



Automatic Test Generation for Functional Testing

with Conformiq Tool Suite

© 2011 Verifysoft Technology GmbH

29.04.2011

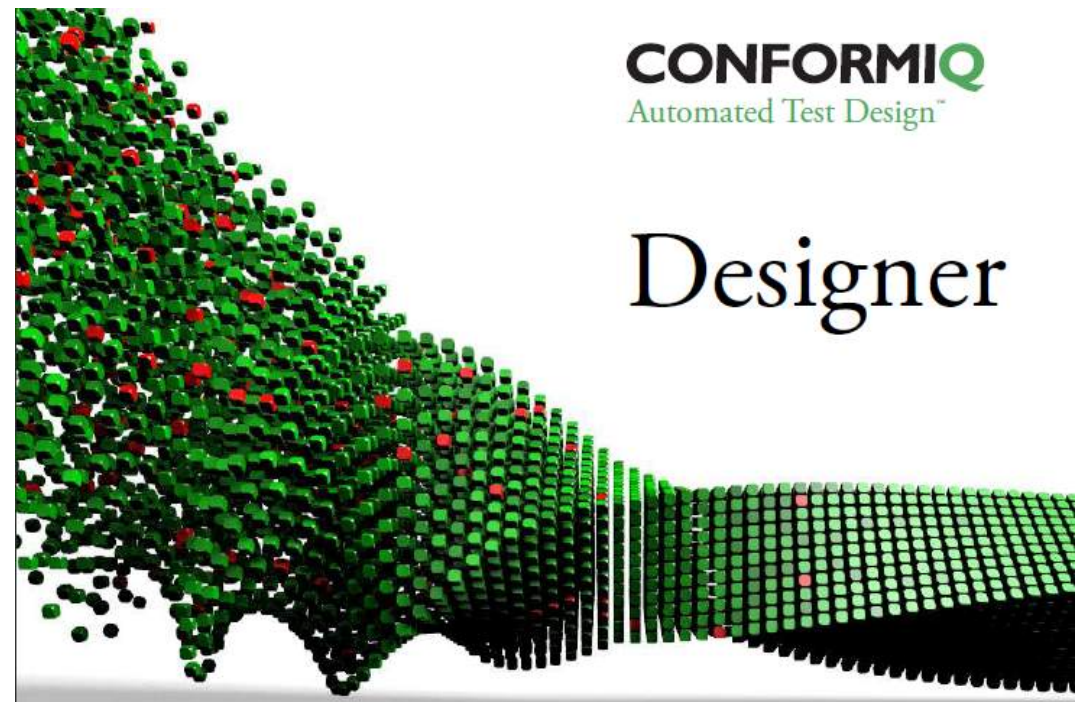

TECHNOLOGY



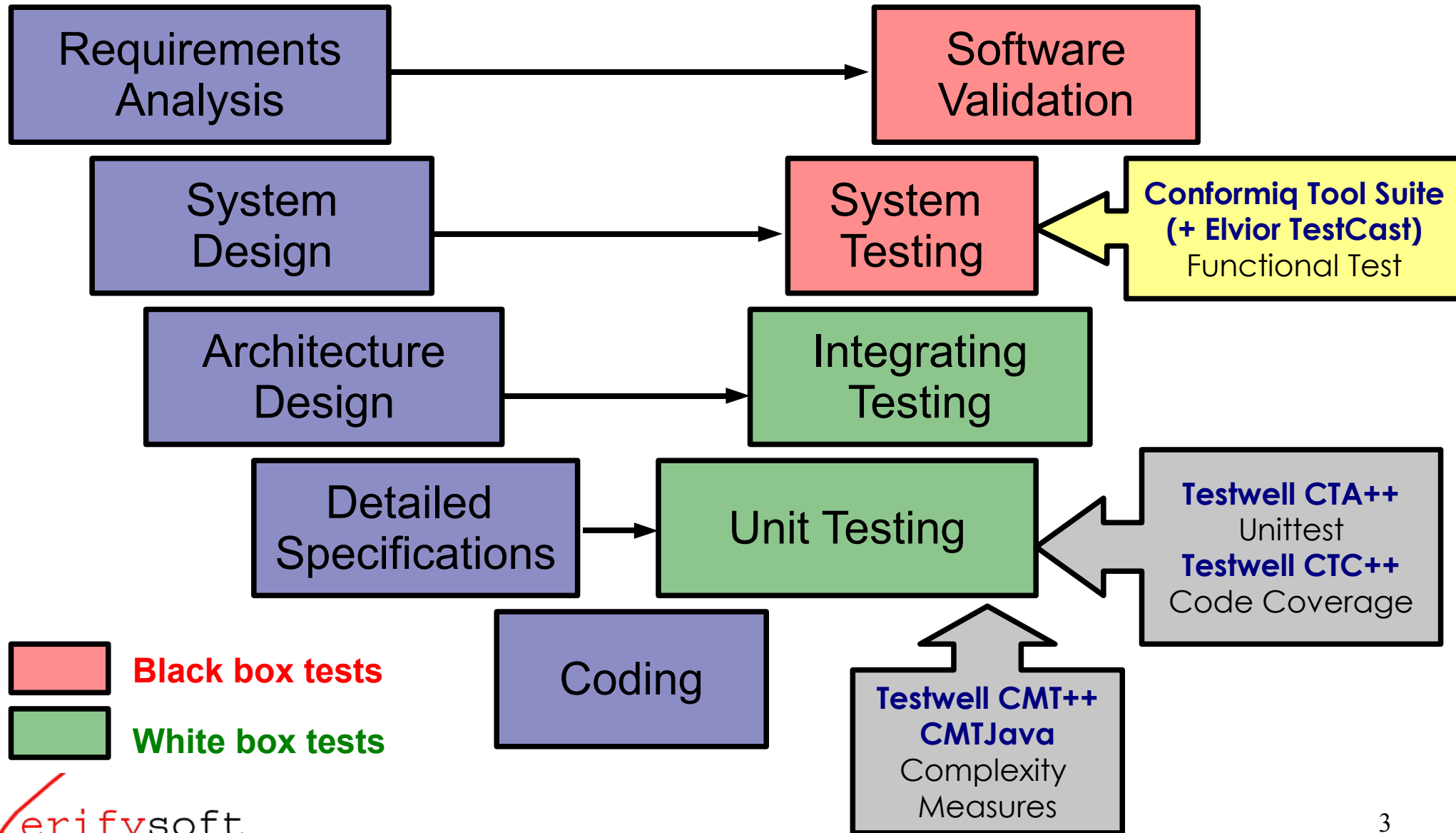
Conformiq Tool Suite

Conformiq Tool Suite:
Model-Based Automated Test Design
for functional tests

(black box tests)
for software and systems



Conformiq Tool Suite



Conformiq Designer

Why automate test generation?

Manual tests takes time...

and leads to risks

- 💣 incorrect tests
- 💣 missed tests
- 🕒 redundant tests
- 🕒 maintenance of tests





Conformiq Designer

Our solution: **Automated Test Design**

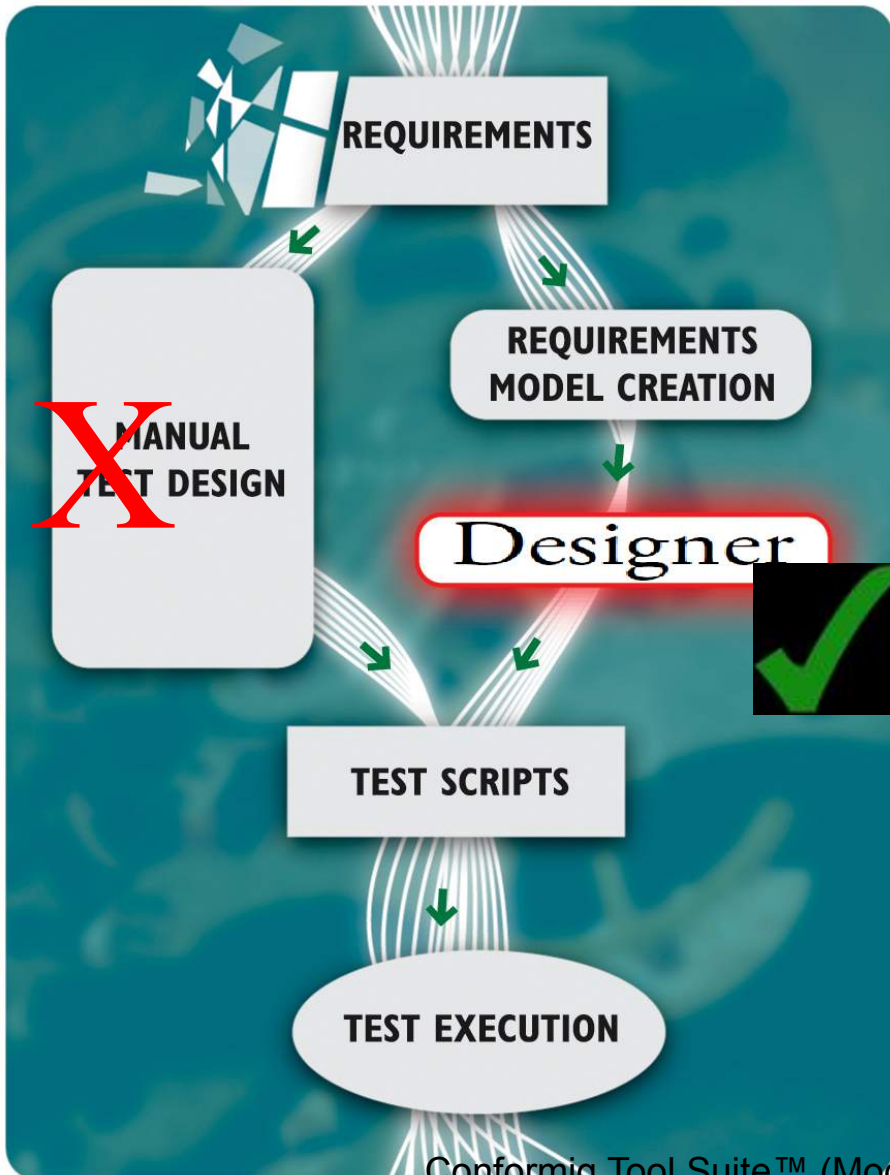
→ model driven testing, model based testing,
specification based testing,
specification driven testing,
...



Manual vs. Automatic

Automatic test generation and execution based on your design models

instead of writing the tests manually



Test Generation

Process:

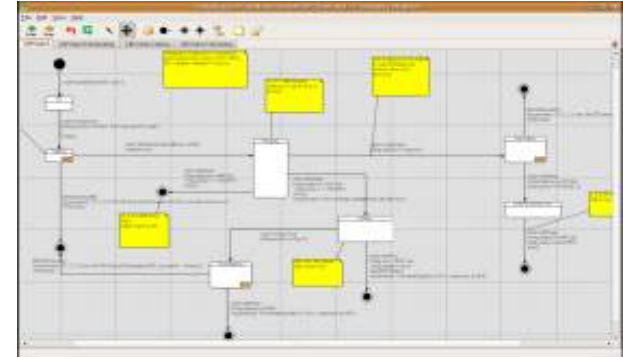
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Conformiq Modeler

C:/Programme/Conformiq Software/Conformiq Qtronic/Example Models/Inventory/InventoryClient.xmi - Conformiq Qtronic Modeler

File Edit View Help

Easy modelling with tool box

InventoryClient

In manual update mode an update is only received in response to an explicit request.

This check is needed so that we only proceed to "get update" processing if the item id received from userIn is correct.

State

Transition

Comment

In auto-update mode the server sends updates automatically whenever the count of an item changes in the Inventory. A client can request auto-updates for multiple items.

userIn:UserGetUpdate/processUserGetUpdate(msg.itemId);

[requestedItem > 0]

Process update request

[else]

Manual update mode

userIn:UserAutoUpdatesFor/processUserAutoUpdatesFor(msg.itemId);

userIn:UserSetAlarmLimit/processUserSetAlarmLimit(msg.itemId, msg.limit);

Auto-update mode

userIn:UserAutoUpdatesOn/NetAutoUpdatesOn(on); networkOut.send(on);

userIn:UserAutoUpdatesOff/NetAutoUpdatesOff(off); networkOut.send(off);

userIn:UserAutoUpdatesFor/processUserAutoUpdatesFor(msg.itemId);

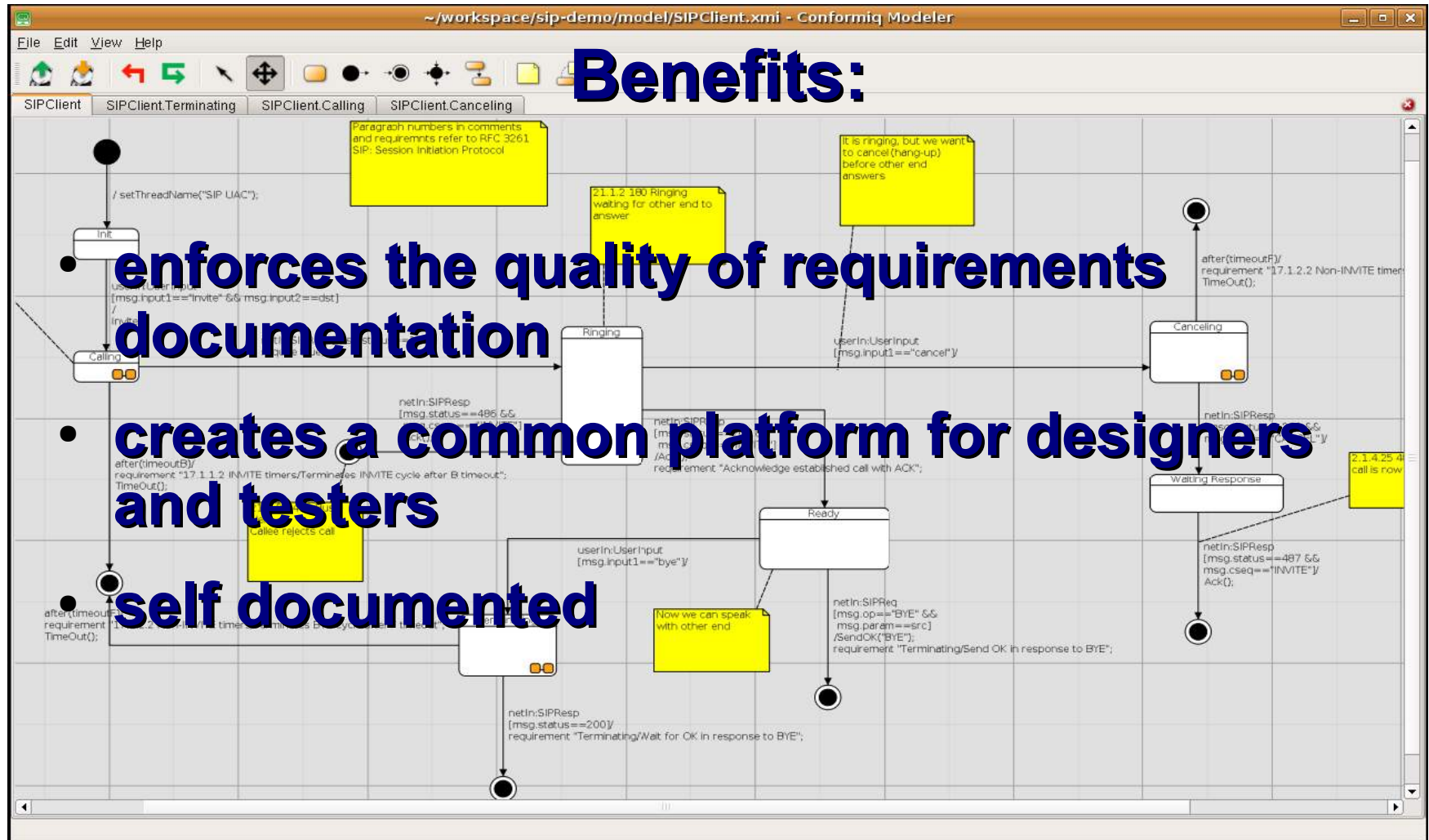
userIn:UserSetAlarmLimit/processUserSetAlarmLimit(msg.itemId, msg.limit);

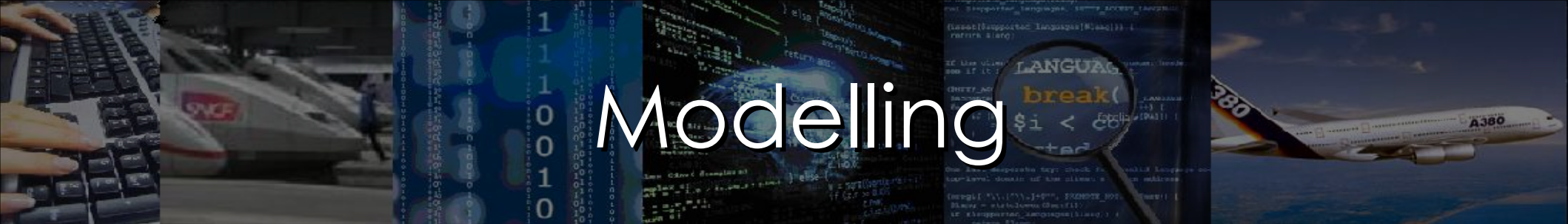
networkIn:NetItemUpdate/processNetItemUpdateAuto(msg.itemId, msg.count);

Veri

Start Modeller - Microsoft ... C:/Programme/Confo... M01 - Paint DE 15:40

Model Driven Testing





Modelling

- Textually in Java (with elements in C#)
- Grafically in UML State Charts (optional)
- The model can be created with:
 - Text editor („Java“)
 - Conformiq Modeler (UML State Charts)
 - or Third Party Modeling (UML) tools

Test Generation

Process:

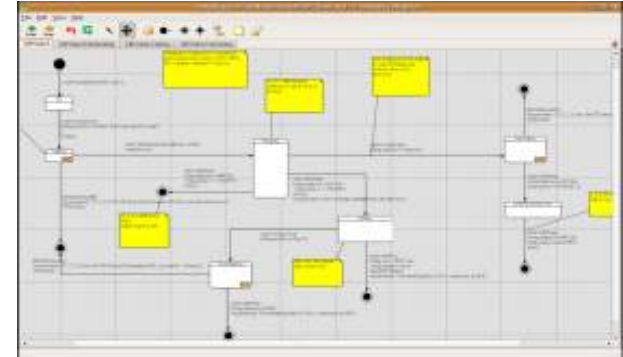
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Test Generation

Process:

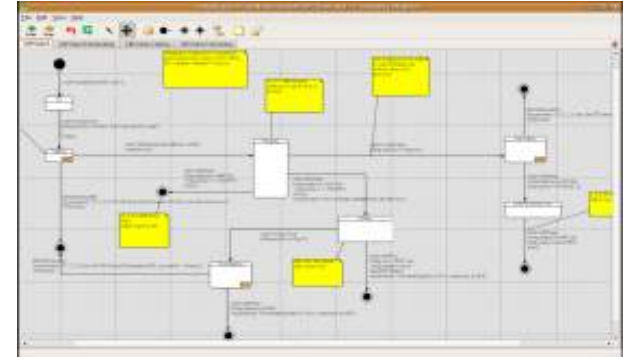
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts

Automatic test generation with Conformiq Designer

Test execution in your environment.





Test Generation

Choice of Test Coverage: State Coverage

(covers every state at least once)

Transition Coverage

(covers every transition at least once)

2-Transition Coverage

(covers every pair of two subsequent transitions at least once)



Test Generation

Choice of Test Coverage (continued):

Boundary Value Analysis

Branch Coverage

Atomic Condition Coverage

Method Coverage

Statement Coverage

Parallel Transition Coverage ... etc...



Test Generation

Formats for the test scripts generation:

Python

TCL

TTCN-3

C, C++

Visual Basic

Java

Junit

Perl

Excel

HTML

Word

Shell Scripts

Test Generation

Process:

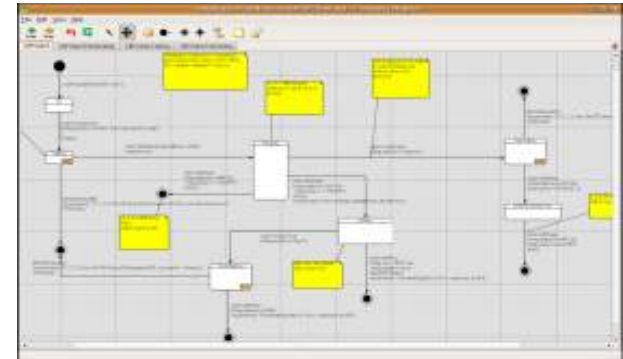
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Test Generation

The screenshot displays the Eclipse IDE interface for a project named 'Qtronic - Requirement Goals for SIPClient'. The interface is divided into several panes:

- Project Explorer:** Shows the project structure with folders for 'Boundary Values', 'Requirement Coverage', and 'model'.
- Requirements Goals Editor:** A table showing testing goals and their coverage. All goals are marked as 100% covered.
- Traceability: Requirement Coverage:** A table showing the coverage of requirements across different test cases (1-6).
- Test Case 4:** A sequence diagram showing the interaction between a 'Tester' and 'SIP UAC'. The diagram includes messages like 'UserInput -> userIn', 'SIPReq <- netOut', and 'SIPResp -> netIn'.
- Steps: Test Case 4 / Execution Trace / Console:** A log of messages and progress updates, including coverage percentages for various checkpoints and test case generation details.

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 E 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

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						

```
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
    SIP UAC-->>Tester: SIPResp -> netIn
    Note over Tester: t=0.0
    Tester->>SIP UAC: SIPReq -> netIn
    Note over SIP UAC: 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=0.0
    Tester->>SIP UAC: SIPReq <- netOut
```

Console Log:

- 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 Generation

**Test scripts
Generation progress**

Boundary Value:

- Currently covered 82% (65/79) of target checkpoints.
- Currently covered 87% (69/79) of target checkpoints.
- Running incremental test generation.
- Finally covered 87% (69/79) of target checkpoints.
- Generated 7 test cases.

Requirement Coverage:

- Running test asset analysis.
- Currently covered 35% (21/61) of target checkpoints.
- Currently covered 38% (23/61) of target checkpoints.
- Currently covered 59% (36/61) of target checkpoints.
- Currently covered 68% (42/61) of target checkpoints.
- Currently covered 83% (51/61) of target checkpoints.
- Currently covered 95% (58/61) of target checkpoints.

DC 2:

- Currently covered 48% (29/61) of target checkpoints.
- Currently covered 50% (31/61) of target checkpoints.
- Currently covered 72% (44/61) of target checkpoints.
- Currently covered 83% (51/61) of target checkpoints.
- Currently covered 90% (55/61) of target checkpoints.
- Currently covered 100% (61/61) of target checkpoints.
- Running incremental test generation.
- 100% coverage reached, stopping.
- Finally covered 100% (61/61) of target checkpoints.
- Generated 7 test cases.

Sequence Diagram:

```
sequenceDiagram
    participant NetIn as netIn
    participant NetOut as netOut
    participant User as userIn
    SIPReq->>NetIn: SIPReq -> netIn
    NetIn-->>SIPResp: SIPResp -> netIn
    NetOut-->>SIPReq: SIPReq -> netOut
    User->>User: UserInput -> userIn
    NetOut-->>SIPReq: SIPReq -> netOut
    NetOut-->>SIPReq: SIPReq -> netOut
```

Test Generation

**Test scripts
Generation progress**

**List of
test scripts**

Qtronic - Requirement Goals for SIPClient - Eclipse SDK

File Edit Navigate Project Qtronic Filters Window Help

Project Explorer

- SIPClient
 - Boundary Values
 - Requirement Coverage
 - model

Requirements Goal

Testing Goals

Requirements

- 17.1.2.2 Non-Interfering
- Resends CA
- Terminates
- Resends BY
- Terminates
- Terminates

Console

SIPClient

Messages

- Boundary Value: Currently covered 82% (65/79) of target checkpoints.
- Boundary Value: Currently covered 87% (69/79) of target checkpoints.
- Boundary Value: Running incremental test generation.
- Boundary Value: Finally covered 87% (69/79) of target checkpoints.
- Boundary Value: Generated 7 test cases.
- Requirement Coverage: Running test asset analysis.
- Requirement Coverage: Currently covered 35% (21/61) of target checkpoints.
- Requirement Coverage: Currently covered 38% (23/61) of target checkpoints.
- Requirement Coverage: Currently covered 59% (36/61) of target checkpoints.
- Requirement Coverage: Currently covered 68% (42/61) of target checkpoints.
- Requirement Coverage: Currently covered 83% (51/61) of target checkpoints.
- Requirement Coverage: Currently covered 95% (58/61) of target checkpoints.

Test Cases: SIPClient

Search:

#	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

Sequence Diagram:

- SIPResp -> netIn
- SIPReq <- netOut
- UserInput -> userIn
- SIPReq <- netOut
- SIPReq <- netOut

Console (continued):

- 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 Generation

Qtronic - Requirement Goals for SIPClient - Eclipse SDK

File Edit Navigate Project Qtronic Filters Window Help

Project Explorer

Requirements Goal

Testing Goals

Requirements

- 17.1.2.2 Non-INVITE timers
- Resends CANCEL after E timeout
- 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
- Terminates INVITE cycle after B timeout
- Resends INVITE after A timeout
- Acknowledge established call with ACK
- State Chart
- 2-Transitions
- Transitions
- Implicit Consumption
- States
- SIPClient-Init
- SIPClient-Calling-initial-f
- SIPClient-Calling-Wait
- SIPClient-Calling-junction-2
- SIPClient-Ready

Test script Generation

Test Cases: SIPClient >

Search:

#	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

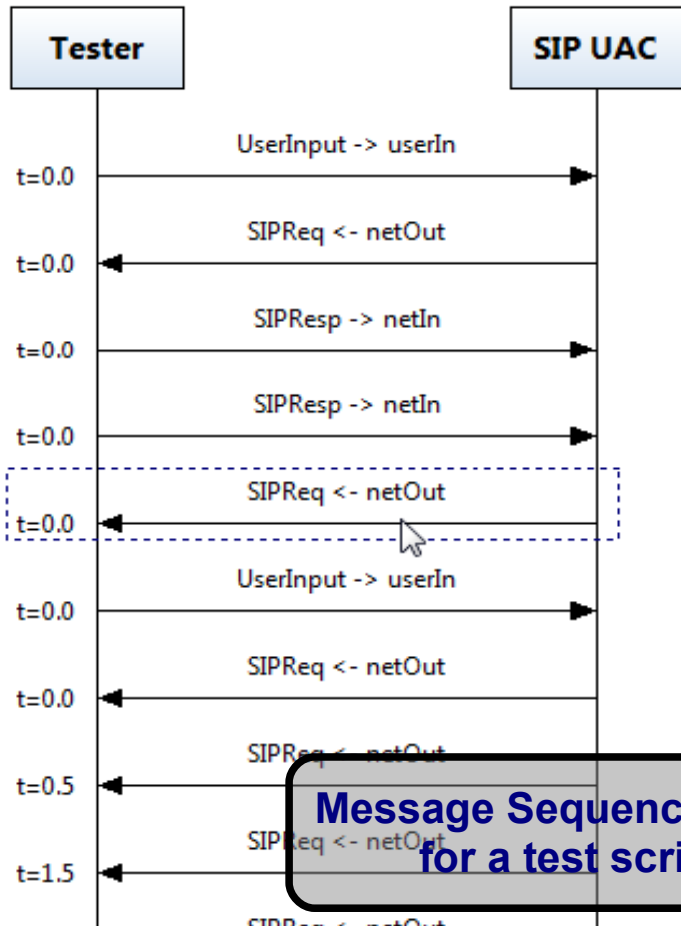
List of test scripts

Traceability Matrix

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
SIPClient-Calling-initial-f		X	X	X	X	X
SIPClient-Calling-Wait		X	X	X	X	X
SIPClient-Calling-junction-2		X	X	X	X	X
SIPClient-Ready		X	X	X	X	X

Requirement Traceability of the test scripts

Test Generation



Message Sequence Chart for a test script

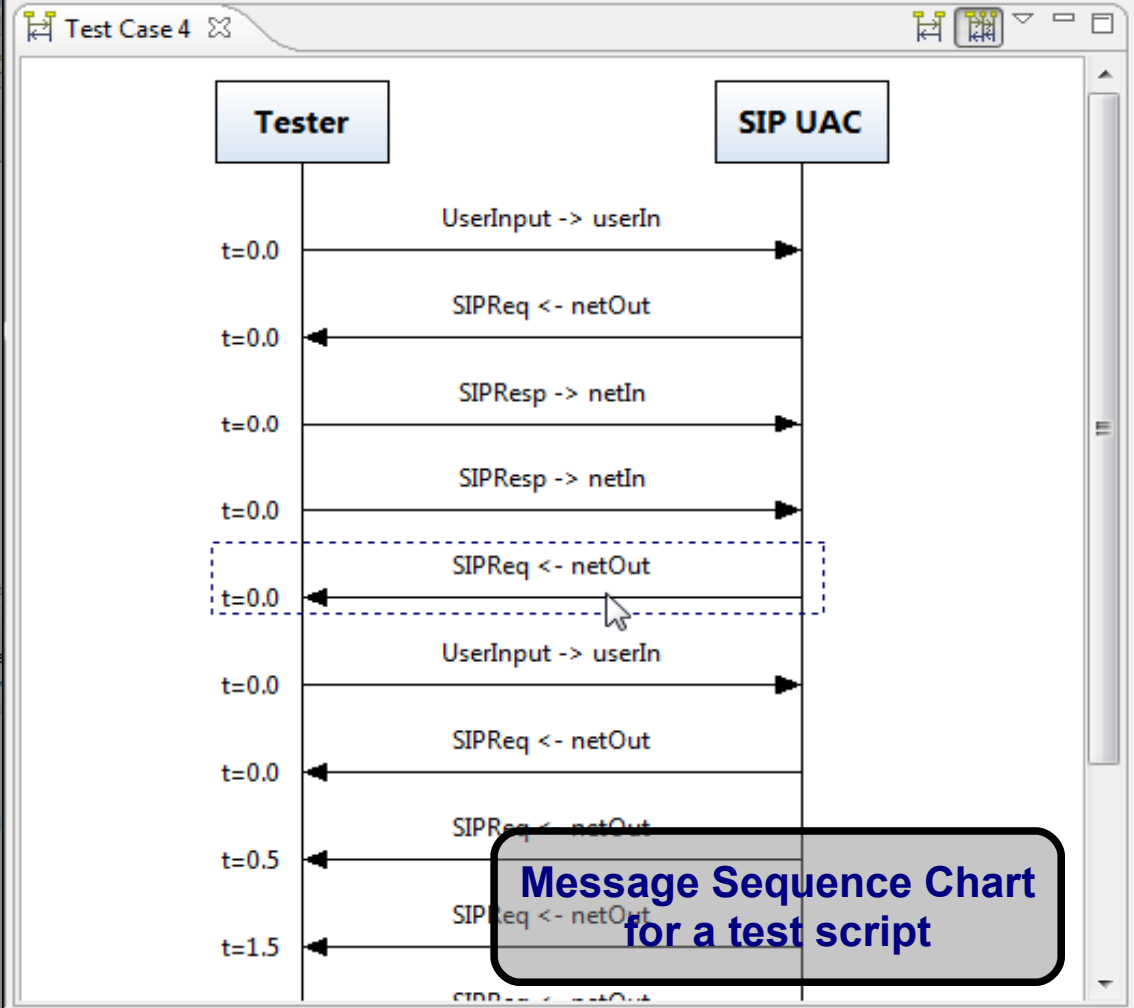
LIST OF test scripts

The Traceability Matrix shows the following data:

	1	2	3	4	5	6
INVITE timers						
CANCEL after E timeout						X
times CANCEL cycle after F timeout						X
BYE after E timeout				X		
times BYE cycle after F timeout				X		
OK in response to BYE			X			
in response to BYE		X				
INVITE timers						
times INVITE cycle after B timeout						
INVITE after A timeout						
Call established call with ACK		X	X	X		
Assumption						
t-Init		X	X	X	X	X
t-Calling-initial		X	X	X	X	X
t-Calling-Wait		X	X	X	X	X
t-Calling-junction-2		X	X	X	X	X
t-Ready		X	X	X	X	X

Requirement Traceability of the test scripts

Test Generation



Message Sequence Chart for a test script

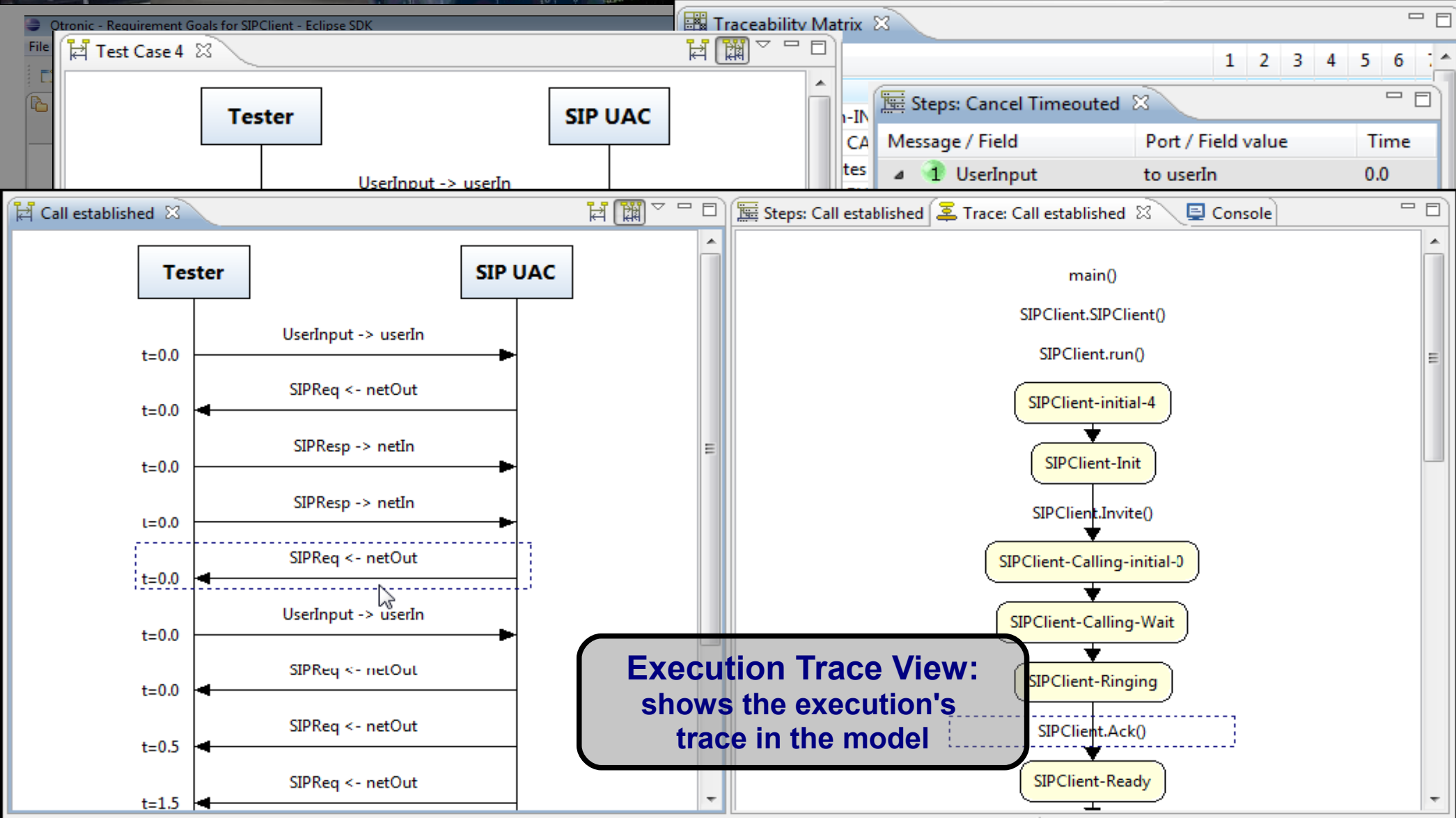
The screenshot shows a "Traceability Matrix" window with a tab titled "Steps: Cancel Timeouted". The table below represents the data shown in the screenshot:

Message / Field	Port / Field value	Time
1 UserInput	to userIn	0.0
input1	invite	
input2	sip:127.0.0.1:5061	
2 SIPReq	from netOut	0.0
op	INVITE	
param	sip:127.0.0.1:5061	
3 SIPResp	to netIn	0.0
status	180	
cseq		
4 UserInput	to userIn	0.0
input1	cancel	
input2		
5 SIPReq	from netOut	0.0
op	CANCEL	
param	sip:127.0.0.1:5061	
6 SIPReq	from netOut	0.5
7 SIPReq	from netOut	1.5
8 SIPReq	from netOut	3.5
9 SIPReq	from netOut	7.5
op	CANCEL	
param	sip:127.0.0.1:5061	
10 TimeOutIndicator	from userOut	8.0

Test Step View with detailed information about contents

List of test scripts

Test Generation



Test Generation

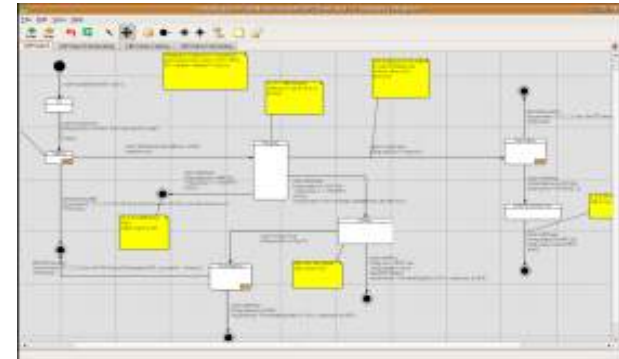
Process:

Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

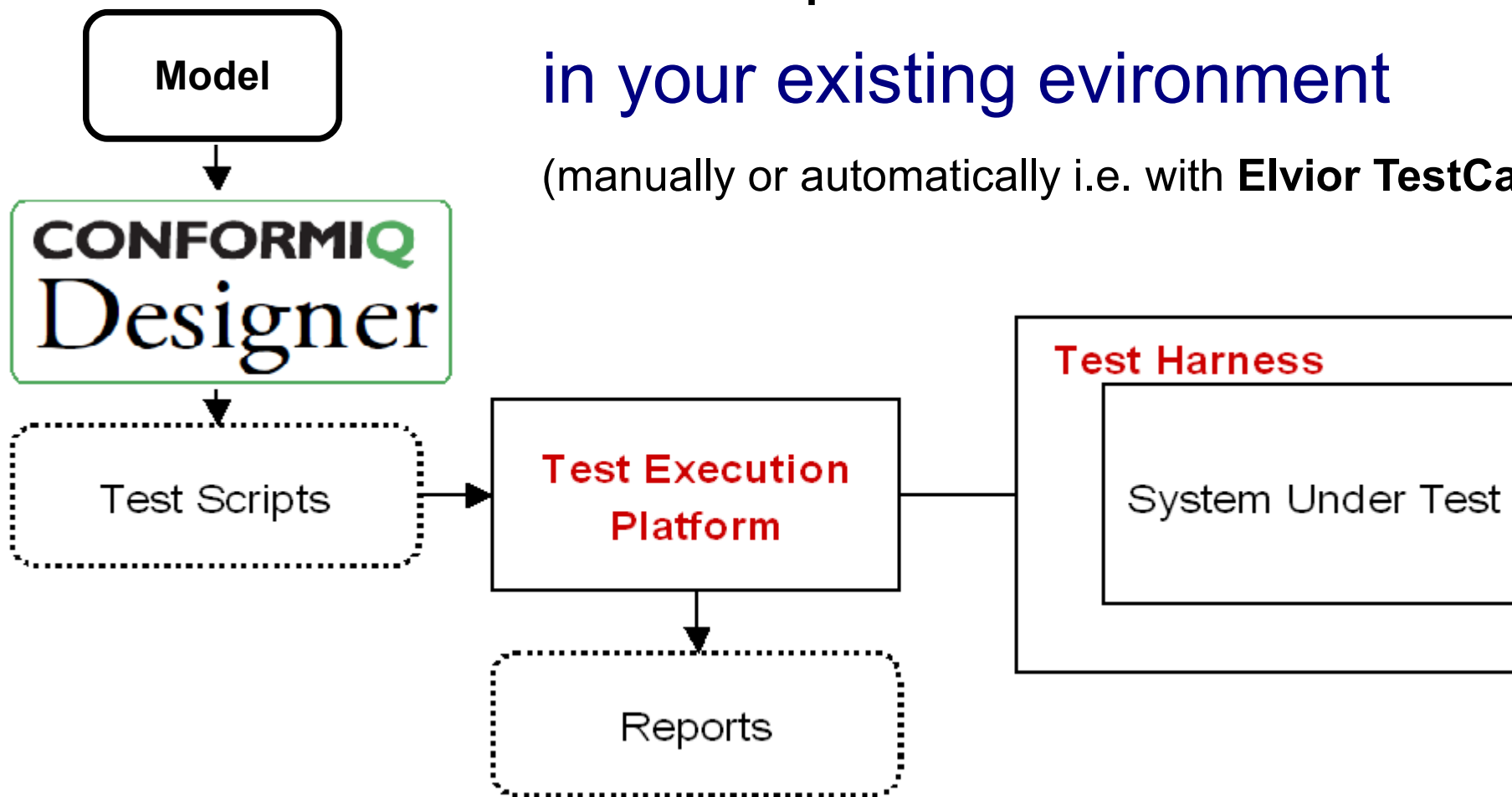


Test execution in your environment

Test Generation and Execution

Test scripts execution in your existing environment

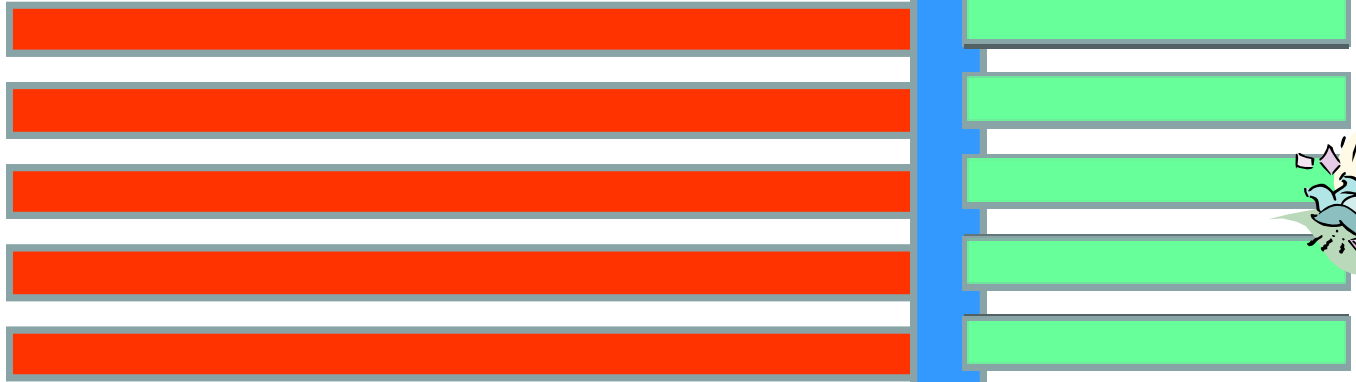
(manually or automatically i.e. with **Elvior TestCast**)



Manual vs. Automatic

Test Plan

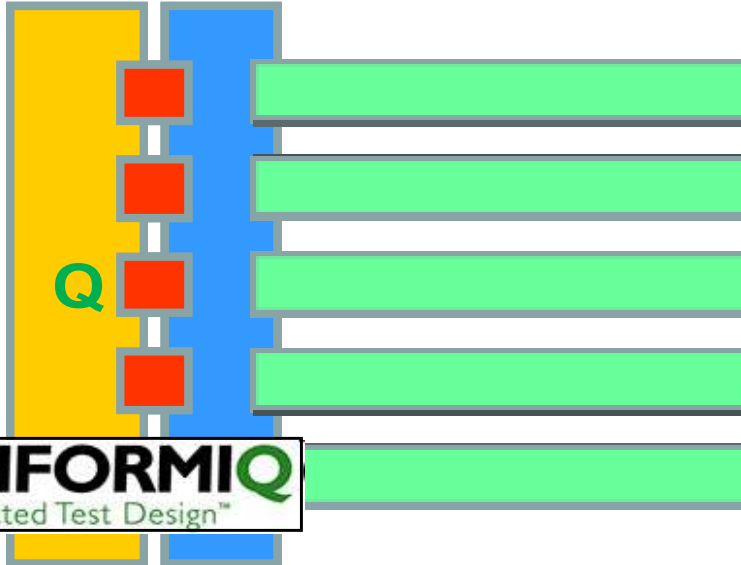
Manual test script



Script execution

System Model

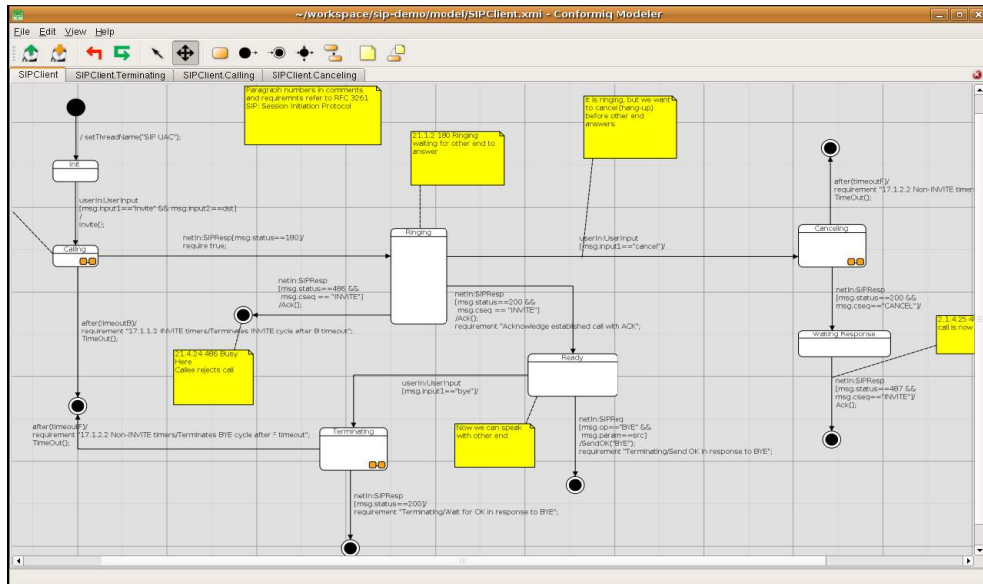
Automatic test generation



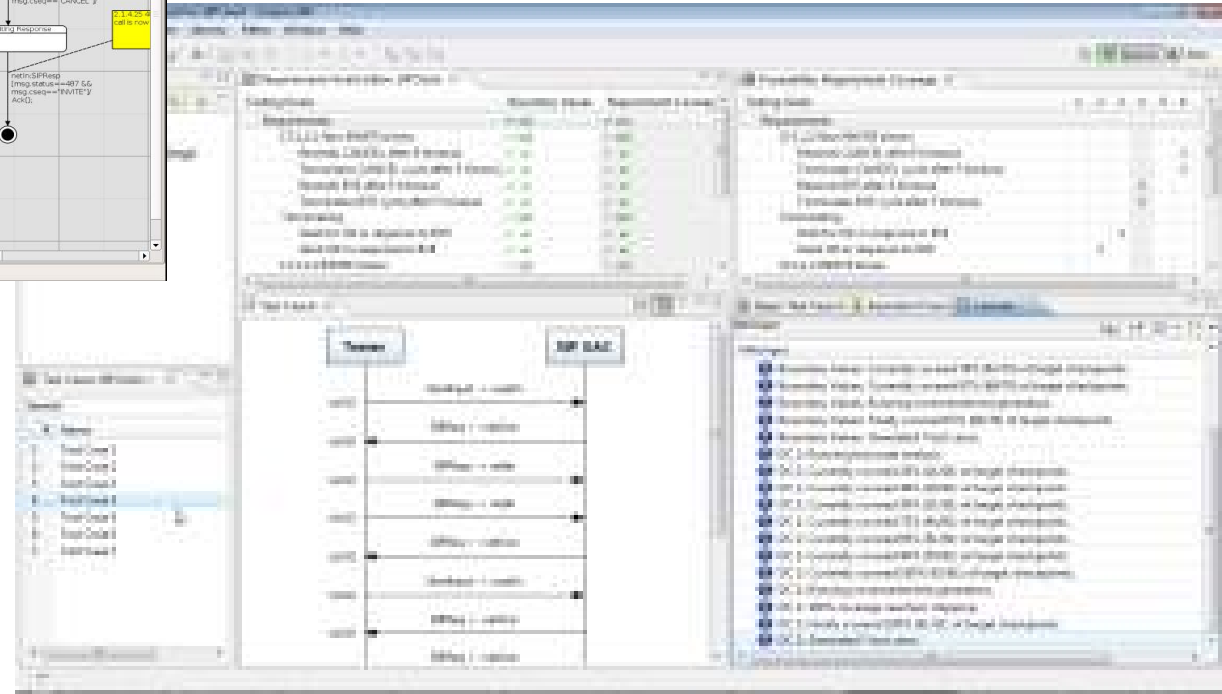
Script execution

Conformiq Tool Suite

Model



Test scripts generation

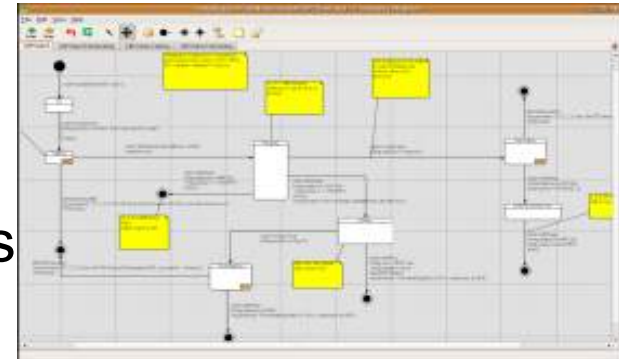


Conformiq Tool Suite: Benefits

Benefits:

Reduces maintenance of test cases

→ instead of maintaining huge amounts of test scripts needs to be updated to new/changing requirements



Creation of the model helps to validate specifications

Better requirements traceability

Increase of test coverage

More errors found compared to manual testing

Avoid problems of manual test generation (time, errors)

Time for testing updates usually very short



Conformiq Tool Suite

What can we do for you?

On-site presentation

Tool Evaluation

Pilot Project



Conformiq Tool Suite



Further information:

www.verifysoft.com