USA	First Class	Aisle
10	Canada	Tourist	Window
11	Mexico	Tourist	Window
12 (defect!)	USA	Tourist	Window
13	Canada	Business Class	Window
14	Mexico	Business Class	Window
15	USA	Business Class	Window
16	Canada	First Class	Window
17	Mexico	First Class	Window
18	USA	First Class	Window

- Assume: USA, Tourist causes the problem
- Pairwise test generation:
  - Test 18:
    - USA, First: in 9,
    - USA, Window: in 15,
    - First, Window: in 17
  - Test 18 is redundant, etc.

PAIRWISE TESTING

Number	Destination	Class	Seat Preference
1	Canada	Tourist	Aisle
3 (defect!)	USA	Tourist	Aisle
5	Mexico	Business Class	Aisle
8	Mexico	First Class	Aisle
9	USA	First Class	Aisle
11	Mexico	Tourist	Window
13	Canada	Business Class	Window
15	USA	Business Class	Window
16	Canada	First Class	Window

- 9 tests instead of 18
- Still finds the problem

PAIRWISE TESTING

- More effective if much higher combinations
  - If 10 variables with 5 values each
    - 5^{10}=9 765 625 exhaustive tests
    - Only 44 pairwise tests
  - If 75 binary variables
    - 2^{75} = 37 778 931 862 957 161 709 568 exhaustive tests
    - Only 28 pairwise tests
- Complicated test generation algorithms
- N-wise coverage
  - If suppose that the problem depends on N values instead 2
  - Number of tests rapidly grows as N increases
  - All-triples can still be practical, but all-quadruples are ~not

FAULT-BASED TESTING

- Pre-specified faults
  - Typically frequently occurred ones
  - Mutation operators: e.g. substitute + with - in expressions
  - Generate tests for each mutant of the original program,
    - Design a test that distinguishes that mutant from the original program
    - The resulting test suite is therefore able to show, which faults are NOT in the SUT
      - Fault-finding Power

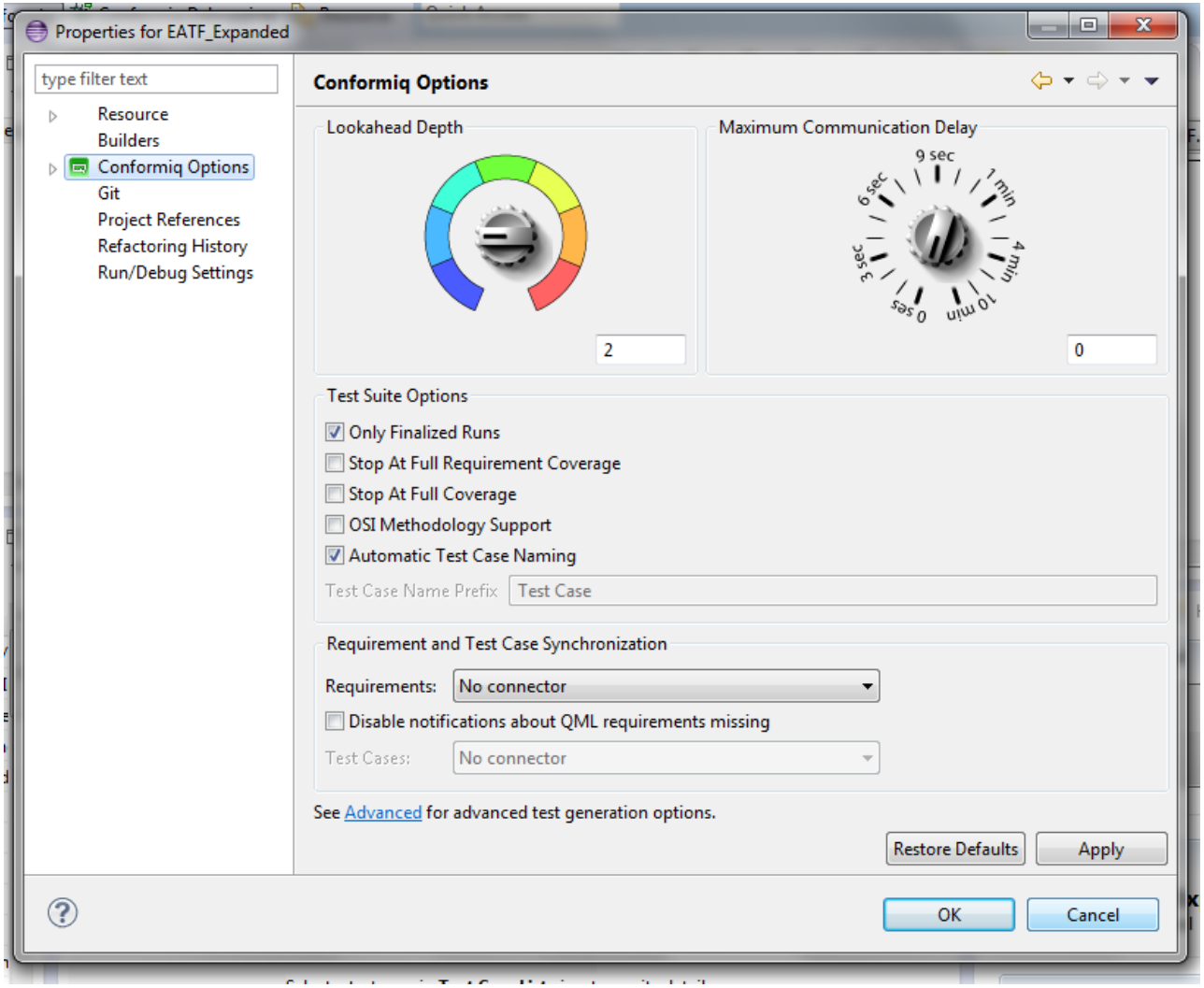
REQUIREMENTS-BASED CRITERIA

- High-level, testable statements of functionalities
- Each requirement shall be tested (e.g. in acceptance tests)
- In MBT two typical solutions:
  - 1. Record the requirements inside the behavior model (as annotations on various parts of the model) so that the test generation process can ensure that all requirements have been tested
  - 2. Formalize each requirement and then use that formal expression as a test selection criterion to drive the automated generation of one or more tests from the behavior model
    - Explicit test case specifications

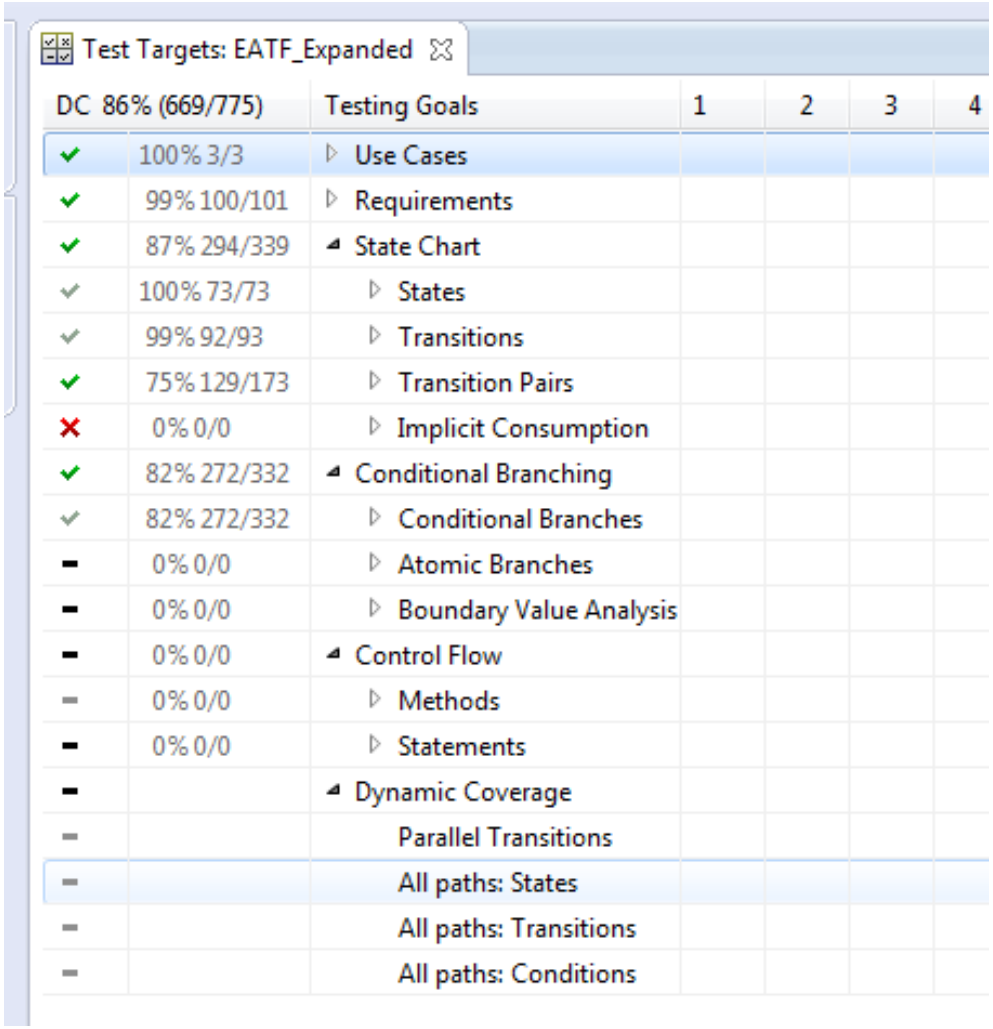
EXPLICIT TEST CASE SPECIFICATION

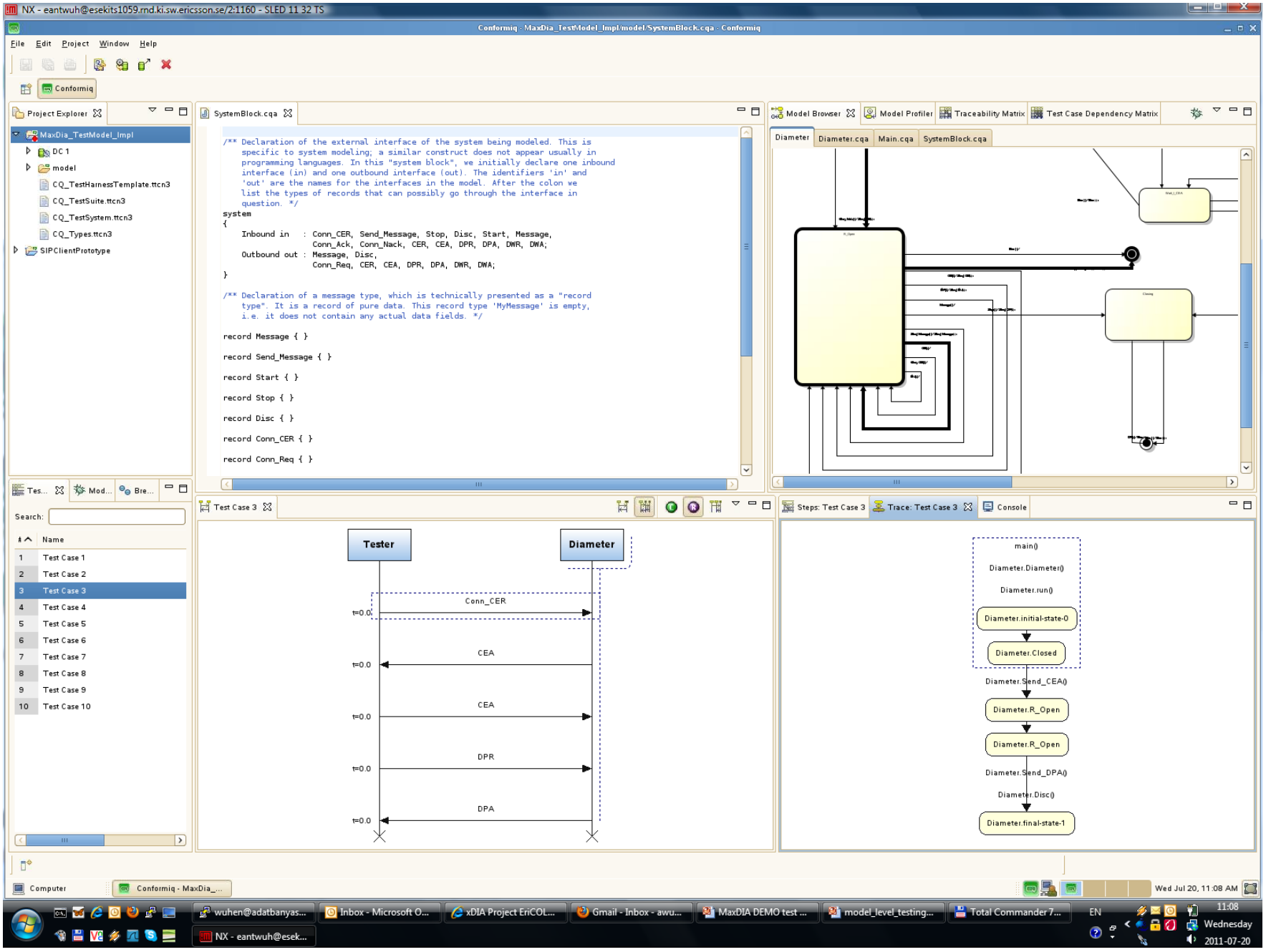
- Some explicit requirements are given in the model
  - E.g.: Test shall contain this state
- Generate test
  - For typical or for less typical cases
  - Just for a given service
  - Etc.

TEST SELECTION IN AN MBT TOOL (CONFORMIQ)



TEST SELECTION IN AN MBT TOOL (CONFORMIQ)




NX - eantwuh@esekits1059.rnd.ki.sw.ericsson.se/2:1160 - SLED 11 32 TS
Conformiq - MaxDia_TestModel_Impl/model/SystemBlock.cqa - Conformiq
File Edit Project Window Help
Conformiq
Project Explorer
MaxDia_TestModel_Impl
DC 1
model
CQ_TestHarnessTemplate.ttcn3
CQ_TestSuite.ttcn3
CQ_TestSystem.ttcn3
CQ_Types.ttcn3
SIPClientPrototype
SystemBlock.cqa
/** Declaration of the external interface of the system being modeled. This is specific to system modeling; a similar construct does not appear usually in programming languages. In this "system block", we initially declare one inbound interface (in) and one outbound interface (out). The identifiers 'in' and 'out' are the names for the interfaces in the model. After the colon we list the types of records that can possibly go through the interface in question. */
system
{
Inbound in : Conn_CER, Send_Message, Stop, Disc, Start, Message, Conn_Ack, Conn_Nack, CER, CEA, DPR, DPA, DWR, DWA;
Outbound out : Message, Disc, Conn_Req, CER, CEA, DPR, DPA, DWR, DWA;
}
/** Declaration of a message type, which is technically presented as a "record type". It is a record of pure data. This record type 'MyMessage' is empty, i.e. it does not contain any actual data fields. */
record Message { }
record Send_Message { }
record Start { }
record Stop { }
record Disc { }
record Conn_CER { }
record Conn_Req { }
Model Browser
Model Profiler
Traceability Matrix
Test Case Dependency Matrix
Diameter
Diameter.cqa
Main.cqa
SystemBlock.cqa
Search:
Name
1 Test Case 1
2 Test Case 2
3 Test Case 3
4 Test Case 4
5 Test Case 5
6 Test Case 6
7 Test Case 7
8 Test Case 8
9 Test Case 9
10 Test Case 10
Test Case 3
Tester
Diameter
Conn_CER
CEA
CEA
DPR
DPA
t=0.0
Steps: Test Case 3
Trace: Test Case 3
Console
main()
Diameter.Diameter()
Diameter.run()
Diameter.initial-state-0
Diameter.Closed
Diameter.Send_CEA()
Diameter.R_Open
Diameter.R_Open
Diameter.Send_DPA()
Diameter.Disc()
Diameter.final-state-1
Computer
Conformiq - MaxDia_...
Wed Jul 20, 11:08 AM
wuhen@adatbanyas...
Inbox - Microsoft O...
xDIA Project EriCOL...
Gmail - Inbox - awu...
MaxDIA DEMO test ...
model_level_testing...
Total Commander 7...
NX - eantwuh@esek...
EN
11:08
Wednesday
2011-07-20