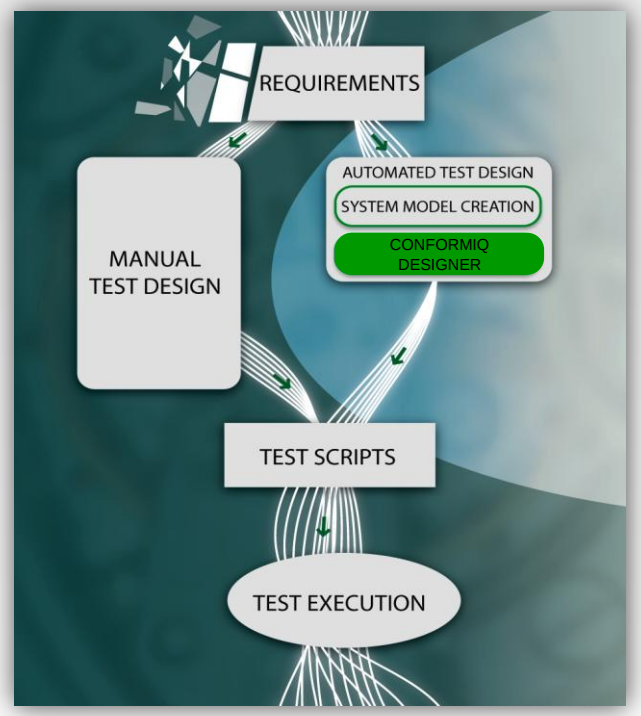
CONFORMIQ

Automated Test Design™

Automation of Test Design with Model-Based Testing

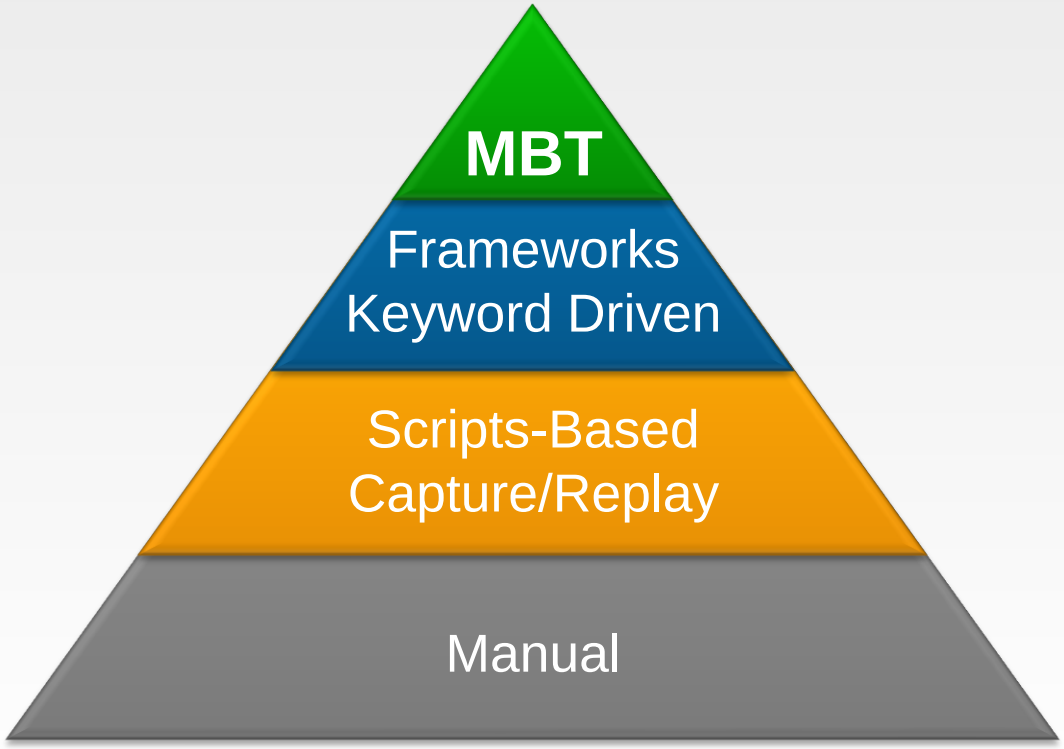
(A Statechart-based Approach)

MBT UC 2011, Berlin, Germany



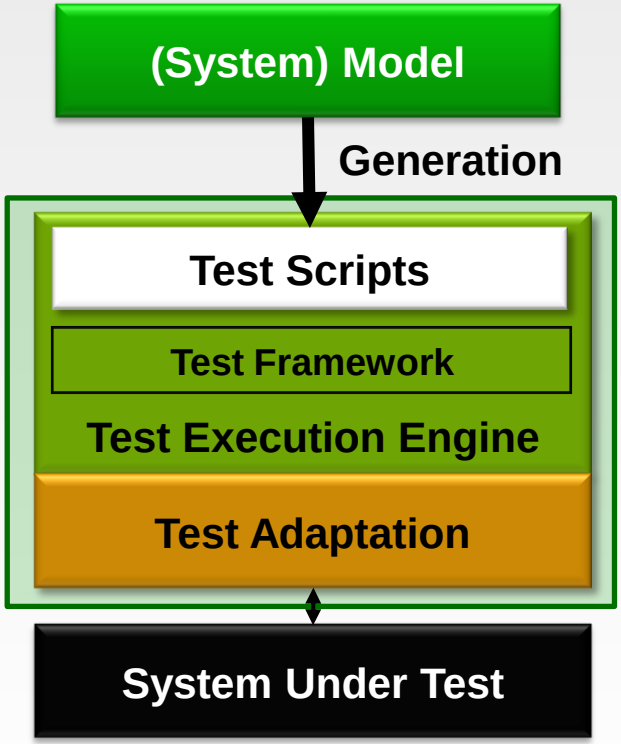
INTRODUCTION TO AUTOMATED TEST DESIGN WITH MBT

Evolution of Software Testing



Model-Based Testing

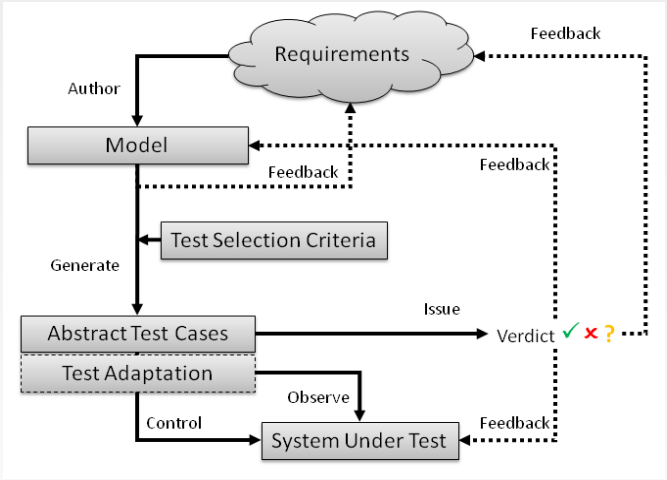
- Umbrella term for any approach that uses models for testing
- One of them is to use MBT for automating *test design*
  - Models reflect externally observable behavior of the system to be tested
  - MBT *complements* test execution
- First industrial standards (ES) being developed at ETSI
  - Multiple tools available in the market
  - Involves stakeholders from different application domains



Outline of ETSI Standard 202 951

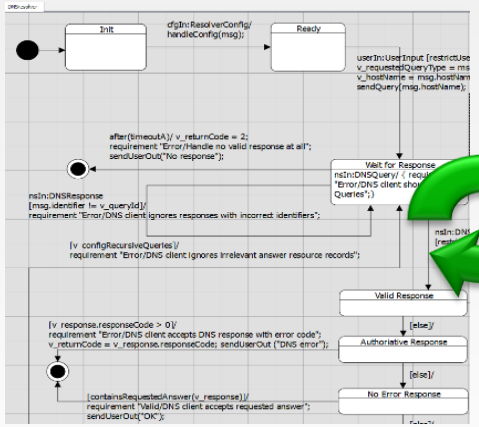
(Requirements for Modelling Notations)

- Model-Based Test Development Overview
- General Modeling Notation Requirements
  - Modularization, Algorithms, Documentation
- Modeling System Interface
  - Actions, Operations, Ports, Configurations
- Modeling System Behavior
  - System State, System State Transition, Non-Determinism
- Examples of Modeling Notation Styles (Annex)
  - Rule-Based/EFSM, Statechart, Process-Oriented

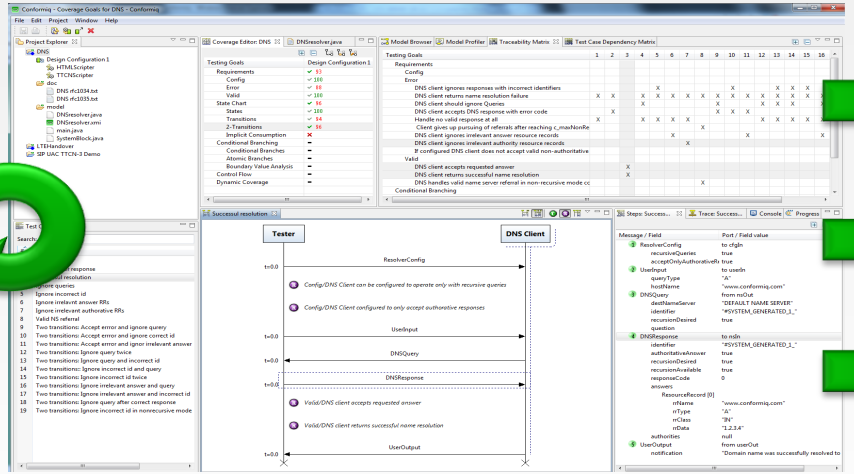


Automated Test Design Workflow

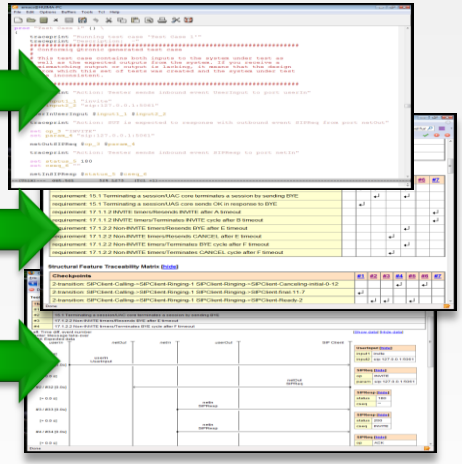
Develop (System) Model



Direct & Review Test Design



Generate Test Scripts & Documentation



Conformiq - Coverage Goals for DNS - Conformiq

File Edit Project Window Help

Project Explorer

- DNS
  - Design Configuration 1
    - HTMLScripter
    - TTCNScripter
  - doc
    - DNS rfc1034.txt
    - DNS rfc1035.txt
  - model
    - DNSresolver.java
    - DNSresolver.xmi
    - main.java
    - SystemBlock.java
- LTEHandover
- SIP UAC TTCN-3 Demo

Coverage Editor: DNS | DNSresolver.java

Testing Goals	Design Configuration 1
Requirements	✔ 93
Config	✔ 100
Error	✔ 88
Valid	✔ 100
State Chart	✔ 96
States	✔ 100
Transitions	✔ 94
2-Transitions	✔ 96
Implicit Consumption	✘
Conditional Branching	–
Conditional Branches	–
Atomic Branches	–
Boundary Value Analysis	–
Control Flow	–
Dynamic Coverage	–

Model Browser | Model Profiler | Traceability Matrix | Test Case Dependency Matrix

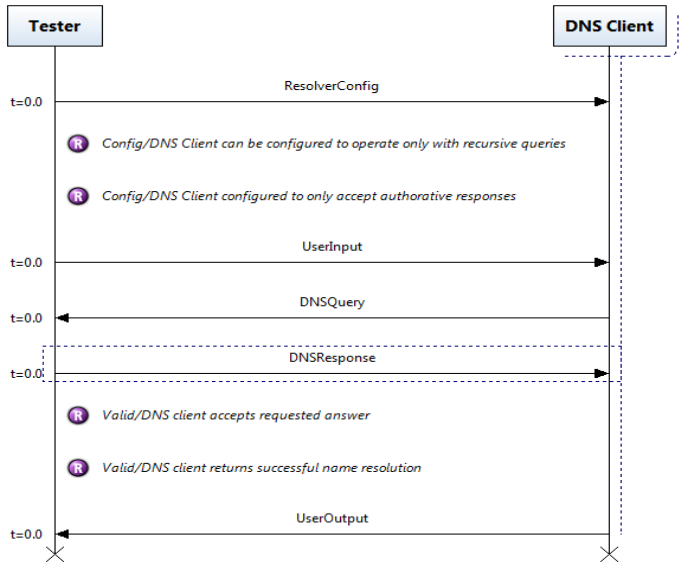
Testing Goals	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Requirements																
Config																
Error																
DNS client ignores responses with incorrect identifiers					X					X			X	X	X	
DNS client returns name resolution failure	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X
DNS client should ignore Queries				X					X			X	X	X		X
DNS client accepts DNS response with error code		X							X	X	X					
Handle no valid response at all	X			X	X	X	X					X	X	X	X	X
Client gives up pursuing of referrals after reaching c_maxNonRe								X								
DNS client ignores irrelevant answer resource records						X					X					X
DNS client ignores irrelevant authority resource records							X									
If configured DNS client does not accept valid non-authoritative																
Valid																
DNS client accepts requested answer			X													
DNS client returns successful name resolution			X													
DNS handles valid name server referral in non-recursive mode cc								X								
Conditional Branching																

Test Cases: DNS

Search:

#	Name
1	No response
2	Accept error response
3	Successul resolution
4	Ignore queries
5	Ignore incorrect id
6	Ignore irrelavnt answer RRs
7	Ignore irrelevant authorative RRs
8	Valid NS referral
9	Two transitions: Accept errror and ignore qurery
10	Two transitions: Accept errror and ignore correct id
11	Two transitions: Accept errror and ignor irrelevant answer
12	Two transitions: Ignore query twice
13	Two transitions: Ignore query and incorrect id
14	Two transitions:: Ignore incorrect id and query
15	Two transitions: Ignore incorrect id twice
16	Two transitions: Ignore irrelevant answer and query
17	Two transitions: Ignore irrelevant answer and incorrect id
18	Two transitions: Ignore query after correct response
19	Two transitions: Ignore incorrect id in nonrecursive mode

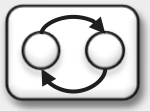
Successul resolution



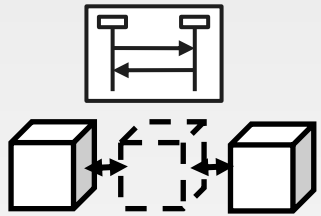
Steps: Success... | Trace: Success... | Console | Progress

Message / Field	Port / Field value
1 ResolverConfig	to cfgIn
recursiveQueries	true
acceptOnlyAuthoritativeR	true
2 UserInput	to userIn
queryType	"A"
hostName	"www.conformiq.com"
3 DNSQuery	from nsOut
destNameServer	"DEFAULT NAME SERVER"
identifier	"#SYSTEM_GENERATED_1_"
recursionDesired	true
question	
4 DNSResponse	to nsIn
identifier	"#SYSTEM_GENERATED_1_"
authoritativeAnswer	true
recursionDesired	true
recursionAvailable	true
responseCode	0
answers	
ResourceRecord [0]	
rrName	"www.conformiq.com"
rrType	"A"
rrClass	"IN"
rrData	"1.2.3.4"
authorities	null
5 UserOutput	from userOut
notification	"Domain name was successfully resolved to

MBT - A New Way of Testing



Functionality vs Path



SUT vs Environment

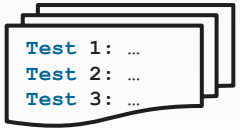
95%

Coverage vs Number of Tests

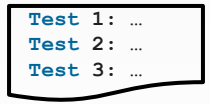
42

Abstraction vs Detail

Optimization vs Expert Opinion



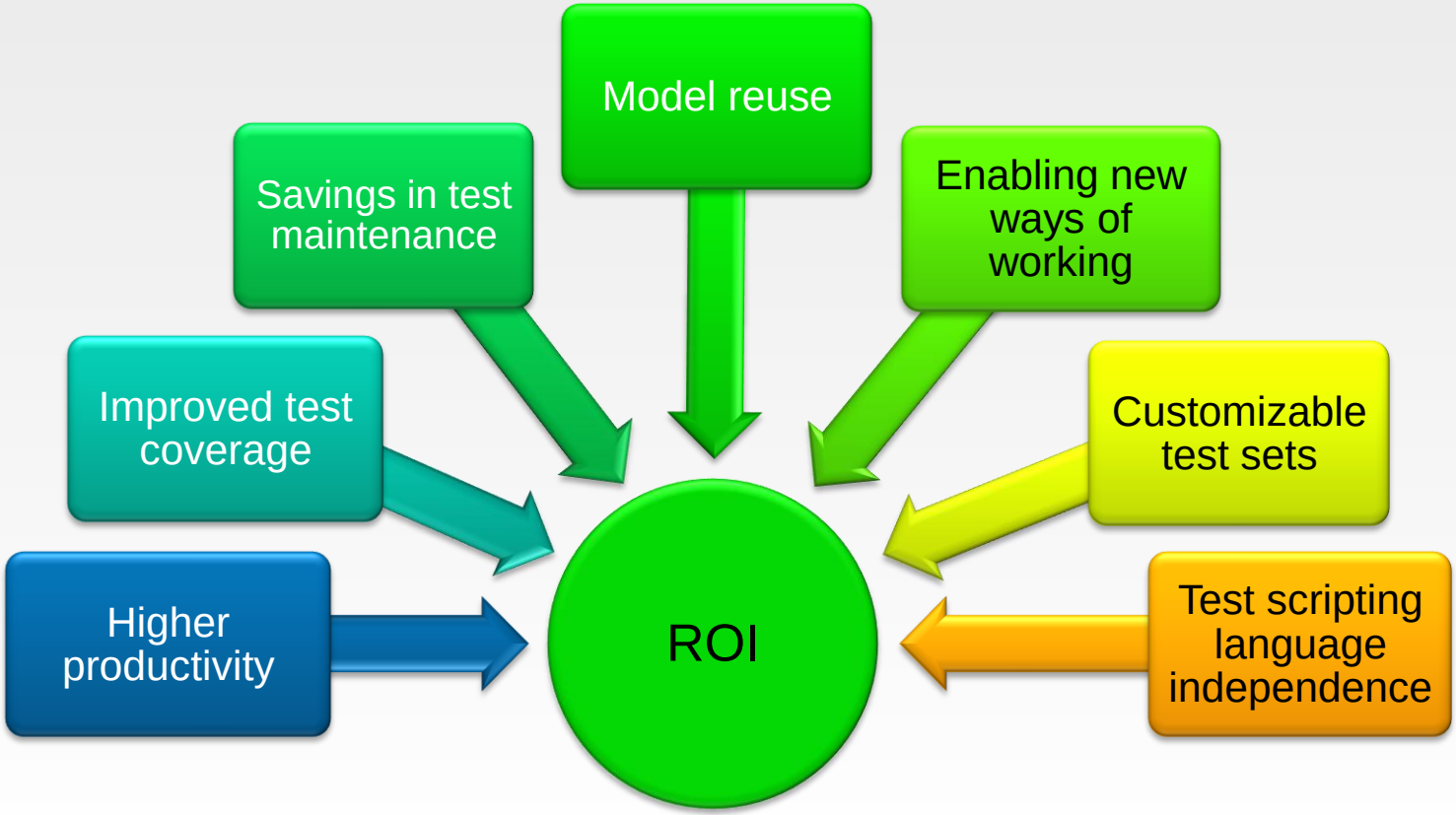
Flexible vs Fixed Test Sets



What Remains the Same?

- The starting point will be an informal specification
  - Customer: ”I want a  ”
- Tests have to be validated
  - Any testing artefact remains as good as the *agreed* understanding of the specification
- Still infinitely many tests have to be executed to proove that a SUT faithfully follows a specification
  - But a *limited* amount of time is available for test execution

The Business Case for Automated Test Design



Automated Test Design Benefits

- Less **customer found defects**
- More available **resources**
- Shorter testing **turnaround time**
- Reliable reports and documentation
- Test suites with optimal coverage

Less Time

Spend less calendar time in test process, get product out faster

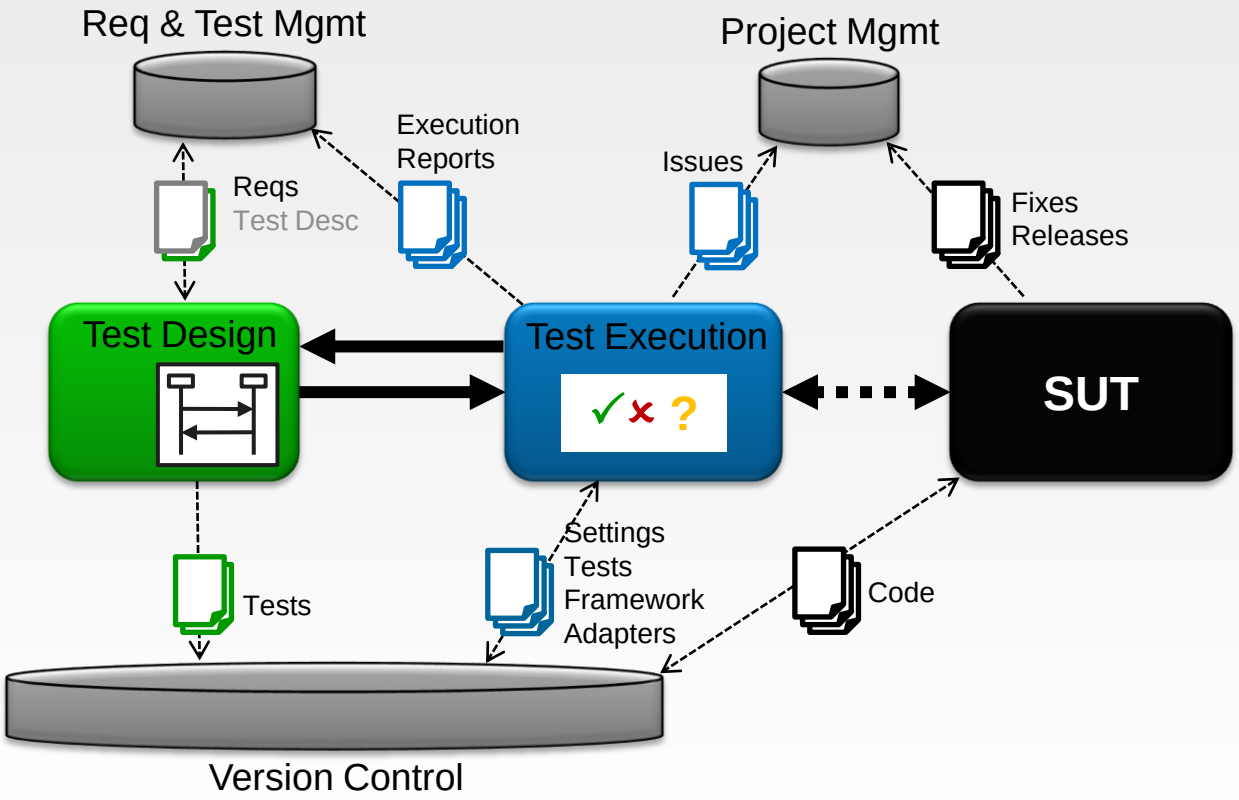
Higher Quality

More effective test sets resulting in a higher quality product

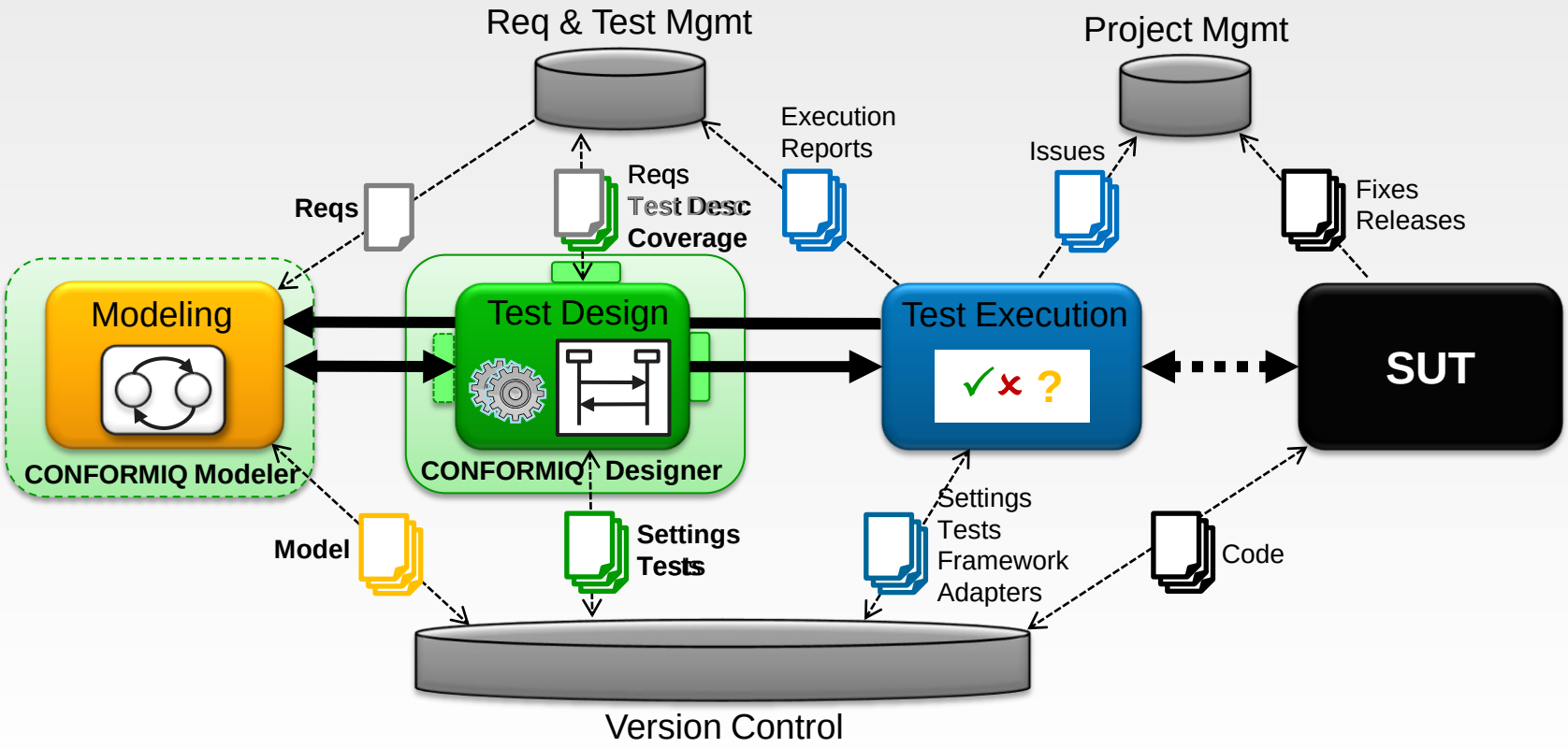
Reduced Cost

With less resource to higher quality

The BIG Picture: From Manual Test Design …



The BIG Picture: … to Automated Test Design



Introduction Summary

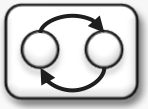
- MBT is next step in evolution of software testing
- First industrial standards attest maturity
- Use MBT allows engineers to work (usually >5x) more effectively and focus on the essense of testing
- Next to new notations & tools MBT requires a change in thinking for management and engineers
- Automatic test generation is only *one* part of MBT

MODELING FOR TESTING

About (System) Models

- Structural view: SUT interface available for testing
  - 1+ (logical) ports and parameterized input & output (actions) reflecting controllable & observable interfaces
  - At least one but possibly multiple parallel model components

- Behavioral view: The expected SUT operation
  - Should focus on one or more aspects to be tested
  - Can be defined using states, transitions, and operations on inputs, outputs and component variables
  - Possibly defined across a set of communicating components



About Modeling

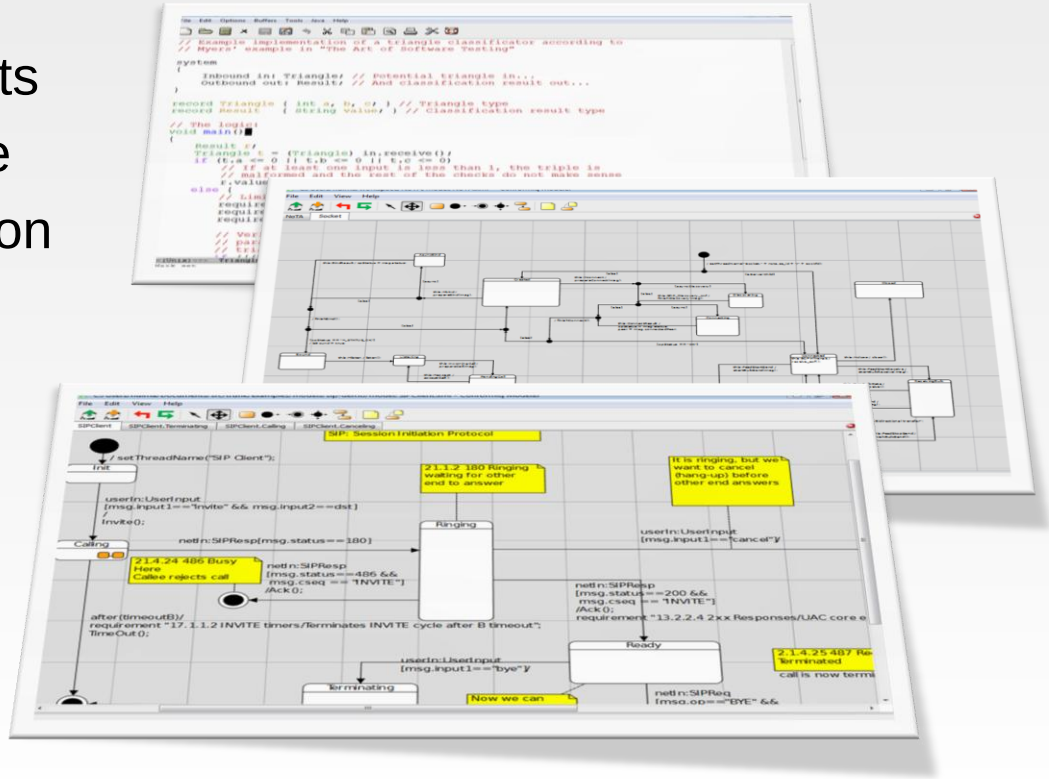
- Who should write models?
  - The people that write or script tests today!
- What is the starting point?
  - Selecting first aspect of the functionality to be tested
  - Identifying a suitable level of abstraction

Abstraction Level

- Level of detail used to describe functionality to be tested
- Selection from the specification
  - Which interfaces are available for testing selected functionality?
  - What are the relevant inputs & outputs on these interfaces?
  - Extend inputs/outputs with parameters only as you model behavior
- Alternative: Selection from existing test framework
  - Model functions/operations available in test automation (framework) as inputs/outputs in system interface specification
  - Extend inputs/outputs with parameters only as you model behavior

Conformiq's Modeling Notation (QML)

- Hierarchical UML Statecharts
- Java based action language
- System interface specification
- Rich type system
- Multi component models
- Timing constraints
- Data constraints



Our Task: Testing of a VoIP Terminal

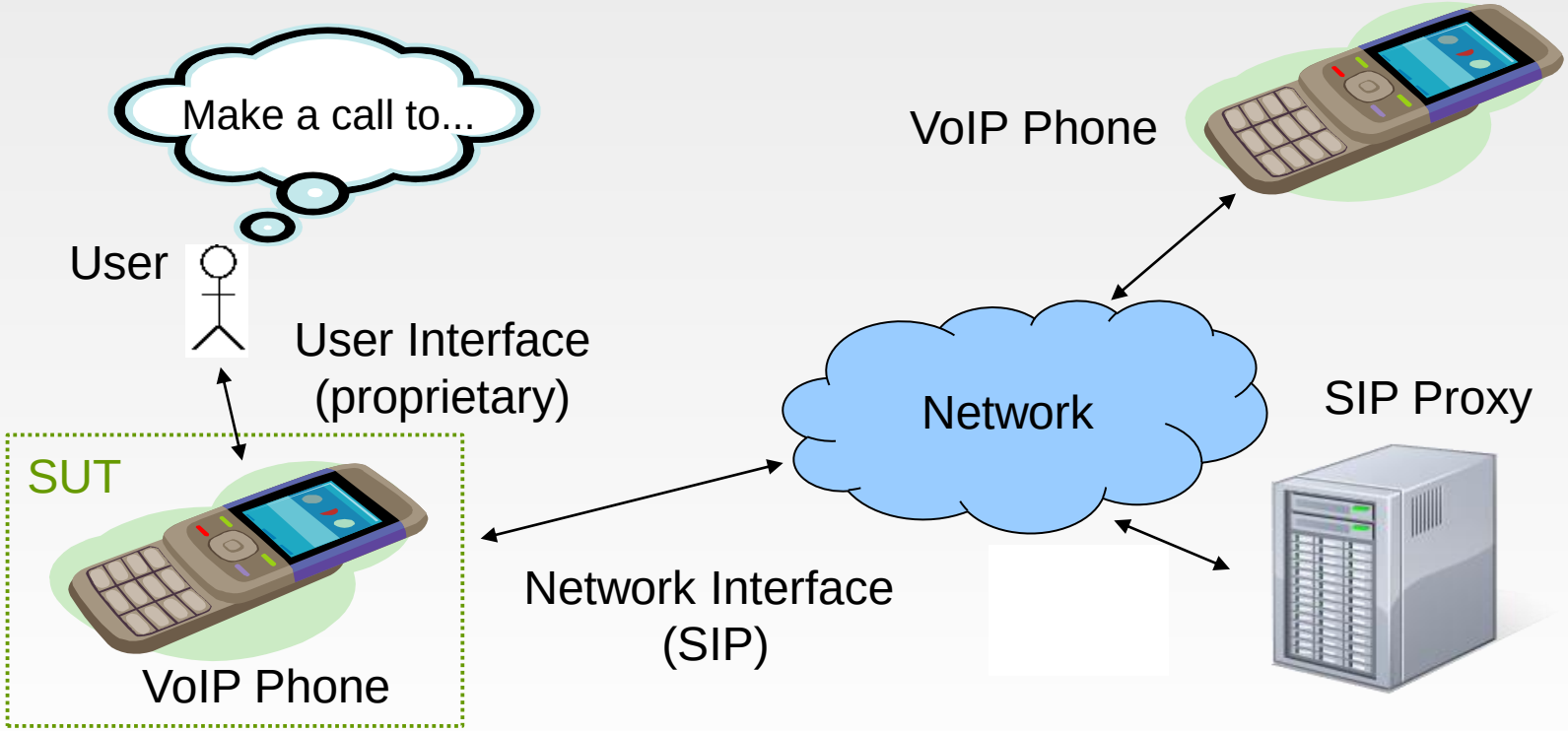
- Subject of Testing: Session Layer
- Specification: RFC 3261 “Session Initiation Protocol”
- Interfaces available: user & network
- Scope: Basic call functionality
  - Call establishment
  - Call termination (callee vs. callee initiated)
  - Call cancelation
  - Call timeouts (re-transmission & transaction)
- See also Conformiq white paper
  - “Case Study: Automated Testing of X-Lite SIP Softphone”

Requirements Extracted from RFC 3261

A SIP User Agent must:

1. Establish a session with SIP ACK (Clause 13.2.2.4)
2. Terminate a session with SIP BYE (Clause 15.1.1)
3. Confirm a SIP BYE with a SIP 200 OK (Clause 15.1.2)
4. Re-send an SIP INVITE after timeout A (Clause 17.1.1.2)
5. Terminate an SIP INVITE after timeout B (Clause 17.1.2.2)
6. Terminate a SIP BYE request after timeout F (Clause 17.1.2.2)
7. Terminate a SIP CANCEL request after timeout F (Clause 17.1.2.2)

Identification of System Boundary

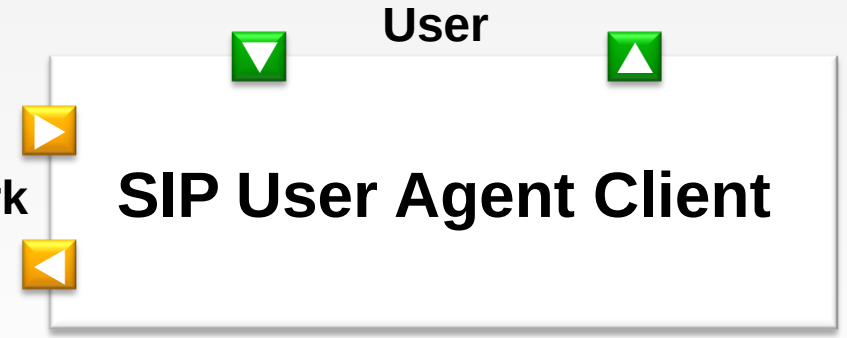


Identification of Logical Interface

UserInput:
”call”, ”hang up”, ”cancel”

UserOutput:
”ringing”, ”call ended” ”call established”, ”timeout”

SipRequest: ”BYE”
SipResponse: “180 Ringing”, ”200 OK”, ”486 Busy Here”, ”487 Request Terminated”



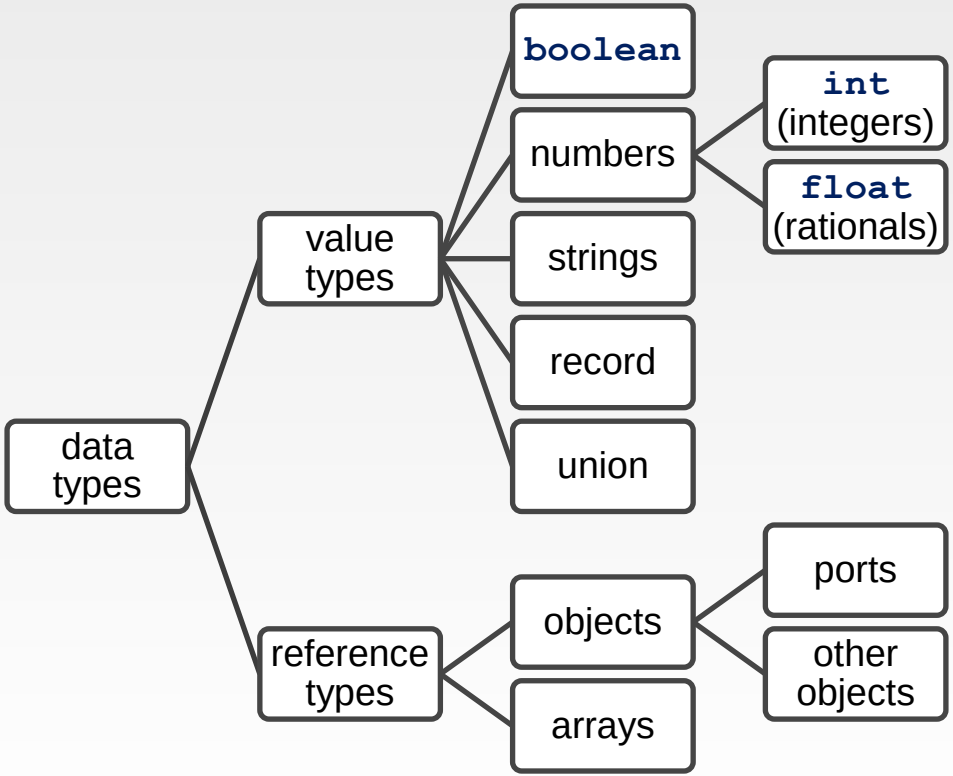
SipRequest: “ACK”, “BYE”, ”CANCEL”, ”INVITE”
SipResponse: ”200 OK”

QML Representation of System Interface

```
system {                          // system block alias system boundary definition
    Inbound  userIn  : UserInput;  // port instance definition with valid message list
    Outbound userOut : UserOutput;
    Inbound  netIn   : SipResponse, SipRequest;
    Outbound netOut  : SipRequest, SipResponse;
}

record SipRequest {          // message type definition (ordered sequence)
    RequestLine startLine; // field of other structured type (definition elsewhere)
    String callId;
    String contact;
    CSeq cSeq;
    From from;
    int maxForwards;
    To to;
    Via[] vias;              // unbounded list of via headers
    String msgbody;
}
```

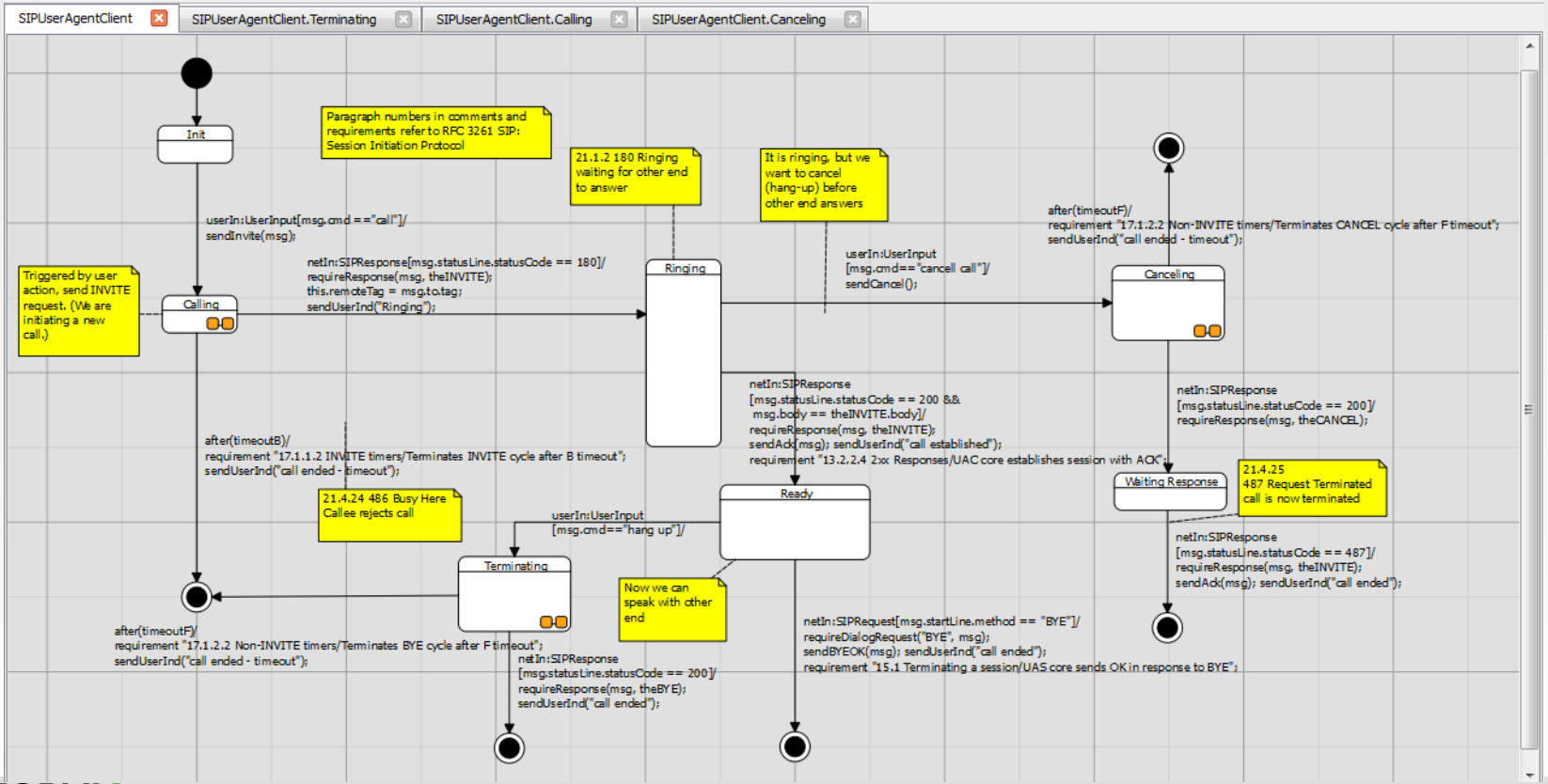
Available QML Data Types



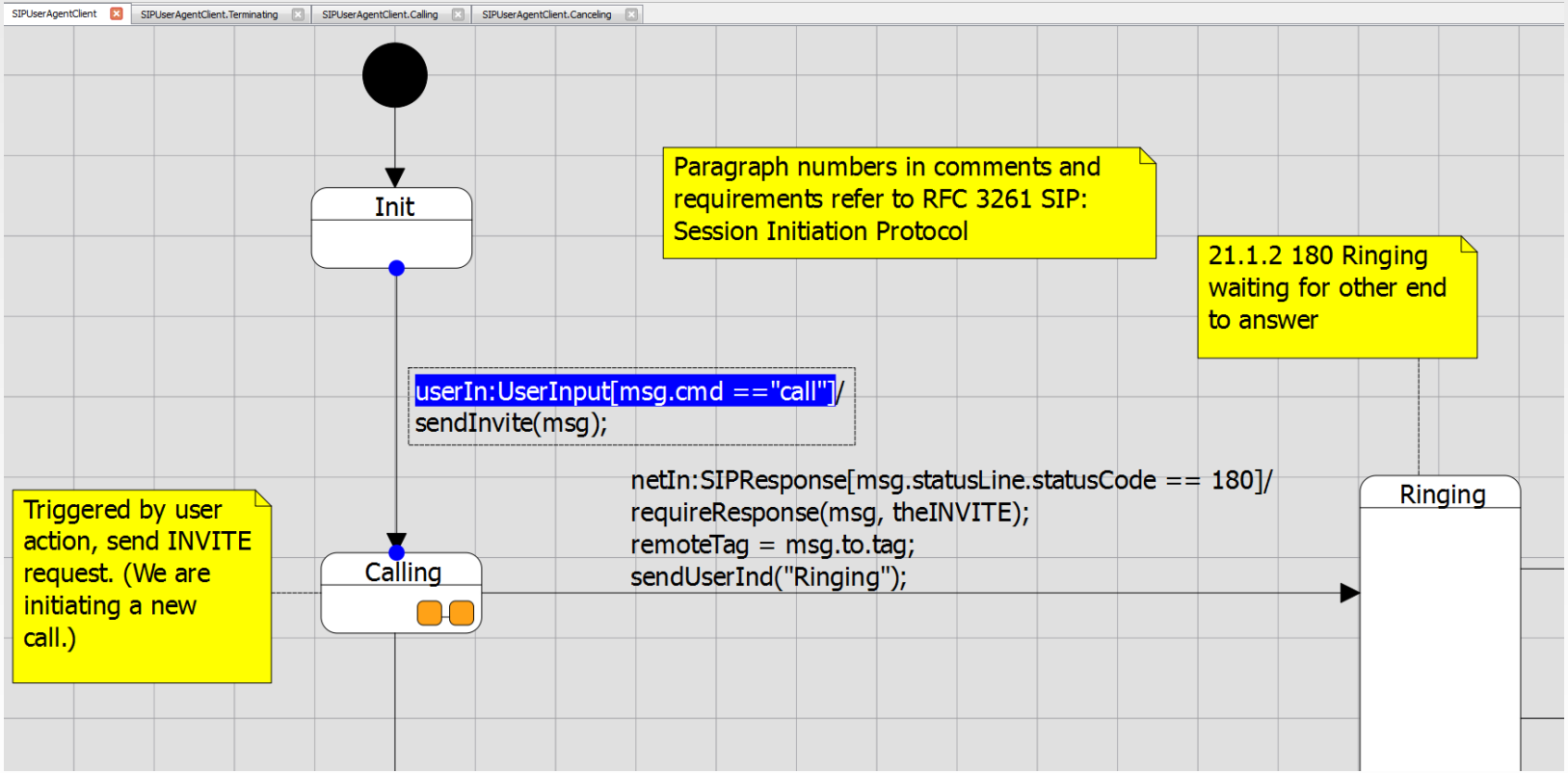
Modeling Behavior: State Machines

- One way to represent the behavior of a system
- At any moment in time the system is ”in” a state
- Transitions define state changes
  - Can be triggered by an external input or timeout and/or fulfilment of guard conditions
  - When triggered can perform one or more actions such as sending external output(s), marking the coverage of a requirement, or operating on received or component data
  - Actions can also be implemented using methods

Example Statechart: Sip User Agent Client



Example Transition: Call initiation

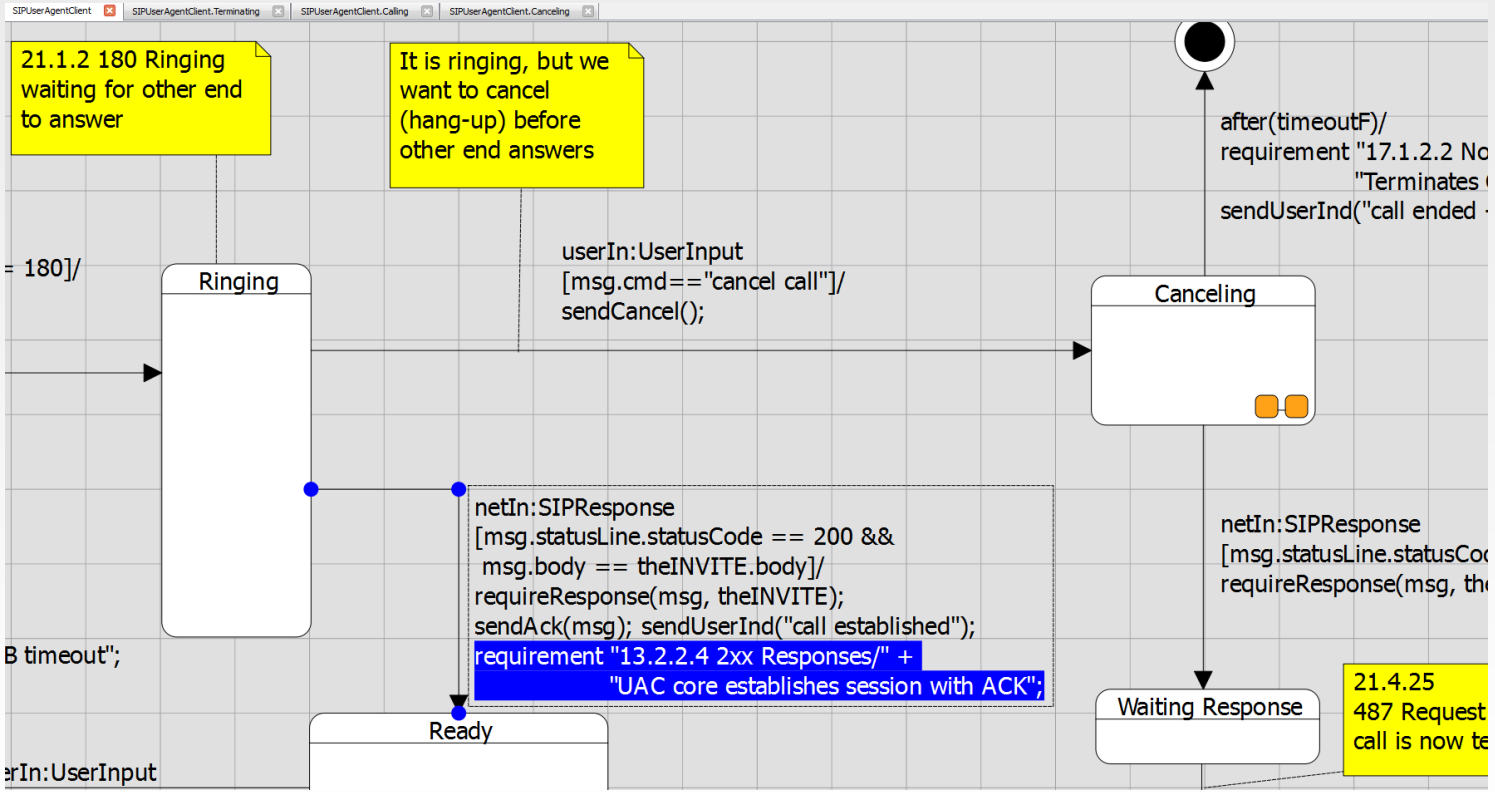


Example QML Method

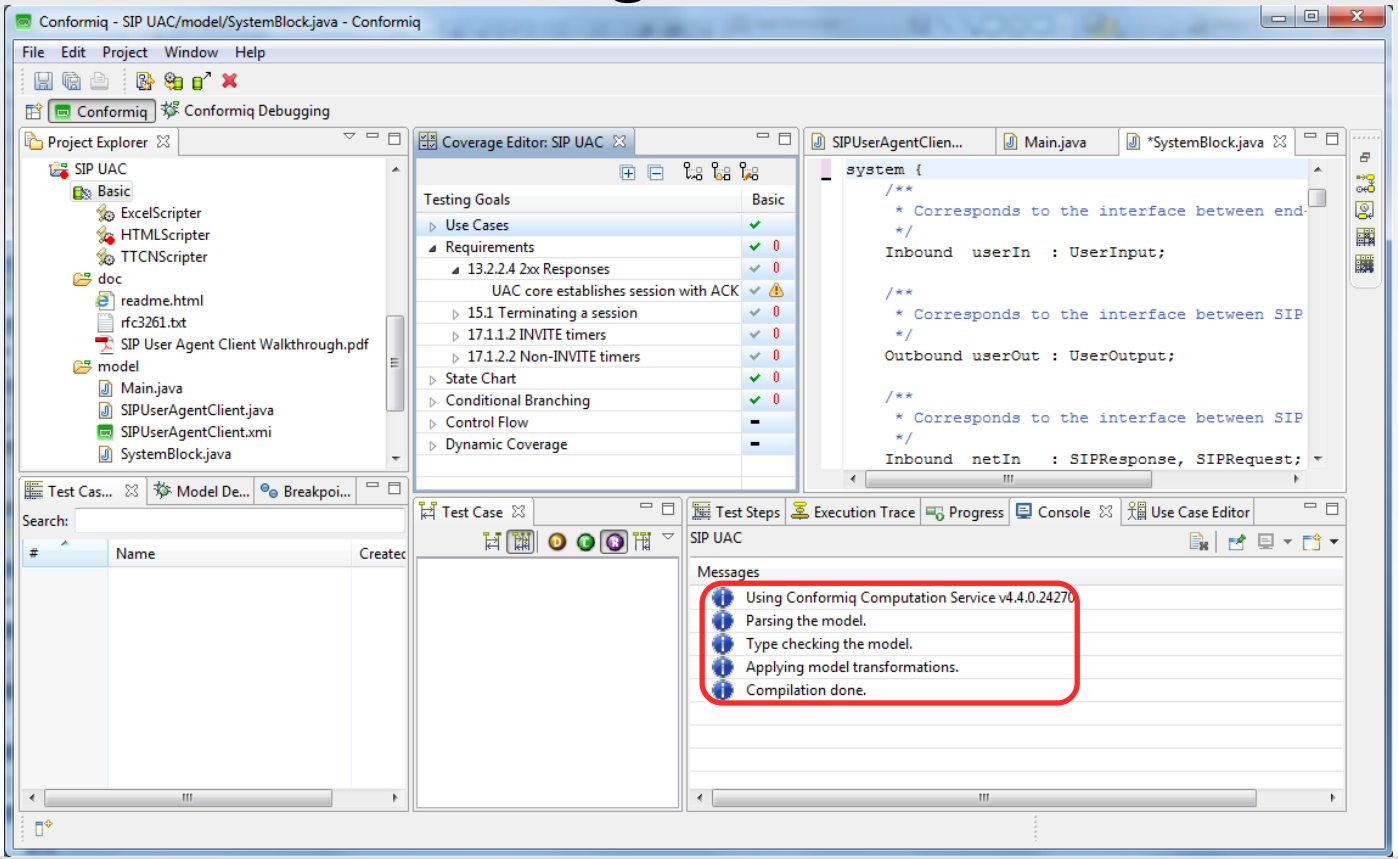
- Implementation of action to send a SIP INVITE request via the network interface:

```
void sendInvite() {
  // construct first SIP request value by calling 'getRequestBase' method
  SipRequest theINVITE = getRequestBase("INVITE", newCallId());
  // store 'from' header tag in component variable for later checks
  localTag = theINVITE.from.tag;
  // overwrite contact header and message body default values
  theINVITE.contact = "sip:" + getCallerSipUri();
  theINVITE.msgbody = newMsgBody();
  netOut.send(theINVITE);
}
```

A Requirement in the Model



Loading the Model



Modeling Summary

- Functional models have structural and behavioral aspects
- System interface is defined via ports and message types
- Handling of inputs and computation of outputs is defined in triggers & actions on Statechart transitions
- Start modeling with a high level of abstraction, e.g., no message parameters, then refine model structure and behavior iteratively
- Model only information required by aspect to be tested

TEST GENERATION

Expectations from Test Generation

- Complete message flow, data, timing, and oracle
- Test generation (regardless of complexity) within minutes
- At least one test per requirement
- Exercise all (possible) SUT source code
- Shortest possible test set execution time
- Do not test ”the same thing” multiple times
- Automatic test maintainence

Test Generation at a Second Glance

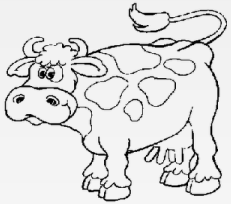
- Big test sets require a lot of validation
  - There needs to be a way to limit test set size
- Where is the test that covers criteria XYZ?
  - Generation results must be easily comprehensible
- Even at 100% coverage my test is not in the test set
  - Optimal test sets do not always fulfill all expectations
  - Users must be able to affect/guide test generation

But Most Important at the End is …

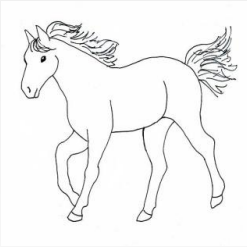
Needed
[Customer]

Specification
[Architect]

Implemented
[Developer]

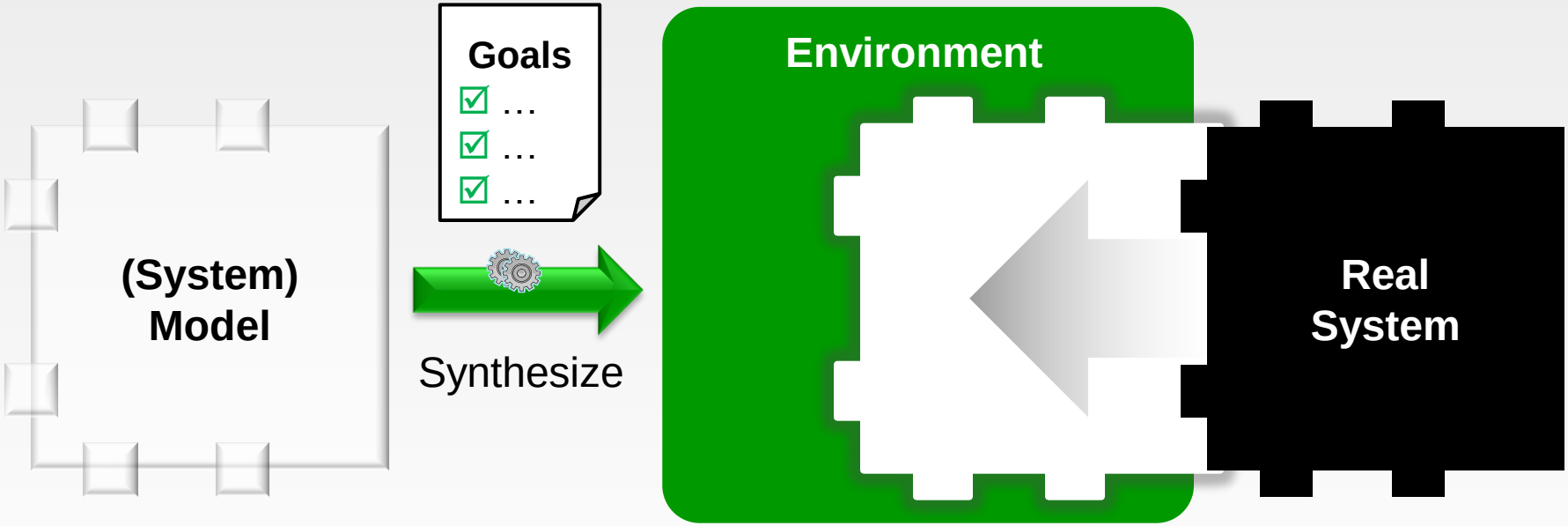


Modeled/Tested
[Tester]



- … good support for sharing and communicating test generation results (and models) with testers *and others* !

From Modeling to Black-Box Testing



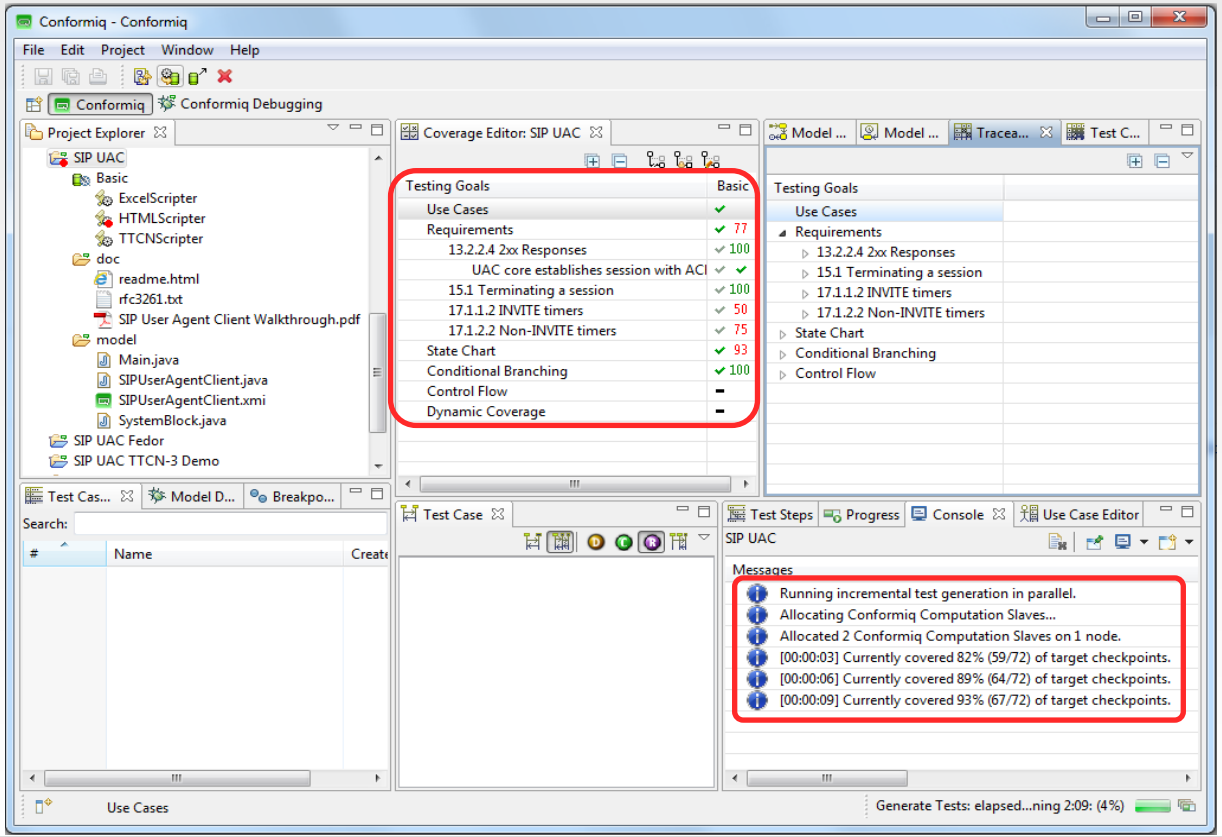
Example Model Coverage Criteria

Name	Explanation	Typically Used For
Requirements Coverage	Cover “requirement” statements	Basic test generation
Use Case Coverage	Cover independently specified use cases	
State Coverage	Cover states of every state chart	
Transition Coverage	Cover transitions of every state chart	
Condition Coverage	Cover “true” *and* “false” branches of conditional constructs	
Parallel Transition Coverage	Cover all interleavings of independent transitions in multi component models	Extended test generation
Data Coverage	Cover all pairs or all combinations of data values	
2-Transition Coverage	Cover combinations of entry/exit transitions of all states	
Atomic Condition Coverage	Cover all “true” *and* “false” evaluations of Boolean expressions	
Boundary Value Analysis	Cover integer boundary conditions	
All Paths - States	Cover all possible states sequences	Exhaustive test generation
All Paths - Transitions	Cover all possible transition sequences	

How does Test Generation work?

- Different technologies available to generate tests
  - Graph traversal vs. symbolic model checking plus constraint solving
- Different approaches how to expose users to results
- What should be expected from a MBT tool today
  - Generation of complete message flow, test data, timing, and oracle
  - Support for specification of coverage criteria to control test suite size
  - Options for generating different types of test sets
  - Incremental & deterministic test generation
  - Automatic marking/elimination of invalid tests in re-generations

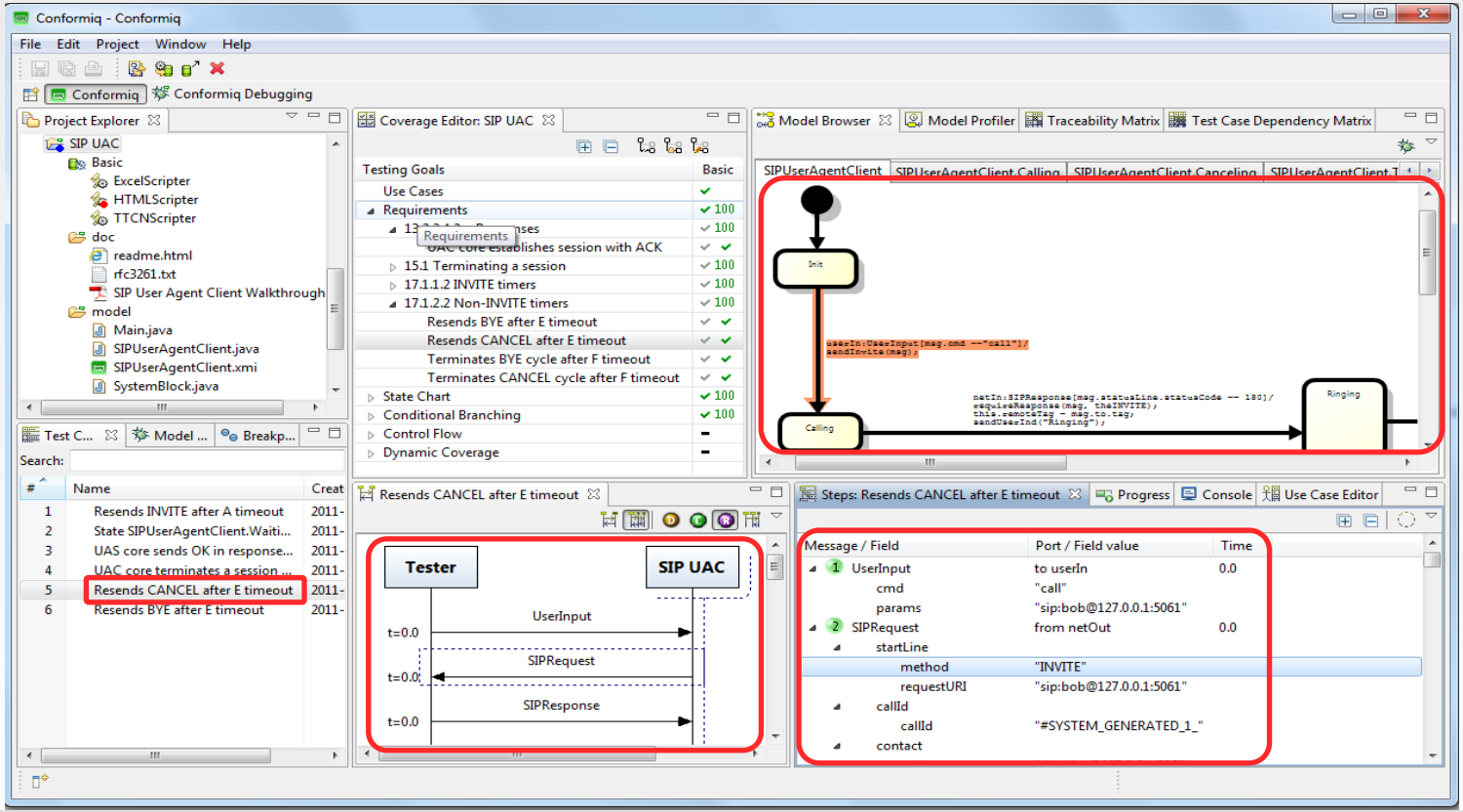
Generating Tests from Models



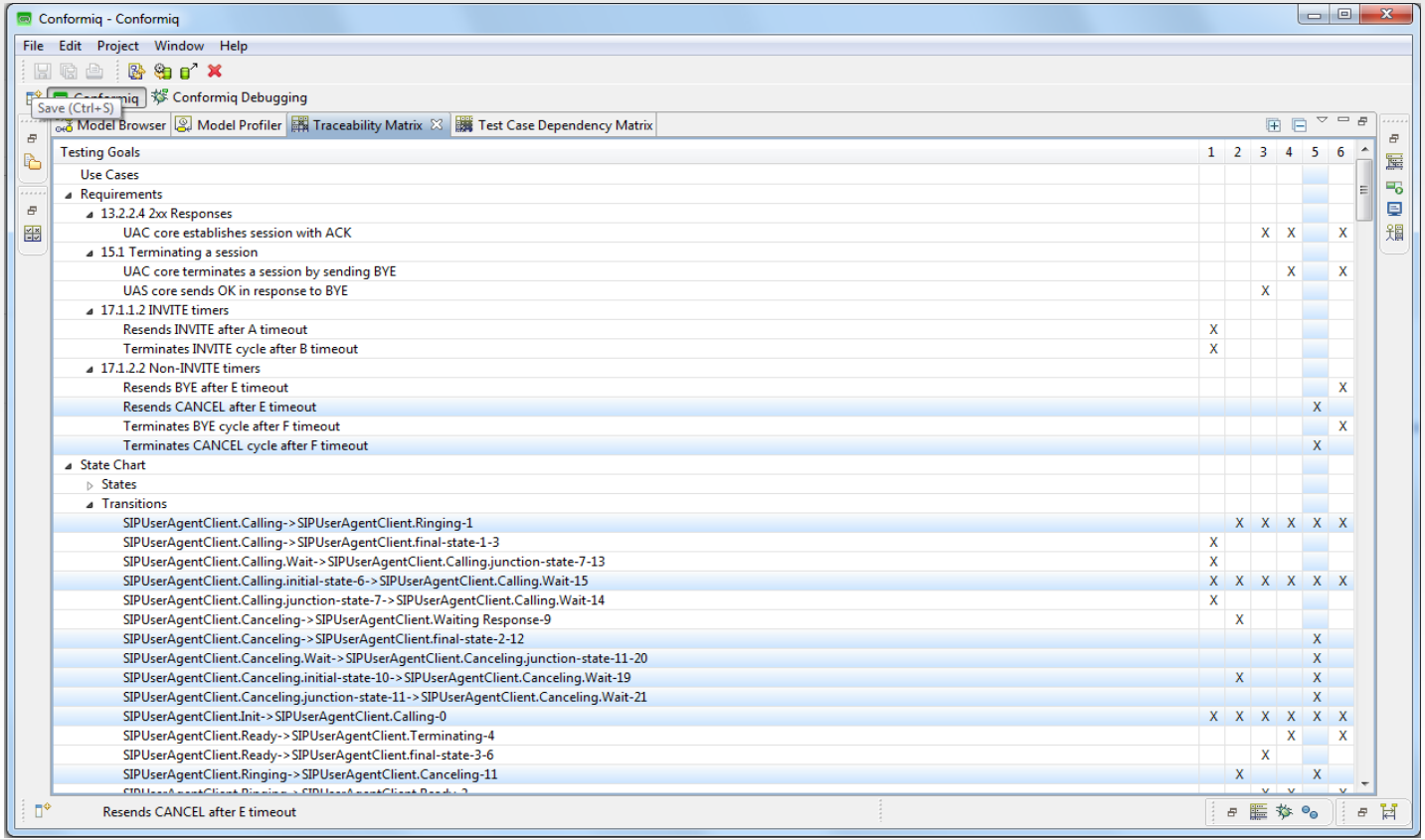
Making Sense of Generation Results

- Integrated graphical IDEs with interconnected views
- Ability to influence test case naming – a first glance should give a good idea
- Traceability matrix linking coverage criteria to tests
- Specific test preview of message flow, test data, and timing
- Highlighting of test and criteria in the (system) model
- Enable analysis of tests and model defects via debugger

Example Test Review



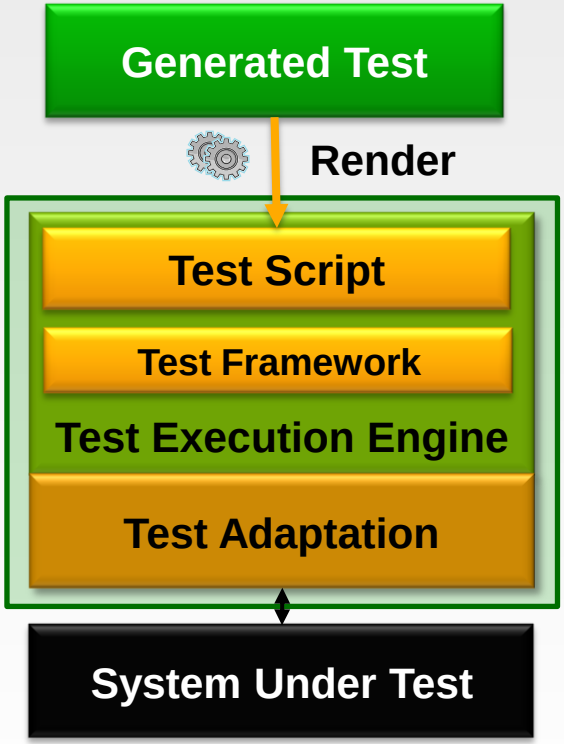
Traceability Matrix



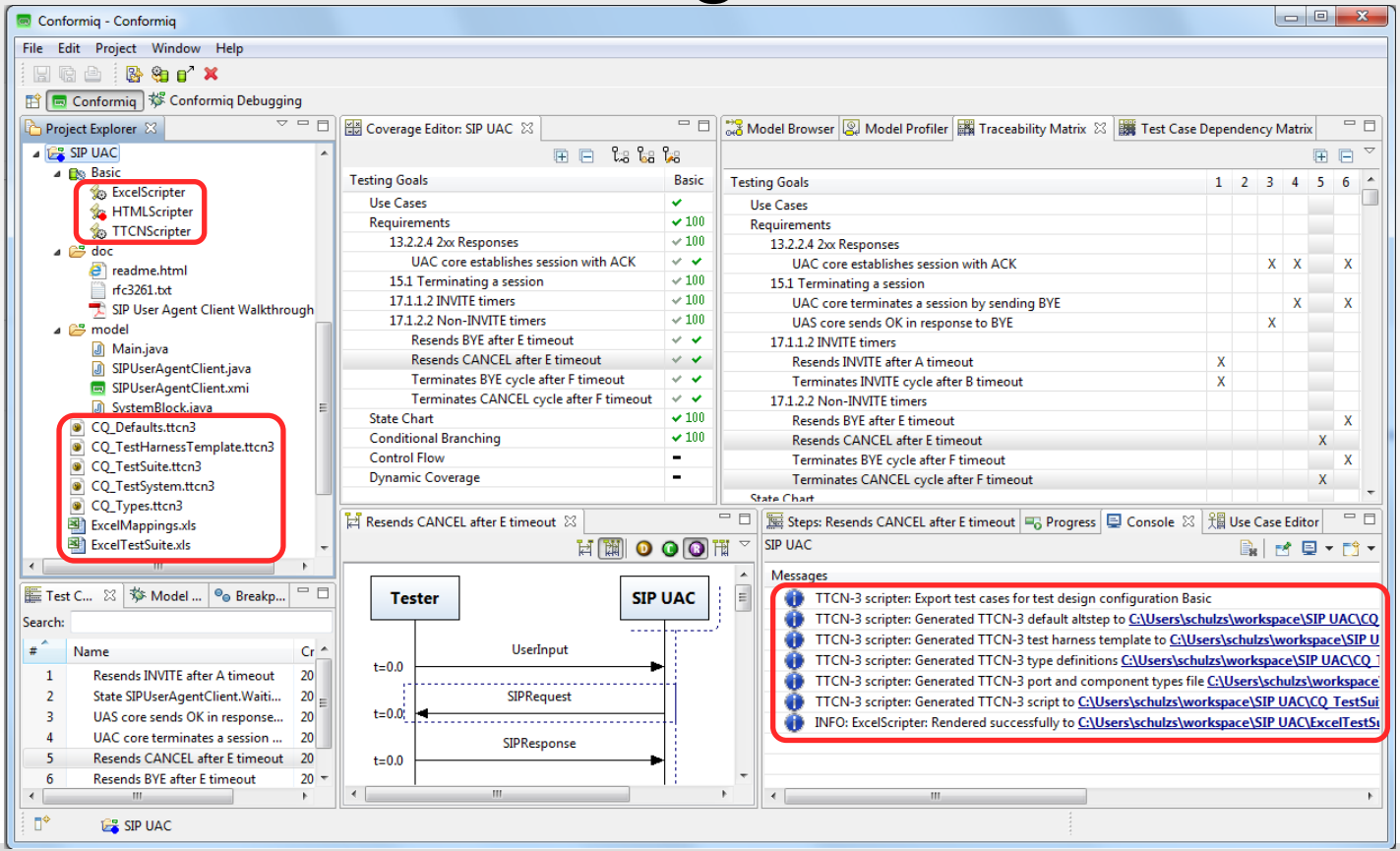
Testing Goals	1	2	3	4	5	6
Use Cases						
Requirements						
13.2.2.4 2xx Responses						
UAC core establishes session with ACK			X	X		X
15.1 Terminating a session						
UAC core terminates a session by sending BYE				X		X
UAS core sends OK in response to BYE			X			
17.1.1.2 INVITE timers						
Resends INVITE after A timeout	X					
Terminates INVITE cycle after B timeout	X					
17.1.2.2 Non-INVITE timers						
Resends BYE after E timeout						X
Resends CANCEL after E timeout					X	
Terminates BYE cycle after F timeout						X
Terminates CANCEL cycle after F timeout					X	
State Chart						
States						
Transitions						
SIPUserAgentClient.Calling->SIPUserAgentClient.Ringing-1		X	X	X	X	X
SIPUserAgentClient.Calling->SIPUserAgentClient.final-state-1-3	X					
SIPUserAgentClient.Calling.Wait->SIPUserAgentClient.Calling.junction-state-7-13	X					
SIPUserAgentClient.Calling.initial-state-6->SIPUserAgentClient.Calling.Wait-15	X	X	X	X	X	X
SIPUserAgentClient.Calling.junction-state-7->SIPUserAgentClient.Calling.Wait-14	X					
SIPUserAgentClient.Canceling->SIPUserAgentClient.Waiting Response-9		X				
SIPUserAgentClient.Canceling->SIPUserAgentClient.final-state-2-12					X	
SIPUserAgentClient.Canceling.Wait->SIPUserAgentClient.Canceling.junction-state-11-20					X	
SIPUserAgentClient.Canceling.initial-state-10->SIPUserAgentClient.Canceling.Wait-19		X			X	
SIPUserAgentClient.Canceling.junction-state-11->SIPUserAgentClient.Canceling.Wait-21					X	
SIPUserAgentClient.Init->SIPUserAgentClient.Calling-0	X	X	X	X	X	X
SIPUserAgentClient.Ready->SIPUserAgentClient.Terminating-4				X		X
SIPUserAgentClient.Ready->SIPUserAgentClient.final-state-3-6			X			
SIPUserAgentClient.Ringing->SIPUserAgentClient.Canceling-11		X			X	

From Test Generation to Test Execution

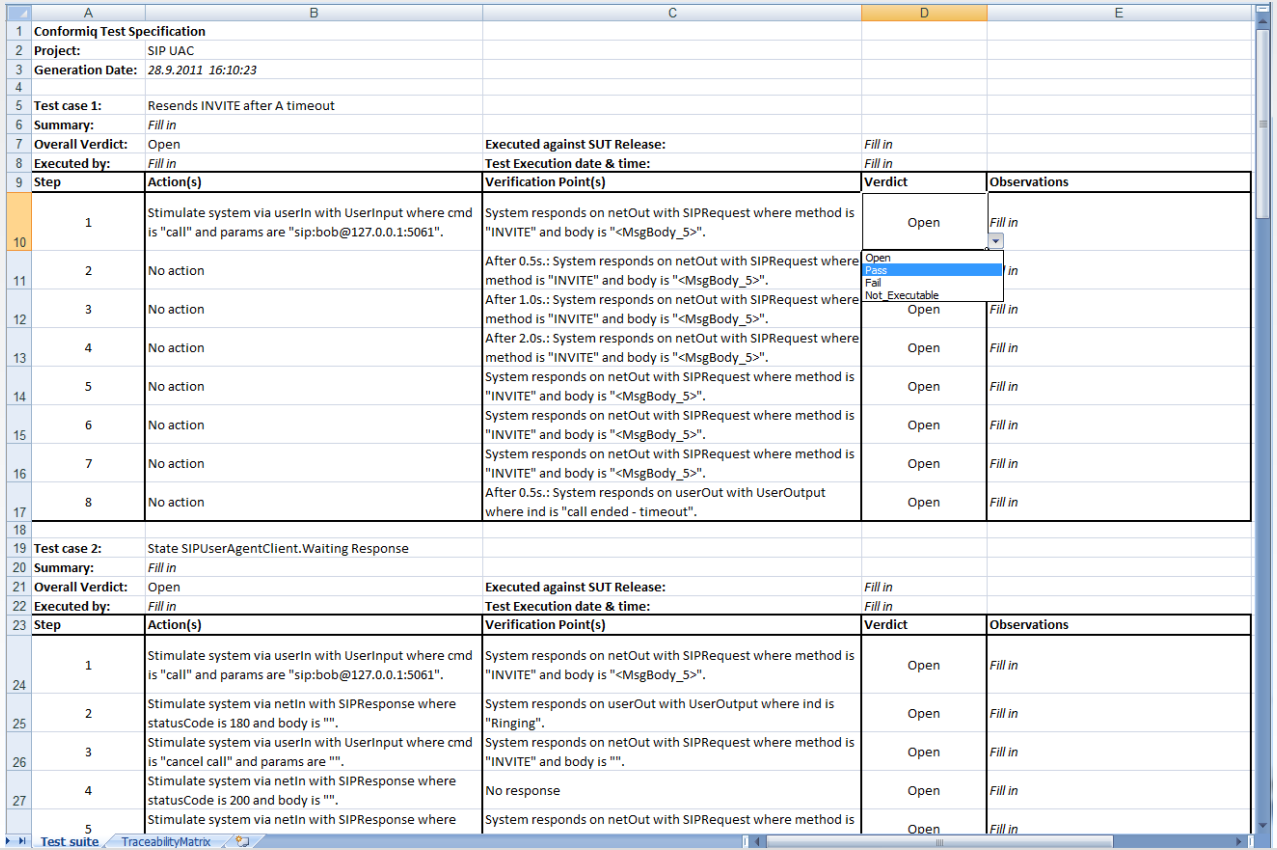
- Modeling and test generation is iterative process
  - Regular reviews of model & tests with stakeholders locate problems *prior to* test execution
- Integration with automated test execution frameworks is realized via scripting backends
  - Full test script generation in case interfaces of model and test framework match
  - In other cases test scripts with stubs for a generic mapping to a test framework
  - Same mechanism can also generate documentation
- Open API allows creation of own or adaptation of generic backends for any testing framework



Rendering of Tests



Example Manual Test Report

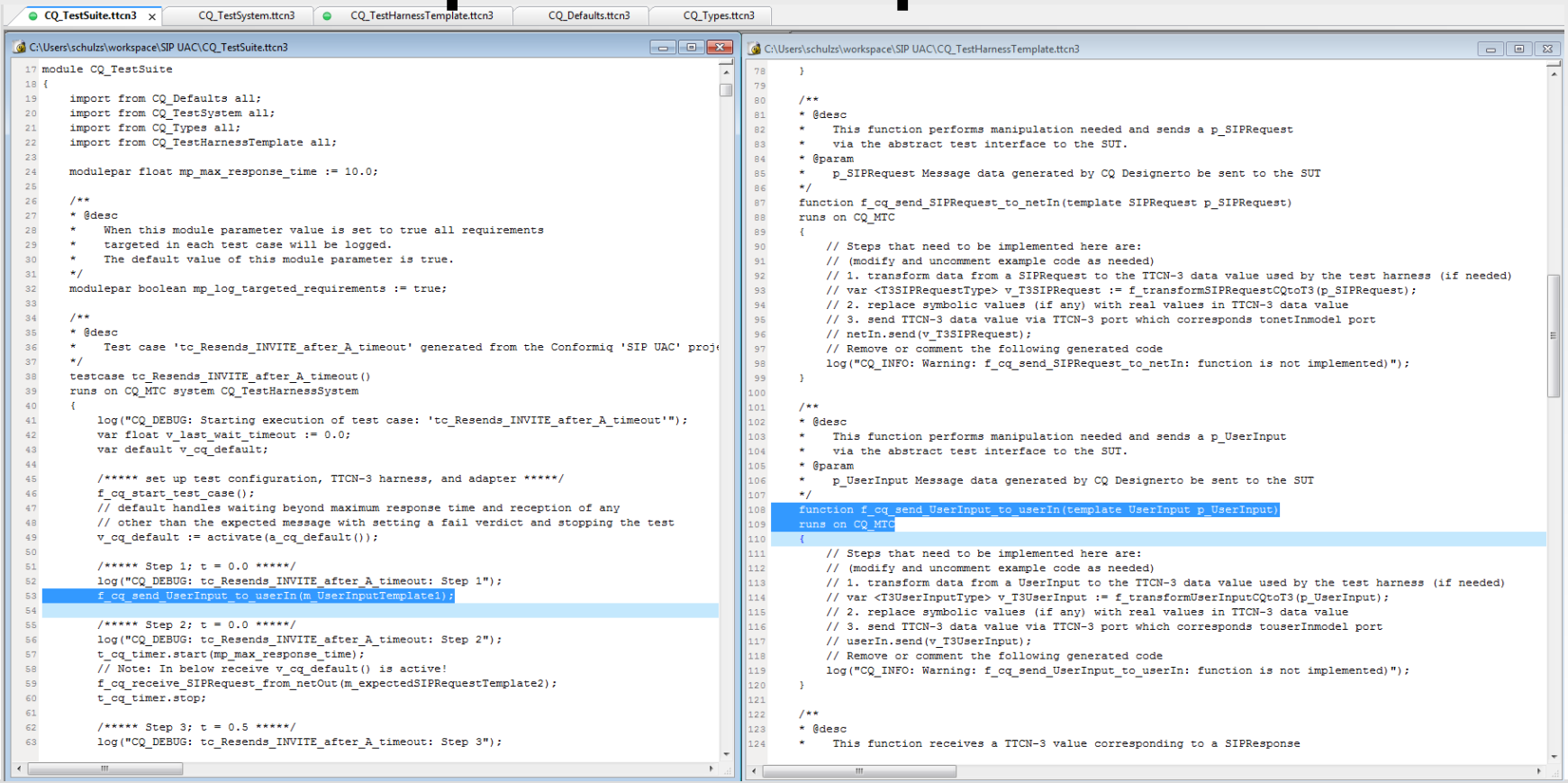


	A	B	C	D	E
1	**Conformiq Test Specification**				
2	**Project:**	SIP UAC			
3	**Generation Date:**	*28.9.2011 16:10:23*			
4					
5	**Test case 1:**	Resends INVITE after A timeout			
6	**Summary:**	*Fill in*			
7	**Overall Verdict:**	Open	**Executed against SUT Release:**	*Fill in*	
8	**Executed by:**	*Fill in*	**Test Execution date & time:**	*Fill in*	
9	**Step**	**Action(s)**	**Verification Point(s)**	**Verdict**	**Observations**
10	1	Stimulate system via userIn with UserInput where cmd is "call" and params are "sip:bob@127.0.0.1:5061".	System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
11	2	No action	After 0.5s.: System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".		*Fill in*
12	3	No action	After 1.0s.: System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
13	4	No action	After 2.0s.: System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
14	5	No action	System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
15	6	No action	System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
16	7	No action	System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
17	8	No action	After 0.5s.: System responds on userOut with UserOutput where ind is "call ended - timeout".	Open	*Fill in*
18					
19	**Test case 2:**	State SIPUserAgentClient.Waiting Response			
20	**Summary:**	*Fill in*			
21	**Overall Verdict:**	Open	**Executed against SUT Release:**	*Fill in*	
22	**Executed by:**	*Fill in*	**Test Execution date & time:**	*Fill in*	
23	**Step**	**Action(s)**	**Verification Point(s)**	**Verdict**	**Observations**
24	1	Stimulate system via userIn with UserInput where cmd is "call" and params are "sip:bob@127.0.0.1:5061".	System responds on netOut with SIPRequest where method is "INVITE" and body is "<MsgBody_5>".	Open	*Fill in*
25	2	Stimulate system via netIn with SIPResponse where statusCode is 180 and body is "".	System responds on userOut with UserOutput where ind is "Ringing".	Open	*Fill in*
26	3	Stimulate system via userIn with UserInput where cmd is "cancel call" and params are "".	System responds on netOut with SIPRequest where method is "INVITE" and body is "".	Open	*Fill in*
27	4	Stimulate system via netIn with SIPResponse where statusCode is 200 and body is "".	No response	Open	*Fill in*
	5	Stimulate system via netIn with SIPResponse where	System responds on netOut with SIPRequest where method is	Open	*Fill in*

Open
Pass
Fail
Not_Executable

Test suite | TraceabilityMatrix

Example Test Script with Stubs




```
module CQ_TestSuite
{
    import from CQ_Defaults all;
    import from CQ_TestSystem all;
    import from CQ_Types all;
    import from CQ_TestHarnessTemplate all;

    modulepar float mp_max_response_time := 10.0;

    /**
    * @desc
    *    When this module parameter value is set to true all requirements
    *    targeted in each test case will be logged.
    *    The default value of this module parameter is true.
    */
    modulepar boolean mp_log_targeted_requirements := true;

    /**
    * @desc
    *    Test case 'tc_Resends_INVITE_after_A_timeout' generated from the Conformiq 'SIP UAC' proj
    */
    testcase tc_Resends_INVITE_after_A_timeout()
    runs on CQ_MTC system CQ_TestHarnessSystem
    {
        log("CQ_DEBUG: Starting execution of test case: 'tc_Resends_INVITE_after_A_timeout'");
        var float v_last_wait_timeout := 0.0;
        var default v_cq_default;

        /***** set up test configuration, TTCN-3 harness, and adapter *****/
        f_cq_start_test_case();
        // default handles waiting beyond maximum response time and reception of any
        // other than the expected message with setting a fail verdict and stopping the test
        v_cq_default := activate(a_cq_default());

        /***** Step 1; t = 0.0 *****/
        log("CQ_DEBUG: tc_Resends_INVITE_after_A_timeout: Step 1");
        f_cq_send_UserInput_to_userIn(m_UserInputTemplate1);

        /***** Step 2; t = 0.0 *****/
        log("CQ_DEBUG: tc_Resends_INVITE_after_A_timeout: Step 2");
        t_cq_timer.start(mp_max_response_time);
        // Note: In below receive v_cq_default() is active!
        f_cq_receive_SIPRequest_from_netOut(m_expectedSIPRequestTemplate2);
        t_cq_timer.stop;

        /***** Step 3; t = 0.5 *****/
        log("CQ_DEBUG: tc_Resends_INVITE_after_A_timeout: Step 3");
```

```
    }

    /**
    * @desc
    *    This function performs manipulation needed and sends a p_SIPRequest
    *    via the abstract test interface to the SUT.
    * @param
    *    p_SIPRequest Message data generated by CQ Designerto be sent to the SUT
    */
    function f_cq_send_SIPRequest_to_netIn(template SIPRequest p_SIPRequest)
    runs on CQ_MTC
    {
        // Steps that need to be implemented here are:
        // (modify and uncomment example code as needed)
        // 1. transform data from a SIPRequest to the TTCN-3 data value used by the test harness (if needed)
        // var <T3SIPRequestType> v_T3SIPRequest := f_transformSIPRequestCQtoT3(p_SIPRequest);
        // 2. replace symbolic values (if any) with real values in TTCN-3 data value
        // 3. send TTCN-3 data value via TTCN-3 port which corresponds tonetInmodel port
        // netIn.send(v_T3SIPRequest);
        // Remove or comment the following generated code
        log("CQ_INFO: Warning: f_cq_send_SIPRequest_to_netIn: function is not implemented)");
    }

    /**
    * @desc
    *    This function performs manipulation needed and sends a p_UserInput
    *    via the abstract test interface to the SUT.
    * @param
    *    p_UserInput Message data generated by CQ Designerto be sent to the SUT
    */
    function f_cq_send_UserInput_to_userIn(template UserInput p_UserInput)
    runs on CQ_MTC
    {
        // Steps that need to be implemented here are:
        // (modify and uncomment example code as needed)
        // 1. transform data from a UserInput to the TTCN-3 data value used by the test harness (if needed)
        // var <T3UserInputType> v_T3UserInput := f_transformUserInputCQtoT3(p_UserInput);
        // 2. replace symbolic values (if any) with real values in TTCN-3 data value
        // 3. send TTCN-3 data value via TTCN-3 port which corresponds touserInmodel port
        // userIn.send(v_T3UserInput);
        // Remove or comment the following generated code
        log("CQ_INFO: Warning: f_cq_send_UserInput_to_userIn: function is not implemented)");
    }

    /**
    * @desc
    *    This function receives a TTCN-3 value corresponding to a SIPResponse
```

Test Generation Summary

- The ability to generate tests automatically is a good start
  - Equally important however is the ability to control & guide test generation and to be able to understand the purpose of a test
  - Coverage criteria selection is one way of steering test generation
- Model and test review clear up misunderstandings & defects earlier where it is (much) cheaper to fix them
  - Makes test quality much higher during test execution
- Finally tests can then be rendered for test execution to any desired scripting language or documentation format

Thank you for your attention!

stephan.schulz@conformiq.com
www.conformiq.com