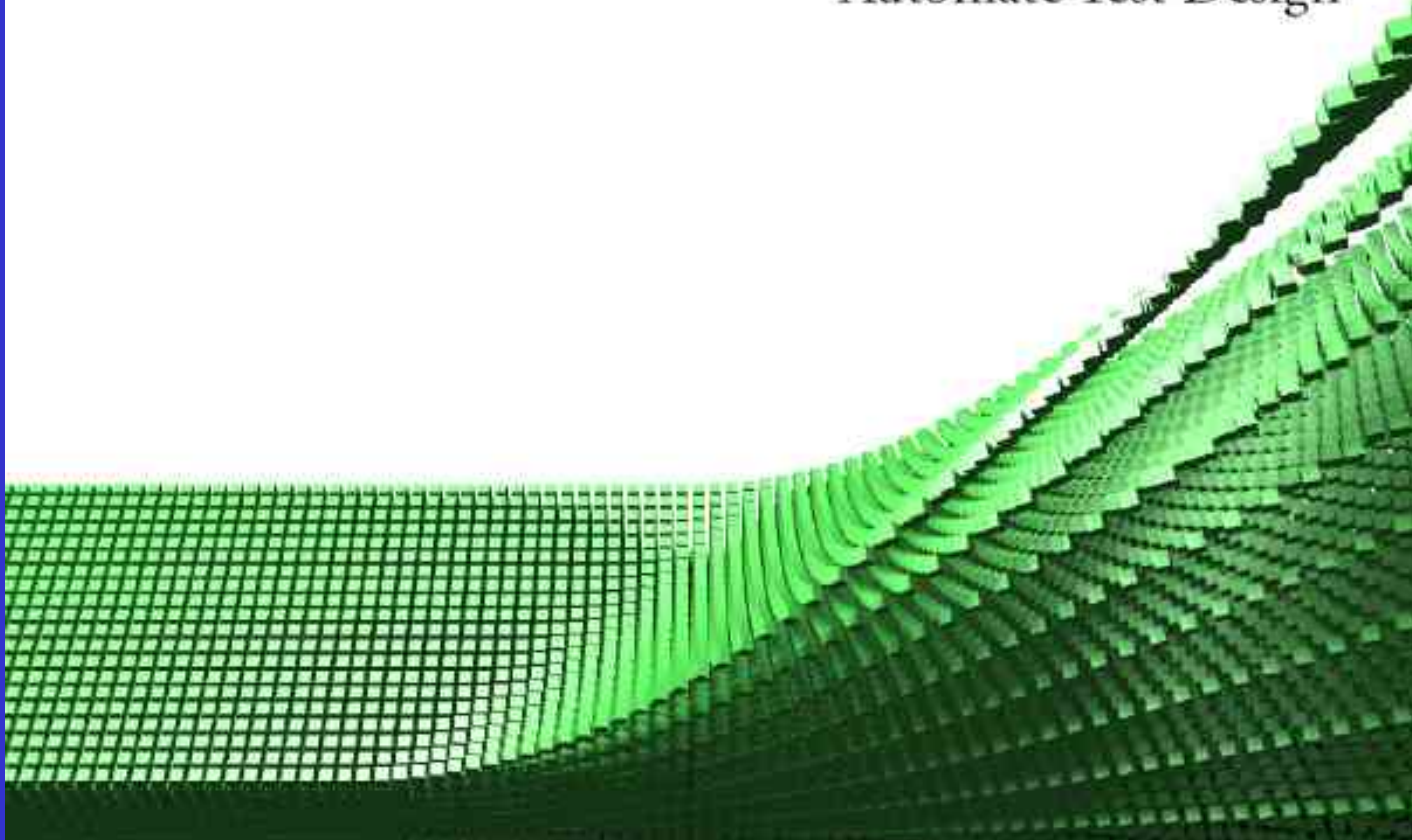


CONFORMIQ

QTRONIC™

Automate Test Design™



Automatic Generation of Test Scripts for Functional Tests (Black-Box-Tests) with Conformiq Qtronic

Verifysoft Technology GmbH

www.verifysoft.com

Verifysoft Technology GmbH

Developer and Distributor of Software Testing Tools

Workshops

Founded 2003 – privately held

Offenburg (Germany)

150+ customers in Europe



Verifysoft Customers



and many more ...

Testwell

Testwell, Tampere (Finland)

Code Coverage, Code Complexity Measures, Unit-Tests

 coverity

Coverity, San Francisco (USA)

Static Code-Analysis: Coverity Prevent, Coverity Extend

CONFORMIQ

Conformiq, Saratoga (USA)

Automatic Test Generator: Conformiq Qtronic



founded 1998

Espoo (Finland)

Headquarter: Saratoga (USA)

Fokus on „model driven testing Tools“

Conformiq Qtronic: since early 2007

Manual Test Case Design takes **time** ...
and creates **risks**:

- 💣 incorrect tests
- 💣 missed tests
- 🕒 redundant tests
- 🕒 maintenance of test scripts takes time



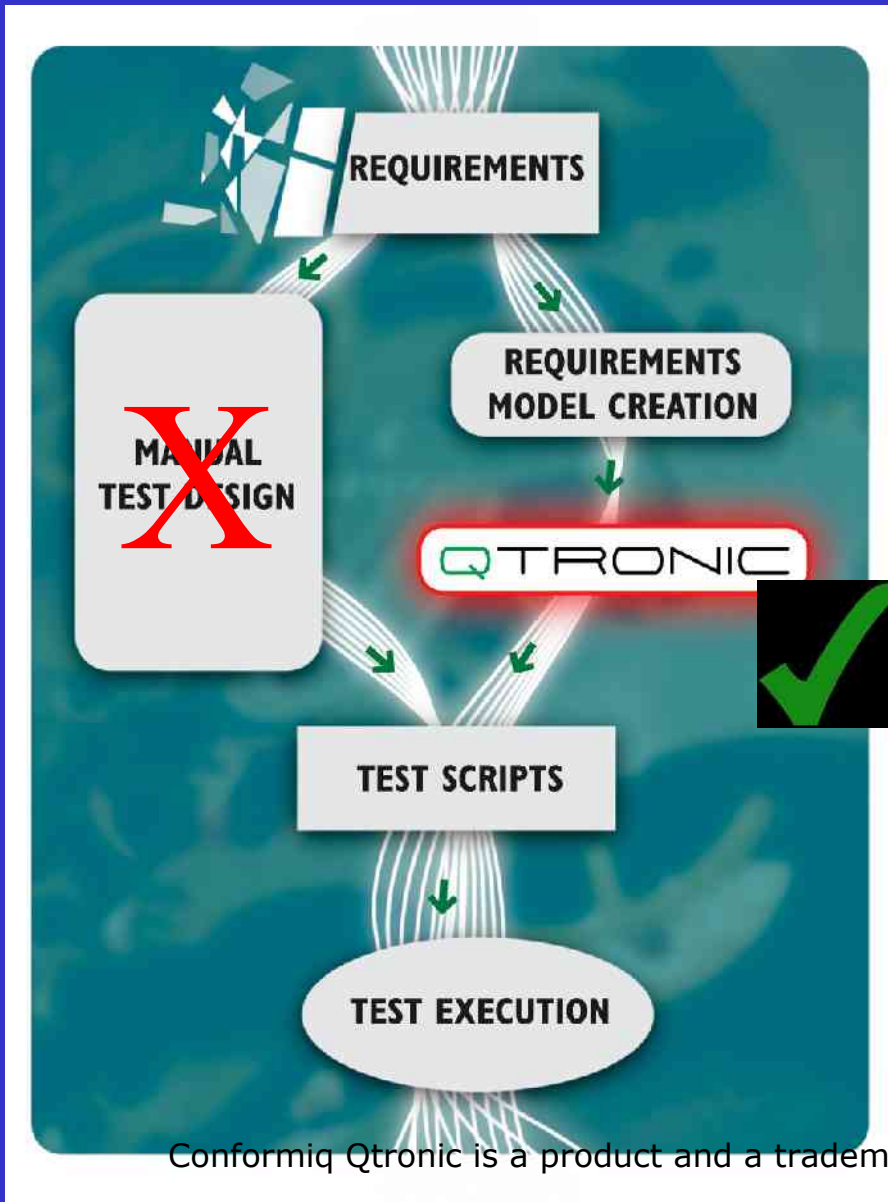
Our solution: **Automated Test Design**

Automated Test Design

-> model driven testing, model based testing,
specification based testing,
specification driven testing,
...



Conformiq Qtronic Workflow

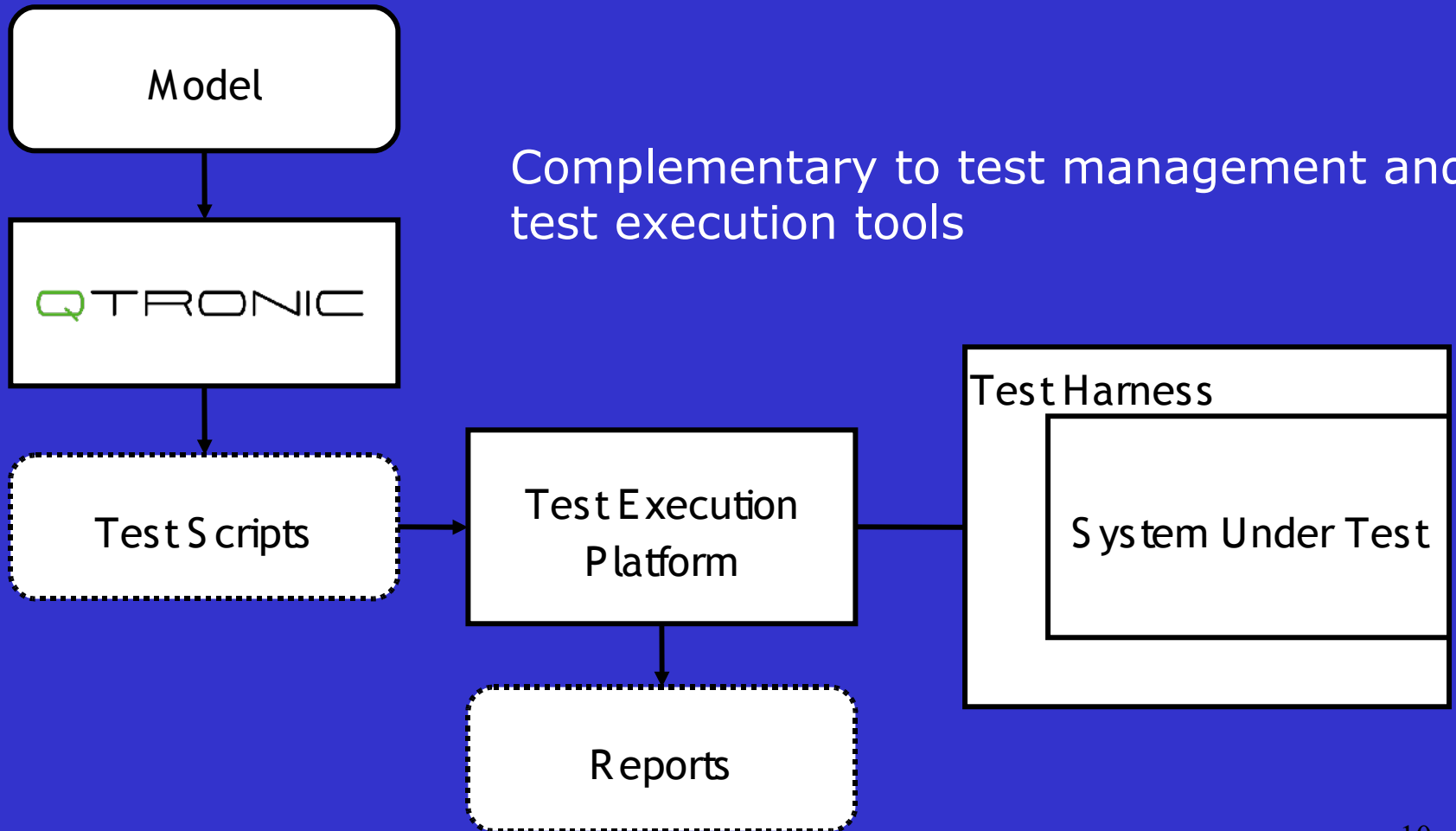


Automatic Test Case Generation based on models

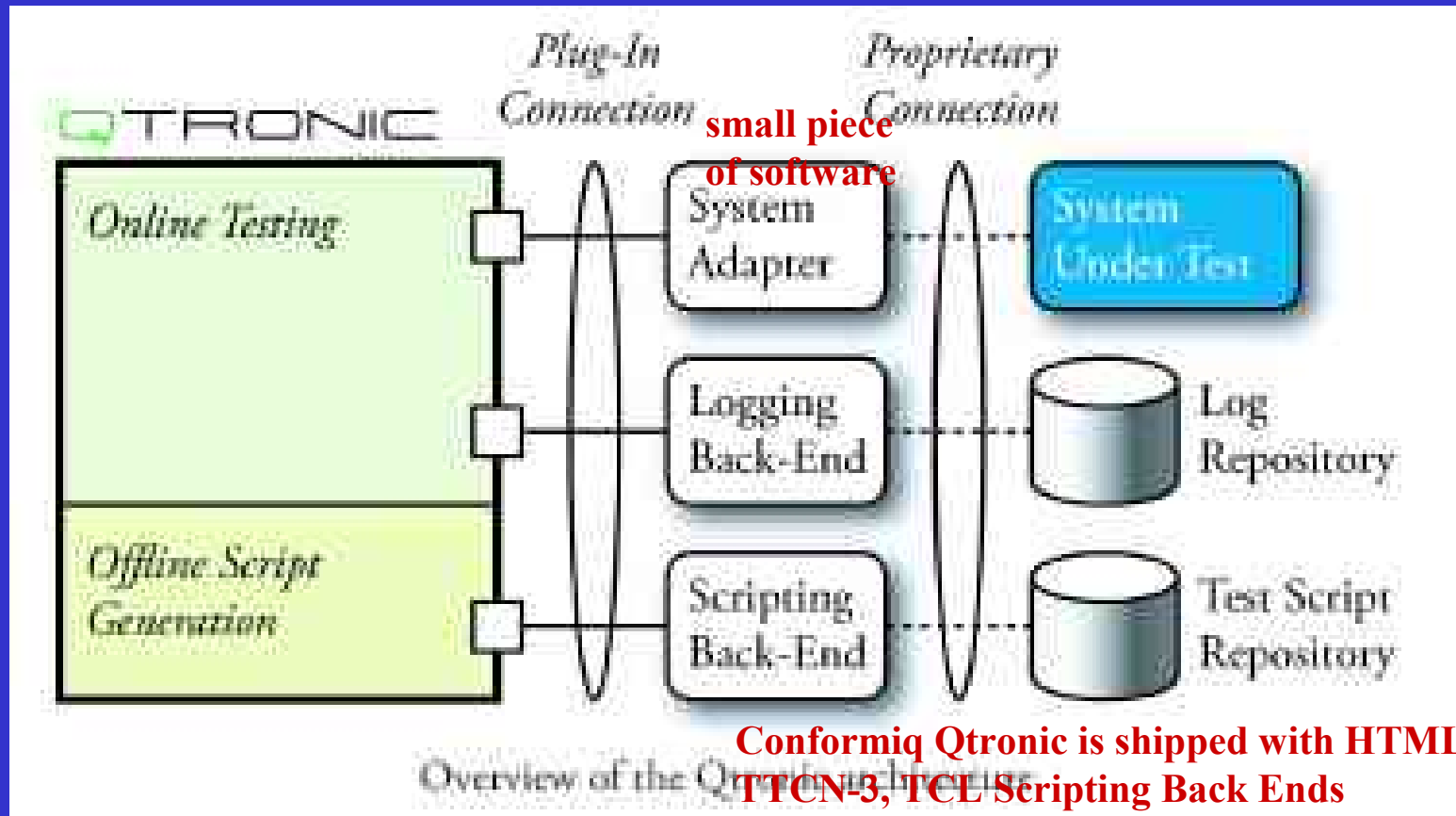
instead of

time consuming manual writing of test cases

Conformiq Qtronic Workflow

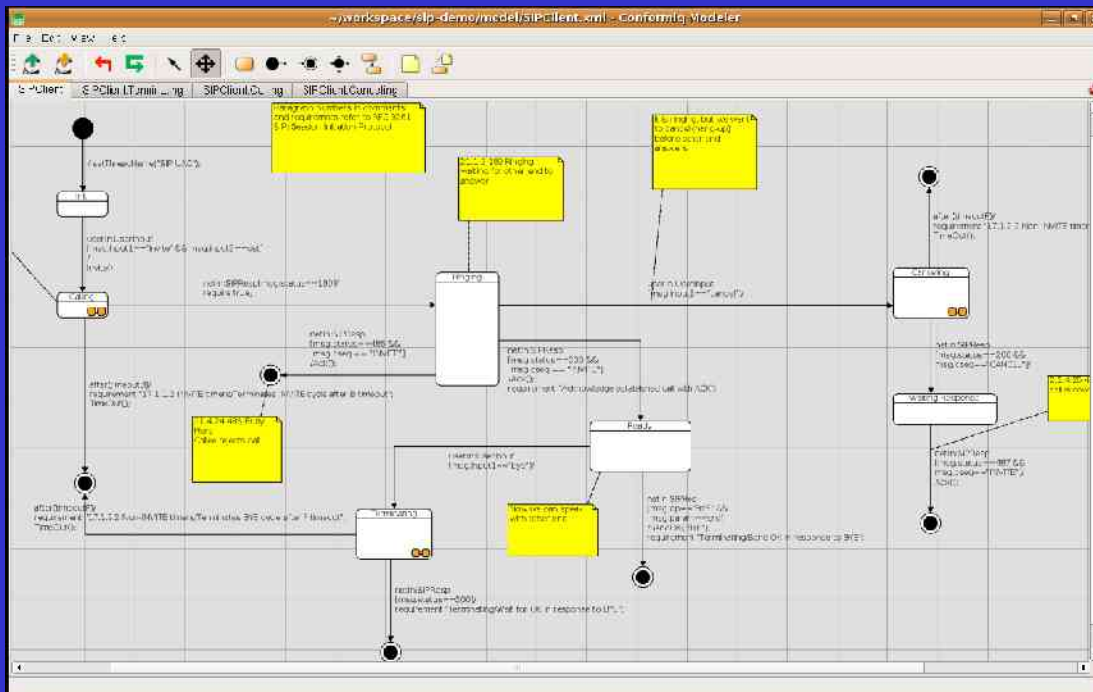


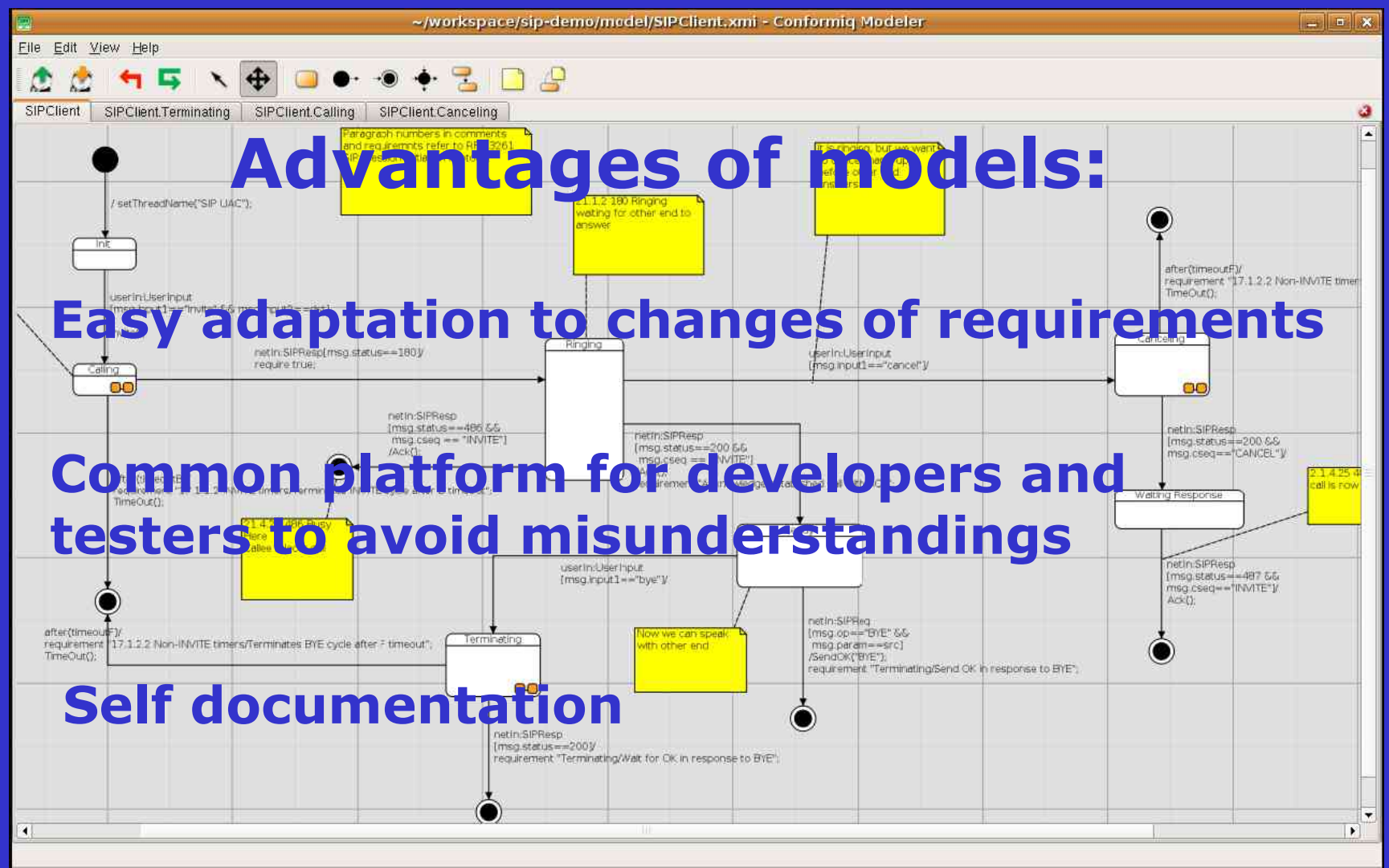
Conformiq Qtronic Architecture



Model Driven Testing

The model describes the attended behaviour of the software (or system) as seen by a user (Black-Box)





Advantages of models:

Easy adaptation to changes of requirements

Common platform for developers and testers to avoid misunderstandings

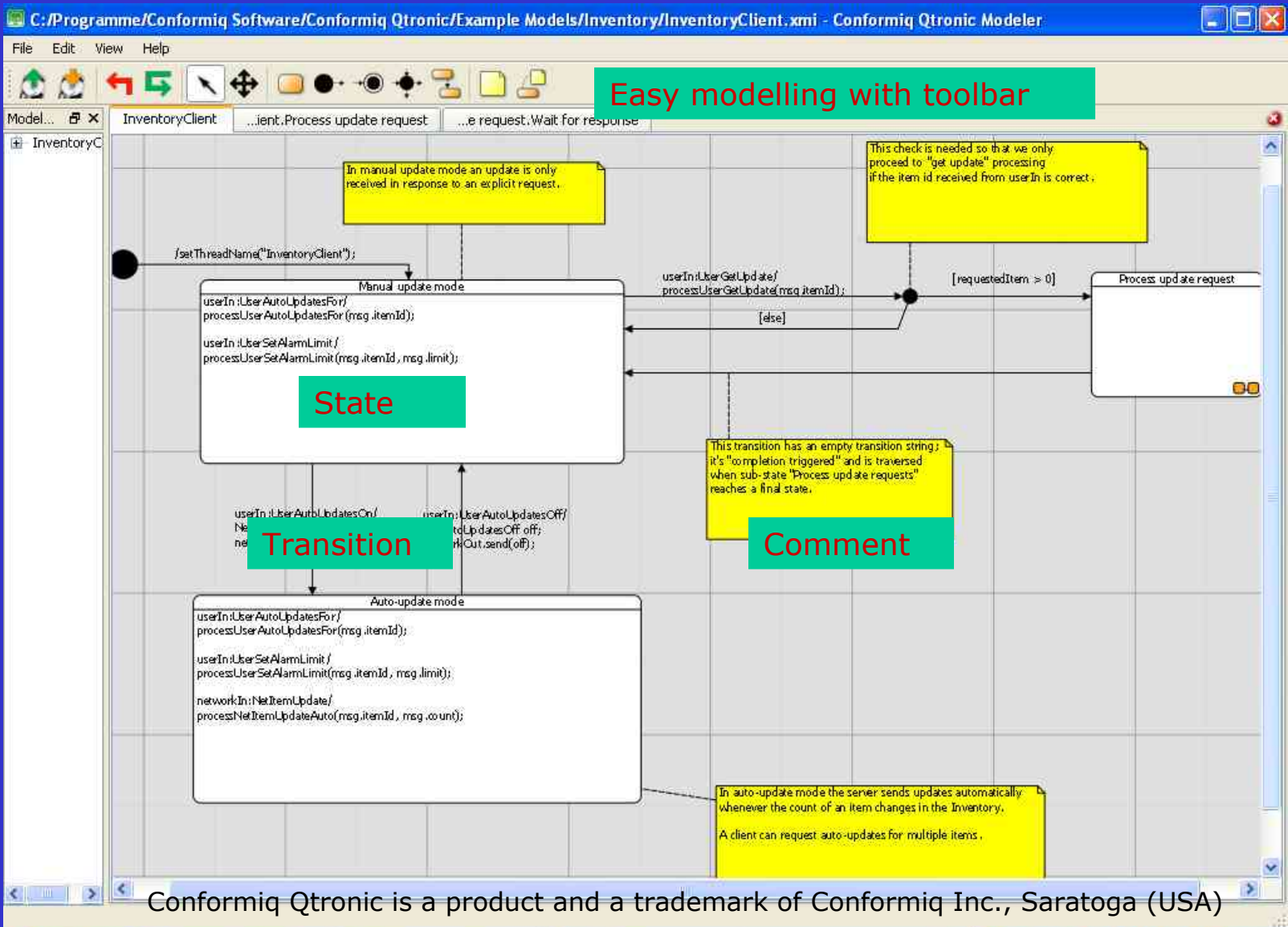
Self documentation

- textual in Java (with some C#-elements)
 - > „Qtronic Modelling Language“ (QML)
- grafical: UML State Charts (optional)

- Model can be set up with
 - text editor („Java“)
 - Qtronic Modeller (UML State Charts)
 - Third Party Modeling (UML) Tools

Supported Constructs

- **Data** (strings, numbers, records, classes, arrays...)
- **Time** (timeouts, dynamic timeouts...)
- **Control structures** (methods, dynamic polymorphism...)
- **Concurrency** (multiple Java threads in model, ITC primitives...)
- **Java** + templates, macros, record values, type inference...



Conformiq Qtronic is a product and a trademark of Conformiq Inc., Saratoga (USA)

The screenshot displays the Conformiq Qtronic Modeler interface. The main window shows a state machine diagram for 'InventoryClient'. A state is labeled 'Manual update mode' with the following transitions:

- Initial state: /setThreadName("InventoryClient");
- Transition to 'Manual update mode': userIn:UserAutoUpdatesFor/processUserAutoUpdatesFor(m.sg.itemId);
- Transitions from 'Manual update mode':
 - userIn:UserSetAlarmLimit/processUserSetAlarmLimit(m.sg.itemId, m.sg.limit);
 - userIn:UserGetUpdate/processUserGetUpdate(m.sg.itemId);
- Transitions to 'Manual update mode':
 - [else]
 - [requestedItem]

Two yellow callout boxes provide context:

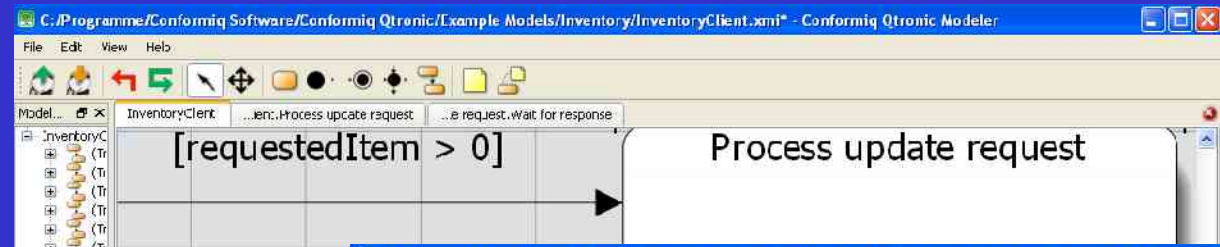
- "In manual update mode an update is only received in response to an explicit request."
- "This check is needed so that we proceed to 'get update' process if the item id received from us..."

An inset window shows the QML code for 'InventoryClientSystemBlock.cqa':

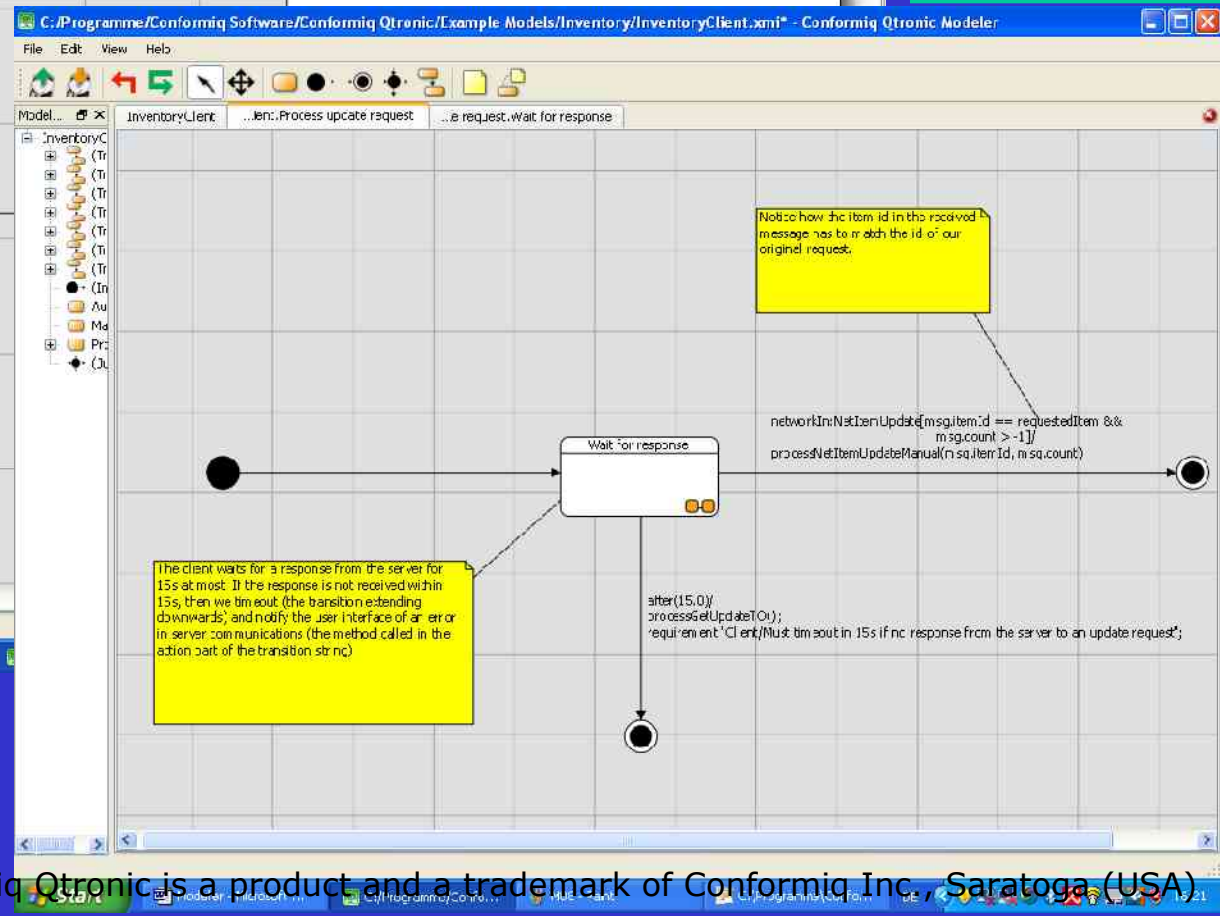
```

1 // -*- cqa -*-
2
3 system
4 {
5     Inbound networkIn: NetItemUpdate;
6     Outbound networkOut: NetAutoUpdatesFor, NetGetUpdate, NetAutoUpdatesOff;
7
8     Inbound userIn: UserAutoUpdatesFor, UserSetAlarmLimit, UserGetUpdate,
9     UserAutoUpdatesOn, UserAutoUpdatesOff;
10    Outbound userOut: UserItemUpdate, UserItemAlarm, UserError;
11
12 }
13
    
```

textual description
in „Java + C#“
(QML)



Substate

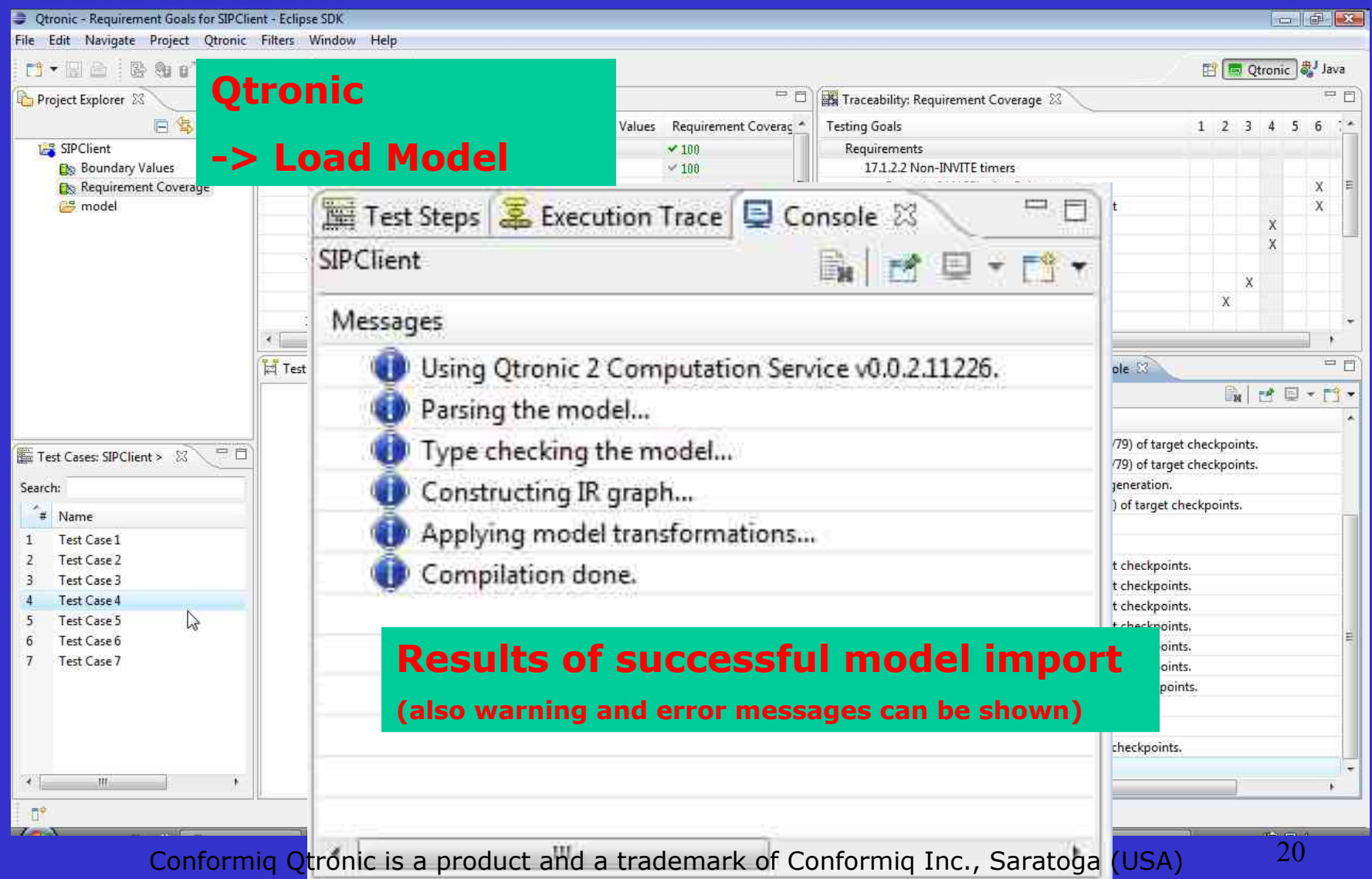


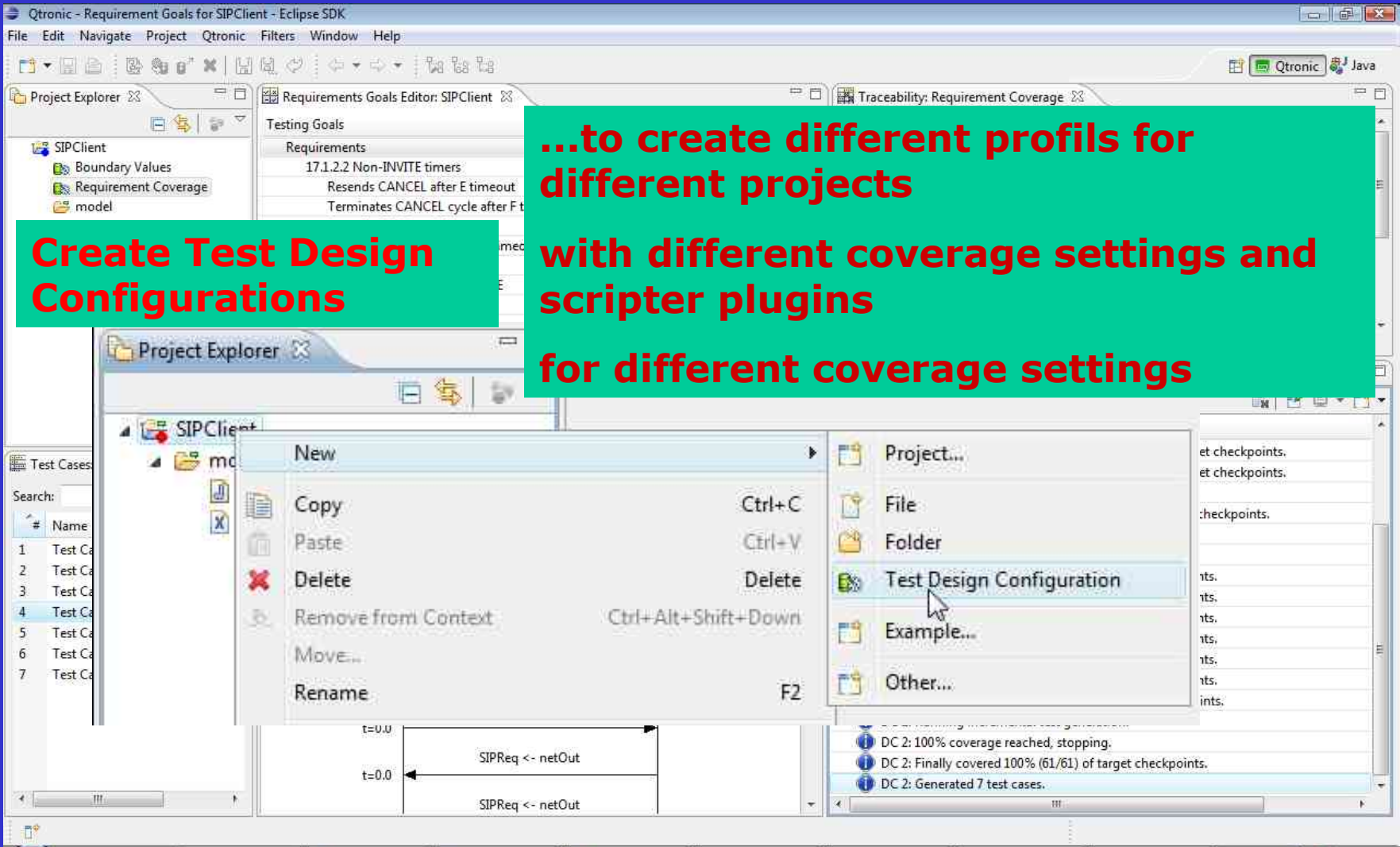
The screenshot displays the Eclipse IDE interface with the Qtronic plugin. The main workspace is divided into several panes:

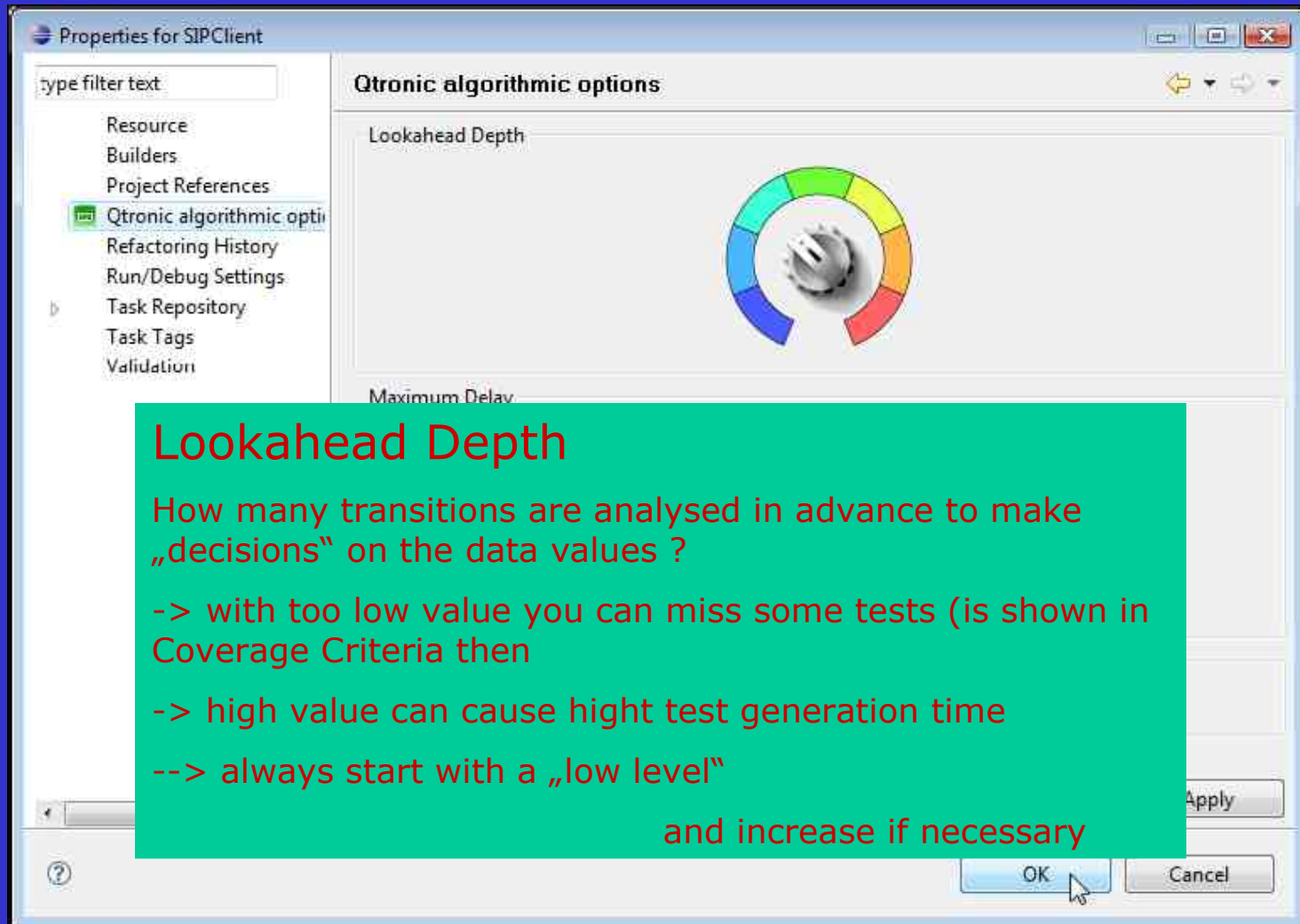
- Project Explorer:** Shows the project structure for 'SIPClient', including folders for 'Boundary Value', 'Requirement Coverage', and 'model', and files 'SIPClient.java' and 'SIPClient.xmi'. A red box highlights the 'Project Explorer' label.
- Requirements Goals Editor: SIPClient:** A table showing testing goals and their coverage.

Testing Goals	Boundary Values	Requirement Coverage
Requirements	✓ 100	✓ 100
Resends CANCEL after E timeout	✓ 100	✓ 100
Terminates CANCEL cycle after F timeout	✓	✓
Resends BYE after E timeout	✓	✓
Terminates BYE cycle after F timeout	✓	✓
Terminating	✓	✓
Wait for OK in response to BYE	✓	✓
Send OK in response to BYE	✓	✓
17.1.1.2 INVITE timers	✓	✓
- Traceability: Requirement Coverage:** A table showing coverage for specific requirements.

Requirements	1	2	3	4	5	6
17.1.2.2 Non-INVITE timers						
Resends CANCEL after E timeout						X
Terminates CANCEL cycle after F timeout						X
Resends BYE after E timeout				X		
Terminates BYE cycle after F timeout				X		
Terminating						
Wait for OK in response to BYE				X		
Send OK in response to BYE		X				
17.1.1.2 INVITE timers						
- Steps: Test Case 4 Execution Trace Console:** Shows a list of messages and progress reports for 'DC 2' test cases, including coverage percentages and test case generation status.
- UML Sequence Diagram:** Shows a participant 'SIP UAC' with a self-call arrow at 't=0.0' and two outgoing messages labeled 'SIPReq <- netOut'.





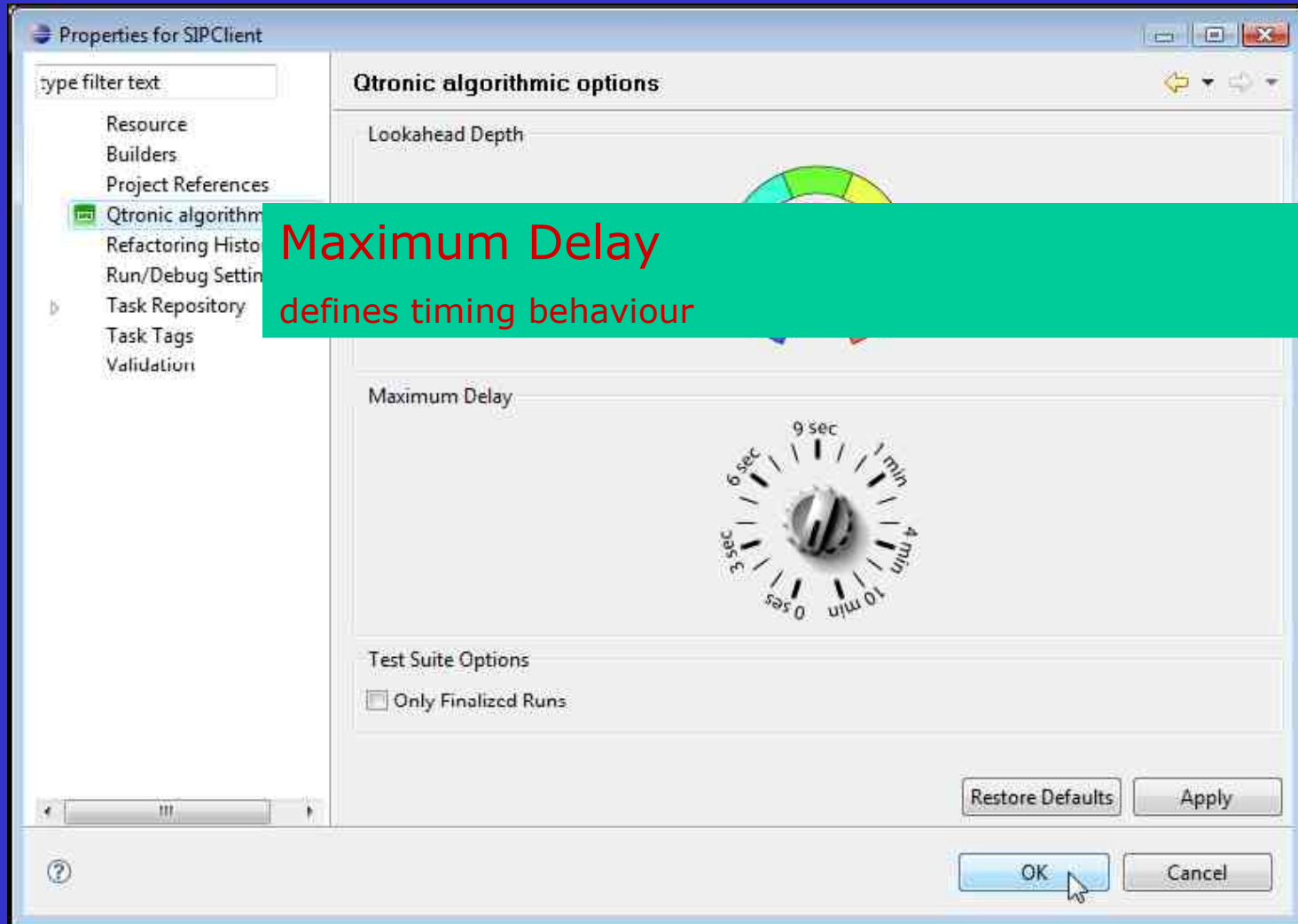


Lookahead Depth

How many transitions are analysed in advance to make „decisions“ on the data values ?

- > with too low value you can miss some tests (is shown in Coverage Criteria then
- > high value can cause high test generation time
- > always start with a „low level“

and increase if necessary



Selecting Coverage Criteria in Coverage Editor

Testing Goals	Boundary Values	Requirement Coverage
▲ Requirements	✓ 100	✓ 100
▲ 17.1.2.2 Non-INVITE timers	✓ 100	✓ 100
Resends CANCEL after E timeout	✓ ✓	✓ ✓
Terminates CANCEL cycle after F timeout	✓ ✓	✓ ✓
Resends BYE after F timeout	✓ ✓	✓ ✓
Terminates BYE cycle after F timeout	✓ ✓	✓ ✓
▲ Terminating	✓ 100	✓ 100
Wait for OK in response to BYE	✓ ✓	✓ ✓
Send OK in response to BYE	✓ ✓	✓ ✓
▲ 17.1.1.2 INVITE timers	✓ 100	✓ 100
Terminates INVITE cycle after B timeout	✓ ✓	✓ ✓
Resends INVITE after A timeout	✓ ✓	✓ ✓
Acknowledge established call with ACK	✓ ✓	✓ ✓
▲ State Chart	✓ 100	✓ 100
▷ 2-Transitions	-	-
▷ Transitions	✓ 100	✓ 100
▷ Implicit Consumption	✗	✗
▷ States	✓ 100	✓ 100
▷ Control Flow	✓ 100	✓ 100
▲ Conditional Branching	✓ 44	✓
▷ Boundary Value Analysis	✓ 44	-
▷ Atomic Branches	-	-
▷ Dynamic Coverage	-	-

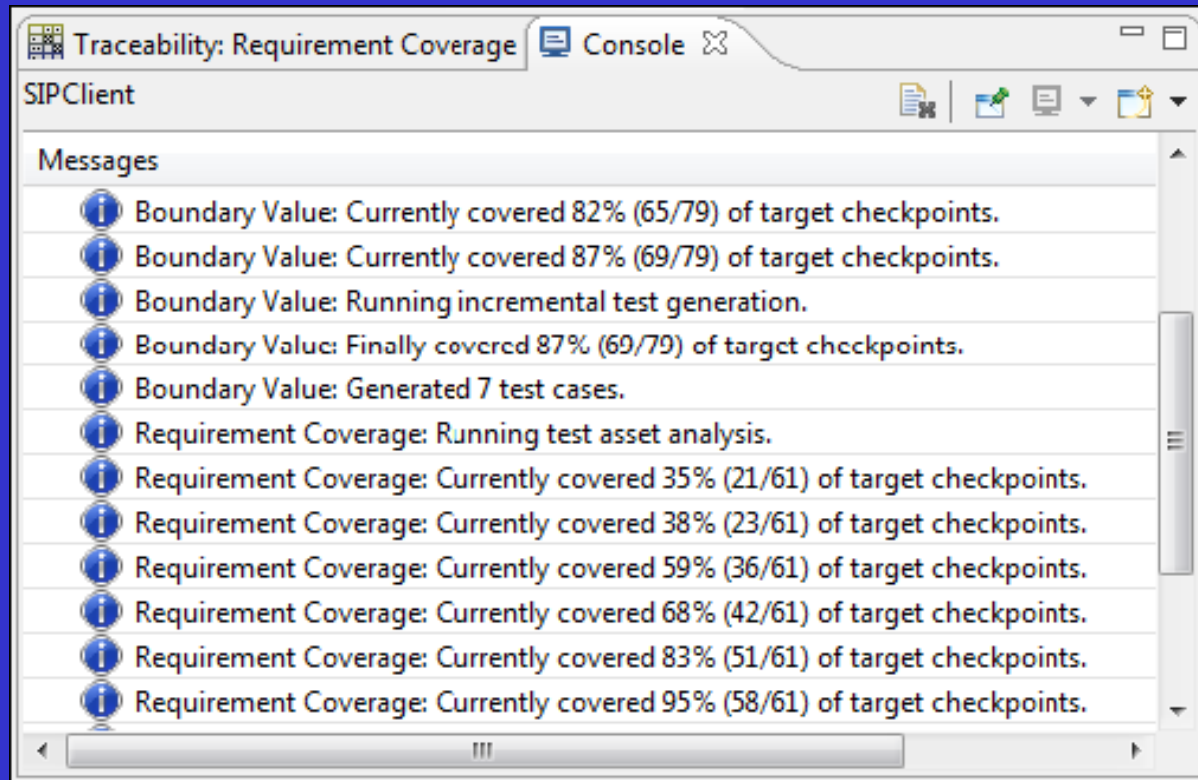
Coverage Criteria

- State Coverage: covers all states in all state charts
- Boundary Value Coverage
- Branch Coverage
- Transition Coverage: covers all transitions in all state charts
- 2-transition Coverage: covers all transition sequences of length 2 in all state charts
- Atomic Condition Coverage
- Method Coverage: covers every QML level method
- Statement Coverage: covers every QML leved statement
- MC/DC-Coverage (coming soon...)

How to generate Tests ?

Once the model files have been selected, the model has been imported to Conformiq Qtronic, test design configuration/coverage criteria have been set

Test Generation can be started by selecting „Update Test Set“



Status of Test Generation shown in Eclipse console

-once the test generation finishes, the final coverage will be shown (for all, for each requirement and structural feature)

Generated Test Cases

Test Cases: SIPClient

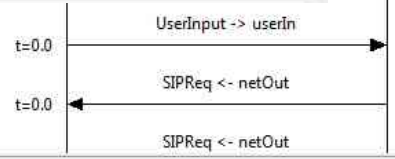
#	Name
1	Test Case 1
2	Test Case 2
3	Test Case 3
4	Test Case 4
5	Test Case 5
6	Cancel Timeouted
7	Test Case 7

Requirement	1	2	3	4	5	6
17.1.2.2 Non-INVITE timers						
Resends CANCEL after E timeout						X
Terminates CANCEL cycle after F timeout						X
Resends BYE after E timeout				X		
Terminates BYE cycle after F timeout				X		
Terminating						
Wait for OK in response to BYE				X		
Send OK in response to BYE		X				
17.1.1.2 INVITE timers						

Messages

- Boundary Values: Currently covered 79% (63/79) of target checkpoints.
- Boundary Values: Currently covered 87% (69/79) of target checkpoints.
- Boundary Values: Running incremental test generation.
- Boundary Values: Finally covered 87% (69/79) of target checkpoints.
- Boundary Values: Generated 7 test cases.
- DC 2: Running test asset analysis.
- DC 2: Currently covered 35% (21/61) of target checkpoints.
- DC 2: Currently covered 48% (29/61) of target checkpoints.
- DC 2: Currently covered 50% (31/61) of target checkpoints.
- DC 2: Currently covered 72% (44/61) of target checkpoints.
- DC 2: Currently covered 83% (51/61) of target checkpoints.
- DC 2: Currently covered 90% (55/61) of target checkpoints.
- DC 2: Currently covered 100% (61/61) of target checkpoints.
- DC 2: Running incremental test generation.
- DC 2: 100% coverage reached, stopping.
- DC 2: Finally covered 100% (61/61) of target checkpoints.
- DC 2: Generated 7 test cases.

Test Case List



Generated Test Cases

Traceability Matrix:
shows for each test case what it covers

Testing Goals	1	2	3	4	5	6
Requirements						
17.1.2.2 Non-INVITE timers						
Resends CANCEL after E timeout						X
Terminates CANCEL cycle after F timeout						X
Resends BYE after E timeout				X		
Terminates BYE cycle after F timeout				X		
Terminating						
Wait for OK in response to BYE			X			
Send OK in response to BYE		X				
17.1.1.2 INVITE timers						
Terminates INVITE cycle after B timeout						
Resends INVITE after A timeout						
Acknowledge established call with ACK		X	X	X		
State Chart						
2-Transitions						
Transitions						
Implicit Consumption						
States						
SIPClient-Init	X	X	X	X	X	X
SIPClient-Calling-initial-0	X	X	X	X	X	X
SIPClient-Calling-Wait	X	X	X	X	X	X
SIPClient-Calling-junction-2						
SIPClient-Ready		X	X	X		

covered 48% (29/61) of target checkpoints,
covered 50% (31/61) of target checkpoints,
covered 72% (44/61) of target checkpoints,
covered 83% (51/61) of target checkpoints,
covered 90% (55/61) of target checkpoints,
covered 100% (61/61) of target checkpoints.
incremental test generation.
coverage reached, stopping.
covered 100% (61/61) of target checkpoints.
37 test cases.

Generated Test Cases

Message Sequence Chart for a test case

```

sequenceDiagram
    participant Tester
    participant SIP UAC
    Note over Tester: t=0.0
    Tester->>SIP UAC: UserInput -> userIn
    Note over SIP UAC: t=0.0
    SIP UAC-->>Tester: SIPReq <- netOut
    Note over Tester: t=0.0
    Tester->>SIP UAC: SIPResp -> netIn
    Note over SIP UAC: t=0.0
    Tester->>SIP UAC: SIPResp -> netIn
    Note over Tester: t=0.0
    SIP UAC-->>Tester: SIPReq <- netOut
    Note over Tester: t=0.0
    Tester->>SIP UAC: UserInput -> userIn
    Note over SIP UAC: t=0.0
    SIP UAC-->>Tester: SIPReq <- netOut
    Note over Tester: t=0.5
    SIP UAC-->>Tester: SIPReq <- netOut
    
```

Coverage Matrix:

	1	2	3	4	5	6
Requirement 1				X		
Requirement 2				X		X
Requirement 3			X			
Requirement 4		X				
Requirement 5						
Requirement 6						
Requirement 7						
Requirement 8						
Requirement 9						
Requirement 10						
Requirement 11						
Requirement 12						
Requirement 13						
Requirement 14						
Requirement 15						
Requirement 16						
Requirement 17						
Requirement 18						
Requirement 19						
Requirement 20						
Requirement 21						
Requirement 22						
Requirement 23						
Requirement 24						
Requirement 25						
Requirement 26						
Requirement 27						
Requirement 28						
Requirement 29						
Requirement 30						
Requirement 31						
Requirement 32						
Requirement 33						
Requirement 34						
Requirement 35						
Requirement 36						
Requirement 37						
Requirement 38						
Requirement 39						
Requirement 40						
Requirement 41						
Requirement 42						
Requirement 43						
Requirement 44						
Requirement 45						
Requirement 46						
Requirement 47						
Requirement 48						
Requirement 49						
Requirement 50						
Requirement 51						
Requirement 52						
Requirement 53						
Requirement 54						
Requirement 55						
Requirement 56						
Requirement 57						
Requirement 58						
Requirement 59						
Requirement 60						
Requirement 61						
Requirement 62						
Requirement 63						
Requirement 64						
Requirement 65						
Requirement 66						
Requirement 67						
Requirement 68						
Requirement 69						
Requirement 70						
Requirement 71						
Requirement 72						
Requirement 73						
Requirement 74						
Requirement 75						
Requirement 76						
Requirement 77						
Requirement 78						
Requirement 79						
Requirement 80						
Requirement 81						
Requirement 82						
Requirement 83						
Requirement 84						
Requirement 85						
Requirement 86						
Requirement 87						
Requirement 88						
Requirement 89						
Requirement 90						
Requirement 91						
Requirement 92						
Requirement 93						
Requirement 94						
Requirement 95						
Requirement 96						
Requirement 97						
Requirement 98						
Requirement 99						
Requirement 100						

Generated Test Cases

The screenshot displays the Verifysoft interface with three main components:

- Test Case List:** A list of seven test cases for 'SIPClient', with 'Test Case 4' selected.
- Tester Timeline:** A vertical timeline for 'Tester' showing execution steps at t=0.0 and t=0.5.
- Steps: Cancel Timeouted View:** A detailed view of a message step (Step 5) showing fields like 'Message / Field', 'Port / Field value', and 'Time'. The message content includes 'SIPReq', 'op: CANCEL', and 'param: sip:127.0.0.1:5061'.
- Matrix:** A grid showing coverage for each test case across various scenarios. The matrix is as follows:

	1	2	3	4	5	6
Scenario 1						
Scenario 2				X		X
Scenario 3				X		X
Scenario 4			X			
Scenario 5		X				
Scenario 6	X	X	X	X	X	X
Scenario 7	X	X	X	X	X	X

Test Case List

Message Step View

Test Step View with more detailed information about the message content

Matrix: each test it covers

Generated Test Cases

The screenshot displays a software testing interface with several windows. The central window shows a sequence diagram between 'Tester' and 'SIP UAC' with messages like 'UserInput -> userIn' and 'SIPReq <- netOut' at various time points. To the right, a flowchart shows the execution path through components like 'SIPClient-initial-4', 'SIPClient-Init', 'SIPClient.Invite()', 'SIPClient-Calling-initial-0', 'SIPClient-Calling-Wait', 'SIPClient-Ringing', and 'SIPClient.Ack()'. A table at the top right shows a grid with columns 1-6. A 'Steps: Cancel Timeouted' window shows a table with columns 'Message / Field', 'Port / Field value', and 'Time', containing the entry 'UserInput to userIn 0.0'. A 'Console' window is also visible.

Execution Trace View
Shows the execution trace of the test case in the model

Test Cases can be exported using plugin scripters

Scripters „translates“ the Conformiq Qtronic internal representation to a predefined syntas used in the test execution harness

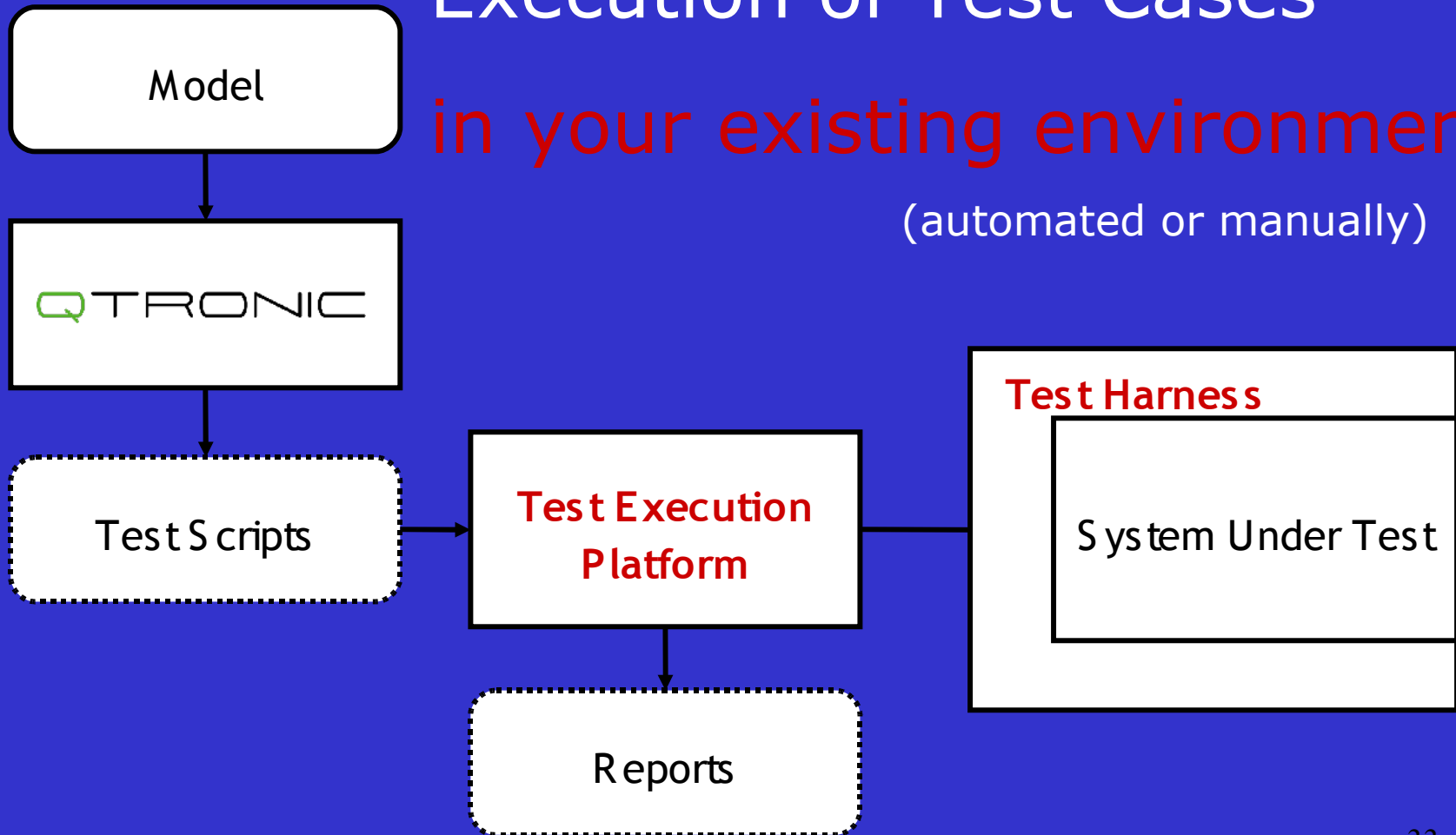
Three default scripters are provided with Conformiq Qtronic:

- HTML TTCN-3 TCL
- other formats can be „written“

Execution of Test Cases

Execution of Test Cases

in your existing environment
(automated or manually)



Tool Evaluation



Presentation



Pilot Project



Verifysoft Technology GmbH
Technologiepark
In der Spoeck 10
77656 OFFENBURG (Germany)

Contact: Klaus LAMBERTZ
Phone +49 781 6392 027 / +33 3 68 33 58 84
quality@verifysoft.com

www.verifysoft.com

Thank you !